Omeka Documentation

Release 2.6

Roy Rosenzweig Center for History and New Media

Jan 09, 2020

Contents

This is the developer documentation for Omeka Classic. If you're working on an Omeka theme or plugin, or on Omeka itself, you've come to the right place.

For documentation for end-users and site administrators, or for information about older Omeka versions, please visit the Omeka Classic Documentation instead. For information about Omeka S, please visit the Omeka S pages

CHAPTER 1

What's new in Omeka

1.1 What's new in Omeka 2.0

1.1.1 Principles of Omeka 2.0

People who are familiar with conventions and patterns from Omeka 1.x will see many changes. Here are some ofbroad principles that went into the changes to Omeka 2.0

Record-independent Functionality

The design of Omeka 1.x tended to treat different database records as superficially very different, despite the fact they all inherit from the same class. Thus, we had functions `item()` alongside `collection()`, both of which performed very similar jobs of retrieving data from a record. Similarly, we had `loop_items()` and `loop_collections()`. That pattern extended into our plugins, e.g. `simple_page()` and `loop_simple_pages()`. This created excessive duplication of functionality.

In Omeka 2.0, we instead concentrated on the similarity across all records, designing single functions for similar tasks that work by either passing in a record or by passing in the name of the record type current in a view. For example, to retrieve the id from the current item in a view, use `metadata('item', 'id')`. For a collection: `metadata('collection', 'id')`.

This requires some attention to the names of your models and how they are assigned to the view. The first parameter to `metadata()` will be made singular, with the assumption that the view object has a record assigned to that property name. Similarly, looping assumes a plural form, an sets the current record to the singular name.

New Function Conventions

To make our function names more consistent and easier to understand, we have introduced the following conventions:

- Functions that return an array or object begin with `get_`
- Functions that return a boolean begin with `is_` or `has_`

- Functions do not echo a string, instead they return the string

New Class Naming Conventions

In anticipation of an eventual move to using Zend Framework 2, we have reorganized our directories and class names to conform with those conventions. Abstract classes and interfaces now reflect that status in their names, and class names are generally laid out to read more naturally.

Also, the classes representing records and the table for them are no longer in the same directory. Instead, there is a `Table` directory inside the directory containing the models. The name of the table class should be the model's class name prefixed with `Table_`, e.g. `Table_SimplePagesPage`.

1.1.2 Migrating your code

There are significant changes moving from Omeka 1.5 to Omeka 2.0.

Here, you will find a skeleton of typical tasks that will be required to migrate your code. Consult the *reference section* for code details.

Omeka an Archive?

While archivists can (and many do) use Omeka as a presentation layer to their digital holdings, Omeka is not archival management software. To underscore this fact, we've removed all mention of the word "archive" in the Omeka codebase and filesystem. This will require at least two additional steps when upgrading from earlier versions to 2.0:

1. Rename the archive/files/ directory to /archive/original/:

```
$ mv /path/to/omeka/archive/files/ /path/to/omeka/archive/original/
```

2. Rename the archive/ directory to files/:

```
$ mv /path/to/omeka/archive/ /path/to/omeka/files/
```

Logging and Debugging

Developers of both themes and plugins will need to be aware of the following changes in Omeka's `.htaccess` file and `config.ini` file in `application/config`. Compare your existing files to the new `.htaccess.changeme` and `config.ini.changeme` files

- `.htaccess` now includes an environment variable for development: `# SetEnv APPLICATION_ENV development`
- `config.ini` now includes a setting for the minimal level of error logging. The default level is WARN. DEBUG is the lowest level of priority, and will show all messages. *_log — Log a message.* allows you to set your a priority, and the setting in `config.ini` must be set appropriately for the messages to be saved.

```
; log.priority
; The minimum priority level of messages that should be logged.
; default: Zend_Log::WARN (Logs warnings and above)
log.priority = Zend_Log::DEBUG
```

Note: *debug — Log a message with 'DEBUG' priority.* uses DEBUG priority, so to see messages logged by that function you must set the log priorty to DEBUG in `config.ini`.

Upgrading Plugins

As you look through the lists of typical tasks below, you might also want to consult *Best Practices for Plugin Development*

Typical tasks you will need to do to upgrade your plugins for Omeka 2.0 are:

- Change the classes your controllers and models extend from.
  - Omeka_Controller_Action becomes *Omeka_Controller_AbstractActionController*
  - Omeka_Record becomes *Omeka_Record_AbstractRecord*
- Update any helper functions you use in hooks for filters.
- Change your hook callbacks to have an array passed in. Typically, the expected variable name passed in in version 1.5 (e.g. `$user`) becomes the key for the corresponding data in the array, e.g. `$user = $args['user'];`. See Updating Plugins for 2.0: Hooks and Filters
- Update any filters you use. The third argument must now be an array to fit with the standard above.
- Change the helper functions used in the views * All functions of the form `loop_{record type}`, like `loop_items()`, become `loop("{record type}")`
- Change usage of functions that previously echoed content. For example, `<?php head(); ?>` should now be `<?php echo head(); ?>`.

Database

Record classes

- The abstract class records extend from is now *Omeka_Record_AbstractRecord*, not `Omeka_Record`
- The following callbacks have been **removed**, along with their associated plugin hooks:
  - `beforeSaveForm`
  - `afterSaveForm`
  - `beforeInsert`
  - `afterInsert`
  - `beforeUpdate`
  - `afterUpdate`
  - `beforeValidate`
  - `afterValidate`

  Any logic currently in the `SaveForm`, `Insert`, or `Updare` callbacks should be moved to *beforeSave* or *afterSave*. Anything using the associated hooks should be moved to *before_save_<model>* or *after_save_<model>*. The `Validate` callbacks should be replaced by code in *_validate*.

  A boolean *insert* argument for the `beforeSave` and `afterSave` callbacks and hooks replaces the insert and update-specific versions.

- The `saveForm` and `forceSave` methods are **removed**. Use *Omeka_Record_AbstractRecord::save* instead.

Table classes

- SQL aliases are no longer the initials of the underlying table, they are the full table name (without the prefix). For example, the Items table alias was `i` in Omeka 1.x, but it is now `items`. You can call *Omeka_Db_Table::getTableAlias* to get the alias.
- Table classes can now optionally use the naming pattern `Table_{Record}` instead of `{Record}Table`. Omeka's built-in tables use this new naming scheme.

Built-in records

- The `Entity`, `EntitiesRelations`, and `EntityRelationships` models, and their underlying tables are **removed**. Any code relying on them must be changed or removed.
- *User* now directly stores the name and email data for users that was previously in the `Entity`.
  - The separate first, middle, and last name fields for Users are combined into a single name field.

Built-in mixins

- All mixins now have a prefix of `Mixin_` on their class name, and have a new naming convention:
  - `Ownable` is now *Mixin_Owner*.
  - `Taggable` is now *Mixin_Tag*.
  - `ActsAsElementText` is now *Mixin_ElementText*.
  - `PublicFeatured` is now *Mixin_PublicFeatured*.

ACL and Permissions

- `Omeka_Acl` is **removed**. All references to `Omeka_Acl` should be to `Zend_Acl` instead.
  - `loadRoleList`, `loadResourceList`, and `loadAllowList` were Omeka-specific methods, and are now gone. Now, just directly make individual calls to `addRole()`, `addResource()`, and `allow()`. You no longer need to use `loadResourceList()` to define the privileges for each resource.
  - `checkUserPermission` is also gone. Use `isAllowed` instead:

```
$acl->isAllowed(current_user(), 'Resource', 'privilege');
```

- The `has_permission` global function is replaced by *is_allowed*.

Controllers

- Many methods that were previously directly called on a Controller are now controller helpers instead.
  - The database wrapper methods `findById()`, `getTable('TableName')`, `getDb()` are **removed** in favor of the Db helper:

```
// old: $record = $this->findById();
$record = $this->_helper->db->findById();

// old: $element = $this->getTable('Element')->find($elementId);
$element = $this->_helper->db->getTable('Element')->find($elementId);

// old: $db = $this->getDb();
$db = $this->_helper->db->getDb();
```

- The Db helper is also now used to set the default model name. The `_modelClass` property is **removed** in favor of `setDefaultModelName` from the Db helper:

```
// 1.x
public function init()
{
    $this->_modelClass = 'MyModel';
}

// 2.0
public function init()
{
    $this->_helper->db->setDefaultModelName('MyModel');
}
```

- The `flash`, `flashSuccess`, and `flashError` methods are **removed** in favor of the FlashMessenger helper:

```
$this->_helper->flashMessenger('A neutral message');

$this->_helper->flashMessenger('A success message!', 'success');

$this->_helper->flashMessenger('An error message.', 'error');
```

Omeka_Context

- `Omeka_Context` is **removed**. Resources are instead available directly through `Zend_Registry` or through the bootstrap object:

```
$acl = Zend_Registry::get('bootstrap')->getResource('Acl');
```

Views

Admin Views

- Many new CSS classes are available and should be used to ensure a consistent look and feel across Omeka plugins. It will be helpful to become familiar with them. For example, this is the new code structure to use if you need to create inputs yourself:

```
<div class="field">
    <div class="two columns alpha">
        <label for="some_input" class="required">Some Input Label</label>
    </div>
    <div class="inputs five columns omega">
```

(continues on next page)

(continued from previous page)

```
        <input type="text" name="some_input">
    </div>
</div>
```

- Admin theme now displays an `<h1>` with the title you set for the page. You can remove those from your admin views.
- Use new save panel features. For ease of use in the most common cases, the *Omeka_Form_Admin* is available.

Updating Themes

The number of global functions has been cut nearly in half in Omeka 2.0. This will require many changes to your themes, but will also make the patterns of usage much easier to follow and much more consistent.

Here are a few of the basic tasks for upgrading.

- Change the various metadata-retrieval functions for different record types (e.g., `item()`, `collection()`, etc) to the generalized *metadata — Get metadata for a record.* function.
- Change the loop structure for the various record types (e.g., `loop_items()`, `loop_collections`, etc) to the generalized *loop — Get an iterator for looping over an array of records.* function. Note that the structure changes from:

```
while(loop_items()):
```

  to:

```
foreach(loop('items') as $item):
```

- Use *get_records* when getting sets of any record within a theme. `get_items`, `get_tags`, and `get_collections` are all replaced by `get_records`.
- Change the structure of any arrays passed to *nav — Create a navigation menu of links.*. `nav` now uses the Zend_Navigation component, which changes the way you need to specify the array of nav links. Zend has some more expansive documentation on the available options, but it's pretty simple to convert the old `label => url` pairs to the new style:

```
echo nav(array(
    array('label' => 'Browse All', 'uri' => url('items'))
    array('label' => 'Browse By Tag', 'uri' => url('items/tags'))
));
```

- Update calls to hooks and filters (wherever you use *fire_plugin_hook — Declare the point of execution for a specific plugin hook.* and *apply_filters — Apply a set of plugin filters to a given value.*). Typically, the expected variable name passed in version 1.5 (e.g. `$user`) becomes the key for the corresponding data in the array, e.g. `$user = $args['user'];` See Updating Plugins for 2.0: Hooks and Filters

Changed Function Names

1.2 What's new in Omeka 2.1

Note: Because of an update to Zend Framework, Omeka's PHP version requirement increases to 5.2.11 in Omeka 2.1.

1.2.1 REST API

Omeka 2.1 adds a REST API. The standard Omeka data types like Item, Collection, File, and Item Type are exposed, and plugins can hook into the API as well to add new resources or add data to existing ones.

For in-depth information about the API, see *the REST API section of the documentation*.

1.2.2 New Functions

- `get_record`

1.2.3 New hooks

- *activate*
- *deactivate*
- *admin_collections_show_sidebar*

1.2.4 New filters

- *api_resources*
- *api_extend_<resource>*

1.3 What's new in Omeka 2.2

1.3.1 Shortcodes

Omeka 2.2 adds support for shortcodes, allowing users to embed complex content into free-text fields with a simple syntax. You can read more about using shortcodes on the Omeka Codex or about supporting shortcodes in your own add-ons in the new tutorial *Working with Shortcodes*.

1.3.2 Pluggable Derivative Strategies

The file derivative process is refactored so administrators can switch between different strategies to change how the file derivatives (thumbnails) are created. Strategies can also take in options from the configuration file to change their behavior without requiring a completely new strategy.

Read the new tutorial *File Derivative Strategies* for more information.

1.3.3 Representative Files

Omeka 2.2 adds a new and generalized system for using Files for representing records beyond just Items. This system allows Omeka to display thumbnails in places like the sitewide search that deal with multiple record types at the same time.

Read the new tutorial *Representative Files for Records* for more information.

1.3.4 Extensible fallback thumbnails for Files

Omeka 2.2 expands from the single fallback image for Files (the "blank page" icon) to allow different fallbacks for every MIME type or family of MIME types. Omeka 2.2 itself adds new specific icons for audio, video, and images, and plugins can also add new fallbacks for specific filetypes.

Read the new tutorial *Changing Fallback Thumbnails* for more information.

1.3.5 Classes

- *Omeka_File_Derivative_AbstractStrategy*
- *Omeka_File_Derivative_Creator*
- *Omeka_File_Derivative_Strategy_ExternalImageMagick*
- *Omeka_File_Derivative_Strategy_Imagick*
- *Omeka_Form_Element_Input*
- *Omeka_Validate_HexColor*
- *Omeka_View_Helper_FormInput*
- *Omeka_View_Helper_Shortcodes*
- Removed `Omeka_File_Derivative_Image_Creator`

1.3.6 Interfaces

- *Omeka_File_Derivative_StrategyInterface*

1.3.7 Methods

- *Omeka_Record_AbstractRecord::getFile*

1.3.8 Functions

- *add_file_fallback_image*
- *add_shortcode*
- *record_image*
- *recent_items*

1.3.9 Hooks

- *admin_users_panel_buttons*

1.3.10 Filters

- *admin_files_form_tabs*
- *admin_navigation_users*

1.4 What's new in Omeka 2.3

1.4.1 Automatic page limit support

Controllers extending *Omeka_Controller_AbstractActionController* can now easily set their controllers to use the admin and public-side page limits configured by the site administrator in Appearance Settings. The user-set `per_page` GET parameter is also now available to all controllers using the built-in pagination support.

To use the configured limits, set the *Omeka_Controller_AbstractActionController::$_browseRecordsPerPage* property to `Omeka_Controller_AbstractActionController::RECORDS_PER_PAGE_SETTING`.

In general, plugins that previously were overriding the `_getBrowseRecordsPerPage` method or manually selecting a hardcoded page size limit should use this new constant instead.

To maintain backwards compatibility, controllers will still default to performing no pagination of the results.

1.4.2 API

All "browse" actions within the API now support `sort_field` and `sort_dir` GET parameters to allow custom sorting of the result set. The parameters behave the same way as for normal, non-API browse actions.

1.4.3 Classes

- *Omeka_File_Derivative_Strategy_GD*
- Removed `Omeka_View_Helper_AbstractSearch`

1.4.4 Methods

- *Mixin_ElementText::getAllElementTextsByElement*

1.4.5 Filters

- *<model>s_browse_default_sort*
- *<model>s_browse_per_page*
- *search_element_texts*
- *search_form*
- *search_form_default_query*
- *search_form_default_query_type*
- *search_form_default_record_types*

1.5 What's new in Omeka 2.4

1.5.1 PHP 5.2 support dropped

Keeping up with adjusted support by our various third-party libraries, the lack of upstream support or updates, and its substantial age, Omeka 2.4 no longer supports PHP 5.2. PHP 5.3.2 is required at a minimum.

1.5.2 REST API changes

The item resource's index now accepts `tags`, `excludeTags`, `hasImage`, and `range` as GET parameters. These behave the same as their preexisting counterparts on the regular items browse page.

1.6 What's new in Omeka 2.5

1.6.1 Unsaved changes warnings

Starting with version 2.5, Omeka will warn users on the admin interface if they attempt to navigate away from a page without saving their changes first. This is accomplished with JavaScript that stores the initial state of every POSTed `<form>` element on an admin page. Before navigation, Omeka checks if the current state of the form differs from its initial state, and if there's a difference on any form (except one being submitted), then we show the warning.

For most forms this should work fine out of the box with no alterations, but for some that are using non-standard methods for mananging the form, we provide some ways to alter the process.

The original state of the form is stored as HTML5 data on the form element with the name `omekaFormOriginalData`. Developers can alter this if the state of a form isn't really initialized until some time after the document's `ready` event.

Another HTML5 data attribute can be used to unconditionally trigger the warning: `omekaFormDirty`. If this is present on the form element and set to true, Omeka will always treat the form state as changed. This can be useful to account for changes that don't actually cause the form's state to be changed.

A custom event `o:before-form-unload` is fired on every form before it's checked. Developers can use this to alter the form state or set the "dirty" flag just in time for the check to occur.

To opt a form out of the unsaved warning system completely, either make sure it isn't using the POST method, or add the class `disable-unsaved-warning` to the `<form>`.

1.6.2 `display_title` property for records

In previous versions of Omeka, getting a title for a record suitable for use in a table or in a `<title>` element could involve checking several pieces of metadata, manually stripping HTML if it was present, and even then could run into problems, particularly when dealing with HTML data.

Version 2.5 introduces a `display_title` property, usable with the *`metadata`* helper function for Items, Collections, and Files. `display_title` willautomatically handle and remove HTML formatting and decoding entities in titles, as well as falling back to a default if no Dublin Core Title exists.

The implementation is shared between the record types, and is actually provided by the *`ElementText mixin`*.

All applications needing a "simple" version of a title for usage in places like attributes, tables, citations, link text, and other places where some content is required or where HTML tags are not allowed or desirable are suggested to use the `display_title` property through the `metadata` helper.

1.6.3 User control of thumbnail type

Omeka 2.5 adds a "Use Square Thumbnails" setting in the appearance settings, to allow administrators to choose between square thumbnails and "regular," aspect-ratio-preserving thumbnails.

Themes and plugins that want to opt in to this behavior will need to change some code if they're displaying their own thumbnails: instead of specifying `'thumbnail'` or `'square_thumbnail'` as the image type, they should

pass `null` (or omit the argument, where possible). *`record_image`* and *`item_image`* have been updated to allow passing null for their imageType arguments.

1.6.4 `ignore_unknown` option for `metadata()`

The *`metadata`* function throws an exception if a developer requests an Element which is unknown in the system. In some particular cases such as Item Type Metadata where users can freely add and delete elements, this can cause undesirable and sometimes confusing errors.

To account for this, a new option is now available for `metadata: ignore_unknown`. When set to true, a call requesting a nonexistent Element will simply return nothing rather than causing an error.

1.6.5 API: simple search enabled for items

The `search` GET parameter is now allowed when browsing for items in the REST API.

1.6.6 New "next" and "previous" filters for items

Many administrators and users prefer the "next" and "previous" item links to have different behavior than the default of going to the next and previous record by ID, globally. To allow this more easily, Omeka 2.5 adds two new filters: *item_next* and *item_previous*, allowing a plugin to alter where those links will go.

1.7 What's new in Omeka 2.6

CHAPTER 2

Tutorials

2.1 Plugin Basics

2.1.1 Plugin Directory Structure

The basics

There are a few things that any plugin must contain: the plugin folder, the plugin INI file, and the main plugin PHP file.

Plugin folder

An Omeka plugin must have one top-level folder that all the other folders and files are contained in. This folder is what users end up putting in their installation's `plugins/` directory.

Omeka uses the folder names to keep track of which plugins are installed and activated, so once you pick a folder name, you should stick with it. The folder name also determines things like the default URLs that will be used if your plugin adds pages of its own.

There are a few principles you should keep in mind when picking a plugin folder name. The name should be:

- **Unique**: the name must not conflict with an existing plugin name
- **Simple**: the name should be concise and to-the-point
- **Descriptive**: the name should describe the plugin, at least to some degree
- **CamelCased**: the name must contain no spaces, and the first letter of every "word" should be capitalized (e.g.: `SimplePages`, `ExhibitBuilder`)

In addition, the human-readable name you set in the plugin.ini should usually be the "normal" equivalent of your folder name, so users can easily tell which folder goes with which plugin.

Main plugin file

The most important file in most plugins is the main plugin file. This is the file Omeka actually loads when loading the plugin, and its where all the hooks and filters are registered and defined.

The main plugin file must have the same name as the plugin folder, with `Plugin.php` tacked onto the end. For example, a plugin in the folder `Foo` will have a main plugin file named `FooPlugin.php`.

The `FooPlugin.php` file must contain a PHP class with the same name (without the `.php`). Generally, this class will be a subclass of *`Omeka_Plugin_AbstractPlugin`*. For more information about how to write and use a main plugin class, see *Understanding Omeka_Plugin_AbstractPlugin*.

Note: Omeka also supports another, older, type of main plugin file, at `plugin.php`. This file is plain PHP code and is not required to contain a class. Current plugins mostly use the class-based main file, however.

Plugin information file (`plugin.ini`)

The INI file for the plugin, `plugin.ini` is used both by Omeka itself and the plugin directory on omeka.org. Most of what the INI file contains is metadata about the plugin, like the name of the plugin, the author, and a description. However, some fields are internally used by Omeka when loading the plugin.

The first line in any plugin.ini file should be `[info]`. The rest of the file consists of various fields, one on each line. The possible fields are:

name The name of the plugin.

author The plugin's author.

description A short description of the plugin.

version The plugin's version number. Omeka uses this to determine when a user has upgraded a plugin.

license The software license the plugin is released under. This key is used by the plugins directory on omeka.org.

link A URL for information or documentation about the plugin.

support_link A URL for users to go to for help with the plugin or to report an issue.

omeka_minimum_version The minimum version of Omeka needed to run the plugin. Omeka will refuse to install or load any plugin that requires a higher Omeka version.

omeka_target_version The most recent version the plugin is intended to work on and was tested on. Omeka itself does not report or use this value, but it does appear on the omeka.org plugin directory.

required_plugins Other plugins that *must* be installed for the plugin to work. Plugins must be specified by their folder names, and multiple required plugins are separated by commas. Omeka will refuse to load the plugin if any of the required plugins are missing, and will make sure the plugin loads after all plugins it requires.

optional_plugins Other plugins that the plugin *can* work with. Plugins must be specified by their folder names, and multiple required plugins are separated by commas. Omeka will make sure the plugin loads after all plugins it requires.

Here is an example plugin.ini that uses all the possible fields:

```
[info]
name = "Foo"
author = "My Name"
description = "Does this, that, and the other thing."
```

(continues on next page)

(continued from previous page)

```
version = "1.0"
license = "GPLv3"
link = "http://example.com/my-plugin"
support_link = "http://example.com/my-plugin/support"
omeka_minimum_version = "2.0"
omeka_target_version = "2.2"
required_plugins = "ExhibitBuilder,SimplePages"
optional_plugins = "Bar,Baz"
```

Other common folders

For a plugin that merely needs to use hooks and filters to modify existing things in Omeka, the bare basics are enough, and pretty much everything can be done within the main plugin file alone.

Adding Pages: Controllers and Views

Plugins that want to add totally new pages to Omeka must do so using Controllers and Views.

Controllers are PHP classes that handle basically the "glue" necessary to make a page work: retrieving data from the database, determining if a user has permission to see what's on the page, and other necessary tasks for getting whatever data is necessary for the page to be displayed.

Views are PHP files containing code for displaying a page. Typically, a view will take data set by its corresponding controller and print it out as HTML.

Controllers

The `controllers/` folder within a plugin contains controller classes. In plugins, the *internal* name of a controller class must prepend the name of the plugin, but the name of the file must not. For example, to create an "index" controller for MyPlugin, you would create a class named `MyPlugin_IndexController` and place it at `controllers/IndexController.php`.

Controllers extend an Omeka class *`Omeka_Controller_AbstractActionController`*, but the controller system is all built off of Zend Framework. See the Zend documentation on controllers for some basic information on controllers.

Views

The `views/` folder contains view files. Generally, each view file corresponds with a *controller* and an *action* (together, a controller and action basically describe one page).

Views are bare PHP files, not classes: they simply contain code to display a page. (Some views actually display just part of a page or something else; these are called "partials.")

In a plugin, the `views/` folder has three subfolders:

- `views/admin/` for view files visible only in the Omeka admin interface
- `views/public/` for view files visible only on public pages
- `views/shared/` for view files available on both the admin and public sides

Under each subfolder the structure is the same: a folder for each controller, and inside that folder, a view file for each action. Names of controllers and actions, when used in views, are written in hyphen-separated-lowercase. As an example, Public-facing views for the previous example's `MyPlugin_IndexController` controller would correspond to a folder `views/public/index`.

Custom database tables: Models

Plugins with simple needs can often store data without needing to create their own database tables, just using existing Omeka systems like options or element texts. For plugins with different or more complicated needs, they can create their own tables and manage them with models in the `models/` folder.

Two types of files are used as models: *record classes* and *table classes*. More information on each is available at their respective links.

Additional code: Libraries

The `libraries/` folder simply contains additional PHP classes used by the plugin. Files under this folder are automatically set up to be autoloaded using the PSR-0 file/folder structure. Code here can include classes written specifically for the plugin, as well as external libraries.

Internationalization

The `languages/` folder contains translations of the plugin's text into different languages. For more information about internationalization, see the *Internationalization* page.

2.1.2 Best Practices for Plugin Development

Method Naming Conventions

Method names must be in camelCase (e.g., `getItem()`).

For private or protected methods, prefix the method name with underscore (_).

See also the Zend Framework Coding Standards.

Maintain expected behaviors in Omeka

Omeka defines some hooks that operate on every record. For example, Omeka's *`Omeka_Record_AbstractRecord::save`* method fires hooks before and after saving. In most cases, you don't need to override it, but if you do, make sure you include `$this->runCallbacks('beforeSave', $callbackArgs);` and `$this->runCallbacks('afterSave', $callbackArgs);` where appropriate.

Similarly, whenever you override methods from abstract classes in Omeka, make sure that you have studied the parent methods' code to include all the expected callback. This will allow other developers to work with the expected behaviors of the objects.

Also, if you use any non-standard routing in your plugin, you must override *`Omeka_Record_AbstractRecord::getRecordUrl`* so that it returns the correct url to the record. Compare the `getRecordUrl()` method on the `SimplePagesPage` model in the "Simple Pages" plugin.

Database changes

Omeka 2.0 switched MySQL database engines from MyISAM to InnoDB. We recommend that you set all plugin tables to InnoDB. Existing plugins may alter their tables during the upgrade hook:

```
public function hookUpgrade($args)
{
    if (version_compare($args['old_version'], $newPluginVersion, '<')) {
        // Change the storage engine to InnoDB.
        $sql = "ALTER TABLE {$this->_db->TableName} ENGINE = INNODB";
        $this->_db->query($sql);
    }
}
```

Record API

We've made some major changes to the record API. The following internal callbacks have been removed:

- beforeSaveForm
- afterSaveForm
- beforeInsert
- afterInsert
- beforeUpdate
- afterUpdate
- beforeValidate
- afterValidate

As such, the following plugin hooks have been removed, where "*" is "record" or a specific record name (e.g. "item", "collection"):

- before_save_form_*
- after_save_form_*
- before_insert_*
- after_insert_*
- before_update_*
- after_update_*
- before_validate_*
- after_validate_*

By removing these callbacks we give you full control over the timing of execution. Any logic that's currently in the *SaveForm*, *Insert*, and *Update* callbacks should be moved to `beforeSave()` and `afterSave()`. Any logic that's currently in the *Validate* callbacks should be moved to `_validate()`. For example:

```
// Note the order of execution.
public function beforeSave($args)
{
    if ($args['insert']) {
        // Do something before record insert. Equivalent to beforeInsert.
```

(continues on next page)

(continued from previous page)

```
    } else {
        // Do something before record update. Equivalent to beforeUpdate.
    }

    // Do something before every record save.

    if ($args['post']) {
        // Do something with the POST data. Equivalent to beforeSaveForm.
    }
}

// Note the order of execution.
public function afterSave($args)
{
    if ($args['insert']) {
        // Do something after record insert. Equivalent to afterInsert.
    } else {
        // Do something after record update. Equivalent to afterUpdate.
    }

    // Do something after every record save.

    if ($args['post']) {
        // Do something with the POST data. Equivalent to afterSaveForm.
    }
}
```

Note that the signature of the `beforeSave()` and `afterSave()` has changed to `beforeSave($args)` and `afterSave($args)`, with no type specified for `$args`. To adhere to strict standards, existing beforeSave and afterSave methods should reflect that change.

Another change is that `Omeka_Record_AbstractRecord::saveForm()` has been merged into *`save()`*. Using `save()` to handle a form in your controller can be done like this:

```
public function editAction()
{
    // Check if the form was submitted.
    if ($this->getRequest()->isPost()) {
        // Set the POST data to the record.
        $record->setPostData($_POST);
        // Save the record. Passing false prevents thrown exceptions.
        if ($record->save(false)) {
            $successMessage = $this->_getEditSuccessMessage($record);
            if ($successMessage) {
                $this->_helper->flashMessenger($successMessage, 'success');
            }
            $this->_redirectAfterEdit($record);
        // Flash an error if the record does not validate.
        } else {
            $this->_helper->flashMessenger($record->getErrors());
        }
    }
}
```

Use View Helpers instead of global functions

View helpers are preferred alternatives to global theming functions. They provide a convenient interface (called directly from the view object) to logic and/or markup that's commonly used in view scripts. If you find yourself using global functions or static methods to support your views, consider using view helpers instead.

First, you must add your view helper directory path to the stack during plugin initialization:

```
public function hookInitialize()
{
    get_view()->addHelperPath(dirname(__FILE__) . '/views/helpers', 'PluginName_View_
↪Helper_');
}
```

Replace *PluginName* with your plugin's name. The helpers/ directory may be anywhere in your plugin's directory structure, but we recommend that you place it in the views/ directory for consistency.

Then create your view helper file in the helpers/ directory (named something like ViewHelperName.php) and in that file write your view helper class:

```
class PluginName_View_Helper_ViewHelperName extends Zend_View_Helper_Abstract
{
    public function viewHelperName($arg1, $arg2)
    {
        // Build markup.
        return $markup;
    }
}
```

Note the use of UpperCamelCase and lowerCamelCase. The `viewHelperName()` method can accept any number of arguments and should return something, most often markup. You may add `__construct()` to the class if the helper needs a one-time setup (e.g. to assign class properties). The constructor will not be called on subsequent calls to the helper.

Now you can call your view helper directly in your view script like so:

```
<p><?php echo $this->viewHelperName() ?></p>
```

Use View Partials

View partials let you separate out parts of long or complicated views into separate files. For example, if you have a browse view that allows different ordering, it is best to use view partials to separate the code for the different orderings to be in different partials. For example:

```
<?php if (isset($_GET['view']) && $_GET['view'] == 'hierarchy'): ?>
    <?php echo $this->partial('index/browse-hierarchy.php', array('simplePages' =>␣
↪get_simple_pages_for_loop())); ?>
<?php else: ?>
    <?php echo $this->partial('index/browse-list.php', array('simplePages' => get_
↪simple_pages_for_loop())); ?>
<?php endif; ?>
```

When using hooks that add markup to views, such as *admin_items_show*, consider using partials instead of outputting markup directly in the callback.

Use Jobs instead of Processes

We highly recommend that all processes that may run longer than a typical web process are sent to a job. The job will mediate the process, reducing the chance of timeout and memory usage errors that can happen even with the best written code. To run a job just write a class that contains the code to run, like so:

```
class YourJob extends Omeka_Job_AbstractJob
{
    public function perform()
    {
        // code to run
    }
}
```

You have two options on how to run the code: *default* and *long-running*. The default way is intended to run processes that, though are more processor-intensive than the typical web process, are usually not in danger of timing out. You can run these processes like so:

```
Zend_Registry::get('bootstrap')->getResource('jobs')->send('YourJob');
```

Your other option is intended for processes that will most likely result in a timeout error if run as a normal web script. Processes that import thousands of records or convert hundreds of images are examples of such processes. You can run these processes like so:

```
Zend_Registry::get('bootstrap')->getResource('jobs')->sendLongRunning('YourJob');
```

It's important to note that nothing that uses the job system should assume or require synchronicity with the web process. If your process has to be synchronous, it shouldn't be a job.

Load Resources for Jobs At Will

In previous versions, long running processes were fired directly through a background process via `ProcessDispatcher::startProcess()`, which loaded resources (e.g. Db, Option, Pluginbroker) in phases. Phased loading is now removed in favor of loading resources when needed.

When using the background process adapter for your jobs (typically used for long running jobs), the following resources are pre-loaded for you: Autoloader, Config, Db, Options, Pluginbroker, Plugins, Jobs, Storage, Mail. If you need other resources, load them like so in your job:

```
Zend_Registry::get('bootstrap')->bootstrap('YourResource');
```

Setting Up Your Plugin's Config Page

You can provide a configuration page that will be linked from the Plugins page. Upon installing or upgrading a plugin that has a configuration form, Omeka redirects the user to the configuration page.

You provide the markup for your configuration form by using the *config_form* hook. Plugins often just `require` or `include` a config_form.php file, but any form of output will work. That hook is also the best place to read the current value of any options or other data that needs to be set on the form.

Omeka will provide a framework for most of the page for you: the page heading, the `<form>` element and the submit button are all provided and you do not need to write them.

To get form elements styled in a way that is consistent with the rest of the Omeka admin interface, you just need to use a simple set of CSS classes:

```
<div class="field">
    <div class="two columns alpha">
        <?php echo get_view()->formLabel('some-element', __('Some Element')); ?>
    </div>
    <div class="inputs five columns omega">
        <p class="explanation">
            <?php echo __('Any explanatory text about the form element.'); ?>
        </p>
        <?php echo get_view()->formInput('some-element', $someElementValue); ?>
    </div>
</div>
```

The submitted data will be sent back in a POST, and you can use the *config* hook to handle that data and actually update the options or other settings.

Building Forms in Admin

Omeka 2.0 admin interface works with modern CSS and design practices, including responsive design. Omeka 2.0 therefore also includes a *Omeka_Form_Admin* class to help you quickly and easily build simple forms. It should be suitable for building basic add/edit forms. The SimplePages plugin makes uses it, can can offer a good example of usage.

It is best to put your form-building logic into your controller, e.g. in a `_getForm()` method. The *Omeka_Form_Admin* class works basically as follows.

If you are editing an existing record, instantiate it like so: `$form = new Omeka_Form_Admin(array('record'=>$record);`

If the form is for a record (which is typically the case), pass the record as one of the options. Additionally, if you want a link to the record's public page on the admin side, pass `'hasPublicPage'=>true` as an option:

```
$options = array('record'=>$record, 'hasPublicPage'=>true);
```

Other options available for *Omeka_Form_Admin* are:

string type Often, this will be the record type (e.g. 'simple_pages_page'), but can be anything. Hooks for the save panel follow the type that you give. See *admin_<type>_panel_buttons* and *admin_<type>_panel_fields*.

string editGroupCssClass Change the CSS classes for the 'main' edit area. This should rarely be necessary.

string saveGroupCssClass Change the CSS classes for the save panel. This should rarely be necessary.

To add your form elements to the main editing area, use *Omeka_Form_Admin::addElementToEditGroup*. You can either pass in a `Zend_Form_Element` you have already built, or pass in the parameters to build the element as if you were creating one. For example, creating a text input looks like this:

```
$form->addElementToEditGroup(
    'text', 'title',
    array(
        'id'=>'simple-pages-title',
        'size'  => 40,
        'value' => metadata($page, 'title'),
        'label' => 'Title',
        'description' => 'The title of the page (required).',
        'required' => true
    )
);
```

The first argument specifies the element type (text, textarea, etc.). The second gives the name to be used on the element in the form. The third gives a keyed array of various attributes for the element, as well as a label and a description.

If you build the Zend_Form_Element yourself, you can simply pass that in as the first parameter and leave the rest empty.

In some cases, it makes sense to add an element directly to the save panel on the right. This should be reserved for small, peripheral data, such as whether a record is public or featured, if the model implements those features.

Doing so works similarly, using the *Omeka_Form_Admin::addElementToSaveGroup* method:

```
$form->addElementToSaveGroup(
    'checkbox', 'is_published',
    array(
        'id' => 'simple_pages_is_published',
        'values' => array(1, 0),
        'checked' => metadata($page, 'is_published'),
        'label' => 'Publish this page?',
        'description' => 'Checking this box will make the page public and it will␣
↪appear in Simple Page navigation.'
    )
);
```

As with `addElementToEditGroup()`, you can build the element yourself and pass it as the first parameter.

For more complex form requiring tabs and a variety of sections, you'll want to familiarize yourself with *Understanding the Admin CSS*.

See also workingwiththeadmintheme, which includes more details of how the HTML is constructed, and the CSS classes involved.

Search

Omeka 2.0 allows any record to be full-text searchable, not just items, but also files, collections, exhibits, etc. This includes records implemented by your plugin.

Individual record indexing and bulk-indexing will only work on record types that have been registered via the new *search_record_types* filter:

```
public function filterSearchRecordTypes($searchableRecordTypes)
{
    // Register the name of your record class. The key should be the name
    // of the record class; the value should be the human readable and
    // internationalized version of the record type.
    $searchableRecordTypes['YourRecord'] = __('Your Record');
    return $searchableRecordTypes;
}
```

Follow this template to make your record searchable:

```
class YourRecord extends Omeka_Record_AbstractRecord
{
    // Add the search mixin during _initializeMixins() and after any mixins
    // that can add search text, such as Mixin_ElementText. Doing this
    // tells Omeka that you want this record to be searchable.
    protected function _initializeMixins()
    {
        // Add the search mixin.
```

(continues on next page)

(continued from previous page)

```
        $this->_mixins[] = new Mixin_Search($this);
    }

    // Use the afterSave() hook to set the record's search text data.
    protected function afterSave($args)
    {
        // A record's search text is public by default, but there are times
        // when this is not desired, e.g. when an item is marked as
        // private. Make a check to see if the record is public or private.
        if ($private) {
            // Setting the search text to private makes it invisible to
            // most users.
            $this->setSearchTextPrivate();
        }

        // Set the record's title. This will be used to identify the record
        // in the search results.
        $this->setSearchTextTitle($recordTitle);

        // Set the record's search text. Records that implement the
        // Mixin_ElementText mixin during _initializeMixins() will
        // automatically have all element texts added. Note that you
        // can add multiple search texts, which simply appends them.
        $this->addSearchText($recordTitle);
        $this->addSearchText($recordText);
    }

    // The search results need a route to the record show page, so build
    // a routing array here. You can also assemble the URL yourself using
    // the URL view helper and return the entire URL as a string.
    public function getRecordUrl($action)
    {
        if ('your-show-action' == $action) {
            return $yourCustomRecordShowUrl;
        }
        return array(
            'module' => 'your-module',
            'controller' => 'your-controller',
            'action' => $action,
            'id' => $this->id,
        );
    }
}
```

Once this is done you should enable the new search record type and re-index all records in your admin interface, under Settings > Search.

Customizing Search Type

Omeka now comes with three search query types: keyword (full text), boolean, and exact match. Full text and boolean use MySQL's native full text engine, while exact match searches for all strings identical to the query.

Plugin authors may customize the type of search by implementing the *search_query_types* filter. For example, if you want to implement a "ends with" query type that searches for records that contain at least one word that ends with a string:

```
public function filterSearchQueryTypes($queryTypes)
{
    // Accept an array and return an array.
    function your_search_query_types_callback($queryTypes)
    {
        // Register the name of your custom query type. The key should be
        // the type's GET query value; the values should be the human
        // readable and internationalized version of the query type.
        $queryTypes['ends_with'] = __('Ends with');
        return $queryTypes;
    }
}
```

Then you must modify the search SQL using the *search_sql* hook, like so:

```
public function hookSearchSql($args)
{
    $params = $args['params'];
    if ('ends_with' == $params['query_type']) {
        $select = $args['select'];
        // Make sure to reset the existing WHERE clause.
        $select->reset(Zend_Db_Select::WHERE);
        $select->where('`text` REGEXP ?', $params['query'] . '[[:>:]]');
    }
}
```

Remember that you're searching against an aggregate of all texts associated with a record, not structured data about the record.

2.1.3 Understanding Hooks

Hooks and *Filters* are the two ways to modify and extend Omeka's capabilities. A hook "fires" in response to some action taking place in Omeka, such as saving a record, deleting a record, rendering some kinds of pages, installing a plugin, etc.

When a hook fires, it sends relevant data to a callback function. The callback function then performs any necessary actions with the data before Omeka moves on to the next callback registered for the hook, or if no more callback remain Omeka continues on with its own processing.

To take a common example, plugins will often add new kinds of information about items in Omeka. Thus, when items are saved, various plugins will need to update their own records to reflect the new changes. Since Omeka itself knows nothing of the extended functionality, this takes place through the *before_save_item* and *after_save_item* hooks.

Those callbacks to those hooks get data about the record and about the `$_POST` data submitted. The callbacks define by each plugin run in turn, updating their own records. After all of the callbacks have run, Omeka finally completes its process for saving the item.

Another common example is the *install* hook. This fires when a plugin is activated, so unlike the previous example only one callback is run – the callback defined by that plugin. This is where a plugin would create its database tables, set options, and do any other initial setup.

Hook callbacks always receive one array of arguments (though that array may be empty). They never return a value. If content is to be printed to the screen, they should use `echo` internally.

2.1.4 Understanding Filters

Hooks and filters are the two ways to modify and extend Omeka's capabilities. Filters are a way for data in Omeka to be passed among different plugins that want to add to it or modify it in some way before final processing by Omeka. Typical examples include adding navigation links and changing default behaviors.

Filter Basics

Filters work similarly to hooks, but are focused on modifying data rather than simply executing code or producing output in some particular place. Filter functions always recieve two parameters.

The first parameter is always the value being filtered. The documentation page for a filter will describe this parameter under the heading "Value." A filter function is expected to take the value passed in that parameter, modify it somehow (though this can include completely replacing it or leaving it unchanged), and return the result. That returned value is what's passed to any later functions attached to the same filter, and eventually back to the code that applied the filter in the first place. Since filters work this way, there's aclear distinction from how hooks work in this case: filter functions filter functions **must** return a value.

The second parameter to a filter function is conventionally called `$args`. Just like with hooks, `$args` is an array of arguments, optional useful pieces of extra data passed along with the hook to provide context or access to objects that might likely be needed by the filter function. The arguments passed with a filter are documented under the heading "Arguments."

Using Filters in Plugins

There are two basic ways plugins can use filters, and both work fairly similarly to hooks: the plugin defines a function and attaches it to the filter being used.

Using Omeka_Plugin_AbstractPlugin

The usual and current way of using filters is to use the features built into the *Omeka AbstractPlugin class*. You add the name of the filter you want to use to the `$_filters` array in the plugin class, and then add a public method to the class with the CamelCased name of the filter and `filter` prepended. As an example, to use the *search_form* filter, the code you would add would look like this:

```
protected $_filters = array(
    'search_form',
);

// ...

public function filterSearchForm($form, $args)
{
    // Modify or replace $form and return the results
}
```

Using `add_filter` directly

Though the preferred method is to use the AbstractPlugin when using filters in a plugin, plugins that are built with the old legacy plugin.php-based code or that need to add filters dynamically may need to directly use Omeka's *`add_filter`* function.

`add_filter` works very similarly to *`add_plugin_hook`*: you simply pass it the name of the filter to attach to and the name of a function you've defined. Using the same `search_form` example, the code for adding a filter this way is pretty similar:

```
add_filter('search_form', 'myplugin_search_form');

function myplugin_search_form($form, $args)
{
    // Modify or replace $form and return the results
}
```

One added wrinkle is that you can also pass in the third argument a "priority" that determines when your attached function is run relative to other functions attached to the same filter. Priority is an integer, and passing a lower priority means your function will run earlier. For most cases, the default priority setting is fine and you can simply omit the third argument.

2.2 Essential Classes in Omeka

2.2.1 Understanding Omeka_Plugin_AbstractPlugin

`Omeka_Plugin_AbstractPlugin` is designed to streamline common tasks for a plugin, such as defining hook and filter callbacks and setting options when the plugin is installed.

To use it, simply create a class for your plugin as its own file in the plugins directory:

```
// In plugins/YourPlugin/YourPluginPlugin.php
class YourPluginPlugin extends Omeka_Plugin_AbstractPlugin
{}
```

Hook and filter callbacks are defined with an array. One change introduced in Omeka 2.0 is arbitrary hook and filter callback names. Before, all callback names followed a predetermined format. While this format remains an option, now a corresponding key in `$_hooks` and `$_filters` will be interpreted as the name of the callback method.

```
protected $_filters = array('admin_navigation_main',
  'public_navigation_main',
  'changeSomething' => 'display_setting_site_title',
  'displayItemDublinCoreTitle' => array(
      'Display',
      'Item',
      'Dublin Core',
      'Title',
  ));

  protected $_hooks = array('install', 'uninstall');
```

When installing your plugin, there might be options that need to be set. You can do this easily within the install callback by adding an `$_options` array and calling *`Omeka_Plugin_AbstractPlugin::_installOptions`* from the install callback, and *`Omeka_Plugin_AbstractPlugin::_uninstallOptions`* in the uninstall callback. The array is name-value pairs for the name of the option and initial value.

```
protected $_hooks = array('install', 'uninstall');

protected $_options = array('my_plugin_option'=>'option_value');

public function install() {
```

(continues on next page)

(continued from previous page)

```
    $this->_installOptions();
}

public function uninstall() {
    $this->_uninstallOptions();
}
```

When creating tables for your plugin in the install hook, use the _db property. Omeka will convert your model names into the appropriate table names.

```
public function hookInstall() {
    $db = $this->_db;
    $sql = "
    CREATE TABLE IF NOT EXISTS `$db->MyPluginModel` (
      `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
      `modified_by_user_id` int(10) unsigned NOT NULL,
      . . .
      ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci";
    $db->query($sql);

}
```

2.2.2 Understanding Omeka_Record_AbstractRecord

Omeka_Record_AbstractRecord is the heart of the ORM for Omeka. It is an abtract class that is extended to create the models for Omeka and its plugins. Each class extending Omeka_Record_AbstractRecord corresponds to a database table, with each row representing an individual record. Table columns correspond to public properties of the class.

In plugins, it's best practice to include the name of the plugin at the beginning of each model name, to avoid any potential conflicts with models from the core or other plugins.

Classes extending Omeka_Record_AbstractRecord are placed in the models folder of a plugin (or application/models in the core). You can choose to keep the model classes all directly under the models folder or organize them into subfolders. Omeka will work with either organizational style, but as with all other class files, additional levels of the folder structure must have corresponding underscores in the class name. For example, a model named MyPluginThing would be located at <plugin folder>/models/MyPluginThing.php, while one named MyPlugin_Thing would be located at <plugin folder/models/MyPlugin/Thing.php.

```
class MyPluginThing extends Omeka_Record_AbstractRecord
{
    /**
     * text Text for the Thing
     */
    public $text;
}
```

Note that an $id property is inherited from Omeka_Plugin_AbstractRecord. All Omeka ORM models have an id property/column that uniquely identifies a particular record. Tables that would normally have a composite primary key, like "junction" tables representing many-to-many relationships, must either have a redundant id column or be managed outside the ORM.

Database tables

The Omeka convention is for table names to be underscore-separated and pluralized (i.e. a `MyPluginThing` model will correspond to a table `my_plugin_things` in the database). This convention can be overridden if necessary in the corresponding `Table` model. Accessing a model name as a property (`$db->MyPluginThing`) from the DB object will return the table name for that model, correctly prepended with the user's configured database prefix. This is the preferred way to get the table name when needed for SQL queries (like a `CREATE TABLE` query).

To create the tables in the database table for a plugin, use the *install hook*. The table must contain a column for each public property of the model, as well as an auto-incrementing `id` column, usually as the primary key. For example, this code in a plugin would create a table for the above model:

```
protected $_hooks = array('install' /* ... */);

/* ... */

public function hookInstall() {
    $db = $this->_db;
    $sql = "
    CREATE TABLE IF NOT EXISTS `$db->MyPluginThing` (
      `id` int(10) unsigned NOT NULL AUTO_INCREMENT PRIMARY KEY,
      `text` text NOT NULL
      ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci";
    $db->query($sql);
}
```

Other features like additional indexes besides the primary key would be specified in the CREATE TABLE statement when necessary.

Plugins that create tables should drop them in their *uninstall hook*.

Events and callbacks

The abstract record contains callback functions creating, reading, updating, and deleting records, both before and after those actions. These are helpful if any automatic processing should be done during those operations.

For example, to append to the `text` column before the record is saved, override the `beforeSave()` callback.

```
protected function beforeSave()
{
    // only append upon insert
    if (! $this->exists()) {
        $this->text = $this->text . "Hello!";
    }
}
```

The *_validate()* function can similarly be overridden to add custom validation code for the record.

2.2.3 Understanding Omeka_Db_Table

Basics

Omeka_Db_Table is part of the Model part of Model-View-Controller in Omeka. It provides the methods for querying the database and returning results as *`Omeka_Record_AbstractRecord`* objects. The most frequently used methods in this class are *`Omeka_Db_Table::findBy`* and *`Omeka_Db_Table::find`*

Omeka_Db_Table::find simply returns a single record by the `id` passed in.

Omeka_Db_Table::findBy returns an array of records that match the criteria passed in with the `$param` argument. When creating your subclasses of *Omeka_Db_Table*, you will want to understand first how to build filters for queries using *Omeka_Db_Table::findBy*.

Omeka_Db_Table::findBy works by manipulating a *Omeka_Db_Select* object, which simplifies the process fo building simple SQL queries.

Filtering Queries

Omeka_Db_Table::findBy calls method *Omeka_Db_Table::applySearchFilters*. It takes a `$select` argument, and a `$params` argument. By default, this method simply checks the `$params` array passed in, looking for keys that match column names. For each match, a `WHERE` clause is generated.

Sometimes, though, it makes sense to add more complex logic. For example, you might want to filter a query by conditions on joining another table, or you might need to call a method in your subclass to process the data passed in the value of the parameter key. In such cases, Omeka has moved toward a pattern of using `filterBy*` methods.

In this pattern, you build more complex logic for adding a `WHERE` clause by passing in the `$select` object, and the value that you want to work with. *Table_Collection* offers a good example.

```
public function filterByPublic($select, $isPublic)
{
    $isPublic = (bool) $isPublic; // this makes sure that empty strings and␣
↪unset parameters are false

    //Force a preview of the public collections
    if ($isPublic) {
        $select->where('collections.public = 1');
    } else {
        $select->where('collections.public = 0');
    }
}
```

This method does a little double-checking on the value passed in for whether the collection has been marked public, then applies the appropriate `WHERE` clause to the `$select` object.

In *Table_Collection::applySearchFilters*, we see the following to call it:

```
if(array_key_exists('public', $params)) {
    $this->filterByPublic($select, $params['public']);
}
```

2.2.4 Understanding Omeka_Controller_AbstractActionController

2.3 The Admin Theme

2.3.1 Understanding the Admin CSS

The Omeka admin theme uses Skeleton development kit for easy, responsive styles. Each page consists of a twelve-column grid, with sections' columns defined by CSS classes. For example, a three column section would have the classes "three columns".

The first two columns are dedicated to navigation, while the latter ten hold the main content. Admin views, including those added by plugins, are therefore contained within ten columns, and plugin writers should look to the structure of other admin views for markup patterns.

Forms

Within the main content, Omeka forms typically take up seven columns, and leave the last three for a save panel that follows the user down the page. The following is an example of markup from a typical admin form. The "alpha" class signifies the first set of columns within a set, and "omega" signifies the last set of columns. These classes modify the left and right margins respectively.

```
<form method="post" enctype="multipart/form-data" action="">
    <section class="seven columns alpha">
        <!-- The main body of the form -->
    </section>
    <section class="three columns omega">
        <div id="save" class="panel">
            <!-- Submit button(s) and other small controls -->
        </div>
    </section>
</form>
```

For reference:

- Official Skeleton website
- Build a Responsive, Mobile-Friendly Web Page With Skeleton (For a more detailed guide on understanding and using Skeleton.)

2.3.2 Understanding Omeka_Form_Admin

Omeka_Form_Admin is designed to make it easier for most plugins to conform to the look and feel of Omeka's

admin interface. In most, but not all, cases, this class will let you easily construct your form. More complex forms might require building the form from scratch.

Like the rest of Omeka, it relies heavily on Zend's form elements. Read *Understanding Form Elements* for a primer.

Basics

The admin editing screen generally consist of two areas, the 'main' area and the 'save panel', which includes the save and delete buttons, as well as miscellaneous buttons and sometimes some form elements. Omeka_Form_Admin is meant to make building those areas in a way that keeps the look and feel as the rest of the admin theme as easy as possible for most kinds of forms. It does this by creating a DisplayGroup for each of the two areas, styling them appropriately, and letting you add your form elements to them.

It is generally best to put the logic for building your form inside a method of your controller, e.g. in `_getForm($record)`. You then add it to your add and edit views the usual way: `$this->view->form = $this->_getForm($record)`. Generally, extending Omeka_Form_Admin is not necessary.

Create the form like so:

```
protected function _getForm($record)
{

    $formOptions = array('type' => 'my_type', 'hasPublicPage'=>true);
    if($record && $record->exists()) {
        $formOptions['record'] = $record;
    }

    $form = new Omeka_Form_Admin($formOptions);

    // build the form elements

    return $form;
}
```

The form option `type` is required. Typically, this is the name of your record type (e.g., 'item'). Hooks into the save panel are based on this type. See *admin_<type>_panel_buttons*, *admin_<type>_panel_fields*, and *admin_<type>_form*

`hasPublicPage` is a flag to determine whether a link to the record's public page should be created in the save panel. Defaults to false.

Building the main edit area

The 'main' content area is where the primary parts of your form go: text inputs, most select dropdowns, etc.

Use the *`Omeka_Form_Admin::addElementToEditGroup`* to add your elements to the main edit area:

Building the save panel

The save and delete buttons will the automatically added to the save panel. Some true form elements also make more sense in this area than in the main editing area. Often, data that is entered via a simple checkbox is a good candidate for having the form element here. Whether a page is public or not is a good example.

```
$form->addElementToSaveGroup('checkbox', 'is_published',
                array('id' => 'my_plugin_is_published',
                        'values' => array(1, 0),
                        'checked' => metadata($record, 'is_published'),
                        'label' => 'Publish this page?',
                        'description' => 'Checking this box will make the page public.
↪'
                ));
```

2.3.3 Understanding Form Elements

Omeka makes heavy use of Zend's Form_Element class for building forms, rather than writing out the HTML explicitly.

Typically, elements are added directly to an *Omeka_Form* object or an *Omeka_Form_Admin* object.

The basic structure is to use the `addElement()` method, with three parameters:

`string` $element A string name for the form element. Typical examples are: button, captcha, checkbox, file, multicheckbox, multiselect, radio, select, submit, text, textarea.

An element object can also be passed in here. In this (rare) case, the other two parameters are usually unnecessary

`string` $name The name attribute for the form element

`array` $options Additional options for the form element. `label`, `description`, and `required` are the most common options. HTML attributes such as `rows` and `cols` can also appear here, as well as `class`

In more complex examples, an array of `validators` can also be passed here

Examples

```
$form->addElement('textarea', 'description', array(
    'label' => __('Description'),
    'description' => __('My record description'),
    'rows' => 10
));

$form->addElement('text', 'user_email', array(
    'label' => __('Email'),
    'description' => __("Prefered email address of the person submitting the record"),
    'validators' => array('EmailAddress'),
    'required' => true
));
```

For a more extended example, see the IndexController::_getForm() method in the SimplePages plugin, bundled with Omeka.

See Also

- *Omeka_Form*
- *Omeka_Form_Admin*
- *Understanding Omeka_Form_Admin*

2.4 Public Themes

2.4.1 Modifying Themes

Overriding default templates

Omeka has a set of default template files that all themes use, and override when the desired page structure is different from the default.

The default theme files are in the folder `/application/views/scripts` in your Omeka installation. Subfolders correspond to the pages that are seen along url patterns. For example, the page displayed at `{YourOmekaSite}/items/show` is produced by the file in `/application/views/scripts/items/show.php`.

Themes might or might not override these files. The default theme, for example, has an `items` directory that overrides two of the default templates: `random-featured.php` and `show.php`

```
items/
    random-featured.php
    show.php
```

If you want to modify a file in a theme, the first place to look is in the theme's own directories. But notice that that will only work if the theme has overridden the default template. In many cases, then, you will need to copy the default template files into the theme, taking care to maintain the directory structure.

So, for example, imagine wanting to modify the show page for items, the browse page for items, and the show page for collections in the default theme.

The `/items/show.php` file is easy, since the default theme already includes it.

For the browse page for items, we need to copy `/application/views/scripts/items/browse.php` to `/default/items/browse.php`

For the browse page for collections, we need to first create the directory: `/default/collections`

Then we can copy `/application/views/scripts/collections/browse.php` to `/default/collections/browse.php`

The result in the default theme will look like:

```
items/
   random-featured.php
   browse.php
   show.php
collections/
   browse.php
```

2.4.2 Working with Public Themes

Themes included in Omeka 2.0, "Thanks, Roy" and "Seasons", have been rebuilt with two new features: Compass project files and Susy responsive grids. Compass is a CSS framework that utilizes the Sass preprocessor to generate styles, making it easier for front-end developers to manage typography, margin units, colors, and more. Susy is a responsive grid library that can be imported into Compass. Responsive grids allow sites to adapt to the browser width, making content more easily readable on a variety of devices.

For users who want to start using Compass and Susy with Omeka themes, here are recommended tutorials for installation. Both Compass and Susy require Ruby to be installed, and users should be comfortable with the command line.

- The Sass Way: Getting started with Sass and Compass
- The official Compass installation tutorial
- The official Susy installation tutorial

Those who simply want to edit the CSS without getting into preprocessors can ignore the 'css/sass' folder completely and focus on .css files. **Note:** if you edit a .css file but later decide to use Sass, you should back up and make note of your changes before compiling for the first time. Changes made in .scss files overwrite any changes made to .css files.

2.4.3 Omeka Theme Style Guide

This style guide is for use in writing themes for Omeka 2.0+. It borrows heavily from the Github style guide. This style guide is a work in progress, so excuse our dust.

Configuration

When setting up the Compass configuration for a theme, the settings should be as follows:

```
http_path = "/"
css_dir = "/"
sass_dir = "sass"
images_dir = "images"
javascripts_dir = "javascripts"
output_style = :expanded
line_comments = false
```

When using Git, make sure you include a `.gitignore` in the root of your theme folder. The most common inclusions are:

```
.DS_Store
.sass-cache/
```

Organization

As such, the */css* directory of an Omeka theme should follow the following structure:

```
|- css
|--- sass
|    |--- _base.scss
|    |--- _normalize.scss
|    |--- _print.scss
|    |--- _screen.scss
|    |--- style.scss
|--- config.rb
|--- style.css
```

Group your styles using commented headings and include a table of contents. Here is an example of a table of contents:

```
/*
Table of Contents
=================
-- General HTML Elements
----- Headings
----- Form Elements
```

(continues on next page)

(continued from previous page)

```
-- Global Classes
-- Navigation
----- Pagination
-- Header
-- Footer
-- Content
*/
```

An example of a section heading:

```
/* !---------- Header ---------- */
```

Use the `/*` style of comments for headings you want to appear in the `.css` file. For Sass-only sections, such as the variables set in `_bass.scss`, use the `//` commenting syntax. The `!` at the beginning of the header helps bookmark sections for theme writers using the Coda web editor.

_base.scss

The `_base.scss` file should include any variables used across multiple `.scss` files. Otherwise, the variables appear at the top of the file in which they are used.

_normalize.scss

We like to use Nicolas Gallagher's normalize.css as our reset stylesheet. Include it as a `.scss` file and import it into `_base.scss`.

_screen.scss

If you are creating a responsive theme, `_screen.scss` is the default file for mobile-first styles. Separate `.scss` files should be used to group styles for different resolutions. Files should be named based on the breakpoint, i.e. `_screen_480px.scss`, `_screen_360px.scss`, `_screen_40em.scss`.

style.scss

If you are not creating a responsive theme, all styles should go in `style.scss`. The `style.scss` file should also import `_base.scss` and `_normalize.scss`. Responsive themes should also import all `_screen_x.scss` files here.

General styles

- Use 4 spaces for indentation. Do not use tabs.
- Use spaces after `:` when writing property declarations.
- Put spaces before `{` in rule declarations.
- Use hex color codes `#000` unless using rgba.

2.5 File Display

2.5.1 Changing Fallback Thumbnails

Omeka displays thumbnails for files in many places. However, useful thumbnails can't be generated for all types of files, and for some files that could have a thumbnail, the server simply doesn't have the libraries it needs to read that type of file and create the thumbnail.

In these situations, there is no thumbnail generated, so Omeka will use a fallback thumbnail instead. All Omeka 2 versions have `images/fallback-file.png`, a simple image of a blank page used to represent files generically. Omeka 2.2 adds more specific fallbacks for audio, video and image files.

Overriding thumbnails in themes

The fallback thumbnails are simply theme assets, so they can be overridden by a theme just including its own image in the same location. The default fallbacks are located in `application/views/scripts/images` and are PNG images with names starting with `fallback-`. `fallback-file.png` is present in all Omeka versions, and Omeka 2.2 adds `fallback-audio.png`, `fallback-video.png` and `fallback-image.png`.

In a theme, you can just put your own versions of those files in the theme's `images` folder. The default images are 200x200 pixel square PNGs.

Adding new fallback thumbnails

In addition to just overriding the existing thumbnails, you can also add new fallbacks with a plugin.

The function *`add_file_fallback_image`* is used to add a new fallback image for a given MIME type or type family. The first argument is the MIME type the fallback will apply to, and the second is the name of the image to use. The image name is internally passed to *`img`*. The plugin should place the actual fallback image in its `views/shared/images` folder.

The fallback will only be used for files lacking thumbnails that match the MIME type (or family) you pass. The "family" is the part of the MIME type before the slash, "image" for "image/jpeg" for example.

The call to `add_file_fallback_image` should be made in the plugin's *initialize* hook.

2.5.2 Representative Files for Records

New in version 2.2.

Traditionally in Omeka, only Files and Items could easily be represented by graphics like thumbnails. Files can be displayed with *`file_markup`* or as thumbnails with *`file_image`*, and Items could use *`files_for_item`* to show all their files, or *`item_image`* to show the thumbnail of their first attached File.

Omeka 2.2 allows records other than Items to show thumbnails and other functionality that depends on File records by choosing a "representative" file. Omeka uses this functionality in the views for some records like Collections and Files, and also in the search results page.

Themes and plugins can use a new helper funtion to display thumbnails for records with representative files, and plugins can also select and expose representatives for their own records.

Showing thumbnails for records

Instead of the existing functions which are different for every type of record that can have a thumbnail, the new representative file system has just one helper function: *record_image*. `record_image` works just like `file_image` or `item_image`, but it will work for any type of record that has a representative file.

So, a theme can use this same function across the views for any number of record types, and can even use it on views where multiple different record types may appear.

To directly use the existing functions that work on Files, you can also call the *Omeka_Record_AbstractRecord::getFile* method on the record, which will return the File object for the representative file. You can pass that File to helpers like *file_markup*.

Using representative files in plugin records

Plugins can also add representative file support to their own records. These records will then work with *record_image*, so the plugin's views can use that function to show thumbnails. Thumbnails will also show when those records appear in search results and any other places already using `record_image`.

To have a representative files for its instances, a Record subclass must override *Omeka_Record_AbstractRecord::getFile*. `getFile` takes no arguments, and the record simply has to return a File object from it. Records can use any strategy they want to select a representative file.

For example, Omeka's own records use a variety of strategies:

- Files return `$this`, since they "represent" themselves
- Items return their first attached File
- Collections select an item in the collection, preferring featured items, and return that item's first File
- Exhibits return the attached File that appears earliest in the exhibit

2.5.3 File Derivative Strategies

New in version 2.2.

As of Omeka 2.2, administrators can switch between different strategies to change the code that's used when creating thumbnails and other file derivatives. Plugins can provide their own new strategies, and administrators can switch between all available strategies and set options through the configuration file.

Configuring a derivative strategy

You switch derivative strategies and set their options through the configuration file, `application/config/config.ini`. All the file derivative settings go under the key `fileDerivatives`.

To set the strategy Omeka will use when creating derivatives, use the `fileDerivatives.strategy` setting. The value of the setting should be the name of the strategy class you want to use. This class must implement *Omeka_File_Derivative_StrategyInterface*. You can swtich between the different strategies provided by Omeka, like `ExternalImageMagick` and `Imagick`, or switch to a new strategy of your own or one provided by a plugin. The string given must be the full name of the class.

Beyond just switching the entire strategy, you can also configure settings for each strategy through the config file. The key `fileDerivatives.strategyOptions` lets you specify named options that will be passed to the current strategy. For example, both the built-in Omeka strategies support an option `gravity`, which sets which part of an image will be kept when cropping for a square thumbnail. A site administrator would change this setting with one line in the config:

```
fileDerivatives.strategyOptions.gravity = "north"
```

Creating a new strategy

Derivative strategies are classes implementing *Omeka_File_Derivative_StrategyInterface*. The interface has one main entry point, the method *createImage*. `createImage` is called by Omeka once per derivative type (thumbnail, square thumbnail, fullsize) per file uploaded, and receives as arguments the source and destination file paths, the type of image being created, the configured size constraint, and the MIME type of the file.

`createImage` simply returns a boolean true/false value. If the return value is true, Omeka expects that a new derivative file will have been created at the destination path. A return value of false indicates that the derivative process failed and there is no derivative to upload. The strategy can use any means, including PHP libraries or extensions, or even shell calls, to create the new derivative, so long as it ends up in the correct location. Using the MIME type parameter and/or the source path, a strategy could even use totally different procedures for making derivatives for different kinds of files.

The interface also requires `setOptions` and `getOptions` methods, so Omeka can inject the settings from the site's configuration file (see above). You can implement these simple methods yourself, or you can extend from *Omeka_File_Derivative_AbstractStrategy*, which provides those methods for you, but still leaves `createImage` as an abstract for you to override. The `AbstractStrategy` class also provides a convenience method, *getOption*, for getting a specific named option with a fallback or default value.

A strategy simply needs to be located in a location Omeka can autoload from to be available for the adiminstrator to select it in the config file. In a plugin this means the class should live in the `libraries` folder, at a path conforming to PSR-0.

2.6 International Locale Support

2.6.1 Internationalization

From Omeka 1.5 on, site admins can pick the locale they want Omeka to use. When working on the core, plugins, or themes, the code must be *internationalized* to make display in different languages and locales possible.

Text

For most plugins and themes, making user-facing text translatable will be the lion's share of the internationalization work. All text strings that are presented to the user and are not editable by the user should be translated. This includes obvious suspects like text in paragraphs and other visible HTML elements, but also less obvious places like the `<title>` element or `title` and `alt` attributes.

Omeka uses one function for enabling text translation, the __ (double-underscore) function. Anything that needs to be translated must be passed through the double-underscore function.

Bare text

Before internationalization, a great deal of the user-facing text may be written directly in HTML or plain text, with no PHP code. Omeka's translation works through a PHP function, so you need to introduce a PHP block.

Untranslatable

```
<p>Some text.</p>
```

Translatable

```
<p><?php echo __('Some text.'); ?></p>
```

Literal PHP strings

PHP strings that will end up being shown to the user also need to get translated. These strings are already in PHP code blocks, so the process is easy. Just wrap the double-underscore function around the string that's already there.

Untranslatable

```
<?php
echo head(array(
    'title' => 'Page Title'
));
?>
```

Translatable

```
<?php
echo head(array(
    'title' => __('Page Title')
));
?>
```

Strings with variables

A common pattern in PHP is to write strings that directly contain variables. These need a slightly different approach to be translatable. The goal is to make translators only have to translate your string once, no matter what the particular values of the variables inside are.

To do this, you replace your variables with *placeholders*, and pass your variables separately into the double-underscore function. (The placeholders used are from PHP's `sprintf` function.)

Single variable

The basic placeholder is `%s`. It's used when your original string simply contained one variable.

Untranslatable

```
<?php
echo "The site contains $numItems items.";
?>
```

Translatable

```
<?php
echo __('The site contains %s items.', $numItems);
?>
```

This will output the same way as the original, but translators will work with the single string `'The site contains %s items.'` instead of many different ones for each possible number.

Multiple variables

The `%s` placeholder is fine for a string with only one variable. However, with two or more, you need to account for the possibility that some translations will need to reorder the variables, because their sentence structure differs from English. With multiple variables, you must instead use **numbered placeholders** like `%1$s`, `%2$s`, and so on.

Untranslatable

```
<?php
echo "Added $file to $item.";
?>
```

Translatable

```
<?php
echo __('Added %s$1 to %s$2.', $file, $item);
?>
```

By using numbered placeholders, translators can reorder where the variables will appear in the string, without modifying the code to do so.

Dates and times

The other major thing you will often want to display differently for different for different locales are dates and times. Omeka comes pre-packaged with date formats for various locales already.

Where translations run through one function, the double-underscore function, dates and times similarly work with one function: *format_date*. `format_date` automatically selects the right format based on the site's configured locale.

`format_date` takes two parameters. The first is the time you want to display. The second, which is optional, is the format you want to use. If you don't pick a format, the default is an appropriate format for displaying a date.

Time

There are two possible types for the time parameter for `format_date`: integer and string. If you pass an integer, the time is interpreted as a Unix timestamp. If you pass a string, the time/date is interpreted according to the ISO 8601 standard (this will, among many other formats, correctly parse the output from MySQL date and time columns).

Format

`format_date` uses Zend_Date internally, so the Zend documentation is the place to go for an exhaustive list of available formats.

Format constants starting with `DATE` are used for displaying dates without a specific time, ones starting with `DATETIME` are used for date/time combinations, and ones starting with `TIME` are for times alone. For each, there are `FULL`, `LONG`, `MEDIUM`, and `SHORT` variants. Each variant will automatically use a format specific to the current locale, including things like the proper order for dates and the correct names of months.

The default format is `Zend_Date::DATE_MEDIUM`. This will display the given date/time value as a date, with medium length. In the standard US English locale, this looks like "May 31, 2013." In a Brazilian locale, it would instead look like "31/05/2013."

Preparing Translation Files

Omeka reads translations from .mo files produced with GNU `gettext`. There are three steps to the process. After the basic work described above is complete, you will need to

1. Create a template file that includes all of the strings to translate
2. Create `.po` files that contain the actual translations
3. Compile `.mo` files that Omeka will use

The guide for these tasks below follows the practices used by the Omeka dev team. There are other tools and approaches that can accomplish the same tasks. The tool we use are

- ant build utility (along with a `build.xml` file described below)
- Transifex client (requires Python)
- podebug (requires Python)

Creating the template file

The simplest way to produce the template file is to follow the examples in Omeka. We begin with a `template.base.pot` file, which contains the basic format required to begin generating translations.

```
# Translation for the Simple Pages plugin for Omeka.
# Copyright (C) 2011 Roy Rosenzweig Center for History and New Media
# This file is distributed under the same license as the Omeka package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: SimplePages\n"
"Report-Msgid-Bugs-To: http://github.com/omeka/plugin-SimplePages/issues\n"
"POT-Creation-Date: 2012-01-09 21:49-0500\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"Language: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
```

This file will be used to generate the `template.pot` file that is used as the template for translations. `template.pot` files will begin with exactly the content shown above and then include pairs of `msgid` s and empty `msgstr`. The `msgid` s contain the English string of text to translate. The `msgstr` s will eventually contain the actual translations.

The `template.base.pot` file is also helpful if your plugin uses strings of text that are not available for the `__()` function described above. For example, if your records include a flag for a permission such as `allowed` or `required` in the database, those strings need to be translated, but might not appear directly in your plugin's display. In such cases, the strings should be added to `template.base.pot` below the last line:

```
msgid "allowed"
msgstr ""

msgid "required"
msgstr ""
```

If you have `ant` installed on your system, you can modify the following `build.xml` file.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="SimplePages" basedir=".">
    <property name="lang.dir" location="languages" />
    <property name="core.pot" location="../../application/languages/Omeka.pot" />
    <target name="update-pot" description="Update the translation template.">
        <property name="pot.file" location="${lang.dir}/template.pot"/>
        <property name="pot.base" location="${lang.dir}/template.base.pot"/>
        <tempfile property="pot.temp" suffix=".pot"/>
        <tempfile property="pot.duplicates" suffix="-duplicates.pot" />
        <copy file="${pot.base}" tofile="${pot.temp}"/>
        <apply executable="xgettext" relative="true" parallel="true" verbose="true">
            <arg value="--language=php"/>
            <arg value="--from-code=utf-8"/>
            <arg value="--keyword=__"/>
            <arg value="--flag=__:1:pass-php-format"/>
            <arg value="--add-comments=/"/>
            <arg value="--omit-header"/>
            <arg value="--join-existing"/>
            <arg value="-o"/>
            <arg file="${pot.temp}"/>
            <fileset dir="." includes="**/*.php **/*.phtml"
                excludes="tests/"/>
        </apply>
        <exec executable="msgcomm">
            <arg value="--omit-header" />
            <arg value="-o" />
            <arg file="${pot.duplicates}" />
            <arg file="${pot.temp}" />
            <arg file="${core.pot}" />
        </exec>
        <exec executable="msgcomm">
            <arg value="--unique" />
            <arg value="-o" />
            <arg file="${pot.temp}" />
            <arg file="${pot.temp}" />
            <arg file="${pot.duplicates}" />
        </exec>
        <move file="${pot.temp}" tofile="${pot.file}"/>
        <delete file="${pot.duplicates}" quiet="true" />
    </target>

    <target name="build-mo" description="Build the MO translation files.">
        <apply executable="msgfmt" dest="${lang.dir}" verbose="true">
            <arg value="-o"/>
            <targetfile />
            <srcfile />
            <fileset dir="${lang.dir}" includes="*.po"/>
            <mapper type="glob" from="*.po" to="*.mo"/>
        </apply>
    </target>
</project>
```

It creates two ant commands. The first one that is important to us here is `ant update-pot` . It will read the `template.base.pot` and generate the `template.pot` file from the strings that are wrapped in `__()`. `template.pot` will then contain all the `msgid` s to be translated.

You will want to double-check that you have found all of the strings that require localization. The `podebug` utility

can be helpful with this. It automatically generates `.po` files that contain pseudo-translations that will help you spot any strings that are not being translated, but should be.

Creating `.po` files

The `.po` files contain the localizations, named according to the ISO 639-1 standard. For example, `es.po` will contain translations into Spanish, and `es_CO.po` will contain the more precise localization to Colombian Spanish.

Omeka uses the Transifex service to produce our translations. Other tools and services also exist to help you produce your translations, but we recommend using Transifex if possible, and setting up your plugin as child project to Omeka. This will widen the pool of translators and languages for your project.

Compiling `.mo` files

Once you have created the `.po` files for your localizations, the final step is to compile them into binary `.mo` files. The second command defined by the `build.xml` file used above, `ant build-mo` will perform this task for you.

All files, `template.base.pot`, `template.pot`, and all `.po` and `.mo` files should be in a `languages` directory at the top level of your plugin.

2.7 Extending Omeka and the Standard Plugins

2.7.1 Extending Exhibit Builder 3

Exhibit Builder 3 allows other plugins to add their own layouts.

Placing layouts within your plugin

The new Exhibit Builder layouts use the same "views" system as other Omeka pages and views.

In any plugin (including the Exhibit Builder itself), layout views go in the path `views/shared/exhibit_layouts`.

Registering a new layout

Exhibit Builder provides a filter named `exhibit_layouts`. To add a new layout, you must hook into this filter and add some information about your new layout. You need to decide three things:

- ID: an internal name for the layout and the name of the layout folder
- Name: the admin user-facing name for the layout
- Description: the admin user-facing description of the layout

The ID is the key into the array of layouts, and the name and description are keys in an internal array. A filter implementation looks like this:

```
public function filterExhibitLayouts($layouts) {
    $layouts['new-layout'] = array(
        'name' => 'New Layout',
        'description' => 'A new layout.'
    );
```

(continues on next page)

(continued from previous page)

```
    return $layouts;
}
```

Basic structure of an exhibit layout

Each layout has its own folder under the path mentioned above. The folder name is the layout's ID, which will be used internally by the Exhibit Builder to find and store the layouts picked by the user. An ID is simply a text string, and the convention is that layout IDs be ASCII, lowercase, and hyphen-separated: `sample-layout`, not `SampleLayout` or `SampleLayout`.

Inside the layout folder, there are several files with predefined names that the Exhibit Builder will look for:

form.php The PHP view file for the admin-side display of a form for the user to configure attachments, text, and options for a single block with this layout.

layout.php The PHP view file for the public-side display of a single block with this layout.

layout.css A CSS stylesheet that is automatically loaded when this layout is used on a public page.

layout.png A PNG preview image for the layout. This image is displayed on the admin side when the user is selecting the layout to use. To fit with the images for the other layouts, the image should be 260 pixels wide by 140 pixels tall.

Your views can reference and use other files with names of your choosing, but the ones specified above are required and must be at the given locations.

The layout form – form.php

The page block that the layout is being used on is accesible in the form.php view as `$block`. This block object is passed to the form helpers and is used when form entries are created manually.

Attachments

The attachment interface is added by a simple helper:

```
echo $this->exhibitFormAttachments($block);
```

Text

The text entry box is also added by a simple helper:

```
echo $this->exhibitFormText($block);
```

Options

Options are handled differently by each layout, so there is no standard form helper for them. Instead, you should use the built-in Zend form helpers to create form inputs for your layout options.

Internally, options are stored as a JSON-serialized array. Exhibit Builder automatically saves any options that are sub-keys of the `options` key in a block form.

Each block needs a different form name "stem," indexed by block. So, when you're manually creating form inputs, you need to get this stem from the block object:

```
$formStem = $block->getFormStem();
```

The pre-existing options are also available on the block object:

```
$options = $block->getOptions();
```

With those pieces of data, you can manually create a form input that will reflect the status of an option. For example, a simple text input for an option named 'some-option':

```
echo $this->formText($formStem . '[options][some-option]', @options['some-option']);
```

The layout view – layout.php

The layout view has four variables assigned to it.

$attachments An array of the attachment objects for items the user has attached to this block.

$text A string, containing the HTML text the user entered for this block. This text is filtered and can be output directly.

$options An array of the options the user selected for this block.

$index An integer, the index within the page of the current block.

Attachments

Single attachments

There are two main helpers for displaying attachments on a layout. The first is simply used for displaying a single attachment using Omeka's file display functionality. This helper is the `exhibitAttachment` helper.

The `exhibitAttachment` helper takes 4 arguments, but only the first, `$attachment`, is required.

$attachment The attachment object to display. This should be retrieved from the `$attachments` array assigned to the view.

$fileOptions An array of options for displaying the attached file. See the Usage section of the *file_markup* documentation for examples of display options.

$linkProps An array of HTML attributes that will appear on the link to the item. This link will appear if an attachment is for an item with no files, or if the `$forceImage` argument is true.

$forceImage A boolean argument. If true, the normal *file_markup*-style display will not be used, and instead *file_image* will be. This can be useful for displaying attachments in grids or other structures where differing sizes and display methods would be unusable. This argument defaults to false.

The simplest possible example of using this helper just passes the attachment:

```
foreach ($attachments as $attachment):
    echo $this->exhibitAttachment($attachment);
endforeach;
```

Attachment galleries

The second helper is the `exhibitAttachmentGallery` helper. The gallery helper takes an array of attachments, and displays them in a gallery format.

`exhibitAttachmentGallery` takes three arguments, only the first of which, `$attachments`, is required.

`$attachments` An array of attachments to display a gallery of.

`$fileOptions` An array of file display options: see the explanation for the previous helper. If no `imageSize` option is given, the square thumbnail size is automatically selected by this function.

`$linkProps` An array of HTML attributes for the link around each displayed attachment. See the explanation for the previous helper.

Again, the simplest possible example simply passes the attachments. A gallery of all attachments on a block in square thumbnail format is simple:

```
echo $this->exhibitAttachmentGallery($attachments);
```

Custom displays

Of course, you may wish to use the attached items and files in a completely different way than the normal Omeka file display. In that case, you can directly access the Item and File objects for each attachment object, and work on them however you wish:

```
foreach ($attachments as $attachment):
    $item = $attachment->getItem();
    $file = $attachment->getFile();
endforeach;
```

Layout style - layout.css

The layout.css file is automatically loaded when the layout is used on a page. Any given page can contain many different layouts simultaneously, so you need to take some care that you don't write styles that will interfere with other layouts.

To help with keeping styles separate, Exhibit Builder automatically wraps your layout output in a div with the class `layout-<your layout id`. In general, styles in layout.css should start with that class selector as the first component of every selector.

Including extra assets

Some scripts, styles or other content can be included directly in layout.php. But, some content may need to be placed in the `<head>`, or may only be included in the page once. For these scenarios, Exhibit Builder provides a hook to add content to the head for exhibit pages. Note: layout.css is automatically included, so you don't need to write anything to include it.

The hook is called *exhibit_builder_page_head*. It sends two arguments, the View object `$view`, and `$layouts`, a keyed array of the layouts in use on the page being shown. You can check for your layout ID in the keys of `$layouts` and include whatever content your layout needs:

```
public function hookExhibitBuilderPageHead($args) {
    if (array_key_exists('my-layout', $args['layouts'])) {
        queue_js_file('my-script');
    }
}
```

2.7.2 Working with Shortcodes

New in version 2.2.

Starting with Omeka 2.2, Omeka offers shortcode support. The core comes with several default shortcodes, and plugins can register their own, too. Users can then insert shortcodes within free text, and Omeka will replace the shortcode with whatever markup is returned from the registered callback. For more information shortcodes from a user's persepective, see the Shortcodes page on the Omeka Codex.

You can add support for shortcodes in two ways. First, plugins can add their own shortcodes. For example, the Exhibit Builder plugin provides several shortcodes for including Exhibits within other pages, and the Geolocation plugin adds a shortcode for embedding a map. Second, plugins and themes can support shortcodes within user-provided text. The Simple Pages plugin is the obvious example of this: users can include shortcodes in the content of pages.

Adding a new shortcode

Shortcodes are added through the *add_shortcode* function. The function takes only two arguments: the name of the shortcode (what the user will type at the beginning of the shortcode) and a callback. A plugin adding a shortcode should call `add_shortcode` in its *initialize* hook.

The callback will be called whenever a shortcode is encountered in text, and is responsible for returning the markup that will actually be rendered on the page in place of the shortcode. Two parameters are passed to the callback. The first is an array of the arguments the user included in the shortcode (i.e. `[my_shortcode arg1="value 1" arg2="value 2"]` results in `array('arg1' => 'value 1', 'arg2' => 'value 2')` as the first parameter). The second argument is an *Omeka_View* instance, to simplify calling view helpers or reading data from the view.

So, in all, a plugin can add a shortcode with very little code:

```
class MyPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_hooks = array('initialize');

    public function hookInitialize()
    {
        add_shortcode('my_shortcode', array($this, 'shortcode'));
    }

    public function shortcode($args, $view)
    {
        return 'This is a very simple shortcode.';
    }
}
```

Supporting shortcodes in user input

The flipside of adding new shortcodes is supporting shortcode replacement in text input by the user. To support shortcodes, you only need to pass the text through the *shortcodes view helper*.

Assuming `$text` is some text input by the user, you barely need to change your code that would have normally output the text:

```
/* No shortcode support: */
echo $text;

/* Shortcode support: */
echo $this->shortcodes($text);
```

You can also use the shortcodes helper on static text:

```
echo $this->shortcodes('[my_shortcode]');
```

2.8 User-Submitted Recipes

2.8.1 Display Different Header Images

Problem

I want to display a different header image if the page is a SimplePages page.

Solution

In the theme's header.php file, usually in theme/common/header.php, add the following code to change the image if the page is a SimplePage. This example comes from default theme:

```
<header role="banner">
    <div id="site-title">
    <?php if ($bodyclass=='page simple-page'):?>
        <img src="my_image.jpg" alt="my_alt_text"/>
    <?php else: ?>
        <?php echo link_to_home_page(theme_logo()); ?>
    <?php endif ?>
    </div>
</header>
```

In the site title div, the original code for the theme was:

```
<header role="banner">
    <div id="site-title">
    <?php echo link_to_home_page(theme_logo()); ?>
    </div>
</header>
```

The changes here check whether we're on a simple page, and if so will display your image of choice. Otherwise, it instead displays the theme logo.

2.8.2 Retaining Search or Sort Order when Paging through Items

Problem

When paging item by item using the next and previous buttons, Omeka calls the next item in the database by its ID, not by the order that you have searched or sorted prior to viewing the individual items.

Solution

My solution requires some changes to each theme.

1. Create a **custom.php** file to hold the new navigation function. The custom.php should be at the theme's root (ex: /APP_DIRECTORY/themes/default/custom.php):

```
function custom_paging()
{
//Starts a conditional statement that determines a search has been run
    if (isset($_SERVER['QUERY_STRING'])) {

        // Sets the current item ID to the variable $current
        $current = metadata('item', 'id');

        //Break the query into an array
        parse_str($_SERVER['QUERY_STRING'], $queryarray);

        //Items don't need the page level
        unset($queryarray['page']);

        $itemIds = array();
        $list = array();
        if (isset($queryarray['query'])) {
                //We only want to browse previous and next for Items
                $queryarray['record_types'] = array('Item');
                //Get an array of the texts from the query.
                $textlist = get_db()->getTable('SearchText')->findBy($queryarray);
                //Loop through the texts ans populate the ids and records.
                foreach ($textlist as $value) {
                        $itemIds[] = $value->record_id;
                        $record = get_record_by_id($value['record_type'], $value[
↪'record_id']);
                        $list[] = $record;
                }
        }
        elseif (isset($queryarray['advanced'])) {
                if (!array_key_exists('sort_field', $queryarray))
                {
                        $queryarray['sort_field'] = 'added';
                        $queryarray['sort_dir'] = 'd';
                }
                //Get an array of the items from the query.
                $list = get_db()->getTable('Item')->findBy($queryarray);
                foreach ($list as $value) {
                        $itemIds[] = $value->id;
                        $list[] = $value;
                }
        }
        //Browsing all items in general
        else
        {
```

(continues on next page)

(continued from previous page)

```
                if (!array_key_exists('sort_field', $queryarray))
                {
                        $queryarray['sort_field'] = 'added';
                        $queryarray['sort_dir'] = 'd';
                }
                $list = get_db()->getTable('Item')->findBy($queryarray);
                foreach ($list as $value) {
                        $itemIds[] = $value->id;
                }
        }

        //Update the query string without the page and with the sort_fields
        $updatedquery = http_build_query($queryarray);
        $updatedquery = preg_replace('/%5B[0-9]+%5D/simU', '%5B%5D',
↪$updatedquery);

        // Find where we currently are in the result set
        $key = array_search($current, $itemIds);

        // If we aren't at the beginning, print a Previous link
        if ($key > 0) {
            $previousItem = $list[$key - 1];
            $previousUrl = record_url($previousItem, 'show') . '?' .
↪$updatedquery;
                $text = __('&larr; Previous Item');
            echo '<li id="previous-item" class="previous"><a href="' . html_
↪escape($previousUrl) . '">' . $text . '</a></li>';
        }

        // If we aren't at the end, print a Next link
        if ($key >= 0 && $key < (count($list) - 1)) {
            $nextItem = $list[$key + 1];
            $nextUrl = record_url($nextItem, 'show') . '?' . $updatedquery;
                $text = __("Next Item &rarr;");
                echo '<li id="next-item" class="next"><a href="' . html_escape(
↪$nextUrl) . '">' . $text . '</a></li>';
        }
    } else {
        // If a search was not run, then the normal next/previous navigation is␣
↪displayed.
        echo '<li id="previous-item" class="previous">'.link_to_previous_item_
↪show().'</li>';
        echo '<li id="next-item" class="next">'.link_to_next_item_show().'</li>';
    }
}
```

2. Make sure the query parameters are being passed: In the **items/browse.php** class, replace the line:

```
<h2><?php echo link_to_item(metadata('item', array('Dublin Core', 'Title')),␣
↪array('class'=>'permalink')); ?></h2>
```

with the following to pass the query and sorting parameters:

```
if(isset($_SERVER['QUERY_STRING']) && !empty($_SERVER['QUERY_STRING']))
{

    $searchlink = record_url('item').'?' . $_SERVER['QUERY_STRING'];
```

(continues on next page)

(continued from previous page)

```
    echo '<h2><a href="'.$searchlink.'">'. metadata('item', array('Dublin␣
↪Core','Title')).'</a></h2>';
}

else
{
    echo '<h2>'.link_to_item(metadata('item', array('Dublin Core','Title')),␣
↪array('class'=>'permalink')).'</h2>';
}
```

3. Have the links changed to use the new custom code from custom.php: In the **items/show.php** class, replace the lines:

```
<li id="previous-item" class="previous"><?php echo link_to_previous_item_show(); ?
↪></li>

<li id="next-item" class="next"><?php echo link_to_next_item_show(); ?></li>
```

   with:

```
<?php custom_paging(); ?>
```

4. Make a copy of the **application/views/scripts/search/index.php** and place it under the **THEME/search** directory (create one if it isn't already there).

5. To preserve the query and sort parameters: In **search/index.php**, replace the line:

```
<td><a href="<?php echo record_url($record, 'show'); ?>"><?php echo $searchText[
↪'title'] ? $searchText['title'] : '[Unknown]'; ?></a></td>
```

   with:

```
<?php

if(isset($_SERVER['QUERY_STRING']) && !empty($_SERVER['QUERY_STRING']))
{
        $searchlink = record_url($record, 'show').'?' . $_SERVER['QUERY_
↪STRING'];

?>



        <td><a href="<?php echo $searchlink; ?>"><?php echo $searchText[
↪'title'] ? $searchText['title'] : '[Unknown]'; ?></a></td>

<?php
}
else
{

?>

        <td><a href="<?php echo record_url($record, 'show'); ?>"><?php echo
↪$searchText['title'] ? $searchText['title'] : '[Unknown]'; ?></a></td>
<?php
```

(continues on next page)

(continued from previous page)

```
}

?>
```

6. To make custom_paging work for items within an Exhibit (and sort by DC Date):
   - Add the following before **//Browsing all items in general** in Step 1.
   - Replace **YOUR_SLUG_HERE** with the slug for your Exhibit.
   - Replace **YOUR_EXHIBIT_ID_HERE** with your Exhibit ID.
   - Do this for each of your Exhibits.:

```
//Browsing exhibit YOUR_EXHIBIT_ID_HERE items
elseif (strpos($_SERVER['HTTP_REFERER'],'exhibits/show/YOUR_SLUG_HERE') !=
↪false) {
        $exhibit_query = "search=&advanced[0][element_id]=&advanced[0][type]=&
↪advanced[0][terms]=&range=&collection=&type=&user=&public=&featured=&
↪exhibit=YOUR_EXHIBIT_ID_HERE&submit_search=Search&sort_field=Dublin+Core
↪%2CDate";
        parse_str($exhibit_query, $queryarray);
        unset($queryarray['page']);

        if (!array_key_exists('sort_field', $queryarray))
        {
                $queryarray['sort_field'] = 'added';
                $queryarray['sort_dir'] = 'd';
        }
        //Get an array of the items from the query.
        $list = get_db()->getTable('Item')->findBy($queryarray);
        foreach ($list as $value) {
                $itemIds[] = $value->id;
                $list[] = $value;
        }

}
```

CHAPTER 3

Reference

3.1 Global (Theming) Functions

3.1.1 All Functions

`__` (double underscore) — Translate a string.

Locale-related functions

Summary

__(*$msgid*)
Translate a string.

Parameters

- **`$msgid`** (*string|array*) – The string to be translated, or Array for plural translations.

Returns string The translated string.

Usage

Examples

```
echo __("String to translate");
```

See Also

- *Internationalization*

`_log` — Log a message.

Log-related functions

Summary

`_log` (*$msg*, *$priority = Zend_Log::INFO*)
Log a message.

Enabled via config.ini: log.errors.

Parameters

- **`$msg`** (*mixed*) – The log message.
- **`$priority`** (*int*) – See Zend_Log for a list of available priorities.

Usage

Examples

See Also

`absolute_url` — Get an absolute URL.

Navigation-related functions

Summary

`absolute_url` (*$options = array()*, *$route = null*, *$queryParams = array()*, *$reset = false*, *$encode = true*)
Get an absolute URL.

This is necessary because Zend_View_Helper_Url returns relative URLs, though absolute URLs are required in some contexts. Instantiates view helpers directly because a view may not be registered.

Parameters

- **`$options`** (*mixed*) – If a string is passed it is treated as an Omeka-relative link. So, passing 'items' would create a link to the items page. If an array is passed (or no argument given), it is treated as options to be passed to Omeka's routing system.
- **`$route`** (*string*) – The route to use if an array is passed in the first argument.
- **`$queryParams`** (*mixed*) – A set of query string parameters to append to the URL
- **`$reset`** (*bool*) – Whether Omeka should discard the current route when generating the URL.
- **`$encode`** (*bool*) – Whether the URL should be URL-encoded

Returns string HTML

Usage

Examples

See Also

`add_file_display_callback` — Add a callback for displaying files with a given mimetype and/or extension.

Plugin-related functions

Summary

`add_file_display_callback` (*$fileIdentifiers*, *$callback*, *$options = array()*)
Add a callback for displaying files with a given mimetype and/or extension.

Parameters

- **`$fileIdentifiers`** (*`array|string`*) – Set of MIME types and/or file extensions to which the provided callback will respond.
- **`$callback`** (*`callback`*) – Any valid callback.
- **`$options`** (*`array`*) –

Usage

Examples

See Also

- *`Omeka_View_Helper_FileMarkup::addMimeTypes`*

`add_file_fallback_image` — Add a fallback image for files of the given mime type or type family.

New in version 2.2.

Plugin-related functions

Summary

`add_file_fallback_image` (*$mimeType*, *$image*)
Add a fallback image for files of the given mime type or type family.

The fallback is used when there are no generated derivative images and one is requested (for example, by a call to file_image()).

Parameters

- **`$mimeType`** (*`string`*) – The mime type this fallback is for, or the mime "prefix" it is for (video, audio, etc.)
- **`$image`** (*`string`*) – The name of the image to use, as would be passed to img()

Usage

Examples

See Also

`add_filter` — Declare a filter implementation.

Plugin-related functions

Summary

`add_filter` (*$name*, *$callback*, *$priority = 10*)
Declare a filter implementation.

Parameters

- **`$name`** (*`string|array`*) – The filter name.
- **`$callback`** (*`callback`*) – The function to call.
- **`$priority`** (*`int`*) – Defaults to 10.

Usage

Examples

See Also

- *Understanding Filters*

`add_plugin_hook` — Declare a plugin hook implementation within a plugin.

Plugin-related functions

Summary

`add_plugin_hook` (*$hook*, *$callback*)
Declare a plugin hook implementation within a plugin.

Parameters

- **`$hook`** (*`string`*) – Name of hook being implemented.
- **`$callback`** (*`mixed`*) – Any valid PHP callback.

Usage

Examples

```
<?php add_plugin_hook('install', 'exhibit_builder_install'); ?>
```

See Also

`add_shortcode` — Add a shortcode.

New in version 2.2.

View-related functions

Summary

`add_shortcode` (*$shortcodeName*, *$function*)
Add a shortcode.

Parameters

- **`$shortcodeName`** (*string*) – Name of the new shortcode.
- **`$function`** (*callback*) – Callback to execute for this shortcode.

Usage

Examples

See Also

`add_translation_source` — Add an translation source directory.

Locale-related functions

Summary

`add_translation_source` (*$dir*)
Add an translation source directory.

The directory's contents should be .mo files following the naming scheme of Omeka's application/languages directory. If a .mo for the current locale exists, the translations will be loaded.

Parameters

- **`$dir`** (*string*) – Directory from which to load translations.

Usage

Examples

```
<?php

function exhibit_builder_initialize()
{
    add_translation_source(dirname(__FILE__) . '/languages');
}

?>
```

See Also

admin_url — Get a URL to the admin theme.

Navigation-related functions

Summary

admin_url()
Get a URL to the admin theme.

Returns string

Usage

This function takes the same arguments as *url*.

Examples

```
<a href="<?php echo admin_url(); ?>">Go to Omeka Admin</a>
```

See Also

all_element_texts — Get all element text metadata for a record.

View-related functions

Summary

all_element_texts(*$record*, *$options = array()*)
Get all element text metadata for a record.

Parameters

- **$record** (*Omeka_Record_AbstractRecord|string*) – The record to get the element text metadata for.
- **$options** (*array*) – Options for getting the metadata.

Returns string|array

Usage

Valid keys for the `$options` array are:

- `show_empty_elements`: Whether to include elements that have no values. If specified, this overrides the corresponding site setting. If a string is passed, empty elements will be displayed and the string will be used as the replacement text shown for the empty elements.
- `show_element_set_headings`: Whether to show the headings naming each element set (e.g., "Dublin Core"). If specified this overrides the corresponding site setting.
- `show_element_sets`: Array of names of element sets to show. If omitted, all sets are included (except totally empty sets when `show_empty_elements` is false).
- `return_type`: What kind of output the function should return. Must be one of `"html"` or `"array"`. HTML output is the default, and the usual use case. Array output, when selected here, returns the text data indexed by element set and element.
- `partial`: Path to the view partial script used to create the HTML output. By default the path is `'common/record-metadata.php'`.

Examples

Use the `show_empty_elements` option to override the site configurations, either always showing or always hiding the empty elements.

```
<?php echo all_element_texts('item', array('show_empty_elements' => true));?>
```

Use the `show_empty_elements` option to set the text for the empty elements. The text will only display if the "Show Empty Elements" is enabled in the site configurations.

```
<?php echo all_element_texts('item', array('show_empty_elements' => 'This data is not␣
↪available.'));?>
```

Use the `show_element_set_headings` option to override the site configurations, either always showing or always hiding the names of the different element sets.

```
<?php echo all_element_texts('item', array('show_element_set_headings' => true));?>
```

Use the `show_element_sets` option to dictate which element sets to display. Available sets include 'Item Type Metadata' and those listed at `'[your_Omeka_url]/admin/element-sets'`

```
<?php echo all_element_texts('item', array('show_element_sets' => array('Dublin Core',
↪ 'Item Type Metadata'))); ?>
```

See Also

`apply_filters` — Apply a set of plugin filters to a given value.

Plugin-related functions

Summary

apply_filters(*$name*, *$value*, *$args = array()*)
Apply a set of plugin filters to a given value.

Parameters

- **$name** (*string|array*) – The filter name.
- **$value** (*mixed*) – The value to filter.
- **$args** (*array*) – Additional arguments to pass to filter implementations.

Returns mixed Result of applying filters to $value.

Usage

Examples

See Also

- *Understanding Filters*

`auto_discovery_link_tags` — Get link tags for the RSS and Atom feeds.

Layout-related functions

Summary

This function outputs a <link> tag for the RSS feed so the browser can auto-discover the feed.

auto_discovery_link_tags()
Get link tags for the RSS and Atom feeds.

Returns string HTML

Usage

Examples

Use in the document `<head>` to add links to the RSS and Atom feeds.

```
<head>

  <?php echo auto_discovery_link_tags(); ?>

</head>
```

See Also

`body_tag` — Get a <body> tag with attributes.

View-related functions

Summary

`body_tag` (*$attributes = array()*)
Get a <body> tag with attributes.

Attributes can be filtered using the 'body_tag_attributes' filter.

Parameters

- **`$attributes`** (*array*) –

Returns string An HTML <body> tag with attributes and their values.

Usage

Examples

Use the `body_tag` function to set the `id` and `class` attributes for an HTML `body` tag.

```
<?php echo body_tag(array('id' => 'home', 'class' => 'two-col')); ?>
```

See Also

`browse_sort_links` — Get the list of links for sorting displayed records.

View-related functions

Summary

`browse_sort_links` (*$links*, *$wrapperTags = array()*)
Get the list of links for sorting displayed records.

Parameters

- **`$links`** (*array*) – The links to sort the headings. Should correspond to the metadata displayed.
- **`$wrapperTags`** (*array*) – The tags and attributes to use for the browse headings - 'list_tag' The HTML tag to use for the containing list - 'link_tag' The HTML tag to use for each list item (the browse headings) - 'list_attr' Attributes to apply to the containing list tag - 'link_attr' Attributes to apply to the list item tag

Returns string

Usage

Generates HTML elements for sorting based on the metadata given.

When the records are presented as a table, as in most admin browse views, `$wrapperTags` should be:

```
array('link_tag' => 'th scope="col"', 'list_tag' => '')
```

Otherwise, you will want to add text and/or styling to make clear that the list presented is for sorting the displayed records.

Examples

Use the `links` array to set the properties on which items can be sorted in the browse views. The labels are arbitrary, while the property names are determined by the element set or the record model.

```
<?php
  $sortLinks[__('Title')] = 'Dublin Core,Title';
  $sortLinks[__('Creator')] = 'Dublin Core,Creator';
  $sortLinks[__('Date Added')] = 'added';
?>
<div id="sort-links">
    <span class="sort-label"><?php echo __('Sort by: '); ?></span><?php echo browse_
↪sort_links($sortLinks); ?>
</div>
```

Include the `wrapperTags` array when creating a sortable table, as in the admin view. Note that the column with the "Type" label is not sortable because no property name has been entered.

```
<?php
$browseHeadings[__('Title')] = 'Dublin Core,Title';
$browseHeadings[__('Creator')] = 'Dublin Core,Creator';
$browseHeadings[__('Type')] = null;
$browseHeadings[__('Date Added')] = 'added';

echo browse_sort_links($browseHeadings, array('link_tag' => 'th scope="col"', 'list_
↪tag' => ''));
?>
```

See Also

`clear_filters` — Clear all implementations for a filter (or all filters).

Plugin-related functions

Summary

`clear_filters`(*$filterName = null*)
Clear all implementations for a filter (or all filters).

Parameters

- **`$filterName`** –

Usage

Examples

See Also

`common` — Get HTML from a view file in the common/ directory.

Layout-related functions

Summary

The common function loads a script, optionally passing in some variables. Themes and plugins can use this function to separate distinct blocks of output into different scripts, even if all the output will appear on one page.

By default, common tries to load the script from the "common" directory for the current theme, but any view directory can be specified.

common (*$file*, *$vars = array()*, *$dir = 'common'*)

Get HTML from a view file in the common/ directory.

Optionally, parameters can be passed to the view, and the view can be loaded from a different directory.

Parameters

- **$file** (*string*) – Filename
- **$vars** (*array*) – A keyed array of variables to be extracted into the script
- **$dir** (*string*) – Defaults to 'common'

Returns string

Usage

Examples

See Also

`css_src` — Get the URL to a local css file.

Asset-related functions

Summary

`css_src()` is a helper function used when referencing a css file within a theme, and commonly included within a theme's `header.php` file. It returns the path to a css file located in the css folder of that theme, usually located in `themes/YourTheme/css`.

Note: The *queue_css_file* helper is preferred to this one for most use cases.

css_src (*$file*, *$dir = 'css'*, *$version = OMEKA_VERSION*)
Get the URL to a local css file.

Parameters

- **$file** (*string*) – Should not include the .css extension
- **$dir** (*string*) – Defaults to 'css'
- **$version** (*mixed*) – Version number. By default OMEKA_VERSION.

Returns string

Usage

Examples

Add a link to a print.css file located in the theme's `css` directory. Use when you need to load css files outside of the head.php file.

```
<head>
    <link href="<?php echo css_src('print'); ?>" media="all" rel="stylesheet" type=
↪"text/css" />
</head>
```

See Also

- *queue_css_file — Add a CSS file or files to the current page.*

current_url — Get the current URL with query parameters appended.

Navigation-related functions

Summary

current_url (*$params = array()*)
Get the current URL with query parameters appended.

Instantiates view helpers directly because a view may not be registered.

Parameters

- **$params** (*array*) –

Returns string

Usage

Examples

See Also

`current_user` — Return the currently logged in User record.

User-related functions

Summary

`current_user`()
Return the currently logged in User record.

Returns User|null Null if no user is logged in.

Usage

Examples

See Also

`debug` — Log a message with 'DEBUG' priority.

Log-related functions

Summary

`debug`(*$msg*)
Log a message with 'DEBUG' priority.

Parameters

- **`$msg`** (*`string`*) –

Usage

Examples

See Also

`delete_option` — Delete an option from the options table.

Option-related functions

Summary

`delete_option`(*$name*)
Delete an option from the options table.

Parameters

- **`$name`** (*`string`*) – The option name.

Usage

Examples

See Also

`element_exists` — Check whether an element set contains a specific element.

ElementSet-related functions

Summary

`element_exists`(*$elementSetName*, *$elementName*)

Check whether an element set contains a specific element.

Parameters

- **`$elementSetName`** (*`string`*) – The element set name.
- **`$elementName`** (*`string`*) – The element name.

Returns bool

Usage

Examples

See Also

`element_form` — Get the HTML for a form input for a given Element.

Form-related functions

Summary

`element_form`(*$element*, *$record*, *$options = array()*)

Get the HTML for a form input for a given Element.

Assume that the given element has access to all of its values (for example, all values of a Title element for a given Item).

This will output as many form inputs as there are values for a given element. In addition to that, it will give each set of inputs a label and a span with class="tooltip" containing the description for the element. This span can either be displayed, hidden with CSS or converted into a tooltip with javascript.

All sets of form inputs for elements will be wrapped in a div with class="field".

Parameters

- **`$element`** (*`Element|array`*) –
- **`$record`** (`Omeka_Record_AbstractRecord`) –
- **`$options`** (*`array`*) –

Returns string HTML

Usage

Examples

See Also

`element_set_form` — Return a element set form for a record.

Form-related functions

Summary

`element_set_form`(*$record*, *$elementSetName*)
Return a element set form for a record.

Parameters

- **`$record`** (`Omeka_Record_AbstractRecord`) –
- **`$elementSetName`** (*`string`*) – The name of the element set or 'Item Type Metadata' for an item's item type data.

Returns string

Usage

Examples

See Also

`file_display_url` — Get the provided file's URL.

Navigation-related functions

Summary

`file_display_url`(*File $file*, *$format = 'fullsize'*)
Get the provided file's URL.

Parameters

- **`$file`** (`File`) –
- **`$format`** (*`string`*) –

Returns string

Usage

Examples

See Also

`file_id3_metadata` — Return HTML for a file's ID3 metadata.

File-related functions

Summary

`file_id3_metadata` (*$options = array()*, *$file = null*)
Return HTML for a file's ID3 metadata.

Parameters

- **`$options`** (*`array`*) – Options for varying the display. Currently ignored.
- **`$file`** (*`File|null`*) – File to get the metadata for. If omitted, the current file is used.

Returns stringlarray

Usage

Examples

See Also

`file_image` — Get a customized file image tag.

New in version 2.2.

File-related functions

Summary

`file_image` (*$imageType*, *$props = array()*, *$file = null*)
Get a customized file image tag.

Parameters

- **`$imageType`** (*`string`*) – Image size: thumbnail, square thumbnail, fullsize
- **`$props`** (*`array`*) – HTML attributes for the img tag.
- **`$file`** –

Usage

Display the image files and assign css selectors.

Takes a string for the image size and an array of html attributes.

Valid strings for `$imageType` include:

- `thumbnail`
- `square_thumbnail`
- `fullsize`
- `original`

Examples

Display a square thumbnail for an item and wrap it with `class = "thumbnail"`

```
<div class="item-collapsed">
    <?php echo file_image('square_thumbnail', array('class' => 'thumbnail'), $file); ?
↪>
</div>
```

See Also

`file_markup` — Get HTML for a set of files.

File-related functions

Summary

`file_markup` (*$files*, *$props = array()*, *$wrapperAttributes = array('class' => 'item-file')*)
Get HTML for a set of files.

Parameters

- **`$files`** (`File`) – A file record or an array of File records to display.
- **`$props`** (*`array`*) – Properties to customize display for different file types.
- **`$wrapperAttributes`** (*`array`*) – Attributes HTML attributes for the div that wraps each displayed file. If empty or null, this will not wrap the displayed file in a div.

Returns string HTML

Usage

The second argument, `$props`, is an array of options to customize the display of files. Which options are valid depend on the particular file type and on whether a plugin has added a new method for displaying that file, but there are some fairly standard available options.

Options for many possible file types can be specified all together. Each file will only use the options that are valid for its type, and ignore the others.

imageSize Available for images. Default: `square_thumbnail`

A string naming the size of image to use. The possible sizes are `thumbnail`, `square_thumbnail`, and `fullsize`.

linkToFile Available for images and unknown-type files. Default: `true`

Whether the display should be wrapped in a direct link directly the file. `true` makes a link to the original file, `false` makes no link, and a string links to the specified image size (thumbnail, square_thumbnail, or fullsize).

linkToMetadata Available for images and unknown-type files. Default: `false`

If `true`, wrap the file in a link to the file's "show" metadata page. This option overrides `linkToFile` when `true`.

imgAttributes Available for images. Default: none

An array of attributes to set on the HTML `<img>` tag. Keys of the array are attribute names, and values of the array are attribute values.

linkText Available for unknown-type files. Default: `null`

The text to display for the file (inside the link if one of the linking options is enabled). If null or omitted, the file's Dublin Core Title (if available) or original filename are used as the text.

width, height Available for most audio and video types. Defaults vary.

The pixel height and width of the audio or video player.

Examples

`file_markup` can be passed just a file or array of files alone to produce the default display:

```
// Display one specific File
echo file_markup($file);

// Display all the given Files (here, the files for an Item)
echo file_markup($item->Files);
```

One of the simplest uses of `file_markup`'s display options is to display a fullsize image instead of a small square thumbnail:

```
echo file_markup($file, array('imageSize' => 'fullsize'));
```

See Also

- *Omeka_View_Helper_FileMarkup*
- *add_file_display_callback*

`files_for_item` — Get HTML for all files assigned to an item.

Item-related functions

Summary

files_for_item(*$options = array()*, *$wrapperAttributes = array('class' => 'item-file')*, *$item = null*)
Get HTML for all files assigned to an item.

Parameters

- **$options** (*array*) –
- **$wrapperAttributes** (*array*) –
- **$item** (*Item|null*) – Check for this specific item record (current item if null).

Returns string HTML

Usage

Valid keys for the `$options` array are the same as those for the `file_markup` function's `$props` argument. For details, see *the Usage section for file_markup*.

Attributes given in `$wrapperAttributes` are also simply passed as-is to the corresponding argument to `file_markup`.

Examples

To display image icons without a link to the image files:

```
<div class="element-text">
    <?php echo files_for_item(
        array(
            'linkToFile' => false
        )
    ); ?>
</div>
```

To add an attribute such as `rel="lightbox"` to the image links for use with lightbox:

```
<div class="element-text">
    <?php echo files_for_item(
        array(
            'linkAttributes' => array('rel' => 'lightbox')
        )
    ); ?>
</div>
```

To add css class and id selectors to the images:

```
<div class="element-text">
    <?php echo files_for_item(
        array(
           'imgAttributes' => array('id' => 'foobar')
        )
    ); ?>
</div>
```

To change the size of the images displayed for the item, for example to a fullsize image (this may be constrained by css):

```
<div class="element-text">
    <?php echo files_for_item(
        array(
            'imageSize' => 'fullsize'
        )
    ); ?>
</div>
```

To change the default icon displayed, particularly if an image is not generated for a particular file associated with an item. Image files must be located in 'application/views/scripts/images'.

```
<div class="element-text">
    <?php echo files_for_item(
        array(
            'icons' => array('audio/mgep' => img('audio.gif'))
        )
    ); ?>
</div>
```

See Also

`fire_plugin_hook` — Declare the point of execution for a specific plugin hook.

Plugin-related functions

Summary

`fire_plugin_hook` (*$name*, *$args = array()*)

Declare the point of execution for a specific plugin hook.

All plugin implementations of a given hook will be executed when this is called. The first argument corresponds to the string name of the hook. The second is an associative array containing arguments that will be passed to the plugin hook implementations.

<code> // Calls the hook 'after_save_item' with the arguments '$item' and '$arg2' fire_plugin_hook('after_save_item', array('item' => $item, 'foo' => $arg2)); </code>

Parameters

- **`$name`** (*`string`*) – The hook name.
- **`$args`** (*`array`*) – Arguments to be passed to the hook implementations.

Usage

Examples

See Also

`flash` — Return a flashed message from the controller.

View-related functions

Summary

flash()
Return a flashed message from the controller.

Returns string

Usage

Examples

Use on pages associated with database actions (CRUD) to display messages from the controllers. Most often used in views for plugins.

```
<?php echo flash(); ?>
```

See Also

foot — Get the view's footer HTML.

Layout-related functions

Summary

This function includes the template footer.php at the end of a page.

foot(*$vars = array()*, *$file = 'footer'*)
Get the view's footer HTML.

Parameters

- **$vars** (*array*) – Keyed array of variables
- **$file** (*string*) – Filename of footer script (defaults to 'footer')

Returns string

Usage

Examples

See Also

format_date — Format a date for output according to the current locale.

Locale-related functions

Summary

format_date(*$date*, *$format = Zend_Date::DATE_MEDIUM*)
Format a date for output according to the current locale.

Parameters

- **$date** (*mixed*) – Date to format. If an integer, the date is intepreted as a Unix timestamp. If a string, the date is interpreted as an ISO 8601 date.
- **$format** (*string*) – Format to apply. See Zend_Date for possible formats. The default format is the current locale's "medium" format.

Returns string

Usage

Examples

See Also

- *Internationalization*

get_acl — Get the ACL object.

User-related functions

Summary

get_acl()
Get the ACL object.

Returns Zend_Acl

Usage

Examples

See Also

get_collection_for_item — Get the Collection object for the current item.

Item-related functions

Summary

get_collection_for_item(*$item = null*)
Get the Collection object for the current item.

Parameters

- **$item** (*Item|null*) – Check for this specific item record (current item if null).

Returns Collection

Usage

Examples

Print the title and description of the Collection the current Item belongs to (if any):

```
<?php
$collection = get_collection_for_item();
if ($collection):
?>
    <h3><?php echo metadata($collection, array('Dublin Core', 'Title')); ?></h3>
    <p><?php echo metadata($collection, array('Dublin Core', 'Description')); ?></p>
<?php endif; ?>
```

See Also

get_current_action_contexts — Get all output formats available in the current action.

OutputFormat-related functions

Summary

get_current_action_contexts()
Get all output formats available in the current action.

Returns array A sorted list of contexts.

Usage

Examples

See Also

get_current_record — Get the current record from the view.

Loop-related functions

Summary

get_current_record(*$recordVar*, *$throwException = true*)
Get the current record from the view.

Parameters

- **$recordVar** (*string*) – View variable the current record is stored in.

- **$throwException** (*bool*) – Whether to throw an exception if no current record was set. The default is to throw.

Returns Omeka_Record_AbstractRecordlfalse

Usage

Examples

Use the $throwException boolean to turn off the default alert when no current record is set.

```
<?php echo get_current_record('item', false); ?>
```

See Also

get_custom_search_record_types — Get all record types that have been customized to be searchable.

Search-related functions

Summary

get_custom_search_record_types()
Get all record types that have been customized to be searchable.

Returns array

Usage

Examples

See Also

get_db — Get the database object.

Db-related functions

Summary

get_db()
Get the database object.

Returns Omeka_Db

Usage

Examples

See Also

`get_html_lang` — Get the HTML "lang" attribute for the current locale.

Locale-related functions

Summary

`get_html_lang`()
Get the HTML "lang" attribute for the current locale.

Returns string

Usage

Examples

See Also

`get_loop_records` — Get records from the view for iteration.

Loop-related functions

Summary

`get_loop_records`(*$recordsVar*, *$throwException = true*)
Get records from the view for iteration.

Note that this function will return an empty array if it is set to the records variable. Use has_loop_records() to check if records exist.

Parameters

- **`$recordsVar`** (*`string`*) – The name of the variable the records are stored in.
- **`$throwException`** (*`bool`*) – Whether to throw an exception if the $recordsVar is unset. Default is to throw.

Returns array|bool

Usage

Examples

See Also

`get_next_item` — Get the next item in the database.

Item-related functions

Summary

`get_next_item`(*$item = null*)
Get the next item in the database.

Parameters

- **`$item`** (*`Item|null`*) – Check for this specific item record (current item if null).

Returns Item|null

Usage

Examples

See Also

`get_option` — Get an option from the options table.

Option-related functions

Summary

`get_option`(*$name*)
Get an option from the options table.

If the returned value represents an object or array, it must be unserialized by the caller before use. For example: <code> $object = unserialize(get_option('plugin_object')); </code>

Parameters

- **`$name`** (*`string`*) – The option name.

Returns string The option value.

Usage

Examples

See Also

`get_plugin_broker` — Get the broker object for Omeka plugins.

Plugin-related functions

Summary

get_plugin_broker()
Get the broker object for Omeka plugins.

Returns Omeka_Plugin_Broker|null

Usage

Examples

See Also

get_plugin_hook_output — Get the output of fire_plugin_hook() as a string.

Plugin-related functions

Summary

get_plugin_hook_output(*$name*, *$args = array()*)
Get the output of fire_plugin_hook() as a string.

Parameters

- **$name** (*string*) – The hook name.
- **$args** (*array*) – Arguments to be passed to the hook implementations.

Returns string

Usage

Examples

See Also

get_plugin_ini — Get specified descriptive info for a plugin from its ini file.

Plugin-related functions

Summary

get_plugin_ini(*$pluginDirName*, *$iniKeyName*)
Get specified descriptive info for a plugin from its ini file.

Parameters

- **$pluginDirName** (*string*) – The directory name of the plugin.
- **$iniKeyName** (*string*) – The name of the key in the ini file.

Returns string|null The value of the specified plugin key. If the key does not exist, it returns null.

Usage

Examples

See Also

`get_previous_item` — Get the previous item in the database.

Item-related functions

Summary

`get_previous_item`(*$item = null*)
Get the previous item in the database.

Parameters

- **`$item`** (*`Item|null`*) – Check for this specific item record (current item if null).

Returns Item|null

Usage

Examples

See Also

`get_random_featured_collection` — Get a random featured collection.

Collection-related functions

Summary

`get_random_featured_collection`()
Get a random featured collection.

Returns Collection

Usage

Examples

See Also

`get_random_featured_items` — Get random featured items.

Item-related functions

Summary

get_random_featured_items (*$num = 5*, *$hasImage = null*)
Get random featured items.

Parameters

- **$num** (*int*) – The maximum number of recent items to return
- **$hasImage** (*bool|null*) –

Returns array|Item

Usage

Examples

To display a given number of featured items on a page, use this function and pass in the number of items desired:

```
<div id="featured-item">
    <h2><?php echo __('Featured Item'); ?></h2>
    <?php echo random_featured_items(2); ?>
</div>
```

See Also

get_recent_collections — Get the most recently added collections.

Collection-related functions

Summary

get_recent_collections (*$num = 10*)
Get the most recently added collections.

Parameters

- **$num** (*int*) – The maximum number of recent collections to return

Returns array

Usage

Examples

See Also

get_recent_files — Get the most recent files.

Db-related functions

Summary

get_recent_files (*$num = 10*)
Get the most recent files.

Parameters

- **$num** (*int*) – The maximum number of recent files to return

Returns array

Usage

Examples

See Also

get_recent_items — Get the most recently added items.

Item-related functions

Summary

get_recent_items (*$num = 10*)
Get the most recently added items.

Parameters

- **$num** (*int*) – The maximum number of recent items to return

Returns array

Usage

Examples

Use the `$num` parameter to set the number of recent items to display. Default is 10 items. Often used with `set_loop_records`.

```
<?php set_loop_records('items', get_recent_items(5)); ?>
```

See Also

get_recent_tags — Get the most recent tags.

View-related functions

Summary

get_recent_tags (*$limit = 10*)
Get the most recent tags.

Parameters

- **$limit** (*int*) – The maximum number of recent tags to return

Returns array

Usage

Examples

Use the parameter to set the maximum number of recent tags to be displayed:

```
<?php echo get_recent_tags(5); ?>
```

See Also

get_record — Get a single record from the database.

New in version 2.1.

Db-related functions

Summary

get_record (*$recordType*, *$params = array()*)
Get a single record from the database.

Parameters

- **$recordType** (*string*) – Type of records to get.
- **$params** (*array*) – Array of search parameters for records.

Returns object An object of result records (of $recordType).

Usage

Examples

See Also

get_record_by_id — Get a record by its ID.

Db-related functions

Summary

get_record_by_id(*$modelName*, *$recordId*)
Get a record by its ID.

Parameters

- **$modelName** (*string*) – Name of the Record model being looked up (e.g., 'Item')
- **$recordId** (*int*) – The ID of the specific record to find.

Returns Omeka_Record_AbstractRecord|null The record, or null if it cannot be found.

Usage

Examples

See Also

get_records — Get a set of records from the database.

Db-related functions

Summary

get_records(*$recordType*, *$params = array()*, *$limit = 10*)
Get a set of records from the database.

Parameters

- **$recordType** (*string*) – Type of records to get.
- **$params** (*array*) – Array of search parameters for records.
- **$limit** (*int*) – Maximum number of records to return.

Returns array An array of result records (of $recordType).

Usage

Examples

`get_records()` retrieves different kind of records from the database. You can use it when other record queries don't work for you.

```
$items = get_records('Item',array('tags'=>$itemTags, 'range'=>$itemIds, 'collection'=>
↪$itemCollection, 'type'=>$itemType),50);
```

This example uses four different parameters in the `$params` array to narrow down the range of **items** selected:

- `tags` uses a string of tag names to lookup items by tag
- `range` allows you to select records by ID ranges. For example: `1-5,8,15`
- `collection` allows you to select a given collection ID.

- `type` allows you to select a given item type by ID.

```
$exhibits = get_records('Exhibit', array('slug'=>$exhibitSlugs, 'tags'=>$exhibitTags),
↪ 50);
```

This example uses two different parameters in the `$params` array to narrow down the range of **exhibits** selected:

- `slug` allows you to select exhibits by comma-separated slugs
- `tags` uses a string of tag names to lookup exhibits by tag

In both cases, you can then iterate through the array of records that is returned in order to display them or perform further processing.

The complete list of parameters that you can filter records by can be found in the model objects that extend **Omeka_Db_Table**. See list of classes in the `application/models/Table` directory. In particular, look at the `applySearchFilters()` method that is attached to each class.

See Also

`get_search_query_types` — Get all available search query types.

Search-related functions

Summary

`get_search_query_types``()`
Get all available search query types.

Plugins may add query types via the "search_query_types" filter. The keys should be the type's GET query value and the respective values should be the human readable and internationalized version of the query type.

Plugins that add a query type must modify the select object via the "search_sql" hook to account for whatever custom search strategy they implement.

Returns array

Usage

Examples

See Also

`get_search_record_types` — Get all record types that may be indexed and searchable.

Search-related functions

Summary

`get_search_record_types``()`
Get all record types that may be indexed and searchable.

Plugins may add record types via the "search_record_types" filter. The keys should be the record's class name and the respective values should be the human readable and internationalized version of the record type.

These record classes must extend Omeka_Record_AbstractRecord and implement this search mixin (Mixin_Search).

Returns array

Usage

Examples

See Also

`get_specific_plugin_hook_output` — Get the output of a specific plugin's hook as a string.

Plugin-related functions

Summary

`get_specific_plugin_hook_output` (*$pluginName*, *$hookName*, *$args = array()*)
Get the output of a specific plugin's hook as a string.

This is like get_plugin_hook_output() but only calls the hook within the provided plugin.

Parameters

- **`$pluginName`** (*`string`*) – Directory name of the plugin to execute the hook for.
- **`$hookName`** (*`string`*) – Name of the hook to fire.
- **`$args`** (*`mixed`*) – Any arguments to be passed to the hook implementation.

Returns string|null

Usage

Examples

See Also

`get_table_options` — Get the options array for a given table.

Form-related functions

Summary

`get_table_options` (*$tableClass*, *$labelOption = null*, *$searchParams = array()*)
Get the options array for a given table.

Parameters

- **`$tableClass`** (*`string`*) –
- **`$labelOption`** (*`string`*) –
- **`$searchParams`** (*`array`*) – search parameters on table.

Usage

Examples

See Also

`get_theme_option` — Get a theme option.

Option-related functions

Summary

`get_theme_option`(*$optionName*, *$themeName* = *null*)
Get a theme option.

Parameters

- **`$optionName`** (*`string`*) – The option name.
- **`$themeName`** (*`string`*) – The theme name. If null, it will use the current public theme.

Returns string The option value.

Usage

Examples

See Also

`get_user_roles` — Get the names of all user roles.

User-related functions

Summary

`get_user_roles`()
Get the names of all user roles.

Returns array

Usage

Examples

See Also

`get_view` — Get the view object.

View-related functions

Summary

get_view()
Get the view object.

Should be used only to avoid function scope issues within other theme helper functions.

Returns Omeka_View

Usage

Examples

Use in the `config_form.php` file of a plugin to load the view object. For more information on the available view helpers, see the Zend documentation

```
<?php $view = get_view(); ?>

<div class="inputs five columns omega">
  <p class="explanation"><?php echo __("Latitude of the map's initial center point,␣
↪in degrees. Must be between -90 and 90."); ?></p>
  <?php echo $view->formText('default_latitude', get_option('geolocation_default_
↪latitude')); ?>
</div>
```

See Also

has_loop_records — Check if records have been set to the view for iteration.

Loop-related functions

Summary

has_loop_records(*$recordsVar*)
Check if records have been set to the view for iteration.

Note that this function will return false if the records variable is set but is an empty array, unlike get_loop_records(), which will return the empty array.

Parameters

- **$recordsVar** (*string*) – View variable to check.

Returns bool

Usage

Examples

See Also

head — Get the view's header HTML.

Layout-related functions

Summary

This function includes the file header.php in a theme. A view script needs to use this function along with *foot* to create the structure of the page.

head(*$vars = array()*, *$file = 'header'*)
Get the view's header HTML.

Parameters

- **$vars** (*array*) – Keyed array of variables
- **$file** (*string*) – Filename of header script (defaults to 'header')

Returns string

Usage

Examples

You can use the $vars array to control the 'id' and 'class' information for the body element of a given page:

```
<?php echo head(array('bodyid'=>'home', 'bodyclass' =>'two-col')); ?>
```

While the page title is generated from the page metadata by default, you can set a page title manually by passing a 'title' to the head function:

```
<?php
$title = __('Explorations');
echo head(array('title' => $title, 'bodyclass' => 'explorations browse'));
?>
```

See Also

head_css — Get the CSS link tags that will be used on the page.

Asset-related functions

Summary

The `head_css()` helper function prints HTML link tags to the page for each stylesheet added with *queue_css_file*. It is commonly used in a theme's `common/header.php` file to print the link tags inside the page head.

head_css()
Get the CSS link tags that will be used on the page.

This should generally be used with echo to print the scripts in the page head.

Returns string

Usage

Examples

Use after `queue_css_file()`, `queue_css_url()`, and/or `queue_css_string()` to add the links to the css files and/or the css declarations to the page head:

```
<?php
    queue_css_file('style');
    head_css();
?>
```

See Also

- *queue_css_file — Add a CSS file or files to the current page.*
- *queue_css_url — Add a CSS file to the current page by URL.*
- *queue_css_string — Add a CSS string to the current page.*

`head_js` — Get the JavaScript tags that will be used on the page.

Asset-related functions

Summary

The `head_js()` helper function prints HTML script tags to the page for each script added with *`queue_js_file`*. It is commonly used in a theme's `common/header.php` file to print the script tags inside the page head.

head_js() will also include Omeka's "default" JavaScript files.

`head_js`(*$includeDefaults = true*)

Get the JavaScript tags that will be used on the page.

This should generally be used with echo to print the scripts in the page head.

Parameters

- **`$includeDefaults`** (*`bool`*) – Whether the default javascripts should be included. Defaults to true.

Returns string

Usage

Examples

Use after all desired javascript files, urls, and strings have been queued to add the javascript information to the page head.

```
<?php
    queue_js_file(array('my_js_file', 'another_js_file'));
    queue_js_url('//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js');
    queue_js_string("
      window.addEventListener('load', function() {
          FastClick.attach(document.body);
      }, false);
");
?>

<?php head_js(); ?>
```

See Also

- *queue_js_file — Add a local JavaScript file or files to the current page.*
- *queue_js_url — Add a JavaScript file to the current page by URL.*
- *queue_js_string — Add a JavaScript string to the current page.*

`html_escape` — Escape a value to display properly as HTML.

Text-related functions

Summary

`html_escape`(*$value*)

Escape a value to display properly as HTML.

This uses the 'html_escape' filter for escaping.

Parameters

- **`$value`** (*string*) –

Returns string

Usage

To avoid Cross Site Scripting attacks, any data coming from a client (`$_GET`, `$_POST`, etc) should be escaped.

Examples

```
<?php echo html_escape($_GET['search_text']); ?>
```

See Also

`img` — Get the URL to a local image file.

Asset-related functions

Summary

`img()` is a helper function used to include an image from within a theme. It returns the URL to an image file located in the images directory of that theme, usually located in themes/YourTheme/images. The resulting path can be used in an <img> tag, or anywhere else an image URL is needed.

img (*$file*, *$dir = 'images'*)
Get the URL to a local image file.

Parameters

- **$file** (*string*) – Filename, including the extension.
- **$dir** (*string*) – Directory within the theme to look for image files. Defaults to 'images'.

Returns string

Usage

Examples

Use `img()` to include a link to an image stored in `themes/YourTheme/images`:

```
<?php
    $themeScreenshot = img('fallback-theme.png');
?>

<div class="screenshot">
    <img src="<?php echo $themeScreenshot; ?>" />
</div>
```

See Also

insert_collection — Insert a collection

Collection-related functions

Summary

insert_collection (*$metadata = array()*, *$elementTexts = array()*)
Insert a collection

Parameters

- **$metadata** (*array*) – Follows the format: <code> array('public' => [true|false], 'featured' => [true|false]) </code>
- **$elementTexts** (*array*) – Array of element texts to assign to the collection. This follows the format: <code> array([element set name] => array([element name] => array(array('text' => [string], 'html' => [false|true]), array('text' => [string], 'html' => [false|true])), [element name] => array(array('text' => [string], 'html' => [false|true]), array('text' => [string], 'html' => [false|true]))), [element set name] => array([element name] => array(array('text' => [string], 'html' => [false|true]), array('text' => [string], 'html' => [false|true])), [element name] => array(array('text' => [string], 'html' => [false|true]), array('text' => [string], 'html' => [false|true])))); </code>

Returns Collection

Usage

Examples

See Also

`insert_element_set` — Insert an element set and its elements into the database.

ElementSet-related functions

Summary

`insert_element_set` (*$elementSetMetadata = array()*, *$elements = array()*)
Insert an element set and its elements into the database.

Parameters

- **`$elementSetMetadata`** (*`string|array`*) – Element set information. <code> [(string) element set name] // OR array('name' => [(string) element set name, required, unique], 'description' => [(string) element set description, optional], 'record_type' => [(string) record type name, optional]); </code>
- **`$elements`** (*`array`*) – An array containing element data. Follows one of more of the following formats: <ol> An array containing element metadata A string of the element name <code> array(array('name' => [(string) name, required], 'description' => [(string) description, optional],), [(string) element name]); </code>

Returns ElementSet

Usage

Examples

See Also

`insert_files_for_item` — Add files to an item.

Item-related functions

Summary

`insert_files_for_item` (*$item*, *$transferStrategy*, *$files*, *$options = array()*)
Add files to an item.

Parameters

- **`$item`** (*`Item|int`*) – Item record or ID of item to add files to

- **$transferStrategy** (*string|Omeka_File_Ingest_AbstractIngest*) – Strategy to use when ingesting the file. The strings 'Url', 'Filesystem' and 'Upload' correspond to those built-in strategies. Alternatively a strategy object can be passed.
- **$files** (*array*) – Information about the file(s) to ingest. See addFiles() for details
- **$options** (*array*) – Array of options to modify the behavior of the ingest. Available options include: - 'ignore_invalid_files': boolean, false by default. Whether or not to throw exceptions when a file is not valid. - 'ignoreNoFile': (for Upload only) boolean, false by default. Whether to ignore validation errors that occur when an uploaded file is missing, like when a file input is left empty on a form.

Returns array The added File records.

Usage

Examples

See Also

insert_item — Insert a new item into the Omeka database.

Item-related functions

Summary

insert_item(*$metadata = array()*, *$elementTexts = array()*, *$fileMetadata = array()*)

Insert a new item into the Omeka database.

Parameters

- **$metadata** (*array*) – Set of metadata options for configuring the item. The array can include the following properties: - 'public' (boolean) - 'featured' (boolean) - 'collection_id' (integer) - 'item_type_id' (integer) - 'item_type_name' (string) - 'tags' (string, comma-delimited) - 'overwriteElementTexts' (boolean) – determines whether or not to overwrite existing element texts. If true, this will loop through the element texts provided in $elementTexts, and it will update existing records where possible. All texts that are not yet in the DB will be added in the usual manner. False by default.
- **$elementTexts** (*array*) – Array of element texts to assign to the item. This follows the following format:: array([element set name] => array([element name] => array(array('text' => [string], 'html' => [false|true]), array('text' => [string], 'html' => [false|true])), [element name] => array(array('text' => [string], 'html' => [false|true]), array('text' => [string], 'html' => [false|true]))), [element set name] => array([element name] => array(array('text' => [string], 'html' => [false|true]), array('text' => [string], 'html' => [false|true])), [element name] => array(array('text' => [string], 'html' => [false|true]), array('text' => [string], 'html' => [false|true]))));
- **$fileMetadata** (*array*) – Settings and data used to ingest files into Omeka and add them to this item. Includes the following options: - 'file_transfer_type' (string = 'Url|Filesystem|Upload' or Omeka_File_Transfer_Adapter_Interface). Corresponds to the $transferStrategy argument for addFiles(). - 'file_ingest_options' (array) Optional array of options to modify the behavior of the ingest. Corresponds to the $options argument for addFiles(). - 'files' (array or string) Represents information indicating the file to ingest. Corresponds to the $files argument for addFiles().

Returns Item

Usage

Examples

See Also

`insert_item_type` — Insert a new item type.

ItemType-related functions

Summary

`insert_item_type` (*$metadata = array()*, *$elementInfos = array()*)
Insert a new item type.

Parameters

- **`$metadata`** (*`array`*) – Follows the format: <code> array('name' => [string], 'description' => [string]); </code>
- **`$elementInfos`** (*`array`*) – An array containing element data. Each entry follows one or more of the following formats: <ol> An array containing element metadata An Element object <code> array(array('name' => [(string) name, required], 'description' => [(string) description, optional], 'order' => [(int) order, optional],), [(Element)],); </code>

Returns ItemType

Usage

Examples

See Also

`is_admin_theme` — Determine whether the script is being executed through the admin interface.

View-related functions

Summary

`is_admin_theme` ()
Determine whether the script is being executed through the admin interface.

Can be used to branch behavior based on whether or not the admin theme is being accessed, but should not be relied upon in place of using the ACL for controlling access to scripts.

Returns bool

Usage

Examples

Use to check if user is accessing pages from within the `admin` views and manage routes accordingly.

```
// Don't add these routes on the admin side to avoid conflicts.
if (is_admin_theme()) {
    return;
}
```

See Also

`is_allowed` — Check whether the current user has a give permission.

User-related functions

Summary

is_allowed(*$resource*, *$privilege*)

Check whether the current user has a give permission.

Parameters

- **$resource** (*string|Zend_Acl_Resource_Interface*) – The name of a resource, or a record implementing Zend_Acl_Resource_Interface
- **$privilege** (*string|null*) – The privilege to check for the resource.

Returns bool

Usage

Examples

See Also

`is_current_url` — Check if the given URL matches the current request URL.

Navigation-related functions

Summary

is_current_url(*$url*)

Check if the given URL matches the current request URL.

Instantiates view helpers directly because a view may not be registered.

Parameters

- **$url** (*string*) –

Returns bool

Usage

Examples

See Also

item_image — Get a customized item image tag.

Item-related functions

Summary

item_image (*$imageType = null*, *$props = array()*, *$index = 0*, *$item = null*)
Get a customized item image tag.

Parameters

- **$imageType** (*string*) – Image size: thumbnail, square thumbnail, fullsize
- **$props** (*array*) – HTML attributes for the img tag
- **$index** (*int*) – Which file within the item to use, by order. Default is the first file.
- **$item** –

Usage

Examples

Use to display an image file associated with an item. Use the parameters to control the image size and combine with `link_to_item` to wrap the image tag in an anchor tag.

```
<?php echo link_to_item(item_image('square_thumbnail', array('alt' => $itemTitle))); ?
↪>
```

See Also

item_image_gallery — Get a gallery of file thumbnails for an item.

Item-related functions

Summary

item_image_gallery (*$attrs = array()*, *$imageType = 'square_thumbnail'*, *$filesShow = null*, *$item = null*)
Get a gallery of file thumbnails for an item.

Parameters

- **$attrs** (*array*) – HTML attributes for the components of the gallery, in sub-arrays for 'wrapper', 'linkWrapper', 'link', and 'image'. Set a wrapper to null to omit it.
- **$imageType** (*string*) – The type of derivative image to display.

- **$filesShow** (*bool*) – Whether to link to the files/show. Defaults to null, uses 'link to file metadata' appearance option.
- **$item** (Item) – The Item to use, the current item if omitted.

Returns string

Usage

Examples

Use the $attrs array to assign HTML attributes for each image in the gallery for an item.

Use 'wrapper' to set the attributes for gallery block div:

```
<?php echo item_image_gallery(
  array('wrapper' => array('class' => 'gallery')));
?>
```

Use 'linkWrapper' to set the attributes on the div that surrounds each image:

```
<?php echo item_image_gallery(
  array('wrapper' => array('class' => 'gallery'),
  'linkWrapper' => array('class' => 'admin-thumb panel')));
?>
```

Use 'link' to set the attributes for the a tag:

```
<?php echo item_image_gallery(
  array('wrapper' => array('class' => 'gallery'),
  'linkWrapper' => array('class' => 'admin-thumb panel'),
  'link' => array('class' => 'link')));
?>
```

Use 'image' to set the attribute for the img tag:

```
<?php echo item_image_gallery(
  array('wrapper' => array('class' => 'gallery'),
  'linkWrapper' => array('class' => 'admin-thumb panel'),
  'link' => array('class' => 'link'),
  'image' => array('class' => 'image')));
?>
```

Omit the default 'wrapper' attributes by setting it to null:

```
<?php echo item_image_gallery(
  array('wrapper' => null,
  'linkWrapper' => array('class' => 'admin-thumb panel'),
  'link' => array('class' => 'link'),
  'image' => array('class' => 'image')));
?>
```

Use the $imageType parameter to set which image derivative is used in the gallery. Available options are square_thumbnail, thumbnail, fullsize, and original:

```
<?php echo item_image_gallery(
  array('wrapper' => null,
  'linkWrapper' => array('class' => 'admin-thumb panel'),
  'link' => array('class' => 'link'),
  'image' => array('class' => 'image')),
  'thumbnail');
?>
```

Set the `$filesShow` boolean to true to link each image file to its file/show page:

```
<?php echo item_image_gallery(
  array('wrapper' => null,
  'linkWrapper' => array('class' => 'admin-thumb panel'),
  'link' => array('class' => 'link'),
  'image' => array('class' => 'image')),
  'thumbnail',
  true);
?>
```

See Also

`item_search_filters` — Get a list of the current search item filters in use.

Search-related functions

Summary

item_search_filters(*$params = null*)
Get a list of the current search item filters in use.

Parameters

- **$params** –

Returns string

Usage

Examples

See Also

`item_type_elements` — Get the set of values for item type elements.

ItemType-related functions

Summary

item_type_elements(*$item = null*)
Get the set of values for item type elements.

Parameters

- **$item** (*Item|null*) – Check for this specific item record (current item if null).

Returns array

Usage

Examples

See Also

items_output_url — Get a URL to an output format page.

Navigation-related functions

Summary

items_output_url(*$output*, *$otherParams = array()*)
Get a URL to an output format page.

Parameters

- **$output** (*string*) –
- **$otherParams** (*array*) –

Returns string

Usage

Examples

See Also

items_search_form — Return the HTML for an item search form.

Search-related functions

Summary

items_search_form(*$props = array()*, *$formActionUri = null*, *$buttonText = null*)
Return the HTML for an item search form.

Parameters

- **$props** (*array*) – Custom HTML attributes for the form element
- **$formActionUri** (*string*) – URL the form should submit to. If omitted, the form submits to the default items/browse page.
- **$buttonText** (*string*) – Custom text for the form submit button. If omitted, the default text 'Search for items' is used.

Returns string

Usage

Examples

items_search_form() can be used to include a search form on a page. Use the $props array to set the id or class information for the form element.

```
<?php echo items_search_form(array('id' => "items-form")); ?>
```

Use the $buttonText parameter to modify the submit button text.

```
<?php echo items_search_form(array(), current_url(), "I'm Feeling Lucky"); ?>
```

See Also

`js_escape` — Escape a value for use in javascript.

Text-related functions

Summary

`js_escape`(*$value*)

Escape a value for use in javascript.

This is a convenience function for encoding a value using JSON notation. Must be used when interpolating PHP output in javascript. Note on usage: do not wrap the resulting output of this function in quotes, as proper JSON encoding will take care of that.

Parameters

- **`$value`** (*string*) –

Returns string

Usage

Examples

See Also

`js_tag` — Get a tag for including a local JavaScript file.

Asset-related functions

Summary

The `js_tag()` helper function retrieves JavaScript files located within the `/javascripts/` folder of a theme. By using `js_tag()`, the relative path to the file is dynamically generated for each page.

Note: The *queue_js_file* helper is preferred to this one for most use cases. Normally, you should only continue to use this helper when you want to output an inline script in the body of the page.

js_tag (*$file*, *$dir = 'javascripts'*, *$version = OMEKA_VERSION*)
Get a tag for including a local JavaScript file.

Parameters

- **$file** (*string*) – The name of the file, without .js extension.
- **$dir** (*string*) – The directory in which to look for javascript files. Recommended to leave the default value.
- **$version** (*mixed*) – Version number. By default OMEKA_VERSION.

Returns string

Usage

Examples

Add a link to a javascript file located in the theme's `javascripts` directory. Use when you need to load js files outside of the head.php file.

```
<?php echo js_tag('items'); ?>
```

See Also

label_table_options — Add a "Select Below" or other label option to a set of select options.

Form-related functions

Summary

label_table_options (*$options*, *$labelOption = null*)
Add a "Select Below" or other label option to a set of select options.

Parameters

- **$options** (*array*) –
- **$labelOption** (*string|null*) –

Returns array

Usage

Examples

See Also

`latest_omeka_version` — Get the latest available version of Omeka.

Utility-related functions

Summary

`latest_omeka_version`()
Get the latest available version of Omeka.

Returns string|false The latest available version of Omeka, or false if the request failed for some reason.

Usage

Examples

See Also

`link_to` — Get a link to a page within Omeka.

Navigation-related functions

Summary

`link_to`(*$record*, *$action = null*, *$text = null*, *$props = array()*, *$queryParams = array()*)
Get a link to a page within Omeka.

The controller and action can be manually specified, or if a record is passed this function will hand off to record_url to automatically get a link to that record (either its default action or one explicitly chosen).

Parameters

- **`$record`** (*`Omeka_Record_AbstractRecord|string`*) – The name of the controller to use for the link. If a record instance is passed, then it inflects the name of the controller from the record class.
- **`$action`** (*`string`*) – The action to use for the link
- **`$text`** (*`string`*) – The text to put in the link. Default is 'View'.
- **`$props`** (*`array`*) – Attributes for the <a> tag
- **`$queryParams`** (*`array`*) – the parameters in the uri query

Returns string HTML

Usage

Examples

See Also

`link_to_admin_home_page` — Get a link to the admin home page.

Navigation-related functions

Summary

Creates a link to the Omeka admin anywhere in your public theme.

`link_to_admin_home_page` (*$text = null*, *$props = array()*)

Get a link to the admin home page.

Parameters

- **`$text`** (*`null|string`*) –
- **`$props`** (*`array`*) –

Returns string

Usage

Examples

See Also

`link_to_collection` — Get a link to a collection.

Navigation-related functions

Summary

`link_to_collection` (*$text = null*, *$props = array()*, *$action = 'show'*, *$collectionObj = null*)

Get a link to a collection.

The only differences from link_to() are that this function will automatically use the "current" collection, and will use the collection title as the link text.

Parameters

- **`$text`** (*`string`*) – text to use for the title of the collection. Default behavior is to use the name of the collection.
- **`$props`** (*`array`*) – Set of attributes to use for the link.
- **`$action`** (*`array`*) – The action to link to for the collection.
- **`$collectionObj`** (*`array`*) – Collection record can be passed to this to override the collection object retrieved by get_current_record().

Returns string

Usage

Examples

See Also

`link_to_collection_for_item` — Get a link to the collection to which the item belongs.

Navigation-related functions

Summary

`link_to_collection_for_item`(*$text = null*, *$props = array()*, *$action = 'show'*)
Get a link to the collection to which the item belongs.

The default text displayed for this link will be the name of the collection, but that can be changed by passing a string argument.

Parameters

- **`$text`** (*`string|null`*) – Text for the link.
- **`$props`** (*`array`*) – HTML attributes for the <a> tag.
- **`$action`** (*`string`*) – 'show' by default.

Returns string

Usage

Examples

See Also

`link_to_file_show` — Get a link to the file metadata page for a particular file.

Navigation-related functions

Summary

`link_to_file_show`(*$attributes = array()*, *$text = null*, *$file = null*)
Get a link to the file metadata page for a particular file.

If no File object is specified, this will determine the file to use through context. The text of the link defaults to the DC:Title of the file record, then to the original filename, unless otherwise specified.

Parameters

- **`$attributes`** (*`array`*) –
- **`$text`** (*`string`*) –
- **`$file`** (*`File|null`*) –

Returns string

Usage

Examples

See Also

`link_to_home_page` — Get a link to the public home page.

Navigation-related functions

Summary

Returns a link to the public home page of an Omeka site.

`link_to_home_page` (*$text = null*, *$props = array()*)

Get a link to the public home page.

Parameters

- **`$text`** (*`null|string`*) –
- **`$props`** (*`array`*) –

Returns string

Usage

Examples

See Also

`link_to_item` — Get a link to an item.

Navigation-related functions

Summary

`link_to_item` (*$text = null*, *$props = array()*, *$action = 'show'*, *$item = null*)

Get a link to an item.

The only differences from link_to are that this function will automatically use the "current" item, and will use the item's title as the link text.

Parameters

- **`$text`** (*`string`*) – HTML for the text of the link.
- **`$props`** (*`array`*) – Properties for the <a> tag.
- **`$action`** (*`string`*) – The page to link to (this will be the 'show' page almost always within the public theme).
- **`$item`** (`Item`) – Used for dependency injection testing or to use this function outside the context of a loop.

Returns string HTML

Usage

Examples

See Also

`link_to_item_search` — Get HTML for a link to the item search form.

Navigation-related functions

Summary

`link_to_item_search` (*$text = null*, *$props = array()*, *$uri = null*)
Get HTML for a link to the item search form.

Parameters

- **`$text`** (*`string`*) – Text of the link. Default is 'Search Items'.
- **`$props`** (*`array`*) – HTML attributes for the link.
- **`$uri`** (*`string`*) – Action for the form. Defaults to 'items/browse'.

Returns string

Usage

Examples

See Also

`link_to_items_browse` — Get HTML for a link to the browse page for items.

Navigation-related functions

Summary

`link_to_items_browse` (*$text*, *$browseParams = array()*, *$linkProperties = array()*)
Get HTML for a link to the browse page for items.

Parameters

- **`$text`** (*`string`*) – Text to display in the link.
- **`$browseParams`** (*`array`*) – Any parameters to use to build the browse page URL, e.g. array('collection' => 1) would build items/browse?collection=1 as the URL.
- **`$linkProperties`** (*`array`*) – HTML attributes for the link.

Returns string HTML

Usage

Examples

See Also

`link_to_items_in_collection` — Get a link to the collection items browse page.

Navigation-related functions

Summary

`link_to_items_in_collection`(*$text = null*, *$props = array()*, *$action = 'browse'*, *$collectionObj = null*)

Get a link to the collection items browse page.

Parameters

- **`$text`** (*`string`*|*`null`*) –
- **`$props`** (*`array`*) –
- **`$action`** (*`string`*) –
- **`$collectionObj`** (`Collection`) –

Returns string

Usage

Examples

See Also

`link_to_items_rss` — Get a link the the items RSS feed.

Navigation-related functions

Summary

`link_to_items_rss`(*$text = null*, *$params = array()*)

Get a link the the items RSS feed.

Parameters

- **`$text`** (*`string`*) – The text of the link.
- **`$params`** (*`array`*) – A set of query string parameters to merge in to the href of the link. E.g., if this link was clicked on the items/browse?collection=1 page, and array('foo'=>'bar') was passed as this argument, the new URI would be items/browse?collection=1&foo=bar.

Usage

Examples

See Also

`link_to_items_with_item_type` — Get a link to item type items browse page.

Navigation-related functions

Summary

`link_to_items_with_item_type` (*$text = null*, *$props = array()*, *$action = 'browse'*, *$itemTypeObj = null*)

Get a link to item type items browse page.

Parameters

- **`$text`** (*`string|null`*) –
- **`$props`** (*`array`*) –
- **`$action`** (*`string`*) –
- **`$itemTypeObj`** –

Returns string

Usage

Examples

See Also

`link_to_next_item_show` — Get a link to the item immediately following the current one.

Navigation-related functions

Summary

`link_to_next_item_show` (*$text = null*, *$props = array()*)

Get a link to the item immediately following the current one.

Parameters

- **`$text`** (*`string`*) –
- **`$props`** (*`array`*) –

Returns string

Usage

Examples

See Also

`link_to_previous_item_show` — Get a link to the item immediately before the current one.

Navigation-related functions

Summary

`link_to_previous_item_show` (*$text = null*, *$props = array()*)
Get a link to the item immediately before the current one.

Parameters

- **`$text`** (*`string`*) –
- **`$props`** (*`array`*) –

Returns string

Usage

Examples

See Also

`loop` — Get an iterator for looping over an array of records.

Loop-related functions

Summary

`loop()` is designed to make looping through records for a browse page easier.

You can either pass in an array of record to loop through or, more commonly, the name of the model (e.g. 'items' or 'collections'), whose records have already been set on the page.

`loop` (*$recordsVar*, *$records = null*)
Get an iterator for looping over an array of records.

Parameters

- **`$recordsVar`** (*`string`*) – The name of the variable the records are stored in.
- **`$records`** (*`array|null`*) –

Returns Omeka_Record_Iterator

Usage

Usually, the `$recordsVar` parameter is a string name of the model whose records have already been set in the view by the controller.

This will be the name of the table, i.e., the pluralized, underscored name of the model. For example, a CamelCase model name like "CsvImport_Import" should be "csv_import_imports".

The string will be converted to the table name. Thus, `csv_import_imports` and `CsvImport_Import` have exactly the same effect (provided the same convention was used in setting the records to loop in the view).

Examples

See Also

- *Omeka_View_Helper_Pluralize*
- *Omeka_View_Helper_Loop*

`max_file_size` — Get the maximum file upload size.

Utility-related functions

Summary

`max_file_size``()`
Get the maximum file upload size.

Returns string

Usage

Examples

See Also

`metadata` — Get metadata for a record.

View-related functions

Summary

The `metadata` function is the easiest and safest way to include record metadata in HTML output. `metadata` offers support for Element Text metadata, data stored directly in database columns, and custom metadata properties.

`metadata` automatically returns output HTML-escaped for inclusion in a page where appropriate. The escaping can be optionally disabled. Values also automatically passed through the *Element Display Filter*, though this too can be optionall disabled.

`metadata`(*$record*, *$metadata*, *$options = array()*)
Get metadata for a record.

Parameters

- **$record** (*Omeka_Record_AbstractRecord|string*) – The record to get metadata for. If an Omeka_Record_AbstractRecord, that record is used. If a string, that string is used to look up a record in the current view.
- **$metadata** (*mixed*) – The metadata to get. If an array is given, this is Element metadata, identified by array('Element Set', 'Element'). If a string, the metadata is a record-specific "property."
- **$options** (*array*) – Options for getting the metadata.

Returns mixed

Usage

Record

The first parameter can be either a record object or the string name of a record type. The string form is used when accessing "current" records set through *set_current_record* or *loop*.

Metadata

For records that carry Element Set metadata, get the value for an element by passing the element set and element name as an array in the second parameter, e.g. `array('Dublin Core', 'Title')`.

You can also get properties of the record by passing the name of the property as a string in the second parameter. Valid property names differ between record types. Every public property of a record (sometimes called the "columns" of the record because this is the data stored in the columns of the database) is a valid metadata name here. Many records have special additional properties that are available as well:

- Collection
  - `total_items`
- File
  - `uri`
  - `fullsize_uri`
  - `thumbnail_uri`
  - `square_thumbnail_uri`
  - `permalink`
  - `display_title`
- Item
  - `item_type_name`
  - `collection_name`
  - `permalink`
  - `has_files`
  - `file_count`
  - `has_thumbnail`

- citation

Since version 2.5, all the Element Set-carrying records (Collection, File and Item) also have the special property `display_title` that returns a simplified version of their title.

The above list only covers core records. For other records, like those added by plugins, see the plugin's documentation or the `getProperty()` method of the record.

Options

Valid keys for the `$options` array are:

- `all`: If true, return an array containing all values for the field.
- `delimiter`: Return the entire set of metadata as a string, where entries are separated by the given delimiter.
- `index`: Return the metadata entry at the given zero-based index.
- `no_escape`: If true, do not escape the resulting values for HTML entities.
- `no_filter`: If true, return the set of metadata without running any *filters*.
- `snippet`: Trim the length of each piece of text to the given length in characters.
- `ignore_unknown`: If true, do not throw an error if the requested metadata element does not exist. (Since version 2.5)
- Passing simply the string 'all' is equivalent to `array('all' => true)`
- Passing simply an integer is equivalent to `array('index' => [the integer])`

Examples

Use the `metadata` function to get the Element Set metadata for a collection. The `snippet` option can be used to limit the information to only the first 150 characters.

```
<div class="collection record">
    <?php
        $description = metadata($collection, array('Dublin Core', 'Description'),␣
↪array('snippet' => 150));
    ?>
    <?php if ($description): ?>
        <p class="collection-description"><?php echo $description; ?></p>
    <?php endif; ?>
</div>
```

When calling the `metadata` function within a loop, pass in a string of the current record type.

```
<h1><?php echo metadata('collection', array('Dublin Core', 'Title')); ?></h1>
```

The `metadata` function also grabs the record metadata.

```
<h1><?php echo metadata('simple_pages_page', 'title'); ?></h1>
```

See Also

`nav` — Create a navigation menu of links.

Navigation-related functions

Summary

nav (*$navLinks*, *$name = null*, *$args = array()*)
Create a navigation menu of links.

Parameters

- **$navLinks** (*array*) – The array of links for the navigation.
- **$name** (*string*) – Optionally, the name of a filter to pass the links through before using them.
- **$args** (*array*) – Optionally, arguments to pass to the filter

Returns Zend_View_Helper_Navigation_Menu The navigation menu object. Can generally be treated simply as a string.

Usage

Examples

```
$navArray = array();
$navArray[] = array('label'=>'Browse All', 'uri'=>url('items'));
$navArray[] = array('label'=>'Browse By Tag', 'uri'=>url('items/tags');

echo nav($navArray);
```

See Also

`option` — Get the value of a particular site setting for display.

View-related functions

Summary

option (*$name*)
Get the value of a particular site setting for display.

Content for any specific option can be filtered by using a filter named 'display_option_(option)' where (option) is the name of the option, e.g. 'display_option_site_title'.

Parameters

- **$name** (*string*) – The name of the option

Returns string

Usage

Examples

See Also

`output_format_list` — Get an HTML list of output formats for the current action.

OutputFormat-related functions

Summary

`output_format_list` (*$list = true*, *$delimiter = ', '*)
Get an HTML list of output formats for the current action.

Parameters

- **`$list`** (*bool*) – If true or omitted, return an unordered list, if false, return a simple string list using the delimiter.
- **`$delimiter`** (*string*) – If the first argument is false, use this as the delimiter for the list.

Returns string|bool HTML

Usage

Examples

Use `output_format_list` to control how the list of outputs is generated. Use true to output as an unordered list (`<ol>`).

```
<?php echo output_format_list(true); ?>
```

Use `false` to output as a string. While the default delimiter is a comma, this can be overwritten using the `$delimiter` parameter.

```
<?php echo output_format_list(false, ' | '); ?>
```

See Also

`pagination_links` — Get HTML for a pagination control for a browse page.

Navigation-related functions

Summary

`pagination_links` (*$options = array()*)
Get HTML for a pagination control for a browse page.

Parameters

- **$options** (*array*) – Configurable parameters for the pagination links. The following options are available: - 'scrolling_style' (string) See Zend_View_Helper_PaginationControl for more details. Default 'Sliding'. - 'partial_file' (string) View script to use to render the pagination HTML. Default is 'common/pagination_control.php'. - 'page_range' (integer) See Zend_Paginator::setPageRange() for details. Default is 5. - 'total_results' (integer) Total results to paginate through. Default is provided by the 'total_results' key of the 'pagination' array that is typically registered by the controller. - 'page' (integer) Current page of the result set. Default is the 'page' key of the 'pagination' array. - 'per_page' (integer) Number of results to display per page. Default is the 'per_page' key of the 'pagination' array.

Returns string HTML for the pagination links.

Usage

Examples

See Also

physical_path_to — Get the filesystem path for a local asset.

Asset-related functions

Summary

physical_path_to(*$file*)

Get the filesystem path for a local asset.

Parameters

- **$file** (*string*) – The filename.

Returns string

Usage

Examples

See Also

pluck — Return one column of a multidimensional array as an array.

Utility-related functions

Summary

pluck(*$col*, *$array*)

Return one column of a multidimensional array as an array.

Parameters

- **$col** (*string|int*) – The column to pluck.

- **$array** (*array*) – The array from which to pluck.

Returns array The column as an array.

Usage

Examples

See Also

plugin_is_active — Determine whether a plugin is installed and active.

Plugin-related functions

Summary

plugin_is_active (*$name*, *$version = null*, *$compOperator = '>='*)

Determine whether a plugin is installed and active.

May be used by theme/plugin writers to customize behavior based on the existence of certain plugins. Some examples of how to use this function:

Check if ExhibitBuilder is installed and activated. <code> if (plugin_is_active('ExhibitBuilder')): </code>

Check if installed version of ExhibitBuilder is at least version 1.0 or higher. <code> if (plugin_is_active('ExhibitBuilder', '1.0')): </code>

Check if installed version of ExhibitBuilder is anything less than 2.0. <code> if (plugin_is_active('ExhibitBuilder', '2.0', '<')): </code>

Parameters

- **$name** (*string*) – Directory name of the plugin.
- **$version** (*string*) – Version of the plugin to check.
- **$compOperator** (*string*) – Comparison operator to use when checking the installed version of ExhibitBuilder.

Returns bool

Usage

Examples

See Also

plural — Transform arguments in an array suitable for __.

Locale-related functions

Summary

plural (*$msgid*, *$msgid_plural*, *$n*)
Transform arguments in an array suitable for __.

<code> $n = count($items); echo __(plural('one item', '%s items', $n), $n); </code>

Parameters

- **$msgid** (*string*) – The string to be translated, singular form
- **$msgid_plural** (*string*) – The string to be translated, plural form
- **$n** (*int*) – Used to determine the plural form

Returns array Array to pass to __

Usage

Examples

See Also

public_nav_items — Get the navigation for items.

Navigation-related functions

Summary

public_nav_items (*$navArray = null*, *$maxDepth = 0*)
Get the navigation for items.

Parameters

- **$navArray** (*array*) –
- **$maxDepth** (*int|null*) –

Returns string

Usage

Used to display secondary navigation elements within the browse and search pages.

The default value of the `$navArray` is:

```
$navArray = array(
        array(
            'label' =>__('Browse All'),
            'uri' => url('items/browse'),
        ));
        if (total_records('Tag')) {
            $navArray[] = array(
                'label' => __('Browse by Tag'),
                'uri' => url('items/tags')
            );
```

(continues on next page)

(continued from previous page)

```
        }
        $navArray[] = array(
            'label' => __('Search Items'),
            'uri' => url('items/search')
        );
```

This displays the 'Browse All', 'Browse by Tag', and 'Search Items' navigation elements. Plugins such as Geolocation also add navigation elements to this array.

Examples

You can use the public_nav_items function to create a custom browse navigation.

```
<?php echo public_nav_items(array (
    array (
        'label' => __('Browse All'),
        'uri' => url('items/browse')
        ),
    array (
        'label' => __('Search'),
        'uri' => url('search')
        ),
    array (
        'label' => __('Browse Text Items'),
        'uri' => url('items/browse?type=1')
        )
)); ?>
```

You can find the item type numbers when viewing the 'item type' in the Admin interface. Changing the default array does not affect the navigation elements added by plugins.

See Also

`public_nav_main` — Get the main navigation for the public site.

Navigation-related functions

Summary

`public_nav_main``()`
Get the main navigation for the public site.

Returns Zend_View_Helper_Navigation_Menu Can be echoed like a string or manipulated by the theme.

Usage

This function returns a Zend Framework view helper called `Zend_View_Helper_Navigation_Menu`. Simply echoing the result of the function like a string will print out the navigation menu, but you can also call methods on it to change the output. See Zend Framework's documentation for details on what methods you can call on the menu helper.

Examples

Printing the normal nav menu with no changes:

```
echo public_nav_main();
```

Changing the class used for the topmost <ul> for the nav menu:

```
echo public_nav_main()->setUlClass('custom-class');
```

See Also

public_url — Get a URL to the public theme.

Navigation-related functions

Summary

public_url()
Get a URL to the public theme.

Returns string

Usage

This function takes the same arguments as *url*.

Examples

See Also

queue_css_file — Add a CSS file or files to the current page.

Asset-related functions

Summary

queue_css_file(*$file*, *$media = 'all'*, *$conditional = false*, *$dir = 'css'*, *$version = OMEKA_VERSION*)
Add a CSS file or files to the current page.

All stylesheets will be included in the page's head. This needs to be called either before head(), or in a plugin_header hook.

Parameters

- **$file** (*string|array*) – File to use, if an array is passed, each array member will be treated like a file.
- **$media** (*string*) – CSS media declaration, defaults to 'all'.

- **$conditional** (*string|bool*) – IE-style conditional comment, used generally to include IE-specific styles. Defaults to false.
- **$dir** (*string*) – Directory to search for the file. Keeping the default is recommended.
- **$version** (*mixed*) – Version number. By default OMEKA_VERSION.

Usage

head_css actually outputs the stylesheet markup. All themes should already call `head_css` themselves, so you can just use the queue functions.

If writing a theme, `queue_css_*` functions must be called before the call to `head_css`. If writing a plugin, you make these calls within the *admin_head* or *public_head* hooks.

Examples

Include the CSS file `style.css` located in the `css` folder in your theme:

```
<?php queue_css_file('style'); ?>
```

Load multiple CSS files using an array:

```
<?php queue_css_file(array('style', 'screen')); ?>
```

Load a CSS file and add a media query:

```
<?php queue_css_file('media/960min', 'only screen and (min-width: 960px)'); ?>
```

See Also

- *queue_css_url — Add a CSS file to the current page by URL.*
- *queue_css_string — Add a CSS string to the current page.*
- *head_css — Get the CSS link tags that will be used on the page.*

queue_css_string — Add a CSS string to the current page.

Asset-related functions

Summary

queue_css_string (*$string*, *$media = 'all'*, *$conditional = false*)

Add a CSS string to the current page.

The inline stylesheet will appear in the head element. This needs to be called either before head() or in a plugin_header hook.

Parameters

- **$string** (*string*) – CSS string to include.
- **$media** (*string*) – CSS media declaration, defaults to 'all'.

- **$conditional** (*string|bool*) – IE-style conditional comment, used generally to include IE-specific styles. Defaults to false.

Usage

head_css actually outputs the stylesheet markup. All themes should already call `head_css` themselves, so you can just use the queue functions.

If writing a theme, `queue_css_*` functions must be called before the call to `head_css`. If writing a plugin, you make these calls within the *admin_head* or *public_head* hooks.

Examples

Load css styles directly into the page head within `<style></style>` tags:

```
<?php queue_css_string("input.add-element {display: block}"); ?>
```

See Also

- *queue_css_file — Add a CSS file or files to the current page.*
- *queue_css_url — Add a CSS file to the current page by URL.*
- *head_css — Get the CSS link tags that will be used on the page.*

queue_css_url — Add a CSS file to the current page by URL.

Asset-related functions

Summary

queue_css_url (*$url*, *$media = 'all'*, *$conditional = false*)

Add a CSS file to the current page by URL.

The stylesheet link will appear in the head element. This needs to be called either before head() or in a plugin_header hook.

Parameters

- **$url** –
- **$media** (*string*) – CSS media declaration, defaults to 'all'.
- **$conditional** (*string|bool*) – IE-style conditional comment, used generally to include IE-specific styles. Defaults to false.

Usage

head_css actually outputs the stylesheet markup. All themes should already call `head_css` themselves, so you can just use the queue functions.

If writing a theme, `queue_css_*` functions must be called before the call to `head_css`. If writing a plugin, you make these calls within the *admin_head* or *public_head* hooks.

Examples

Use to load in external stylesheets, such as Google Fonts.

```
<?php queue_css_url('http://fonts.googleapis.com/css?family=Raleway:400,600'); ?>
```

See Also

- *queue_css_file — Add a CSS file or files to the current page.*
- *queue_css_string — Add a CSS string to the current page.*
- *head_css — Get the CSS link tags that will be used on the page.*

`queue_js_file` — Add a local JavaScript file or files to the current page.

Asset-related functions

Summary

`queue_js_file`(*$file*, *$dir = 'javascripts'*, *$options = array()*, *$version = OMEKA_VERSION*)

Add a local JavaScript file or files to the current page.

All scripts will be included in the page's head. This needs to be called either before head(), or in a plugin_header hook.

Parameters

- **`$file`** (*`string|array`*) – File to use, if an array is passed, each array member will be treated like a file.
- **`$dir`** (*`string`*) – Directory to search for the file. Keeping the default is recommended.
- **`$options`** (*`array`*) – An array of options.
- **`$version`** (*`mixed`*) – Version number. By default OMEKA_VERSION.

Usage

`head_js` actually outputs the scripts. All themes should already call `head_js` themselves, so you can just use the queue functions.

If writing a theme, `queue_js_*` functions must be called before the call to `head_js`. If writing a plugin, you make these calls within the *admin_head* or *public_head* hooks.

The `$options` array is passed as the `$attrs` parameter to the Zend Framework headScript helper. This can be used to set some attributes on the `<script>` tag as well as to enable some specific functionality of the ZF helper like IE conditional comments.

Examples

Load one javascript file from the default 'javascripts' folder within the theme directory:

```
<?php queue_js_file('my_js_file'); ?>
```

Load two javascript files from the default 'javascripts' folder:

```
<?php queue_js_file(array('my_js_file', 'another_js_file')); ?>
```

Load one javascript file from the default 'javascripts' folder with a conditional statement:

```
<?php queue_js_file('my_js_file', 'javascripts', array('conditional' => '(gte IE 6)&
↪(lte IE 8)')); ?>
```

See Also

- *queue_js_url — Add a JavaScript file to the current page by URL.*
- *queue_js_string — Add a JavaScript string to the current page.*
- *head_js — Get the JavaScript tags that will be used on the page.*

`queue_js_string` — Add a JavaScript string to the current page.

Asset-related functions

Summary

queue_js_string(*$string*, *$options* = *array()*)
Add a JavaScript string to the current page.

The script will appear in the head element. This needs to be called either before head() or in a plugin_header hook.

Parameters

- **$string** (*string*) – JavaScript string to include.
- **$options** (*array*) – An array of options.

Usage

head_js actually outputs the scripts. All themes should already call `head_js` themselves, so you can just use the queue functions.

If writing a theme, `queue_js_*` functions must be called before the call to `head_js`. If writing a plugin, you make these calls within the *admin_head* or *public_head* hooks.

The `$options` array is passed as the `$attrs` parameter to the Zend Framework headScript helper. This can be used to set some attributes on the `<script>` tag as well as to enable some specific functionality of the ZF helper like IE conditional comments.

Examples

Use to add arbitrary javascript code to a page.

```
<?php queue_js_string("
      window.addEventListener('load', function() {
          FastClick.attach(document.body);
      }, false);
"); ?>
```

See Also

- *queue_js_file — Add a local JavaScript file or files to the current page.*
- *queue_js_url — Add a JavaScript file to the current page by URL.*
- *head_js — Get the JavaScript tags that will be used on the page.*

`queue_js_url` — Add a JavaScript file to the current page by URL.

Asset-related functions

Summary

`queue_js_url` (*$url*, *$options = array()*)

Add a JavaScript file to the current page by URL.

The script link will appear in the head element. This needs to be called either before head() or in a plugin_header hook.

Parameters

- **`$url`** –
- **`$options`** (*array*) – An array of options.

Usage

head_js actually outputs the scripts. All themes should already call `head_js` themselves, so you can just use the queue functions.

If writing a theme, `queue_js_*` functions must be called before the call to `head_js`. If writing a plugin, you make these calls within the *admin_head* or *public_head* hooks.

The `$options` array is passed as the `$attrs` parameter to the Zend Framework headScript helper. This can be used to set some attributes on the `<script>` tag as well as to enable some specific functionality of the ZF helper like IE conditional comments.

Examples

Use to load external javascript files, such as the jquery library.

```
<?php queue_js_url('//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js'); ?>
```

See Also

- *queue_js_file — Add a local JavaScript file or files to the current page.*
- *queue_js_string — Add a JavaScript string to the current page.*
- *head_js — Get the JavaScript tags that will be used on the page.*

`random_featured_collection` — Get HTML for displaying a random featured collection.

View-related functions

Summary

`random_featured_collection`()
Get HTML for displaying a random featured collection.

Returns string

Usage

Examples

Use to display a random featured collection. If no featured collection, will return *No featured collections are available.*

```
<?php echo random_featured_collection(); ?>
```

See Also

`random_featured_items` — Get HTML for random featured items.

Item-related functions

Summary

`random_featured_items`(*$count = 5*, *$hasImage = null*)
Get HTML for random featured items.

Parameters

- **`$count`** (*int*) – Maximum number of items to show.
- **`$hasImage`** (*bool*) – Whether or not the featured items must have images associated. If null, as default, all featured items can appear, whether or not they have files. If true, only items with files will appear, and if false, only items without files will appear.

Returns string

Usage

Examples

See Also

`recent_items` — Get HTML for recent items.

New in version 2.2.

Item-related functions

Summary

`recent_items`(*$count = 10*)
Get HTML for recent items.

Parameters

- **`$count`** (*`int`*) – Maximum number of recent items to show.

Returns string

Usage

Examples

See Also

`record_image` — Get an image tag for a record.

View-related functions

Summary

`record_image`(*$record*, *$imageType = null*, *$props = array()*)
Get an image tag for a record.

Parameters

- **`$record`** (*`Omeka_Record_AbstractRecord|string`*) –
- **`$imageType`** (*`string`*) – Image size: thumbnail, square thumbnail, fullsize
- **`$props`** (*`array`*) – HTML attributes for the img tag

Returns string

Usage

Examples

Get the image associated with an individual record, such as a collection or an exhibit. Use the `$imageType` string to set the image size.

```
<?php if ($collectionImage = record_image('collection', 'square_thumbnail')): ?>
    <?php echo link_to_collection($collectionImage, array('class' => 'image')); ?>
<?php endif; ?>
```

See Also

`record_url` — Get a URL to a record.

Navigation-related functions

Summary

`record_url` (*$record*, *$action = null*, *$getAbsoluteUrl = false*, *$queryParams = array()*)
Get a URL to a record.

Parameters

- **`$record`** (*`Omeka_Record_AbstractRecord|string`*) –
- **`$action`** (*`string|null`*) –
- **`$getAbsoluteUrl`** (*`bool`*) –
- **`$queryParams`** (*`array`*) –

Returns string

Usage

Examples

See Also

`release_object` — Release an object from memory.

Utility-related functions

Summary

`release_object` (*$var*)
Release an object from memory.

Use this fuction after you are done using an Omeka model object to prevent memory leaks. Required because PHP 5.2 does not do garbage collection on circular references.

Parameters

- **$var** –

Usage

Examples

See Also

`revert_theme_base_url` — Revert the base URL to its previous state.

Navigation-related functions

Summary

revert_theme_base_url()
Revert the base URL to its previous state.

Usage

Examples

See Also

`search_filters` — Get a list of current site-wide search filters in use.

Search-related functions

Summary

search_filters(*$options = array()*)
Get a list of current site-wide search filters in use.

Parameters

- **$options** (*array*) – Valid options are as follows: - id (string): the value of the div wrapping the filters.

Returns string

Usage

Examples

See Also

`search_form` — Get the site-wide search form.

Search-related functions

Summary

search_form(*$options = array()*)
Get the site-wide search form.

Parameters

- **$options** (*array*) – Valid options are as follows: - show_advanced (bool): whether to show the advanced search; default is false. - submit_value (string): the value of the submit button; default "Submit". - form_attributes (array): an array containing form tag attributes.

Returns string The search form markup.

Usage

Examples

Submit all of the options as an array.

Use the `show_advanced` boolean to show the advanced search:

```
<?php echo search_form(array('show_advanced' => true)); ?>
```

Use the `submit_value` string to change the text value of the submit button:

```
<?php echo search_form(array('show_advanced' => true, 'submit_value' => 'I Feel Lucky
↪')); ?>
```

Use the `form_attributes` array to add html attributes and ARIA roles to the `form` element:

```
<?php echo search_form(array('form_attributes' => array('role' => 'search', 'class' =>
↪ 'form'))); ?>
```

See Also

- *search_form*
- *search_form_default_action*
- *search_form_default_query*
- *search_form_default_query_type*
- *search_form_default_record_types*

set_current_record — Set a record to the view as the current record.

Loop-related functions

Summary

set_current_record(*$recordVar*, *Omeka_Record_AbstractRecord $record*, *$setPreviousRecord = false*)
Set a record to the view as the current record.

Parameters

- **$recordVar** (*string*) – View variable to store the current record in.
- **$record** (Omeka_Record_AbstractRecord) –
- **$setPreviousRecord** (*bool*) – Whether to store the previous "current" record, if any. The default is to not store the previous record.

Usage

Examples

See Also

set_loop_records — Set records to the view for iteration with loop().

Loop-related functions

Summary

set_loop_records (*$recordsVar*, *$records*)
Set records to the view for iteration with loop().

Parameters

- **$recordsVar** (*string*) – The name of the variable to store the records in.
- **$records** (*array*) – The records to store for later looping.

Usage

Examples

See Also

set_option — Set an option to the options table.

Option-related functions

Summary

set_option (*$name*, *$value*)
Set an option to the options table.

Note that objects and arrays must be serialized before being saved.

Parameters

- **$name** (*string*) – The option name.
- **$value** (*string*) – The option value.

Usage

Examples

See Also

`set_theme_base_url` — Set the base URL for the specified theme.

Navigation-related functions

Summary

`set_theme_base_url` (*$theme = null*)
Set the base URL for the specified theme.

Parameters

- **`$theme`** (*`string`*) –

Usage

Examples

See Also

`set_theme_option` — Set a theme option.

Option-related functions

Summary

`set_theme_option` (*$optionName*, *$optionValue*, *$themeName = null*)
Set a theme option.

Parameters

- **`$optionName`** (*`string`*) – The option name.
- **`$optionValue`** (*`string`*) – The option value.
- **`$themeName`** (*`string`*) – The theme name. If null, it will use the current public theme.

Usage

Examples

See Also

`snippet` — Return a substring of a given piece of text.

Text-related functions

Summary

snippet (*$text*, *$startPos*, *$endPos*, *$append = '...'*)
Return a substring of a given piece of text.

Note: this will only split strings on the space character. this will also strip html tags from the text before getting a snippet

Parameters

- **$text** (*string*) – Text to take snippet of
- **$startPos** (*int*) – Starting position of snippet in string
- **$endPos** (*int*) – Maximum length of snippet
- **$append** (*string*) – String to append to snippet if truncated

Returns string Snippet of given text

Usage

Examples

See Also

snippet_by_word_count — Return a substring of the text by limiting the word count.

Text-related functions

Summary

snippet_by_word_count (*$text*, *$maxWords = 20*, *$ellipsis = '...'*)
Return a substring of the text by limiting the word count.

Note: it strips the HTML tags from the text before getting the snippet

Parameters

- **$text** (*string*) –
- **$maxWords** (*int*) –
- **$ellipsis** (*string*) –

Returns string

Usage

Examples

See Also

src — Get a URL for a given local file.

Asset-related functions

Summary

src (*$file*, *$dir = null*, *$ext = null*, *$version = null*)
Get a URL for a given local file.

Parameters

- **$file** (*string*) – The filename.
- **$dir** (*string|null*) – The file's directory.
- **$ext** (*string*) – The file's extension.
- **$version** (*mixed*) – Optional version number.

Returns string

Usage

Examples

See Also

strip_formatting — Strip HTML tags from a string.

Text-related functions

Summary

strip_formatting (*$str*, *$allowableTags = ''*, *$fallbackStr = ''*)
Strip HTML tags from a string.

This is essentially a wrapper around PHP's strip_tags() function, with the added benefit of returning a fallback string in case the resulting stripped string is empty or contains only whitespace.

Parameters

- **$str** (*string*) – The string to be stripped of HTML formatting.
- **$allowableTags** (*string*) – The string of tags to allow when stripping tags.
- **$fallbackStr** (*string*) – The string to be used as a fallback.

Returns The stripped string.

Usage

Examples

See Also

tag_attributes — Generate attributes for an HTML tag.

View-related functions

Summary

tag_attributes(*$attributes*)
Generate attributes for an HTML tag.

Parameters

- **$attributes** (*array|string*) – Attributes for the tag. If this is a string, it will assign both 'name' and 'id' attributes that value for the tag.

Returns string

Usage

Examples

Adds attributes to HTML elements. Used mostly as a helper to other functions.

```
<form <?php echo tag_attributes($formAttributes); ?>>
```

See Also

tag_cloud — Create a tag cloud made of divs that follow the hTagcloud microformat

Tag-related functions

Summary

tag_cloud(*$recordOrTags = null*, *$link = null*, *$maxClasses = 9*, *$tagNumber = false*, *$tagNumberOrder = null*)
Create a tag cloud made of divs that follow the hTagcloud microformat

Parameters

- **$recordOrTags** (*Omeka_Record_AbstractRecord|array*) – The record to retrieve tags from, or the actual array of tags
- **$link** (*string|null*) – The URI to use in the link for each tag. If none given, tags in the cloud will not be given links.
- **$maxClasses** (*int*) –
- **$tagNumber** (*bool*) –
- **$tagNumberOrder** (*string*) –

Returns string HTML for the tag cloud

Usage

Examples

Add a tag cloud to a page. Use the `$link` parameter to set the base url for the tags.

```
<?php echo tag_cloud($tags, 'exhibits/browse'); ?>
```

Use the `$maxClasses` parameter to set the max count used to calculate tag size. Range supported by themes is 1-9.

```
<?php echo tag_cloud($tags, 'exhibits/browse', 5); ?>
```

Use the `$tagNumber` and `$tagNumberOrder` parameters to display the count for each tag. The possible values for `$tagNumberOrder` are 'before' and 'after'.

```
<?php echo tag_cloud($tags, 'exhibits/browse', 5, true, 'before'); ?>
```

See Also

`tag_string` — Return a tag string given an Item, Exhibit, or a set of tags.

Tag-related functions

Summary

`tag_string` (*$recordOrTags = null*, *$link = 'items/browse'*, *$delimiter = null*)
Return a tag string given an Item, Exhibit, or a set of tags.

Parameters

- **`$recordOrTags`** (*`Omeka_Record_AbstractRecord|array`*) – The record to retrieve tags from, or the actual array of tags
- **`$link`** (*`string|null`*) – The URL to use for links to the tags (if null, tags aren't linked)
- **`$delimiter`** (*`string`*) – ', ' (comma and whitespace) is the default tag_delimiter option. Configurable in Settings

Returns string HTML

Usage

Examples

Use to gather all of the tags associated with an item or exhibit. The $link parameter defaults to 'items/browse' but can be overwritten for different contexts, such as listing tags associated with exhibits.

```
<?php echo tag_string('exhibit', 'exhibits'); ?>
```

See Also

`text_to_id` — Convert a word or phrase to a valid HTML ID.

Text-related functions

Summary

text_to_id(*$text*, *$prepend = null*, *$delimiter = '-'*)

Convert a word or phrase to a valid HTML ID.

For example: 'Foo Bar' becomes 'foo-bar'.

This function converts to lowercase, replaces whitespace with hyphens, removes all non-alphanumerics, removes leading or trailing delimiters, and optionally prepends a piece of text.

Parameters

- **$text** (*string*) – The text to convert
- **$prepend** (*string*) – Another string to prepend to the ID
- **$delimiter** (*string*) – The delimiter to use (- by default)

Returns string

Usage

Examples

See Also

text_to_paragraphs — Replace newlines in a block of text with paragraph tags.

Text-related functions

Summary

text_to_paragraphs(*$str*)

Replace newlines in a block of text with paragraph tags.

Looks for 2 consecutive line breaks resembling a paragraph break and wraps each of the paragraphs with a <p> tag. If no paragraphs are found, then the original text will be wrapped with line breaks.

Parameters

- **$str** (*string*) –

Returns string

Usage

Examples

See Also

theme_header_background — Get the theme's header background image style.

Head-related functions

Summary

theme_header_background()
Get the theme's header background image style.

Returns string|null

Usage

Examples

This function is used to set the header image for a theme. It creates an internal css block and is used within the <head> of the HTML page.

```
<head>
    <?php echo theme_header_background(); ?>
</head>
```

Results in:

```
<head>
    <style>
        header {background:transparent url("http://mysite/files/theme_uploads/
↪0e4a046bfc9d5f4d988300168f237b22.jpg") center left no-repeat;}
    </style>
</head>
```

See Also

theme_header_image — Get the theme's header image tag.

Head-related functions

Summary

theme_header_image()
Get the theme's header image tag.

Returns string|null

Usage

Examples

Pull header image information from the theme options and write it to the HTML page:

```
<?php echo theme_header_image(); ?>
```

which results in:

```
<div id="header-image">
    <img src="http://mysite/files/theme_uploads/4c62db4de9d04b8f494ca8c132392920.jpg"␣
↪/>
</div>
```

See Also

`theme_logo` — Get the theme's logo image tag.

Head-related functions

Summary

`theme_logo`()
Get the theme's logo image tag.

Returns string|null

Usage

Examples

Use to display a logo image, which is set in the theme options. Combine with `link_to_home_page()` to wrap the image tag with a link to the home page:

```
<div id="site-title">
    <?php echo link_to_home_page(theme_logo()); ?>
</div>
```

See Also

`total_records` — Get the total number of a given type of record in the database.

Db-related functions

Summary

`total_records`(*$recordType*)
Get the total number of a given type of record in the database.

Parameters

- **`$recordType`** (*string*) – Type of record to count.

Returns int Number of records of $recordType in the database.

Usage

Examples

See Also

`update_collection` — Update an existing collection.

Collection-related functions

Summary

`update_collection`(*$collection*, *$metadata = array()*, *$elementTexts = array()*)
Update an existing collection.

Parameters

- **`$collection`** (*`Collection|int`*) – Either an Collection object or the ID for the collection.
- **`$metadata`** (*`array`*) – Set of options that can be passed to the collection.
- **`$elementTexts`** (*`array`*) – The element texts for the collection.

Returns Collection

Usage

Examples

See Also

`update_item` — Update an existing item.

Item-related functions

Summary

`update_item`(*$item*, *$metadata = array()*, *$elementTexts = array()*, *$fileMetadata = array()*)
Update an existing item.

Parameters

- **`$item`** (*`Item|int`*) – Either an Item object or the ID for the item.
- **`$metadata`** (*`array`*) – Set of options that can be passed to the item.
- **`$elementTexts`** (*`array`*) – Element texts to assign. See insert_item() for details.
- **`$fileMetadata`** (*`array`*) – File ingest data. See insert_item() for details.

Returns Item

Usage

Examples

See Also

`url` — Get a URL given the provided arguments.

Navigation-related functions

Summary

Returns a URL to a page in an Omeka site. This function can be used to create links between different pages on the site.

Plugin and theme writers should always use this helper when linking between pages. Hand-written or "bare" URLs are generally only valid for a particular Omeka installation. This helper generates relative URLs that are valid regardless of where an Omeka installation is located on a server.

`url` (*$options = array()*, *$route = null*, *$queryParams = array()*, *$reset = false*, *$encode = true*)
Get a URL given the provided arguments.

Instantiates view helpers directly because a view may not be registered.

Parameters

- **`$options`** (*`mixed`*) – If a string is passed it is treated as an Omeka-relative link. So, passing 'items' would create a link to the items page. If an array is passed (or no argument given), it is treated as options to be passed to Omeka's routing system. Note that in the Url Helper, if the first argument is a string, the second argument, not the third, is the queryParams
- **`$route`** (*`string`*) – The route to use if an array is passed in the first argument.
- **`$queryParams`** (*`mixed`*) – A set of query string parameters to append to the URL
- **`$reset`** (*`bool`*) – Whether Omeka should discard the current route when generating the URL.
- **`$encode`** (*`bool`*) – Whether the URL should be URL-encoded

Returns string HTML

Usage

Examples

See Also

`Omeka_View_Helper_Url::url`

`url_to_link` — Convert any URLs in a given string to links.

Text-related functions

Summary

url_to_link(*$str*)
Convert any URLs in a given string to links.

Parameters

- **$str** (*string*) – The string to be searched for URLs to convert to links.

Returns string

Usage

Examples

See Also

url_to_link_callback — Callback for converting URLs with url_to_link.

Text-related functions

Summary

url_to_link_callback(*$matches*)
Callback for converting URLs with url_to_link.

Parameters

- **$matches** (*array*) – preg_replace_callback matches array

Returns string

Usage

Examples

See Also

web_path_to — Get the URL for a local asset.

Asset-related functions

Summary

web_path_to(*$file*)
Get the URL for a local asset.

Parameters

- **$file** (*string*) – The filename.

Returns string

Usage

Examples

See Also

`xml_escape` — Escape a value for use in XML.

Text-related functions

Summary

`xml_escape`(*$value*)
Escape a value for use in XML.

Parameters

- **`$value`** (*string*) –

Returns string

Usage

Examples

See Also

3.1.2 By Typical Usage

Global Functions

Up to *Packages*

Db-related functions

Up to *Global Functions*

Collection-related functions

Up to *Db-related functions*

ElementSet-related functions

Up to *Db-related functions*

Item-related functions

Up to *Db-related functions*

ItemType-related functions

Up to *Db-related functions*

Option-related functions

Up to *Db-related functions*

Locale-related functions

Up to *Global Functions*

Log-related functions

Up to *Global Functions*

Plugin-related functions

Up to *Global Functions*

Search-related functions

Up to *Global Functions*

Text-related functions

Up to *Global Functions*

User-related functions

Up to *Global Functions*

Utility-related functions

Up to *Global Functions*

View-related functions

Up to *Global Functions*

Asset-related functions

Up to *View-related functions*

File-related functions

Up to *View-related functions*

Form-related functions

Up to *View-related functions*

Head-related functions

Up to *View-related functions*

Item-related functions

Up to *View-related functions*

ItemType-related functions

Up to *View-related functions*

Layout-related functions

Up to *View-related functions*

Loop-related functions

Up to *View-related functions*

Navigation-related functions

Up to *View-related functions*

OutputFormat-related functions

Up to *View-related functions*

Search-related functions

Up to *View-related functions*

Tag-related functions

Up to *View-related functions*

3.2 Hooks

Hooks are one of the main ways plugins can alter and extend Omeka. This page is a reference list of all the available hooks Omeka provides for plugins to use. See *Understanding Hooks* for general information about using hooks in plugins.

3.2.1 All Hooks

<model>_browse_sql

Usage

Alters the *Omeka_Db_Select* object when finding records of type <model> using the `Omeke_Db_Table::findBy` method.

`<model>` should be the plural name of the model, e.g. `items_browse_sql`

Arguments

Omeka_Db_Select **select** The select object

array params The parameters being used to build the select object. For web requests, GET parameters and Zend routing params will appear in this array. For internal queries, the params here will be the ones passed to `Omeke_Db_Table::findBy`.

Examples

activate

New in version 2.1.

Usage

The activate hook is fired whenever a plugin is activated. Most things plugins do will automatically be handled by normal deactivation, but a plugin could use this hook along with *deactivate* to manage changes it has made external to the plugin that should be enabled and disabled.

For most use cases, plugins should be using *install* or *initialize* instead of this hook.

Arguments

This hook has no arguments.

Examples

add_<model>_tag

Usage

Fires when tags are added to a record of type <model>

Arguments

Omeka_Record_AbstractRecord **record** The record tags are being added to.

array added Array of tags added

Examples

admin_<type>_form

Usage

Add form elements to a form created via *Omeka_Form_Admin*

Arguments

Omeka_Form_Admin **form** The form object

Omeka_Record_AbstractRecord **| null record** The record being edited, if it is included in the form

admin_<type>_panel_buttons

Usage

Append content just below the save panel buttons when editing using the *Omeka_Form_Admin*. Often <type> will be a model type.

Arguments

Omeka_View **view** The view object

Omeka_Record_AbstractRecord **|null record** The record being edited, if a record was passed in when creating the form. Otherwise, null.

Examples

admin_<type>_panel_fields

Usage

Append content just below the save panel fields when editing using the *Omeka_Form_Admin*. Often <type> will be a model type.

Arguments

Omeka_Record_AbstractRecord|**null record** The record being edited, if one was passed in when the form was created. Otherwise, null

Omeka_View **view** The view object

Examples

admin_appearance_settings_form

Usage

Arguments

Omeka_Form **form** The form object

Omeka_View **view** The view object

Examples

admin_collections_browse

Usage

Arguments

array collection Array of :php:class:‘Collection‘s

Omeka_View **view** The view object

Examples

admin_collections_form

Usage

Append to the form for the collection being edited. Content appears underneath the tabs

Arguments

Omeka_View **view** The view object

Collection **collection** The collection being edited

Examples

admin_collections_show

Usage

Append content to the admin collection show page.

Arguments

Collection **collection** The collection

Omeka_View **view** The view object

Examples

admin_collections_show_sidebar

New in version 2.1.

Usage

Append content to the sidebar panels on the collections show page.

Content should be wrapped in `<div class="panel">`

Note: To add content to the save panel, use *admin_collections_panel_buttons* or *admin_collections_panel_fields*.

Arguments

Collection **collection** The item object

Omeka_View **view** The view object

Examples

admin_dashboard

Usage

Appends content to the bottom of the dashboard.

Content should be wrapped in `<section>` tags

Arguments

Omeka_View **view** The view object

Examples

admin_files_form

Usage

Append content to the main area of a file edit page.

Arguments

File **file** The file object

Omeka_View **view** The view object

Examples

admin_files_show

Usage

Append content to the main area of a file show page.

Arguments

File **file** The file object

Omeka_View **view** The view object

Examples

admin_file_show_sidebar

Usage

Append content to the info panels area of a file show page.

Note: To add content to the save panel, use *admin_files_panel_buttons* or *admin_files_panel_fields*.

Arguments

File **file** The file object

Omeka_View **view** The view object

Examples

admin_footer

Usage

Adds content to the footer of the admin theme.

Arguments

Omeka_View **view** The view object

Examples

admin_form_files

Usage

Append content to the "Files" part of the item form.

To work with the files themselves, get the `$item` from the `$args` array and look at `$item->Files`

Arguments

Item **item** The item object

Omeka_View **view** The view object

Examples

admin_general_settings_form

Usage

Manipulate the form for admin general settings.

Arguments

Omeka_Form **form** The form

Examples

admin_head

Usage

Adds content to the header of the admin theme.

This hook is fired by the `header.php` script, which is loaded by a call to `head()` while it is populating the `<head>` tag for the admin theme.

Functions attaching to this hook can directly output HTML into the `<head>`, or use functions like *queue_css_file* and *queue_js_file*.

Plugins will likely not need to add content to the admin-side `<head>` for every page, but the passed request object can be used to include content on particular pages only.

Arguments

Omeka_View **view** The view object

Examples

See Also

- *queue_css_file*
- *queue_css_string*
- *queue_css_url*
- *queue_js_file*
- *queue_js_string*
- *queue_js_url*

admin_item_types_browse

Usage

Arguments

array item_types The array of :php:class:‘ItemType‘s objects

Omeka_View **view** The view object

Examples

admin_item_types_form

Usage

Add content to the admin item type edit form

Arguments

ItemType **item_type** The item type object

Omeka_View **view** The view object

Examples

admin_item_types_show

Usage

Add content to the item types show page

Arguments

ItemType **item_type** The item type object

Omeka_View **view** The view object

Examples

admin_items_batch_edit_form

Usage

Appends content to the end of the batch editing screen, but before the delete option.

If adding form elements, name them "custom"

See *Understanding the Admin CSS* for details on CSS styling

Arguments

Omeka_View **view** The view object

Examples

See Also

- *batch_edit_error*
- *items_batch_edit_custom*

admin_items_browse

Usage

Append content to the admin items browse page

Arguments

array items Array of :php:class:'Item's

Omeka_View **view** The view object

Examples

admin_items_browse_detailed_each

Usage

Append content to the details view of an item on the items browse page

Arguments

Item **item** The item object

Omeka_View **view** The view object

Examples

```
public function hookAdminItemsBrowseDetailedEach($args)
{
  $item = $args['item'];
  $embedTable = get_db()->getTable('Embed');
  $totalEmbeds = $embedTable->totalEmbeds($item->id);
  $html = '<p>';
  $html .= "<a href='" . url('embed-codes/item/' . $item->id) . "'>" . __('Embeds (%d)
↪', $totalEmbeds) . "</a>";
  $html .= '</p>';
  echo $html;
}
```

admin_items_browse_simple_each

Usage

Adds content to each item's row on the items browse page.

Arguments

Item **item** The item object

Omeka_View **view** The view object

Examples

admin_items_form_files

Append content to the "Files" part of the item form.

To work with the files themselves, get the `$item` from the `$args` array and look at `$item->Files`

Arguments

Item **item** The item object

Omeka_View **view** The view object

Examples

admin_items_form_tags

Usage

Add content to the tags form for an item.

To work with the tags themselves, get the `$item` from the `$args` array and do `$item->getTags()`

Arguments

Item **item** The item object

Omeka_View **view** The view object

Examples

admin_items_search

Usage

Append form elements to the advanced search area.

Arguments

Omeka_View **view** The view object

Examples

admin_items_show

Usage

Append content to the main item show area

Arguments

Item **item** The item object

Omeka_View **view** The view object

Examples

admin_items_show_sidebar

Usage

Append content to the sidebar panels on the items show page.

Content should be wrapped in `<div class="panel">`

Note: To add content to the save panel, use *admin_items_panel_buttons* or *admin_items_panel_fields*.

Arguments

Item **item** The item object

Omeka_View **view** The view object

Examples

admin_themes_browse_each

Usage

Add content to a theme's description

Arguments

Theme **theme** The theme object

Omeka_View **view** The view object

Examples

after_delete_<model>

Usage

Fires after any database model of type <model> is deleted.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

Examples

after_delete_record

Usage

Fires after any database model is deleted.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

Examples

after_ingest_file

Usage

Fires after a file is ingested, but before it is added to the item.

Arguments

File **file** The file object being processed

Item **item** The item object

Examples

after_save_<model>

Usage

Fires after a database record of type <model> is saved.

The id is available, but changes made to the record will not be saved.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

array or false post The post data, or false if none

boolean insert Whether the record is being inserted

Examples

after_save_record

Usage

Fires after any database record is saved. This includes both insert and update operations.

Since this hook fires after the record is saved, the record ID will be available, but changes made to any other record properties will not be saved.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

array or false post The post data, or false if none

boolean insert Whether the record is being inserted

Examples

before_delete_<model>

Usage

Fires before any database record of type <model> is deleted.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

Examples

before_delete_record

Usage

Fires before any database record is deleted.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

Examples

before_save_<model>

Usage

Fires before any database record of type <model> is saved. This includes both insert and update operations.

Since this hook fires before the record is saved, the record ID will not be available, but changes made to any other record properties will be saved.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

array or false post The post data, or false if none

boolean insert Whether the record is being inserted

Examples

before_save_record

Usage

Fires before any database record is saved. This includes both insert and update operations.

Since this hook fires before the record is saved, the record ID will not be available, but changes made to any other record properties will be saved.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

array or false post The post data, or false if none

boolean insert Whether the record is being inserted

Examples

config

Usage

This hook processes posted data from the configuration form created by the config_form hook. Most use cases will involve setting configuration options to the database using set_option().

Both the config and config_form hooks must be used to creating a configuration form for a plugin.

To reject the passed configuration and report an error to the user, throw an *Omeka_Validate_Exception*.

Arguments

array post The post data

Examples

```
class ExamplePlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_hooks = array('config', 'config_form');

    /* --snip-- */

    /**
     * Set the options from the config form input.
     */
    function hookConfig($args)
    {
        set_option('example_plugin_setting', $args['post']['example_plugin_setting']);
    }
}
```

See Also

- *config_form*

config_form

Usage

This hook runs when the 'Configure' button is clicked for a specific plugin on the admin plugins panel.

Plugins use this hook to directly output the HTML that goes inside the <form> tag for the configuration form.

The config hook is used to process the submitted data from the form that was created with this hook.

Arguments

None

Examples

```
class SimplePagesPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_hooks = array('config_form');

    /**
     * Display the plugin config form.
     */
    function hookConfigForm()
    {
```

(continues on next page)

(continued from previous page)

```
        require dirname(__FILE__) . '/config_form.php';
    }
}
```

See Also

- *config*

deactivate

New in version 2.1.

Usage

The deactivate hook is fired whenever a plugin is deactivated. Most things plugins do will automatically be handled by normal deactivation, but a plugin could use this hook along with *activate* to manage changes it has made external to the plugin that should be enabled and disabled.

For most use cases, plugins should be using *uninstall* instead of this hook, or simply relying on the fact that its hooks and filters will not run when it is deactivated.

Arguments

This hook has no arguments.

Examples

define_acl

Usage

This hook runs when Omeka's ACL is instantiated. It allows plugin writers to manipulate the ACL that controls user access in Omeka.

In general, plugins use this hook to restrict and/or allow access for specific user roles to the pages that it creates.

Arguments

`array` acl The Zend_Acl object

Examples

```
class SimplePagesPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_hooks = array('define_acl');

    function hookDefineAcl($args)
    {
        $acl = $args['acl'];

        $indexResource = new Zend_Acl_Resource('SimplePages_Index');
        $pageResource = new Zend_Acl_Resource('SimplePages_Page');
        $acl->add($indexResource);
        $acl->add($pageResource);

        $acl->allow(array('super', 'admin'), array('SimplePages_Index', 'SimplePages_
↪Page'));
        $acl->allow(null, 'SimplePages_Page', 'show');
        $acl->deny(null, 'SimplePages_Page', 'show-unpublished');
    }

}
```

define_routes

Usage

This hook allows the plugin writer to add routes to Omeka.

Routes can be used to create custom URLs that are different than the default ones provided by Omeka for plugin pages.

Arguments

`Zend_Controller_Router_Rewrite` router

Examples

```
class SimplePagesPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_hooks = array('define_routes');

    function hookDefineRoutes($args)
    {
        // Don't add these routes on the admin side to avoid conflicts.
        if (is_admin()) {
            return;
        }

        $router = $args['router'];

        // Add custom routes based on the page slug.
        $pages = get_db()->getTable('SimplePagesPage')->findAll();
        foreach ($pages as $page) {
            $router->addRoute(
```

(continues on next page)

(continued from previous page)

```
                'simple_pages_show_page_' . $page->id,
                new Zend_Controller_Router_Route(
                    $page->slug,
                    array(
                        'module'       => 'simple-pages',
                        'controller'   => 'page',
                        'action'       => 'show',
                        'id'           => $page->id
                    )
                )
            );
        }
    }

}
```

See Also

For possible types and configurations of routes, see Zend Framework's router documentation.

html_purifier_form_submission

Usage

Use this hook to run HTMLPurifier on submitted text. After purifying, reset the post data on the request object.

Arguments

`HTMLPurifier` purifier The HTMLPurifier object

Examples

```
class SimplePagesPlugin extends Omeka_Plugin_AbstractPlugin
{

    protected $_hooks = array('html_purifier_form_submission');

    /**
     * Filter the 'text' field of the simple-pages form, but only if the
     * 'simple_pages_filter_page' setting has been enabled from within the
     * configuration form.
     *
     * @param array $args Hook args, contains:
     *  'request': Zend_Controller_Request_Http
     *  'purifier': HTMLPurifier
     */
    function hookHtmlPurifierFormSubmission($args)
    {
        $request = Zend_Controller_Front::getInstance()->getRequest();
        $purifier = $args['purifier'];
```

(continues on next page)

(continued from previous page)

```
        // If we aren't editing or adding a page in SimplePages, don't do anything.
        if ($request->getModuleName() != 'simple-pages' or !in_array($request->
↪getActionName(), array('edit', 'add'))) {
            return;
        }

        // Do not filter HTML for the request unless this configuration directive is␣
↪on.
        if (!get_option('simple_pages_filter_page')) {
            return;
        }

        $post = $request->getPost();
        $post['text'] = $purifier->purify($post['text']);
        $request->setPost($post);
    }
}
```

initialize

Usage

This hook runs during every request on installed/activated plugins. It is primarily used to modify the runtime behavior of Omeka for every request and response.

Arguments

None

Examples

```
class SimplePagesPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_hooks = array('initialize');

    function hookInitialize()
    {
        add_translation_source(dirname(__FILE__) . '/languages');
    }

}
```

install

Usage

This hook is primarily used to set options to the database and create tables, and any other initial actions required for your plugin to function, such as initial records or setting options.

To abort installation and report an error to the user, throw an *Omeka_Plugin_Installer_Exception*.

Arguments

integer plugin_id The id of the plugin

Examples

```
class SimplePagesPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_hooks = array('install');

    protected $_options = array('simple_pages_filter_page' => '0');

    function hookInstall()
    {
        // Create the table.
        $db = get_db();
        $sql = "
        CREATE TABLE IF NOT EXISTS `$db->SimplePagesPage` (
          `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
          `modified_by_user_id` int(10) unsigned NOT NULL,
          `created_by_user_id` int(10) unsigned NOT NULL,
          `is_published` tinyint(1) NOT NULL,
          `title` tinytext COLLATE utf8_unicode_ci NOT NULL,
          `slug` tinytext COLLATE utf8_unicode_ci NOT NULL,
          `text` text COLLATE utf8_unicode_ci,
          `updated` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_
↪TIMESTAMP,
          `inserted` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
          `order` int(10) unsigned NOT NULL,
          `parent_id` int(10) unsigned NOT NULL,
          `template` tinytext COLLATE utf8_unicode_ci NOT NULL,
          `use_tiny_mce` tinyint(1) NOT NULL,
          PRIMARY KEY (`id`),
          KEY `is_published` (`is_published`),
          KEY `inserted` (`inserted`),
          KEY `updated` (`updated`),
          KEY `created_by_user_id` (`created_by_user_id`),
          KEY `modified_by_user_id` (`modified_by_user_id`),
          KEY `order` (`order`),
          KEY `parent_id` (`parent_id`)
        ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci";
        $db->query($sql);

        // Save an example page.
        $page = new SimplePagesPage;
        $page->modified_by_user_id = current_user()->id;
        $page->created_by_user_id = current_user()->id;
        $page->is_published = 1;
        $page->parent_id = 0;
        $page->title = 'About';
        $page->slug = 'about';
        $page->text = '<p>This is an example page. Feel free to replace this content,␣
↪or delete the page and start from scratch.</p>';
```

(continues on next page)

(continued from previous page)

```
        $page->save();

        $this->_installOptions();
    }

}
```

items_batch_edit_custom

Usage

Handles custom content added to the batch edit form by plugins using the *admin_items_batch_edit_form* hook.

The hook fires once for each item being edited.

Note: changes made in this hook are not automaticaly saved, so you must call `save()` on the item yourself, or use a function like *update_item* that also saves.

Arguments

Item **item** The Item currently being edited.

mixed **custom** The "custom" data from the batch edit form.

Examples

See Also

admin_items_batch_edit_form

make_<model>_featured

Usage

For models that use the *Mixin_PublicFeatured*, performs actions after the record is made featured.

Fires after the record is saved.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

Examples

make_<model>_not_featured

Usage

For models that use the *Mixin_PublicFeatured*, performs actions after the record is made not featured.

Fires after the record is saved.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

Examples

make_<model>_not_public

Usage

For models that use the *Mixin_PublicFeatured*, performs actions after the record is made not public.

Fires after the record is saved.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

Examples

make_<model>_public

Usage

For models that use the *Mixin_PublicFeatured*, performs actions after the record is made public.

Fires after the record is saved.

Arguments

Omeka_Record_AbstractRecord **record** The record being edited

Examples

navigation_form

Usage

Use this hook to modify the main navigation form

Arguments

`Omeka_Form_Navigation` form The navigation form

Examples

See Also

- *`Omeka_Navigation`*
- *public_navigation_main*

public_body

Usage

Allows plugins to hook in to the top of the `<body>` of public themes. Content added with this hook will appear at the very top of the `<body>`, outside any other markup on the page.

Arguments

***`Omeka_View`* view** The view object

Examples

public_collections_browse

Usage

Append content at the end of the collections browse page.

Arguments

***`Omeka_View`* view** The view object

`array` collections The array of *`Collection`* objects

Examples

public_collections_browse_each

Usage

Adds content at the end of each collection on the public collections browse page, before the link to the collection.

Arguments

Omeka_View **view** The view object

Colllection collection The current collection

Examples

public_collections_show

Usage

Append content to the end of a collections show page.

Arguments

Omeka_View **view** The view object

Collection **collection** The collection object

Examples

public_content_top

Usage

Inserts content at the top of the page.

Arguments

Omeka_View **view** The view object

Examples

public_footer

Usage

Add content to the end of the public footer.

Arguments

Omeka_View **view** The view object

Examples

public_head

Usage

Adds content to the <head> element. Usually used to add javascript and/or css via the queue_js_* or queue_css_ family of functions.

Arguments

Omeka_View **view** The view object

Examples

```
class MyPlugin extends Omeka_Plugin_Abstract
{
    protected $_hooks = array('public_head');

    public function hookPublicHead($args)
    {
        queue_css_file('myplugin'); // assumes myplugin has a /views/public/css/
↪myplugin.css file
    }
}
```

See Also

- *queue_css_file*
- *queue_css_string*
- *queue_css_url*
- *queue_js_file*
- *queue_js_string*
- *queue_js_url*

public_header

Usage

Adds content at the beginning of the <header> element, before the header image.

Arguments

Omeka_View **view** The view object

Examples

public_home

Usage

Add content at the end of the home page

Arguments

Omeka_View **view** The view object

Examples

public_items_browse

Usage

Append content to the items browse page

Arguments

Omeka_View **view** The view object

array items The array of *Item* objects

Examples

public_items_browse_each

Usage

Append content to each item in the items browse page.

Arguments

Omeka_View **view** The view object

Item **item** The current item object

Examples

public_items_search

Usage

Use this hook to add form elements to the advanced search form.

Additional fields should be wrapped with `<div class="field">`.

Arguments

Omeka_View **view** The view object

Examples

public_items_show

Usage

Add content to the items show page

Arguments

Omeka_View **view** The view object

Item **item** The current item object

Examples

remove_<model>_tag

Usage

Fires when tags are removed from a record of type <model>

Arguments

Omeka_Record_AbstractRecord **record** The record tags are being removed from.

`array` **added** Array of tags removed

Examples

search_sql

Usage

Modifies the SQL query used.

This hook can be used when a custom search strategy is added using the *search_query_types* filter.

Arguments

Omeka_Db_Select **select** The select object to modify

array params Array of parameters to modify the search.

Examples

See Also

get_search_query_types

<model>_browse_sql

search_query_types

uninstall

Usage

Runs when a plugin is uninstalled to remove database tables, settings, etc.

Arguments

None

Examples

```
class SimplePagesPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_hooks = array('uninstall');

    function hookUninstall()
    {
        // Drop the table.
        $db = get_db();
        $sql = "DROP TABLE IF EXISTS `$db->SimplePagesPage`";
        $db->query($sql);

        $this->_uninstallOptions();
    }

}
```

uninstall_message

Usage

Add a message for when administrators uninstall your plugin.

It will be translated if your plugin provides a translation.

The default message says that all data associated with the plugin will be deleted from the database. If that is not true for your plugin, you should provide a message indicating the consequences of uninstalling.

Arguments

None

Examples

upgrade

Usage

When Omeka detects that a user has installed a new version of the plugin, it automatically deactivates the plugin and presents an "Upgrade" button. This hook runs when a superuser clicks the "Upgrade" button. It is primarily used to migrate database tables and other data stored by older versions of the plugin.

To abort the upgrade and report an error to the user, throw an *Omeka_Plugin_Installer_Exception*.

Arguments

string old_version The previous version that was installed and is being upgraded from.

string new_version The new, current version of the plugin.

Examples

users_form

Usage

Adds content to the user form.

Note: If you are looking up any information about the user, wrap the operations in an if statement like so:

```
if($user->id) {
    //your operations and output
}
```

This is necessary because the admin view of users includes a form to add a user, which includes a new User object. Because it is new, it does not yet exist in the database. The result is that if your plugin tries to do any operations on it, it throws a database error. Checking for $user->id ensures that the user exists in the database first.

Arguments

***User* user** The user object

***Omeka_Form* form** The form to modify

Examples

3.2.2 Plugin-specific

3.2.3 Application

3.2.4 Database

3.2.5 Records

Item Records

Collection Records

3.2.6 Users

3.2.7 Admin Theme

3.2.8 Public Theme

3.3 Filters

Filters are one of the main ways plugins can modify and extend Omeka. Filters allow you to directly modify data by modifying and returning data passed to the filter in the first argument.

This page is a list of all the filters Omeka provides. For more about using filters in plugins, see *Understanding Filters*.

3.3.1 All Filters

<model>s_browse_default_sort

New in version 2.3.

Usage

Filters the sort parameters of <model> records returned in default browse queries. If a sort param is passed in the query, this will not apply.

Note: Use the plural form of the <model>

Value

`array` $sortArray Array of parameters, with the first element being the sort_field parameter, and the second (optionally) the sort_dir.

Arguments

array params All params passed to the search query

<model>s_browse_params

Usage

Filters the parameters used when generating the browse page for records (the model)

Note: Use the plural form of the <model>

Value

array $params Params passed to the browse query. May include both filtering and sorting parameters

Arguments

None

Examples

```
class MyPlugin extends Omeka_Plugin_AbstractPlugin
{

    protected $_filters = array('items_browse_params');

    public function filterItemsBrowseParams($params)
    {
        //always sort by title instead of order
        $params['sort_field'] = "Dublin Core,Title";
        return $params;
    }
}
```

See Also

- *<model>_browse_sql*
- *Omeka_Db_Table::getSelectForFindBy*

<model>s_browse_per_page

New in version 2.3.

Usage

Filters the number of <model> records returned per page in browse queries.

Note: Use the plural form of the <model>

Value

`int` $perPage The number of records to return

Arguments

***`Omeka_Controller_AbstractActionController`* controller** The controller issuing the query

<model>s_select_options

Usage

Filters the options for a select element for the <model>'s table. Useful if your plugin adds relations between the <model>'s table and another table.

Note: Use the plural form of the <model>

Value

`array` $options Array of key-value pairs, where the key is usually a record ID and the value is determined by the model's table

Arguments

None

See Also

- *get_table_options — Get the options array for a given table.*
- *label_table_options — Add a "Select Below" or other label option to a set of select options.*
- *`Omeka_Db_Table::findPairsForSelectForm`*

ElementSetForm Filter

Usage

Filters the form to be used for an element set

The name of the filter is an array:

```
array('ElementSetForm', $recordType, $elementSet, $elementName);
```

- $recordType: The type of Omeka object to filter the metadata for. Most commonly, this will be 'Item'.
- $elementSetName: The name of the element set containing the metadata field to be filtered. Possible values include 'Dublin Core' and 'Item Type Metadata'.
- $elementName: The name of the specific element within the set to be filtered.

Value

`array` $elements Array of :php:class:'Element's to filter

Arguments

`string` recordType The type of record

***`Omeka_Record_AbstractRecord`* record** The record whose form is being filtered

`string` elementSetName The name of the element set whose elements should be filtered

Examples

See Also

Element Display Filter

Usage

Alter the way an element is displayed.

The name of the filter is an array:

```
array('Display', $recordType, $elementSetName, $elementName);
```

- $recordType: The type of Omeka object to filter the metadata for. Most commonly, this will be 'Item'.
- $elementSetName: The name of the element set containing the metadata field to be filtered. Possible values include 'Dublin Core' and 'Item Type Metadata'.
- $elementName: The name of the specific element within the set to be filtered.

Value

`string` $text The text to display.

Arguments

Omeka_Record_AbstractRecord **record** The record being displayed

ElementText **element_text** The ElementText object

Examples

```
class MyPlugin extends Omeka_Plugin_AbstractPlugin
{

    protected $_filters = array('concealDescription' => array('Display', 'Item',
↪'Dublin Core', 'Description'));

    public function concealDescription($text, $args)
    {
        //conceal the description to non-logged in users
        if(!get_current_user()) {
            return str_rot13($text);
        }
        return $text;
    }
}
```

Element ElementForm Filter

Usage

Customize the form for a particular element. This only applies to forms generated by the element_form() helper function.

The name is an array:

```
array('ElementForm', $recordType, $elementSetName, $elementName);
```

- $recordType: The type of Omeka object to filter the metadata for. Most commonly, this will be 'Item'.
- $elementSetName: The name of the element set containing the metadata field to be filtered. Possible values include 'Dublin Core' and 'Item Type Metadata'.
- $elementName: The name of the specific element within the set to be filtered.

Value

array $components The form components, like:

```
$components = array(
    'label' => $labelComponent,
    'inputs' => $inputsComponent,
    'description' => $descriptionComponent,
    'comment' => $commentComponent,
    'add_input' => $addInputComponent,
    'html' => null
);
```

Arguments

Omeka_Record_AbstractRecord **record** The model being edited.

Element **element** The element being edited

array options An array of additional options for the form

Examples

```
class MyPlugin extends Omeka_Plugin_AbstractPlugin
{

    $_filters = array('relabelItemTitle' => array('ElementForm', 'Item', 'Dublin Core
↪', 'Title'));

    public function relabelItemTitle($components, $args)
    {
        $components['label'] = "Label";
        return $components;
    }
}
```

See Also

Element ElementInput Filter

Element ElementInput Filter

Usage

Customize the input for a particular element. This only applies to forms generated by the element_form() helper function.

The name of the filter is an array:

```
array('ElementInput', $recordType, $elementSet, $elementName);
```

- $recordType: The type of Omeka object to filter the metadata for. Most commonly, this will be 'Item'.
- $elementSetName: The name of the element set containing the metadata field to be filtered. Possible values include 'Dublin Core' and 'Item Type Metadata'.
- $elementName: The name of the specific element within the set to be filtered.

Value

array $components The input components, like:

```
$components = array(
    'input' => $formTextarea,
    'form_controls' => $removeButton,
```

(continues on next page)

(continued from previous page)

```
    'html_checkbox' => $useHtmlCheckbox,
    'html' => null
);
```

Arguments

string input_name_stem The name of the input, e.g., `Elements[1][0]`

string value The value for the input

***Omeka_Record_AbstractRecord* record** The model being edited

***Element* element** The element being edited

string index The index of the input for the element

boolean is_html Whether the input uses html

Examples

```
class MyPlugin extends Omeka_Plugin_AbstractPlugin
{

    protected $_filters = array('shortenSubjectField' => array('ElementInput', 'Item',
↪ 'Dublin Core', 'Title'));

    public function shortenSubjectField($components, $args)
    {
        $components['input'] = get_view()->formText($args['input_name_stem'] . '[text]
↪', $args['value']);
        return $components;
    }
}
```

Element Flatten Filter

Usage

'Flatten' the array of data for an element into the text to use.

The array looks like:

```
array('text' => 'The text', 'html'=>0);
```

Where the `html` key is a boolean for whether the element text uses HTML.

If no Element Flatten filters operate, the value of the `text` key is used.

This filter's name is actually an array of strings. The first string must always be 'Flatten', but the last three can change depending on exactly what values you want to filter.

`array('Flatten', $recordType, $elementSetName, $elementName)`

- $recordType: The type of Omeka object to filter the metadata for.

- $elementSetName: The name of the element set containing the metadata field to be filtered. Possible values will include 'Dublin Core' and 'Item Type Metadata'.
- $elementName: The name of the specific element within the set to be filtered.

Value

`string` $flatText The flattened text to return. Initially `null`.

Arguments

`array` post_array The posted data, like `array('text' => 'The text', 'html'=>0);`

***`Element`* element** The element being saved.

Examples

```
$_filters = array('prependJulianToDate' => array('Flatten', 'Item', 'Dublin Core',
↪'Date'));

public function prependJulianToDate($flatText, $args)
{
    $postArray = $args['post_array'];
    $value = $postArray['text'];
    return "Julian Calendar: $value";
}
```

See Also

`ElementText::getTextStringFromFormPost`

Element Save Filter

Usage

Customize element texts for a particular element before validating and saving a record. This is helpful if you want to prepare form data for validation automatically, limiting the possibility of a validation error.

`array('Save', $recordType, $elementSetName, $elementName)`

- $recordType: The type of Omeka object to filter the metadata for. Most commonly, this will be 'Item'.
- $elementSetName: The name of the element set containing the metadata field to be filtered. Possible values include 'Dublin Core' and 'Item Type Metadata'.
- $elementName: The name of the specific element within the set to be filtered.

Value

`string` $text The original text for the element

Arguments

Omeka_Record_AbstractRecord **record** The record that this text applies to. The type will be the same as the filter's `$recordType`.

Element **element** The Element record for this text.

Examples

```
class MyPlugin extends :php:class:`Omeka_Plugin_AbstractPlugin`
{

    protected $_filters = array('addIsbnToDcId'=>array('Save', 'Item', 'Dublin Core',
↪'Identifier');

    public function addIsbnToId($text, $args)
    {
        return "ISBN: $text";
    }
}
```

Element Validation Filter

Usage

Perform a custom validation on the texts for any particular element.

This filter's name is actually an array of strings. The first string must always be 'Validate', but the last three can change depending on exactly what values you want to validate.

`array('Validate', $recordType, $elementSetName, $elementName)`

- $recordType: The type of Omeka object to filter the metadata for.
- $elementSetName: The name of the element set containing the metadata field to be filtered. Possible values will include 'Dublin Core' and 'Item Type Metadata'.
- $elementName: The name of the specific element within the set to be filtered.

Value

`bool` $isValid Whether the element text is valid.

Arguments

`string` text The text for the element

Omeka_Record_AbstractRecord **record** The record (subclass of Omeka_Record_AbstractRecord) being validated.

Element **element** The element (e.g., Dublin Core Title) for the element text.

Examples

```
class MyPlugin extends Omeka_Plugin_AbstractPlugin
{

    protected $_filters = array('itemTitleLengthValidator' => array('Validate', 'Item
↪', 'Dublin Core', 'Title'));

    public function itemTitleValidator($isValid, $args)
    {
        $text = $args['text'];
        if(strlen($text) > 100) {
            return false;
        }
        return true;
    }

}
```

See Also

`ElementText::_elementTextIsValid`

action_contexts

Usage

Action contexts work in combination with *response_contexts* to output additiona formats, such as RSS or JSON.

Typically, you check whether the controller passed in the arguments is one defined by your plugin, then add your context (output format) to the array of values.

Then, response contexts you define direct Omeka to the views you have defined for the output.

Value

`array` $contexts Array of the contexts available for a controller's views.

Arguments

`Omeka_Controller_Action` controller The controller that is producing the output.

'Zend_Controller_ActionHelper_ContextSwitch <http://framework.zend.com/manual/1.12/en/zend.controller.actionhelpers.html#zend.controller.actionhelpers.contextswitch> context_switch

Examples

Make an RSS feed of your records available at `/my-records/browse?output=rss`

```
class MyRecordController extends Omeka_Controller_AbstractActionController
{
    //nothing to do here, since we don't need to override anything
}

class MyPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_filters = array('action_contexts');

    public filterActionContexts($contexts, $args)
    {
        if($args['controller'] instanceof MyRecordController) {
            $contexts['browse'][] = 'rss';
        }
        return $contexts;
    }
}
```

See Also

response_contexts

admin_collections_form_tabs

Usage

Filter tabs used on the collections add/edit page

Value

array $tabs The array of tabs. The key is the label for the tab, and the value is the html to display.

Arguments

***Collection* collection** The collection being edited

Examples

admin_dashboard_panels

Usage

Add a panel to the dashboard. HTML content is automatically wrapped in the appropriately styled `<div>`

Value

array $panels Array of HTML content for each panel

Arguments

Omeka_View **view** The current view object

Examples

admin_dashboard_stats

Usage

Add content to the stats row at the top of the dashboard

Value

array $stats Array of links for the stats row

Arguments

Omeka_View **view** The current view object

Examples

Original value looks like:

```
$stats = array(
    array(link_to('items', null, total_records('Item')), __('items')),
    array(link_to('collections', null, total_records('Collection')), __('collections
↪')),
    array(link_to('tags', null, total_records('Tag')), __('tags'))
);
```

admin_files_form_tabs

Usage

Add to or alter the form sections and tabs on the edit page for a File.

Value

array $tabs The array of tabs. The key is the label for the tab, and the value is the HTML to display.

Arguments

File **file** The file being edited

Examples

admin_items_form_tabs

Usage

The admin_items_form_tabs filter allows you to add, remove, or modify the tabs on the edit item form.

Value

array $tabs An array of the item form tabs. The keys are the tab names, and the values are the HTML content of each tab. A filter function should modify this array and return it.

Arguments

***Item* item** The Item being edited. Filter functions can use this parameter to change the tabs or content on an item-by-item basis.

Examples

```
class MyPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_filters = array('admin_items_form_tabs');

    public function filterAdminItemsFormTabs($tabs, $args)
    {
        $item = $args['item'];
        $tabs["My Plugin"] = uri('/my-plugin/edit/id/' . $item->id);
    }
}
```

admin_navigation_global

Usage

Filters the global navigation of the admin theme.

Value

array $nav An array of arrays as used by Zend_Navigation.

Arguments

None

Examples

The array that forms the basis for settings navigation is:

```
$globalNav = array(
    array(
        'label' => __('Plugins'),
        'uri' => url('plugins'),
        'resource' => 'Plugins',
        'privilege' => 'edit'
        ),
    array(
        'label' => __('Appearance'),
        'uri' => url('appearance'),
        'resource' => 'Appearance',
        'privilege' => 'edit'
        ),
    array(
        'label' => __('Users'),
        'uri' => url('users'),
        'resource' => 'Users',
        'privilege' => 'edit'
        ),
    array(
        'label' => __('Settings'),
        'uri' => url('settings'),
        'resource' => 'Settings',
        'privilege' => 'edit'
        )
    );
```

To add a new link to a plugin's admin interface, you would use this filter like this:

```
public function filterAdminNavigationGlobal($nav)
{
    $nav[] = array(
                    'label' => __('My Plugin'),
                    'uri' => url('my-plugin')
                  );
    return $nav;
}
```

admin_navigation_main

Usage

Modifies the top-level navigation for the admin theme.

Value

array $nav An array of arrays as used by Zend_Navigation.

Arguments

None

Examples

The array that forms the basis for admin navigation is:

```
$mainNav = array(
    array(
        'label' => __('Dashboard'),
        'uri' => url('')
    ),
    array(
        'label' => __('Items'),
        'uri' => url('items')
    ),
    array(
        'label' => __('Collections'),
        'uri' => url('collections')
    ),
    array(
        'label' => __('Item Types'),
        'uri' => url('item-types')
    ),
    array(
        'label' => __('Tags'),
        'uri' => url('tags')
    )
);
```

To add a new link to a plugin's admin interface, you would use this filter like this:

```
public function filterAdminNavigationMain($nav)
{
    $nav[] = array(
                    'label' => __('My Plugin'),
                    'uri' => url('my-plugin')
                );
    return $nav;
}
```

admin_navigation_settings

Usage

Filters the navigation for the settings page of the admin theme.

Value

array $nav An array of arrays as used by Zend_Navigation.

Arguments

None

Examples

The array that forms the basis for settings navigation is:

```
$navArray = array(
    array(
        'label' => __('General Settings'),
        'uri' => url('settings')
    ),
    array(
        'label' => __('Element Sets'),
        'uri' => url('element-sets'),
        'resource' => 'ElementSets',
        'privilege' => 'browse'
    ),
    array(
        'label' => __('Security Settings'),
        'uri' => url('security'),
        'resource' => 'Security',
        'privilege' => 'edit'
    ),
    array(
        'label' => __('Search Settings'),
        'uri' => url('search/settings')
    )
);
```

To add a new link to a plugin's admin interface, you would use this filter like this:

```
public function filterAdminNavigationSettings($nav)
{
    $nav[] = array(
                    'label' => __('My Plugin Settings'),
                    'uri' => url('my-plugin/settings')
                );
    return $nav;
}
```

admin_navigation_users

Usage

Filters the navigation for the users/edit pages of the admin theme.

Value

array $nav An array of arrays as used by Zend_Navigation.

Arguments

User **user** The user being edited

Examples

admin_theme_name

Usage

Filters the currently-selected admin theme. Changing the theme name through this filter will cause Omeka to use a different theme to display the admin interface.

Value

string **$theme_name** The name of the theme being used

Arguments

None

Examples

Force Omeka to use a different admin theme

```
public function filterAdminThemeName($theme_name)
{
    return 'my-custom-admin-theme';
}
```

admin_whitelist

Usage

Allows admin controller views to be accessible without logging in

Value

array **$whitelist** An array containing arrays of controller-view pairs that do not require logging in to access

Arguments

None

Examples

The default whitelist is:

```
array(
    array('controller' => 'users', 'action' => 'activate'),
    array('controller' => 'users', 'action' => 'login'),
    array('controller' => 'users', 'action' => 'forgot-password'),
    array('controller' => 'installer', 'action' => 'notify'),
    array('controller' => 'error', 'action' => 'error')
);
```

To make your plugin's admin page accessible without login

```
public function filterAdminWhitelist($whitelist)
{
    $whitelist[] = array('controller' => 'my-controller' , 'action' => 'my-view');
    return $whitelist;
}
```

api_extend_<resource>

New in version 2.1.

Usage

Extends an existing resource registered with the API. If your plugin creates content related to items, for example, use this to add that data to an item's representation. `<resource>` is the name of the resource, e.g. `items`

Value

`array` $extend Array of representations that extend the resource

Arguments

Omeka_Record_AbstractRecord **record** The record whose representation in the API you are extending

Examples

```
public function filterApiExtendItems($extend, $args)
{
    $item = $args['record'];
    $location = $this->_db->getTable('Location')->findBy(array('item_id' => $item->
↪id));
    if (!$location) {
        return $extend;
    }
    $locationId = $location[0]['id'];
    $extend['geolocations'] = array(
        'id' => $locationId,
```

(continues on next page)

(continued from previous page)

```
        'url' => Omeka_Record_Api_AbstractRecordAdapter::getResourceUrl("/
↪geolocations/$locationId"),
    );
    return $extend;
}
```

See also *Extending the API*

api_resources

New in version 2.1.

Usage

Allows plugins to add their content to the Omeka API

Value

`array` $apiResources An array of the resources registered with the API

Arguments

None

Examples

```
    <?php
    protected $_filters = array('api_resources');

    public function filterApiResources($apiResources)
    {
        $apiResources['geolocations'] = array(
            'record_type' => 'Location',
            'actions' => array('get', 'index', 'post', 'put', 'delete'),
        );
        return $apiResources;
    }

See also :doc:`/Reference/api/extending`
```

batch_edit_error

Usage

Add or alter the error message when batch editing

Value

string $errorMessage The message to display

Arguments

array metadata The array of metadata about the items to change

array custom Array of custom data added by plugins

array item_ids Array of item ids being edited

See Also

admin_items_batch_edit_form

body_tag_attributes

Usage

Filters the tags applied to the <body> element of the page

Value

array $attributes Attributes to add to or alter

Arguments

None

Examples

```
class MyPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_filters = array('body_tag_attributes');

    public function filterBodyTagAttributes($attributes)
    {
        //add a class when users are logged in

        if(current_user()) {
            $attributes['class'] = trim("logged-in " . $attributes['class']);

        }
        return $attributes;
    }
}
```

browse_plugins

Usage

Allows you to filter the list of plugins on the admin plugins browse page

Value

array $plugins Array of plugins keyed by their directory name.

Arguments

None

Examples

Remove plugins that do not meet minimum Omeka version unless current user is super

```
class MyPlugin extends :php:class:`Omeka_Plugin_AbstractPlugin`
{

    protected $_filters = array('browse_plugins');

    public filterBrowsePlugins($plugins)
    {
        $user = current_user();

        if($user->getRole() != 'super') {
            foreach($plugins as $key=>$plugin) {
                if(!$plugin->meetsOmekaMinimumVersion()) {
                    unset($plugins[$key]);
                }
            }
        }
        return $plugins;
    }
}
```

browse_themes

Usage

Allows you to filter the list of themes on the admin themes browse page

Value

array $themes Array of themes keyed by their directory name.

Arguments

None

Examples

Ignore themes without images

```
class MyPlugin extends :php:class:`Omeka_Plugin_AbstractPlugin`
{

    protected $_filters = array('browse_themes');

    public filterBrowseThemes($themes)
    {
        foreach($themes as $key=>$theme) {
            if(empty($theme->image)) {
                unset($themes['key'];
            }
        }

        return $themes;
    }
}
```

display_elements

Usage

Filters the element sets and elements displayed by the *all_element_texts* function.

Value

array $elementSets All the elements, keyed by element set.

Arguments

None

Examples

Prevent display of the Dublin Core Title and Description elements:

```
class MyPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_filters = array('display_elements');

    public function filterDisplayElements($elementSets)
    {
        foreach($elementSets as $set => $elements) {
```

(continues on next page)

(continued from previous page)

```
            if ($set == 'Dublin Core') {
                foreach ($elements as $key => $element) {
                    if ($element->name == 'Title' || $element->name == 'Description')
↪{
                        unset($elementSets['Dublin Core'][$key]);
                    }
                }
            }
        }
        return $elementSets;
    }
}
```

display_option_*

Usage

This family of filters is used whenever the *option* theme function is called, to filter the value returned from the database before it is displayed.

The name of the particular filter that should be used is made by appending the name of the option to "display_option_".

Value

`string` $option The original value of the option.

Examples

To print the site title, a theme writer would do the following:

```
<?php echo option('site_title'); ?>
```

If you want to prepend "Omeka: " to the title of every printed page:

```
public function filterDisplayOptionSiteTitle($option)
{
    return "Omeka: $option";
}
```

file_ingest_validators

Usage

Allows you to add or remove validators to be used when a file is uploaded.

Value

`array` $validators A keyed array of validators to use. Validators are subclasses of Zend_Validate_Abstract.

Arguments

None

Examples

The default validators are:

```
array(
      'extension whitelist'=> new Omeka_Validate_File_Extension,
      'MIME type whitelist'=> new Omeka_Validate_File_MimeType
    )
```

To swap out Omeka's Mime Type validator for your own:

```
public function filterFileIngestValidators($validators)
{
    unset($validators, 'MIME type whitelist');
    $validators['my mime validator'] = new MyPlugin_Validate_File_MimeType;
    return $validators;
}
```

file_markup

Usage

Filter the HTML for displaying a file

Value

`string` $html The HTML for displaying the file

Arguments

***`File`* file** The file object

`function` callback A callback to use to generate HTML

`array` options Options to pass to the callback

`string` wrapper_attributes Attributes to apply to the wrapping `<div>` element

See Also

`file_markup`, the function

file_markup_files

New in version 2.7.

Usage

Filter the set of files to be displayed by *file_markup*

Omeka uses functions like *files_for_item* to render all the files attached to an item; this filter allows a developer to alter that list, for example to exclude some files from being displayed.

Value

array $files Array of *File* objects to be displayed

Arguments

array options Options passed to file_markup

file_markup_options

New in version 2.7.

Usage

Modify the options passed to *file_markup*

Unlike the *file_markup* filter, this filter allows you to change options that Omeka uses when rendering HTML for a file, meaning you can alter the output without having to write completely new markup yourself.

Value

array $options Array of options for file_markup

Arguments

***File* file** The file object

Examples

Add target="_blank" to all links to original files:

```
class MyPlugin extends :php:class:`Omeka_Plugin_AbstractPlugin`
{
    protected $_filters = array('file_markup_options');

    public filterHtmlPurifierConfigSetup($options, $args)
    {
        $options['linkAttributes']['target'] = '_blank';
        return $options;
    }
}
```

See Also

file_markup documentation for a list of possible options

files_for_item

New in version 2.7.

Usage

Filter the HTML for displaying all files for an item.

The *file_markup* filter is used to modify the HTML markup for displaying a single file, but this filter can be used to modify the markup for all the files being displayed as a whole (for example, to add markup that comes before or after, or wraps around, all the displayed files).

Value

string $html The HTML for displaying the files

Arguments

***Item* item** The item object

array options Options to pass to the callback

string wrapperAttributes Attributes to apply to the wrapping `<div>` element

See Also

files_for_item

html_escape

Usage

Adds additional escaping to a string

Value

string $string The string to escape

Arguments

None

Examples

See Also

html_escape

html_purifier_config_setup

New in version 2.7.

Usage

Alters the configuration settings for the HTMLPurifier library.

Value

HTMLPurifier_Config $purifierConfig The config object for HTMLPurifier

Arguments

array defaults The default configuration keys (these have already been set to the passed config object)

array allowedHtmlElements List of user-configured enabled elements

array allowedHtmlAttributes List of user-configured enabled attributes

Examples

Force a prefix on all user-provided `id` attributes (for other options see the HTMLPurifier documentation pages linked below:

```
class MyPlugin extends :php:class:`Omeka_Plugin_AbstractPlugin`
{
    protected $_filters = array('html_purifier_config_setup');

    public filterHtmlPurifierConfigSetup($purifierConfig, $args)
    {
        $purifierConfig->set('Attr.IDPrefix', 'foo_');
        return $purifierConfig;
    }
}
```

See Also

- HTMLPurifier customization documentation
- HTMLPurifier list of configuration options

image_tag_attributes

New in version 2.7.

Usage

Filter the HTML attributes used for an record's image.

This filter affects `<img>` tags created by *file_markup*, *files_for_item*, *record_image*, *item_image*, and *file_image* (essentially meaning it covers anywhere derivative images are used for a record).

Value

array $attrs Array of HTML attributes for the `<img>` tag.

Arguments

Omeka_Record record The record being displayed (could be an Item or Collection in the case of a thumbnail)

***File* file** The file being displayed. This is always the File that directly corresponds with the image, while `record` can be another record like an Item, Collection or Exhibit that simply uses the File as its representative image.

string format Type of derivative image being displayed (e.g., `'thumbnail'`, `'fullsize'`, `'square_thumbnail'`)

item_citation

Usage

Filters or replaces the default citation displayed for an item.

Value

string $citation The citation created by Omeka

Arguments

***Item* item** The item for the citation

Examples

```
class MyPlugin extends :php:class:`Omeka_Plugin_AbstractPlugin`
{
    protected $_filters = array('item_citation');

    public filterItemCitation($citation, $args)
    {
        $citation = "";
```

(continues on next page)

(continued from previous page)

```
        return $citation;
    }
}
```

See Also

`item_citation`

item_next

New in version 2.5.

Usage

Replaces the item returned by *`Item::next()`*, and therefore also the item linked to by the "Next Item" link on the item show page.

Value

`Item` $nextItem The "next" item relative to the one passed as the `item` argument. Note, the initial filter passes *null* here. Returning *null* will cause Omeka to use the default "next" item.

Arguments

***`Item`* item** The current item

Examples

See Also

item_previous

item_previous

New in version 2.5.

Usage

Replaces the item returned by *`Item::previous()`*, and therefore also the item linked to by the "Previous Item" link on the item show page.

Value

Item $previousItem The "previous" item relative to the one passed as the `item` argument. Note, the initial filter passes *null* here. Returning *null* will cause Omeka to use the default "previous" item.

Arguments

***Item* item** The current item

Examples

See Also

item_next

item_search_filters

Usage

Use this filter in conjunction with the *<model>_browse_sql* hook to display what search filters are being applied to the item search.

If your plugin uses `item_browse_sql` to provide an additional filter when searching, you should use this filter to display the results.

Look through the `request_array` passed in the arguments for the key/value pair with key matching the additional filter you have created. Add the display value to the array as a new key/value pair, with the key being the name of your filter and the value being the text to display.

Value

array $displayArray Array of filters for the search/browse page. Keys are possible `$_GET` parameters. Values are values being filtered.

Arguments

array request_array Array of `$_GET` parameters

Examples

The MultiCollections plugin clobbers the core collection filtering message by checking if a `multi-collection` value is among the `$_GET` parameters and displaying the collection name.

```
public function filterItemSearchFilters($displayArray, $args)
{
    $request_array = $args['request_array'];
    if(isset($request_array['multi-collection'])) {
        $db = get_db();
```

(continues on next page)

(continued from previous page)

```
        $collection = $db->getTable('Collection')->find($request_array['multi-
↪collection']);
        $displayValue = strip_formatting(metadata($collection, array('Dublin Core',
↪'Title')));
        $displayArray['collection'] = $displayValue;
    }
    return $displayArray;
}
```

See Also

Omeka_View_Helper_ItemSearchFilters

items_advanced_search_link_default_url

Usage

Changes the url to use for `items/advanced-search`

Value

`string` $url The url for items advanced search

Arguments

None

Examples

locale

Usage

Set Omeka's locale. Value must correspond to an existing .mo file in Omeka's translations directory

Value

`string` locale The locale name

Arguments

None

login_adapter

Usage

The login adapter filter may be used to override the default way Omeka authenticates users, for example by checking against a different table.

Value

Omeka_Auth_Adapter_UserTable **$authAdapter** The adapter to use for authenticating a user. You can return your own adapter if necessary for more complex authentication systems.

Arguments

Omeka_Form_Login login_form The form to use for logging in. Can be replaced via the *login_form* filter.

Examples

login_form

Usage

Use this filter to modify or replace the default login form

Value

Omeka_Form_Login $form Omeka's login form.

Arguments

None

Examples

page_caching_blacklist

Usage

Arguments

Omeka_Record_AbstractRecord **record** The record being saved

string action The action taking place

Examples

```
class SimplePagesPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_filters = array('page_caching_blacklist');

    /**
     * Add pages to the blacklist
     *
     * @param $blacklist array An associative array urls to blacklist,
     * where the key is a regular expression of relative urls to blacklist
     * and the value is an array of Zend_Cache front end settings
     * @param $record
     * @param $args Filter arguments. contains:
     * - record: the record
     * - action: the action
     * @return array The blacklist
     */
    function filterPageCachingBlacklistForRecord($blacklist, $args)
    {
        $record = $args['record'];
        $action = $args['action'];

        if ($record instanceof SimplePagesPage) {
            $page = $record;
            if ($action == 'update' || $action == 'delete') {
                $blacklist['/' . trim($page->slug, '/')] = array('cache'=>false);
            }
        }

        return $blacklist;
    }
}
```

page_caching_whitelist

Usage

Arguments

None

Examples

```
class SimplePagesPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_filters = array('page_caching_whitelist');

    /**
     * Specify the default list of urls to whitelist
     *
     * @param $whitelist array An associative array urls to whitelist,
     * where the key is a regular expression of relative urls to whitelist
```

(continues on next page)

(continued from previous page)

```
     * and the value is an array of Zend_Cache front end settings
     * @return array The whitelist
     */
    function filterPageCachingWhitelist($whitelist)
    {
        // Add custom routes based on the page slug.
        $pages = get_db()->getTable('SimplePagesPage')->findAll();
        foreach($pages as $page) {
            $whitelist['/' . trim($page->slug, '/')] = array('cache'=>true);
        }

        return $whitelist;
    }

}
```

public_navigation_admin_bar

Usage

Filters the navigation links in public theme's admin bar (the bar of links for logged-in users).

Value

array $navLinks Array of navigation links

Begins as:

```
$navLinks = array(
    array(
        'label' => __('Welcome, %s', $user->name),
        'uri' => admin_url('/users/edit/'.$user->id)
    ),
    array(
        'label' => __('Omeka Admin'),
        'uri' => admin_url('/')
    ),
    array(
        'label' => __('Log Out'),
        'uri' => url('/users/logout')
    )
 );
```

Arguments

None

Examples

public_navigation_items

Usage

Modifies the navigation option on public items browse page

Value

array $navArray An array of arrays as used by Zend_Navigation.

Arguments

None

Examples

The base navigation looks like:

```
$navArray = array(
    array(
        'label' =>__('Browse All'),
        'uri' => url('items/browse'),
    ),
    array(
        'label' => __('Browse by Tag'),
        'uri' => url('items/tags')
    )
);
```

So you could add to it with:

```
function filterPublicNavigationItems($navArray)
{
    $navArray[] = array('label'=> __('My Plugin Items'),
                        'uri' => url('myplugin/items')
                        );
    return $navArray;

}
```

public_navigation_main

Usage

Modifies the top-level navigation for the public theme.

Value

array $nav An array of arrays as used by Zend_Navigation.

Arguments

None

Examples

To add a new link to a plugin's public interface, you would use this filter like this:

```
public function filterPublicNavigationMain($nav)
{
    $nav[] = array(
                    'label' => __('My Plugin'),
                    'uri' => url('my-plugin')
                );
    return $nav;
}
```

public_theme_name

Filters the currently-selected public theme. Changing the theme name through this filter will cause Omeka to use a different theme to display the public interface.

Value

`string` $theme_name The name of the theme being used

Arguments

None

Examples

Force Omeka to use a different public theme

```
public function filterPublicThemeName($theme_name)
{
    return 'my-custom-admin-theme';
}
```

response_contexts

Usage

Filters the array of response contexts as passed to Zend Framework's ContextSwitch helper.

Response contexts are used to serve the same data (an Item, for example) in different formats. Omeka includes by default response contexts for JSON, RSS and XML response contexts, in addition to the default context, which produces the normal HTML output.

Suffix values added to the contexts correspond to view files. Thus, to add an RSS feed at url `my-records/browse?output=rss`, you will have a view named `browse.rss.php`

Value

`array` $contexts Array of contexts, corresponding to action contexts defined by using *action_contexts*

Arguments

None

Examples

Add an RSS feed at `my-records/browse?output=rss`

```
class MyPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_filters = array('response_contexts');

    public function filterResponseContexts($contexts)
    {
        $contexts['rss'] = array('suffix' => 'rss',
                                 'headers' => array('Content-Type' => 'text/xml'));
        return $contexts;

    }
}
```

search_element_texts

New in version 2.3.

Usage

Filters the element texts that will be included in searches.

Value

`array` $allElementTexts Array of all element text objects to be saved for the record.

Arguments

None

search_form

New in version 2.3.

Usage

Alter the sitewide search form created by *search_form*.

The original form can be altered, added to, or replaced using this filter.

Value

string $form The HTML of the search form

Arguments

array options Options passed to the form partial as $options

array filters Filters passed to the form partial as $filters

array query_types Query types for the form to make available

array record_types Record types for the form to make available

search_form_default_action

Usage

Alter the `action` attribute setting the URL to use when the sitewide search form is submitted. This filter only changes the default value; if a specific `action` is set for a call to *search_form*, that will take precedence.

Value

string $url The url for the form's action attribute

Arguments

None

search_form_default_query

New in version 2.3.

Usage

Set a default query text for the sitewide search form.

Value

string $query The text for the query

Arguments

None

search_form_default_query_type

New in version 2.3.

Usage

Set the default query type for the search form. The unfiltered default type is 'keyword'. This will affect the default selection when the advanced search form is enabled, and will specify the search type performed when the advanced form is disabled.

Value

`string` $searchQueryType The type for the query

Arguments

None

search_form_default_record_types

New in version 2.3.

Usage

Filter default selected record types for the search form.

Using this filter, a site can have record types that are enabled for indexing but will not appear in search results unless specifically requested by a user performing an advanced search.

Value

`array` $recordTypes Array of record types. The defaults are all the types enabled in the site's Search Settings

Arguments

None

search_query_types

Usage

Add a new search query type for the advanced search form

Value

array $searchQueryTypes Array of query types

Arguments

None

Examples

The original array is

```
$searchQueryTypes = array(
    'keyword'     => __('Keyword'),
    'boolean'     => __('Boolean'),
    'exact_match' => __('Exact match'),
);
```

To implement an 'ends with' search query type, you must use both this filter, and the *search_sql* hook

```
class EndsWithPlugin extends Omeka_Plugin_AbstractPlugin
{
    protected $_filters = array('search_query_types');

    protected $_hooks = array('search_sql');

    public function filterSearchQueryTypes($queryTypes)
    {
        // Register the name of your custom query type. The key should be the
        // type's GET query value; the values should be the human readable and
        // internationalized version of the query type.
        $queryTypes['ends_with'] = __('Ends with');
        return $queryTypes;
    }

    public function hookSearchSql($args)
    {
        $params = $args['params'];
        if ('ends_with' == $params['query_type']) {
            $select = $args['select'];
            // Make sure to reset the existing WHERE clause.
            $select->reset(Zend_Db_Select::WHERE);
            $select->where('`text` REGEXP ?', $params['query'] . '[[:>:]]');
        }

    }
}
```

See Also

get_search_query_types

<model>_browse_sql

search_record_types

Usage

Add a record type to the list of options for the search form. Plugins that add searchable content should use this to allow administrators to choose whether to include that content in the search options.

Value

array $searchRecordTypes A key-value array where keys are names of record types and values are the internationalized forms of their labels:

```
$searchRecordTypes = array(
    'Item'       => __('Item'),
    'File'       => __('File'),
    'Collection' => __('Collection'),
);
```

Arguments

None

Examples

```
class SimplePagesPlugin extends Omeka_Plugin_AbstractPlugin
{

    /**
     * Add SimplePagesPage as a searchable type.
     */
    function filterSearchRecordTypes($recordTypes)
    {
        $recordTypes['SimplePagesPage'] = __('Simple Page');
        return $recordTypes;
    }
}
```

There are times when theme writers will want to change the record type label on the search results page. For example, when the "Exhibit" is a gallery and the "Exhibit Pages" are individual artists. To do this, in the theme's custom.php file, add the following

```
add_filter('search_record_types', 'my_site_search_record_types');
function my_site_search_record_types($recordTypes)
{
    $searchRecordTypes['Exhibit'] = __('Gallery');
    $searchRecordTypes['ExhibitPage'] = __('Artist');
    return $searchRecordTypes;
}
```

See also

get_search_record_types

simple_search_default_uri

Usage

Value

Arguments

Examples

storage_path

Usage

Changes the path to where a file is stored. This can be a simpler solution than writing your own storage adapter class.

Value

string $path The path to the file, e.g. `files/image1.jpg`

Arguments

string type The type of path, e.g. `files`

string filename The filename, e.g. `image1.jpg`

Examples

Store files in different directories by extension.

```
class MyPlugin extends :php:class:`Omeka_Plugin_AbstractPlugin`
{

    protected $_filters = array('storage_path');

    public filterStoragePath($path, $args)
    {
        $explodedFileName = explode('.', $args['filename']);
        $extension = $explodedFileName[count($explodedFileName)-1];
        return 'files/' . $extension . '/' . $args['filename'];
    }
}
```

See Also

Omeka_Storage

system_info

Usage

Filter the list of information about the system, such as browser, PHP version, and more

Value

`array` $info The array of system information

Arguments

None

Examples

See Also

- *SystemInfoController::_getInfoArray*

theme_options

Usage

Filters the theme configuration options before they are returned by a call to *get_theme_option*.

Plugins can use this filter to modify settings for particular themes, or store and use alternative settings.

The options will be provided as a serialized string, so in order to modify the options, a plugin must unserialize() the array, make whatever changes are desired, then serialize() again before returning.

Value

`string` $themeOptions The set of all theme configuration options for a theme. This is a serialized array.

Arguments

`string` theme_name The name of the theme

Examples

Exhibit Builder adds theme settings on a per-exhibit basis.

```
class ExhibitBuilderPlugin extends :php:class:`Omeka_Plugin_AbstractPlugin`
{

    protected $_filters = array('theme_options');

    public filterThemeOptions($options, $args)
    {
        if (Omeka_Context::getInstance()->getRequest()->getModuleName() == 'exhibit-
↪builder' && function_exists('__v')) {
            if ($exhibit = exhibit_builder_get_current_exhibit()) {
                $exhibitThemeOptions = $exhibit->getThemeOptions();
            }
        }
        if (!empty($exhibitThemeOptions)) {
            return serialize($exhibitThemeOptions);
        }
        return $themeOptions;
    }
}
```

3.3.2 Database

3.3.3 File and Image Display

3.3.4 Records

Record Metadata

3.3.5 Search

3.3.6 Text

3.3.7 Views

Admin Views

Forms

Public Views

3.4 models

3.4.1 models/Api

Api_Collection

Package: *Record\Api*

class Api_Collection

extends *Omeka_Record_Api_AbstractRecordAdapter*

Api_Collection::**getRepresentation**(*Omeka_Record_AbstractRecord $record*)
Get the REST API representation for Collection

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Api_Collection::**setPostData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set POST data to a Collection.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Api_Collection::**setPutData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set PUT data to a Collection.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Api_Element

Package: *Record\Api*

class Api_Element

extends *Omeka_Record_Api_AbstractRecordAdapter*

Api_Element::**getRepresentation**(*Omeka_Record_AbstractRecord $record*)
Get the REST API representation for an element.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Api_Element::**setPostData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set POST data to an Element.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (Element) –

Api_Element::**setPutData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set PUT data to an Element.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (Element) –

Api_ElementSet

Package: *Record\Api*

class Api_ElementSet

extends *Omeka_Record_Api_AbstractRecordAdapter*

Api_ElementSet::**getRepresentation**(*Omeka_Record_AbstractRecord $record*)
Get the REST API representation for an element set.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Api_ElementSet::**setPostData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set data to an ElementSet.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (ElementSet) –

Api_File

Package: *Record\Api*

class Api_File

extends *Omeka_Record_Api_AbstractRecordAdapter*

Api_File::**getRepresentation**(*Omeka_Record_AbstractRecord $record*)
Get the REST API representation for a file.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Api_File::**setPostData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set POST data to a file.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Api_File::**setPutData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set PUT data to a file.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Api_Item

Package: *Record\Api*

class Api_Item

extends *Omeka_Record_Api_AbstractRecordAdapter*

Api_Item::**getRepresentation**(*Omeka_Record_AbstractRecord $record*)
Get the REST representation of an item.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Api_Item::**setPostData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set POST data to an item.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Api_Item::**setPutData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set PUT data to an item.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Api_ItemType

Package: *Record\Api*

class Api_ItemType

extends *Omeka_Record_Api_AbstractRecordAdapter*

Api_ItemType::**getRepresentation**(*Omeka_Record_AbstractRecord $record*)
Get the REST API representation for an item type.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Api_ItemType::**setPostData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set POST data to an item type.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (ItemType) –

Api_ItemType::**setPutData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set PUT data to an item type.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

- **$data** (ItemType) –

Api_Tag

Package: *Record\Api*

class Api_Tag

extends *Omeka_Record_Api_AbstractRecordAdapter*

Api_Tag::**getRepresentation**(*Omeka_Record_AbstractRecord $record*)
Get the REST representation of a tag.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Api_User

Package: *Record\Api*

class Api_User

extends *Omeka_Record_Api_AbstractRecordAdapter*

Api_User::**getRepresentation**(*Omeka_Record_AbstractRecord $record*)
Get the REST API representation for a user.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

3.4.2 models/Builder

Builder_Collection

Package: *Record\Builder*

class Builder_Collection

extends *Omeka_Record_Builder_AbstractBuilder*

Build a collection.

property Builder_Collection::$**_recordClass**
protected

property Builder_Collection::$**_settableProperties**
protected

Builder_Collection::**setElementTexts**(*$elementTexts*)
Set the element texts for the collection.

Parameters

- **$elementTexts** (*array*) –

Builder_Collection::**_addElementTexts**()
Add element texts to a record.

Builder_Collection::**_replaceElementTexts**()
Replace all the element texts for existing element texts.

Builder_Collection::**_beforeBuild**(*Omeka_Record_AbstractRecord $record*)
Add elements associated with the collection.

Parameters

- **$record** (Omeka_Record_AbstractRecord) – The collection record

Builder_ElementSet

Package: *Record\Builder*

class Builder_ElementSet

extends *Omeka_Record_Builder_AbstractBuilder*

Build an element set.

property Builder_ElementSet::$**_settableProperties**
protected

property Builder_ElementSet::$**_recordClass**
protected

Builder_ElementSet::**setElements**(*$elements*)
Set the elements to add to the element set.

Parameters

- **$elements** (*array*) –

Builder_ElementSet::**_beforeBuild**(*Omeka_Record_AbstractRecord $record*)
Add elements to be associated with the element set.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Builder_Item

Package: *Record\Builder*

class Builder_Item

extends *Omeka_Record_Builder_AbstractBuilder*

Build an item.

constant Builder_Item::**IS_PUBLIC**

property Builder_Item::$**_recordClass**
protected

property Builder_Item::$**_settableProperties**
protected

Builder_Item::**setElementTexts**(*$elementTexts*)
Set the element texts for the item.

Parameters

- **$elementTexts** (*array*) –

Builder_Item::**setFileMetadata**(*$fileMetadata*)
Set the file metadata for the item.

Parameters

- **$fileMetadata** (*array*) –

Builder_Item::**setRecordMetadata**(*$metadata*)
Overrides setRecordMetadata() to allow setting the item type by name instead of ID.

Parameters

- **$metadata** (*array*) –

Builder_Item::**_addElementTexts**()
Add element texts to a record.

Builder_Item::**_replaceElementTexts**()
Replace all the element texts for existing element texts.

Builder_Item::**_addTags**()
Add tags to an item (must exist in database).

Builder_Item::**addFiles**(*$transferStrategy*, *$files*, *$options = array()*)
Add files to an item.

'Url|Filesystem' => string|array If a string is given, this represents the source identifier of a single file (the URL representing the file, or the absolute file path, respectively). If an array is given, it assumes that each entry in the array must be either an array or a string. If it an array, there are several default keys that may be present: <ul> 'source' => Any identifier that is appropriate to the transfer strategy in use. For 'Url', this should be a valid URL. For 'Filesystem', it must be an absolute path to the source file to be transferred. 'name' => OPTIONAL The filename to give to the transferred file. This can be any arbitrary filename and will be listed as the original filename of the file. This will also be used to generate the archival filename for the file. If none is given, this defaults to using the getOriginalFileName() method of the transfer adapter. 'metadata' => OPTIONAL This could contain any metadata that needs to be associated with the file. This should be indexed in the same fashion as for items. See ActsAsElementText::addTextsByArray()

Parameters

- **$transferStrategy** (*string|Omeka_File_Ingest_AbstractIngest*) – This can either be one of the following strings denoting built-in transfer methods: 'Upload', 'Filesystem', 'Url' Or it could be an implemented Omeka_File_Ingest_AbstractIngest class.
- **$files** (*string|array*) – This can be a single string, an array of strings, or an array of arrays, depending on the parameters that are needed by the underlying strategy. Expected parameters for the built in strategies are as follows: <ul> 'Upload' => null|string If a string is given, it represents the POST parameter name containing the uploaded file(s). If null is given, all files in the POST will be ingested.
- **$options** (*array*) – OPTIONAL May contain the following flags where appropriate: <ul> 'ignore_invalid_files' => Do not throw exceptions when attempting to ingest invalid files. Instead, skip to the next file in the list and continue processing. False by default. (all except Upload). 'ignoreNoFile' => Ignore errors resulting from POSTs that do not contain uploaded files as expected (only for Upload).

Returns array Set of File records ingested. May be empty if no files were ingested.

Builder_Item::**_addIngestValidators**(*Omeka_File_Ingest_AbstractIngest $ingester*)
Add the default validators for ingested files.

The default validators are whitelists for file extensions and MIME types, and those lists can be configured via the admin settings form.

These default validators can be disabled by the 'disable_default_file_validation' flag in the settings panel.

Plugins can add/remove/modify validators via the 'file_ingest_validators' filter.

Parameters

- **$ingester** (Omeka_File_Ingest_AbstractIngest) –

Builder_Item::**_beforeBuild**(*Omeka_Record_AbstractRecord $record*)

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Builder_ItemType

Package: *Record\Builder*

class Builder_ItemType

extends *Omeka_Record_Builder_AbstractBuilder*

Build an item type.

property Builder_ItemType::$**_recordClass**
protected

property Builder_ItemType::$**_settableProperties**
protected

Builder_ItemType::**setElements**(*$elementMetadata*)
Set the elements that will be attached to the built ItemType record.

Parameters

- **$elementMetadata** (*array*) –

Builder_ItemType::**_beforeBuild**(*Omeka_Record_AbstractRecord $record*)
Add elements to be associated with the Item Type.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

3.4.3 Collection

Package: *Record*

class Collection

extends *Omeka_Record_AbstractRecord*

implements Zend_Acl_Resource_Interface

A collection and its metadata.

property Collection::$**public**
bool

Whether or not the collection is publicly accessible.

property Collection::$**featured**
bool

Whether or not the collection is featured.

property Collection::$**added**
string

Date the collection was added.

property Collection::$**modified**
string

Date the collection was last modified.

property Collection::$**owner_id**
int

ID for the User that created this collection.

property Collection::$**_related**
protected

Related records.

Collection::**_initializeMixins**()
Initialize the mixins.

Collection::**getProperty**(*$property*)
Get a property about this collection.

Valid properties for a Collection include: * (int) public * (int) featured * (string) added * (string) modified * (int) owner_id * (int) total_items

Parameters

- **$property** (*string*) – The property to get, always lowercase.

Returns mixed The value of the property

Collection::**totalItems**()
Get the total number of items in this collection.

Returns int

Collection::**setAddedBy**(*User $user*)
Set the user who added the collection.

Note that this is not to be confused with the collection's "contributors".

Parameters

- **$user** (User) –

Collection::**getResourceId**()
Required by Zend_Acl_Resource_Interface.

Identifies Collection records as relating to the Collections ACL resource.

Returns string

`Collection::hasContributor()`
Return whether the collection has at least 1 contributor element text.

Returns bool

`Collection::filterPostData(`*$post*`)`
Filter the POST data from the form.

Converts public/featured flags to booleans.

Parameters

- **$post** (*array*) –

Returns array

`Collection::_delete()`
All of the custom code for deleting an collection.

Delete the element texts for this record.

`Collection::_dissociateItems()`
Set items attached to this collection back to "no collection."

`Collection::beforeSave(`*$args*`)`
Before-save hook.

Fire the before-save element texts code.

Parameters

- **$args** –

`Collection::afterSave(`*$args*`)`
After-save hook.

Handle public/private status for search.

Parameters

- **$args** –

`Collection::getFile()`
Get a representative file for this Collection.

Returns File|null

3.4.4 Element

Package: *Record*

class Element

extends *Omeka_Record_AbstractRecord*

implements `Zend_Acl_Resource_Interface`

A metadata element within an element set or item type.

property `Element::$element_set_id`
int

ID of the ElementSet this Element belongs to.

property Element::$**order**
int

This Element's order within the parent ElementSet.

property Element::$**name**
string

A human-readable name

property Element::$**description**
string

A human-readable description

property Element::$**comment**
string

A user-generated comment

Element::**setElementSet**(*$elementSetName*)
Set the parent ElementSet by name.

Parameters

- **$elementSetName** (*string*) –

Element::**getElementSet**()
Return the parent ElementSet object for this element.

Returns ElementSet|null

Element::**setOrder**(*$order*)
Set the order of the element within its element set.

Parameters

- **$order** (*int*) –

Element::**setName**(*$name*)
Set the Element's name.

Parameters

- **$name** (*string*) –

Element::**setDescription**(*$description*)
Set the Element's description.

Parameters

- **$description** (*string*) –

Element::**setComment**(*$comment*)
Set the Element's comment.

Parameters

- **$comment** (*string*) –

Element::**setArray**(*$data*)
Set the data for the Element in bulk.

- name
- description
- comment

- order
- element_set_id
- element_set

Parameters

- **$data** (*array|string*) – If string, the name of the element. Otherwise, array of metadata for the element. The array may contain the following keys:

`Element::_validate()`
Validate the element prior to being saved.

Checks the following criteria:

- Name is not empty.
- Name does not already exist within the given element set.

`Element::_delete()`
Delete associated records when deleting the Element.

Cascade delete to all element texts and item type assignments associated with the element.

`Element::_getElementSetId(`*$elementSetName*`)`
Get an element set ID from a name.

Parameters

- **$elementSetName** –

Returns int

`Element::_nameIsInSet(`*$elementName*, *$elementSetId*`)`
Calculate whether the element's name already belongs to the current set.

Parameters

- **$elementName** –
- **$elementSetId** –

Returns bool

`Element::getResourceId()`
Identify Element records as relating to the Elements ACL resource.

Required by Zend_Acl_Resource_Interface.

Returns string

3.4.5 ElementSet

Package: *Record*

class ElementSet

extends *Omeka_Record_AbstractRecord*

implements `Zend_Acl_Resource_Interface`

An element set and its metadata.

constant ElementSet::**ITEM_TYPE_NAME**
The name of the item type element set.

This is used wherever it is important to distinguish this special case element set from others.

property ElementSet::$**record_type**
string

Type of record this set applies to.

property ElementSet::$**name**
string

Name for the element set.

property ElementSet::$**description**
string

Description for the element set.

property ElementSet::$**_elementsToSave**
protected array

Child Element records to save when saving this set.

ElementSet::**getElements**()
Get the Elements that are in this set.

Returns array

ElementSet::**addElements**(*$elements*)
Add elements to the element set.

Parameters

- **$elements** (*array*) –

ElementSet::**_buildElementRecord**(*$options*)
Create a new Element record with the given data.

Parameters

- **$options** (*array*) – Data to set on the Element.

Returns Element

ElementSet::**afterSave**(*$args*)
After-save hook.

Save the $_elementsToSave and set their orders.

Parameters

- **$args** –

ElementSet::**_delete**()
Delete all the elements associated with an element set.

ElementSet::**_getNextElementOrder**()
Get an order value to place an Element at the end of this set.

Returns int

ElementSet::**_validate**()
Validate the element set.

Tests that name is non-empty and unique.

ElementSet::**getResourceId**()
Identify ElementSet records as relating to the ElementSets ACL resource.

Required by Zend_Acl_Resource_Interface.

Returns string

3.4.6 ElementText

Package: *Record*

class ElementText

extends *Omeka_Record_AbstractRecord*

An element text and its metadata.

property ElementText::$**record_id**
int

ID of the associated record.

property ElementText::$**record_type**
string

Type of the associated record.

property ElementText::$**element_id**
int

ID of this text's Element.

property ElementText::$**html**
int

Whether this text is HTML.

property ElementText::$**text**
string

The text itself.

ElementText::**_validate**()
Validate the element text prior to saving.

Test for a positive record_id and element_id, and a non-empty record_type.

ElementText::**__toString**()
Use the actual text when printing an ElementText as a string.

Returns string

ElementText::**setText**(*$text*)
Set the text.

Parameters

- **$text** (*string*) –

ElementText::**getText**()
Get the text.

Returns string

ElementText::**isHtml**()
Get whether this text is HTML.

Returns bool

3.4.7 File

Package: *Record*

class File

extends *Omeka_Record_AbstractRecord*

implements Zend_Acl_Resource_Interface

A file and its metadata.

constant File::**DISABLE_DEFAULT_VALIDATION_OPTION**
Option name for whether the file validation is disabled.

constant File::**DERIVATIVE_EXT**
File extension for all image derivatives.

property File::$**item_id**
int

ID of the Item this File belongs to.

property File::$**order**
int

Relative order of this File within the parent Item.

property File::$**filename**
string

Current filename, as stored.

property File::$**original_filename**
string

Original filename, as uploaded.

property File::$**size**
int

Size of the file, in bytes.

property File::$**authentication**
string

MD5 hash of the file.

property File::$**mime_type**
string

MIME type of the file.

property File::$**type_os**
string

Longer description of the file's type.

property File::$**has_derivative_image**
int

Whether the file has derivative images.

property File::$**added**
string

Date the file was added.

property File::$**modified**
string

Date the file was last modified.

property File::$**stored**
int

Whether the file has been moved to storage.

property File::$**metadata**
array

Embedded metadata from the file.

File::**getProperty**(*$property*)
Get a property or special value of this record.

Parameters

- **$property** (*string*) –

Returns mixed

File::**_initializeMixins**()
Initialize the mixins.

File::**filterPostData**(*$post*)
Unset immutable properties from $_POST.

Parameters

- **$post** (*array*) –

Returns array

File::**beforeSave**(*$args*)
Before-save hook.

Parameters

- **$args** (*array*) –

File::**afterSave**(*$args*)
After-save hook.

Parameters

- **$args** (*array*) –

File::**getItem**()
Get the Item this file belongs to.

Returns Item

File::**getPath**(*$type = 'original'*)
Get a system path for this file.

Local paths are only available before the file is stored.

Parameters

- **$type** (*string*) –

Returns string

File::**getWebPath**(*$type = 'original'*)
Get a web path for this file.

Parameters

- **$type** (*string*) –

Returns string

File::**getDerivativeFilename**()
Get the filename for this file's derivative images.

Returns string

File::**hasThumbnail**()
Determine whether this file has a thumbnail image.

Returns bool

File::**hasFullsize**()
Determine whether this record has a fullsize image.

This is an alias for hasThumbnail().

Returns bool

File::**getExtension**()
Get the original file's extension.

Returns string

File::**setDefaults**(*$filepath*, *$options = array()*)
Set the default values that will be stored for this record in the 'files' table.

Parameters

- **$filepath** –
- **$options** –

File::**unlinkFile**()
Unlink the file and file derivatives belonging to this record.

File::**_delete**()
Perform any further deletion when deleting this record.

File::**createDerivatives**()
Create derivatives of the original file.

File::**extractMetadata**()
Extract ID3 metadata associated with the file.

Returns bool Whether getID3 was able to read the file.

File::**_getId3**()
Read the file's embedded metadata with the getID3 library.

Returns getID3|bool Returns getID3 object, or false if there was an exception.

File::**storeFiles**()
Store the files belonging to this record.

File::**getStoragePath**(*$type = 'original'*)
Get a storage path for the file.

Parameters

- **$type** (*string*) –

Returns string

File::**setStorage**(*$storage*)
Set the storage object.

Parameters

- **$storage** (Omeka_Storage) –

File::**getStorage**()
Get the storage object.

Returns Omeka_Storage

File::**getResourceId**()
Get the ACL resource ID for the record.

File records are 'Files' resources.

Returns string

File::**isOwnedBy**(*$user*)
Return whether this file is owned by the given user.

Proxies to the Item's isOwnedBy.

Parameters

- **$user** (User) –

Returns bool

File::**getFile**()
Return the representative File for the record (this File itself).

Returns File

3.4.8 models/Installer

Installer_Default

Package: *Install*

class Installer_Default

implements *Installer_InstallerInterface*

The default installer, which extracts values from the installer form to create the default Omeka installation.

Installer_Default::**__construct**(*Omeka_Db $db*)
Constructor.

Parameters

- **$db** (Omeka_Db) –

Installer_Default::**setForm**(*Zend_Form $form*)
Set the form from which to extract data for the installer.

Parameters

- **$form** (*Zend_Form*) –

Installer_Default::**getDb**()

Installer_Default::**install**()

Installer_Default::**_getValue**(*$fieldName*)

Parameters

- **$fieldName** –

Installer_Default::**_createSchema**()

Installer_Default::**_createUser**()

Installer_Default::**_setupMigrations**()

Installer_Default::**_addOptions**()

Installer_Default::**isInstalled**()

Installer_Exception

Package: *Install*

class Installer_Exception

extends Exception

implements Throwable

property Installer_Exception::$**message**
protected

property Installer_Exception::$**code**
protected

property Installer_Exception::$**file**
protected

property Installer_Exception::$**line**
protected

Installer_Exception::**__clone**()

Installer_Exception::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Installer_Exception::**__wakeup**()

Installer_Exception::**getMessage**()

Installer_Exception::**getCode**()

Installer_Exception::**getFile**()

Installer_Exception::**getLine**()

Installer_Exception::**getTrace**()

Installer_Exception::**getPrevious**()

Installer_Exception::**getTraceAsString**()

Installer_Exception::**__toString**()

Installer_InstallerInterface

Package: *Install*

interface Installer_InstallerInterface

Interface for creating different installers for Omeka.

__construct(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

install()

isInstalled()

Installer_Requirements

Package: *Install*

class Installer_Requirements

check()

getErrorMessages()

getWarningMessages()

hasError()

hasWarning()

setDbAdapter(*Zend_Db_Adapter_Abstract $db*)

Parameters

- **$db** (*Zend_Db_Adapter_Abstract*) –

setStorage(*Omeka_Storage $storage*)

Parameters

- **$storage** (Omeka_Storage) –

_checkPhpVersionIsValid()

_checkPhpExtensionsAreAvailable()

_checkMysqliIsAvailable()

_checkMysqlVersionIsValid()

_checkHtaccessFilesExist()

_checkRegisterGlobalsIsOff()

_checkExifModuleIsLoaded()

_checkFileStorageSetup()

_checkFileinfoIsLoaded()

models/Installer/Task

Installer_Task_Exception

Package: *Install*

class Installer_Task_Exception

extends *Installer_Exception*

Class for exceptions thrown by installer tasks that failed.

Installer_Task_Migrations

Package: *Install*

class Installer_Task_Migrations

implements *Installer_TaskInterface*

Installer_Task_Migrations::**install**(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

Installer_Task_Options

Package: *Install*

class Installer_Task_Options

implements *Installer_TaskInterface*

Installer task for inserting options into the options table.

Installer_Task_Options::**setOptions**(*$options*)

Set the key value pairs that will correspond to database options.

Parameters

- **$options** –

Installer_Task_Options::**install**(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

Installer_Task_Schema

Package: *Install*

class Installer_Task_Schema

implements *Installer_TaskInterface*

Load the database schema for an Omeka installation.

Schema should be defined in an SQL file.

`Installer_Task_Schema::`**`addTable`**(*$tableName*, *$sqlFilePath*)
Add an SQL table to the list of tables to create.

Parameters

- **`$tableName`** (*string*) –
- **`$sqlFilePath`** (*string*) –

`Installer_Task_Schema::`**`addTables`**(*$tables*)
Add a set of SQL tables to the list.

Parameters

- **`$tables`** (*array*) –

`Installer_Task_Schema::`**`setTables`**(*$tables*)
Set the list of SQL tables.

Parameters

- **`$tables`** (*array*) –

`Installer_Task_Schema::`**`removeTable`**(*$tableName*)
Remove an SQL table from the list.

Parameters

- **`$tableName`** (*string*) –

`Installer_Task_Schema::`**`getTables`**()
Retrieve list of tables being installed.

`Installer_Task_Schema::`**`useDefaultTables`**()
Add all tables corresponding to the default Omeka installation.

`Installer_Task_Schema::`**`install`**(*Omeka_Db $db*)

Parameters

- **`$db`** (`Omeka_Db`) –

Installer_Task_User

Package: *InstallNTask*

`class Installer_Task_User`

implements *`Installer_TaskInterface`*

Create a default user for an Omeka installation.

`Installer_Task_User::`**`setUsername`**(*$username*)

Parameters

- **`$username`** –

`Installer_Task_User::`**`setPassword`**(*$password*)

Parameters

- **`$password`** –

Installer_Task_User::**setEmail**(*$email*)

Parameters

- **$email** –

Installer_Task_User::**setName**(*$name*)

Parameters

- **$name** –

Installer_Task_User::**setIsActive**(*$active*)

Parameters

- **$active** –

Installer_Task_User::**setRole**(*$role*)

Parameters

- **$role** –

Installer_Task_User::**install**(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

Installer_TaskInterface

Package: *Install*

interface Installer_TaskInterface

Interface for Installer tasks.

install(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

Installer_Test

Package: *Install*

class Installer_Test

extends *Installer_Default*

implements *Installer_InstallerInterface*

Installer for test cases that require database access.

Installer_Test::**_getValue**(*$fieldName*)

Overridden to retrieve values only from a predefined array.

Parameters

- **$fieldName** –

Installer_Test::**install**()

Installer_Test::**addItem**(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

Installer_Test::**__construct**(*Omeka_Db $db*)
Constructor.

Parameters

- **$db** (Omeka_Db) –

Installer_Test::**setForm**(*Zend_Form $form*)
Set the form from which to extract data for the installer.

Parameters

- **$form** (*Zend_Form*) –

Installer_Test::**getDb**()

Installer_Test::**_createSchema**()

Installer_Test::**_createUser**()

Installer_Test::**_setupMigrations**()

Installer_Test::**_addOptions**()

Installer_Test::**isInstalled**()

3.4.9 Item

Package: *Record*

class Item

extends *Omeka_Record_AbstractRecord*

implements Zend_Acl_Resource_Interface

An item and its metadata.

property Item::$**item_type_id**
int

The ID for this Item's ItemType, if any.

property Item::$**collection_id**
int

The ID for this Item's Collection, if any.

property Item::$**featured**
int

Whether this Item is featured.

property Item::$**public**
int

Whether this Item is publicly accessible.

property Item::$**added**
string

The date this Item was added.

property Item::$**modified**
string

The date this Item was last modified.

property Item::$**owner_id**
int

ID of the User who created this Item.

property Item::$**_related**
protected array

Records related to an Item.

Item::**_initializeMixins**()
Initialize the mixins.

Item::**getCollection**()
Get this Item's Collection, if any.

Returns Collection|null

Item::**getItemType**()
Get the ItemType record associated with this Item.

Returns ItemType|null

Item::**getFiles**()
Get the set of File records associated with this Item.

Returns array

Item::**getFile**(*$index = 0*)
Get a single File associated with this Item, by index.

The default is to get the first file.

Parameters

- **$index** (*int*) –

Returns File

Item::**getItemTypeElements**()
Get a set of Elements associated with this Item's ItemType.

Each one of the Element records that is retrieved should contain all the element text values associated with it.

Returns array Element records that are associated with the item type of the item. This array will be empty if the item does not have an associated type.

Item::**getProperty**(*$property*)
Get a property for display.

Parameters

- **$property** (*string*) –

Returns mixed

Item::**beforeSave**(*$args*)
Before-save hook.

Parameters

- **$args** (*array*) –

Item::**afterSave**(*$args*)
After-save hook.

Parameters

- **$args** (*array*) –

Item::**_delete**()
All of the custom code for deleting an item.

Item::**_deleteFiles**(*$fileIds = array()*)
Delete files associated with the item.

If the IDs of specific files are passed in, this will delete only those files (e.g. form submission). Otherwise, it will delete all files associated with the item.

Parameters

- **$fileIds** (*array*) – Optional

Item::**_uploadFiles**()
Iterate through the $_FILES array for files that have been uploaded to Omeka and attach each of those files to this Item.

Item::**saveFiles**()
Save all the files that have been associated with this item.

Item::**filterPostData**(*$post*)
Filter post data from form submissions.

Parameters

- **$post** –

Returns array Clean post data

Item::**fileCount**()
Get the number of files assigned to this item.

Returns int

Item::**previous**()
Get the previous Item in the database.

Returns Item|false

Item::**next**()
Get the next Item in the database.

Returns Item|false

Item::**hasThumbnail**()
Determine whether or not the Item has a File with a thumbnail image (or any derivative image).

Returns bool

Item::**getCitation**()
Return a valid citation for this item.

Generally follows Chicago Manual of Style note format for webpages. Implementers can use the item_citation filter to return a customized citation.

Returns string

Item::**addFile**(*File $file*)
Associate an unsaved (new) File record with this Item.

These File records will not be persisted in the database until the item is saved or saveFiles() is invoked.

Parameters

- **$file** (File) –

Item::**getResourceId**()
Identify Item records as relating to the Items ACL resource.

Required by Zend_Acl_Resource_Interface.

Returns string

Item::**_validate**()
Validate this item.

3.4.10 ItemType

Package: *Record*

class ItemType

extends *Omeka_Record_AbstractRecord*

implements Zend_Acl_Resource_Interface

An item type and its metadata.

Item types are like specialized element sets that only apply to Items and which can vary between items.

constant ItemType::**ITEM_TYPE_NAME_MIN_CHARACTERS**
Minimum length of an ItemType name.

constant ItemType::**ITEM_TYPE_NAME_MAX_CHARACTERS**
Maximum length of an ItemType name.

property ItemType::$**name**
string

Name of this ItemType.

property ItemType::$**description**
string

Description for this ItemType.

property ItemType::$**_related**
protected array

Records related to an ItemType.

ItemType::**getElements**()
Get an array of element objects associated with this item type.

Returns array All the Element objects associated with this item type.

ItemType::**getItems**(*$count = 10*, *$recent = true*)
Get an array of Items that have this item type.

Parameters

- **$count** (*int*) – The maximum number of items to return.

- **$recent** (*bool*) – Whether the most recent items should be chosen.

Returns array The Item objects associated with the item type.

ItemType::**_validate**()
Validate this ItemType.

The name field must be between 1 and 255 characters and must be unique.

ItemType::**filterPostData**(*$post*)
Filter incoming POST data from ItemType form.

Parameters

- **$post** –

ItemType::**_delete**()
Clean up the associated records for this Item Type.

Delete all the ItemTypesElements rows joined to this type, and remove the type ID from any associated items.

ItemType::**afterSave**(*$args*)
After-save hook.

Save Element records that are associated with this Item Type.

Parameters

- **$args** –

ItemType::**reorderElements**(*$elementOrderingArray*)
Reorder the elements for this type.

This extracts the ordering for the elements from the form's POST, then uses the given ordering to reorder each join record from item_types_elements into a new ordering, which is then saved.

Parameters

- **$elementOrderingArray** (*array*) – An array of element_id => order pairs

ItemType::**addElements**(*$elements = array()*)
Add a set of elements to the Item Type.

Parameters

- **$elements** (*array*) – Either an array of elements or an array of metadata, where each entry corresponds to a new element to add to the item type. If an element exists with the same id, it will replace the old element with the new element.

ItemType::**addElementById**(*$elementId*)
Add a new element to the item type, giving the Element by its ID.

Parameters

- **$elementId** –

ItemType::**removeElements**(*$elements*)
Remove an array of Elements from this item type

The elements will not be removed until the object is saved.

Parameters

- **$elements** (*array*) – An array of Element objects or element id strings

ItemType::**removeElement**(*$element*)
Remove a single Element from this item type.

The element will not be removed until the object is saved.

Parameters

- **$element** (*Element|string*) – The element object or the element id.

ItemType::**_removeElement**(*$element*)
Immediately remove a single Element from this item type.

Parameters

- **$element** (*Element|string*) –

ItemType::**hasElement**(*$element*)
Determine whether this ItemType has a particular element.

This method does not handle elements that were added or removed without saving the item type object.

Parameters

- **$element** (*Element|string*) – The element object or the element id.

Returns bool

ItemType::**totalItems**()
Get the total number of items that have this item type.

Returns int The total number of items that have this item type.

ItemType::**getItemTypeElementSet**()
Get the 'Item Type' element set.

Returns ElementSet

ItemType::**getResourceId**()
Identify ItemType records as relating to the ItemTypes ACL resource.

Required by Zend_Acl_Resource_Interface.

Returns string

ItemType::**_dissociateItems**()
Set items attached to this item type back to null.

3.4.11 ItemTypesElements

Package: *Record*

class ItemTypesElements

extends *Omeka_Record_AbstractRecord*

Record linking an Element with an ItemType.

property ItemTypesElements::$**item_type_id**
int

ID for the ItemType being linked.

property ItemTypesElements::$**element_id**
int

ID for the Element being linked.

property ItemTypesElements::$**order**
int

Relative order of the Element within the ItemType.

3.4.12 models/Job

Job_FileProcessUpload

Package: *Job*

class Job_FileProcessUpload

extends *Omeka_Job_AbstractJob*

Job_FileProcessUpload::**perform**()

Job_FileProcessUpload::**_getFile**()

Job_ItemBatchEdit

Package: *Job*

class Job_ItemBatchEdit

extends *Omeka_Job_AbstractJob*

Job_ItemBatchEdit::**perform**()

Job_ItemBatchEdit::**_getItem**(*$id*)

Parameters

- **$id** –

Job_ItemBatchEditAll

Package: *Job*

class Job_ItemBatchEditAll

extends *Omeka_Job_AbstractJob*

property Job_ItemBatchEditAll::$**_table**
protected

property Job_ItemBatchEditAll::$**_aclHelper**
protected

Job_ItemBatchEditAll::**perform**()

Job_ItemBatchEditAll::**_performItem**(*$itemId*)
Check if the item can be processed, then process it if possible.

Parameters

- **$itemId**(*int*) –

Returns bool

Job_ItemBatchEditAll::**_logProcessedItem**(*$message*, *$priority*)
Log a message about the current processed item.

Parameters

- **$message** (*string*) –
- **$priority** (*string*) –

Job_SearchTextIndex

Package: *Job*

class Job_SearchTextIndex

extends *Omeka_Job_AbstractJob*

Job_SearchTextIndex::**perform**()
Bulk index all valid records.

3.4.13 Key

Package: *Record*

class Key

extends *Omeka_Record_AbstractRecord*

property Key::$**user_id**

property Key::$**label**

property Key::$**key**

property Key::$**ip**

property Key::$**accessed**

3.4.14 models/Mixin

Mixin_ElementText

Package: *Record\Mixin*

class Mixin_ElementText

extends *Omeka_Record_Mixin_AbstractMixin*

Record mixin class for associating elements, element texts and their corresponding behaviors to a record.

property Mixin_ElementText::$**_textsByNaturalOrder**
protected array

ElementText records stored in the order they were retrieved from the database.

property Mixin_ElementText::$**_textsByElementId**
protected array

ElementText records indexed by the element_id.

property Mixin_ElementText::$**_elementsBySet**
protected array

Element records indexed by set name and element name, so it looks like:

$elements['Dublin Core']['Title'] = Element instance;

property Mixin_ElementText::$**_elementsById**
protected array

Element records indexed by ID.

property Mixin_ElementText::$**_elementsOnForm**
protected array

List of elements that were output on the form. This can be used to determine the DELETE SQL to use to reset the elements when saving the form.

property Mixin_ElementText::$**_textsToSave**
protected array

Set of ElementText records to save when submitting the form. These will only be saved to the database if they successfully validate.

property Mixin_ElementText::$**_recordsAreLoaded**
protected bool

Whether the elements and texts have been loaded yet.

property Mixin_ElementText::$**_replaceElementTexts**
protected bool

Flag to indicate whether elements added to this save will replace existing element texts, not add them.

Mixin_ElementText::**afterSave**(*$args*)
Omeka_Record_AbstractRecord callback for afterSave. Saves the ElementText records once the associated record is saved. Adds the record's element texts to the search text.

Parameters

- **$args** –

Mixin_ElementText::**_getDb**()
Get the database object from the associated record.

Returns Omeka_Db

Mixin_ElementText::**_getRecordType**()
Get the class name of the associated record (Item, File, etc.).

Returns string Type of record

Mixin_ElementText::**loadElementsAndTexts**(*$reload = false*)
Load all the ElementText records for the given record (Item, File, etc.). These will be indexed by [element_id].

Also load all the Element records and index those by their name and set name.

Parameters

- **$reload** (*bool*) – Whether or not reload all the data that was previously loaded.

Mixin_ElementText::**_loadElements**(*$reload = false*)

Parameters

- **$reload** –

Mixin_ElementText::**_getElementTextRecords**()
Retrieve all of the ElementText records for the given record.

Returns array Set of ElementText records for the record.

Mixin_ElementText::**_getElementRecords**()
Retrieve all of the Element records for the given record.

Returns array All Elements that apply to the record's type.

Mixin_ElementText::**getElementTextsByRecord**(*$element*)
Retrieve all of the record's ElementTexts for the given Element.

Parameters

- **$element** (Element) –

Returns array Set of ElementText records.

Mixin_ElementText::**getElementTexts**(*$elementSetName*, *$elementName*)
Retrieve all of the record's ElementTexts for the given element name and element set name.

Parameters

- **$elementSetName** (*string*) – Element set name
- **$elementName** (*string*) – Element name

Returns array Set of ElementText records.

Mixin_ElementText::**getAllElementTexts**()
Retrieve all of the record's ElementTexts, in order.

Returns array Set of ElementText records.

Mixin_ElementText::**getAllElementTextsByElement**()
Retrieve all of the record's ElementTexts, indexed by element ID.

Returns array Set of ElementText records, indexed by element_id.

Mixin_ElementText::**getElementsBySetName**(*$elementSetName*)
Retrieve the Element records for the given ElementSet.

Parameters

- **$elementSetName** –

Returns array Set of Element records

Mixin_ElementText::**getAllElements**()
Retrieve ALL the Element records for the object, organized by ElementSet. For example, $elements['Dublin Core'] = array(Element instance, Element instance, ...)

Returns array Set of Element records

Mixin_ElementText::**getElement**(*$elementSetName*, *$elementName*)
Retrieve the Element record corresponding to the given element name and element set name.

Parameters

- **$elementSetName** (*string*) –
- **$elementName** (*string*) –

Returns Element

Mixin_ElementText::**getElementById**(*$elementId*)
Retrieve the Element with the given ID.

Parameters

- **$elementId** (*int*) –

Returns Element

Mixin_ElementText::**_indexTextsByElementId**(*$textRecords*)
Index a set of ElementTexts based on element ID.

Parameters

- **$textRecords** (*array*) – Set of ElementText records

Returns array The provided ElementTexts, indexed by element ID.

Mixin_ElementText::**_indexElementsBySet**(*$elementRecords*)
Index a set of Elements based on their name. The result is a doubly associative array, with the first key being element set name and the second being element name.

i.e., $indexed['Dublin Core']['Creator'] = Element instance

Parameters

- **$elementRecords** (*array*) – Set of Element records

Returns array The provided Elements, indexed as described

Mixin_ElementText::**_indexElementsById**(*$elementRecords*)
Indexes the elements returned by element ID.

Parameters

- **$elementRecords** –

Returns array

Mixin_ElementText::**addTextForElement**(*$element*, *$elementText*, *$isHtml = false*)
Add a string of text for an element.

Creates a new ElementText record, populates it with the specified text value and assigns it to the element.

saveElementTexts() must be called after this in order to save the element texts to the database.

Parameters

- **$element** (Element) – Element which text should be created for
- **$elementText** (*string*) – Text to be added
- **$isHtml** (*bool*) – Whether the text to add is HTML

Mixin_ElementText::**addElementTextsByArray**(*$elementTexts*)
Add element texts for a record based on a formatted array of values. The array must be formatted as follows:

<code> 'Element Set Name' => array('Element Name' => array(array('text' => 'foo', 'html' => false))) </code>

Since 1.4, the array can also be formatted thusly:

<code> array(array('element_id' => 1, 'text' => 'foo', 'html' => false)) </code>

Parameters

- **$elementTexts** (*array*) –

Mixin_ElementText::**_addTextsByElementName**(*$elementTexts*)

Parameters

- **$elementTexts** –

Mixin_ElementText::**_addTextsByElementId**(*$texts*)

Parameters

- **$texts** –

Mixin_ElementText::**beforeSaveElements**(*$post*)

The application flow is thus:

1) Build ElementText objects from the POST.

2) Validate the ElementText objects and assign error messages if necessary. 3) After the item saves correctly, delete all the ElementText records for the Item. 4) Save the new ElementText objects to the database.

Parameters

- **$post** –

Mixin_ElementText::**_getElementTextsToSaveFromPost**(*$post*)

The POST should have a key called "Elements" that contains an array that is keyed to an element's ID. That array should contain all the text values for that element. For example:

<code>

array('Elements' => array('50' => array(array('text' => 'Foobar', //element id 50, e.g. DC:Title 'html' => 0)), '41' => array(array('text' => ' Baz baz baz ', //element id 41, e.g. DC:Description 'html' => 1))))

</code>

Parameters

- **$post** –

Mixin_ElementText::**getTextStringFromFormPost**(*$postArray*, *$element*)

Retrieve a text string for an element from POSTed form data.

Parameters

- **$postArray** –
- **$element** –

Returns string

Mixin_ElementText::**_validateElementTexts**()

Validate all the elements one by one. This is potentially a lot slower than batch processing the form, but it gives the added bonus of being able to encapsulate the logic for validation of Elements.

Mixin_ElementText::**_elementTextIsValid**(*$elementTextRecord*)

Return whether the given ElementText record is valid.

Parameters

- **$elementTextRecord** (ElementText) –

Returns bool

Mixin_ElementText::**setReplaceElementTexts**(*$replaceElementTexts = true*)
Set the flag to indicate whether elements added to this save will replace existing element texts, not add them.

Parameters

- **$replaceElementTexts** –

Mixin_ElementText::**saveElementTexts**()
Save all ElementText records that were associated with a record.

Typically called in the afterSave() hook for a record.

Mixin_ElementText::**deleteElementTextsByElementId**(*$elementIdArray = array()*)
Delete all the element texts for element_id's that have been provided.

Parameters

- **$elementIdArray** –

Returns bool

Mixin_ElementText::**deleteElementTexts**()
Delete all the element texts assigned to the current record ID.

Returns bool

Mixin_ElementText::**hasElementText**(*$elementSetName*, *$elementName*)
Returns whether or not the record has at least 1 element text

Parameters

- **$elementSetName** (*string*) – Element set name
- **$elementName** (*string*) – Element name

Returns bool

Mixin_ElementText::**getElementTextCount**(*$elementSetName*, *$elementName*)
Returns the number of element texts for the record

Parameters

- **$elementSetName** (*string*) – Element set name
- **$elementName** (*string*) – Element name

Returns bool

Mixin_ElementText::**getDisplayTitle**(*$default = null*)
Return the title of this record for display/interface purposes

If no title is present or the title contains no text, returns the passed $default value. If no $default is given, returns the translated string [Untitled].

Parameters

- **$default** –

Returns string Raw (unescaped) title string for the record.

Mixin_Owner

Package: *Record\Mixin*

class Mixin_Owner

extends *Omeka_Record_Mixin_AbstractMixin*

Mixin for models that have a user that is their "owner."

property Mixin_Owner::$**_record**
protected

property Mixin_Owner::$**_column**
protected

Mixin_Owner::**__construct**(*$record*, *$column = 'owner_id'*)

Parameters

- **$record** –
- **$column** –

Mixin_Owner::**beforeSave**(*$args*)

Parameters

- **$args** –

Mixin_Owner::**setOwner**(*User $user*)
Set the record's owner.

Parameters

- **$user** (User) –

Mixin_Owner::**getOwner**()
Get the record's owner.

If the record has no user, this method returns null.

Returns User|null

Mixin_Owner::**isOwnedBy**(*User $user*)
Check if the given User owns this record.

Parameters

- **$user** (User) –

Returns bool

Mixin_PublicFeatured

Package: *Record\Mixin*

class Mixin_PublicFeatured

extends *Omeka_Record_Mixin_AbstractMixin*

Adds default behavior associated with the 'public' and 'featured' flags.

Mixin_PublicFeatured::**__construct**(*$record*)
Constructor

Parameters

- **$record** (Omeka_Record_AbstractRecord) – The underlying record

Mixin_PublicFeatured::**isPublic**()
Returns whether the record is public or not.

Returns bool

Mixin_PublicFeatured::**setPublic**(*$flag*)
Sets whether the record is public or not.

Parameters

- **$flag** (*bool*) – Whether the record is public or not

Mixin_PublicFeatured::**isFeatured**()
Returns whether the record is featured or not.

Returns bool

Mixin_PublicFeatured::**setFeatured**(*$flag*)
Sets whether the record is featured or not.

Parameters

- **$flag** (*bool*) – Whether the record is featured or not

Mixin_PublicFeatured::**beforeSave**(*$args*)

Parameters

- **$args** –

Mixin_PublicFeatured::**afterSave**(*$args*)

Parameters

- **$args** –

Mixin_PublicFeatured::**_fireHook**(*$state*, *$flag*)
Fires a hooks like 'make_item_public', 'make_collection_not_featured', etc.

Parameters

- **$state** (*string*) – Currently, 'public' or 'featured'
- **$flag** (*bool*) –

Mixin_PublicFeatured::**_getHookName**(*$state*, *$flag*)
Retrieve formatted hooks like 'make_item_public', 'make_collection_not_featured', etc.

Parameters

- **$state** (*string*) – Currently, 'public' or 'featured'
- **$flag** (*bool*) –

Returns string The hook name

Mixin_Search

Package: *Record\Mixin*

class Mixin_Search

extends *Omeka_Record_Mixin_AbstractMixin*

Make an Omeka record fulltext searchable.

Any class that extends Omeka_Record_AbstractRecord can be made searchable by pushing an instance of this mixin into Omeka_Record::$_mixins during Omeka_Record::_initializeMixins(). It must be pushed after all mixins that can add search text–for example, after ElementText.

The record type must also be registered using the search_record_types filter in order for the records to be searchable.

This mixin leverages the Omeka_Record_AbstractRecord::afterSave() and Omeka_Record_Mixin_AbstractMixin::afterSave() callbacks, so note their order of execution. Records that initialize ActsAsElementText will automatically add their element texts to the search text.

property Mixin_Search::$**_text**
protected

property Mixin_Search::$**_title**
protected

property Mixin_Search::$**_public**
protected

Mixin_Search::**__construct**(*$record*)

Parameters

- **$record** –

Mixin_Search::**addSearchText**(*$text*)
Add search text to this record.

This method is meant to be called during afterSave().

Parameters

- **$text** (*string*) –

Mixin_Search::**setSearchTextTitle**(*$title*)
Add a title to this record.

This method is meant to be called during afterSave().

Parameters

- **$title** (*string*) –

Mixin_Search::**setSearchTextPrivate**()
Mark this record's search text as not public.

This method is meant to be called during afterSave().

Mixin_Search::**afterSave**(*$args*)
Save the accumulated search text to the database.

Parameters

- **$args** –

Mixin_Search::**afterDelete**()
Delete this record's search text after it has been deleted.

Mixin_Search::**saveSearchText**(*$recordType*, *$recordId*, *$text*, *$title*, *$public = 1*)
Save a search text row.

Call this statically only when necessary. Used primarily when in a record that does not implement Mixin_Search but contains text that is needed for another record's search text. For example, when saving a child record that contains search text that should be saved to its parent record.

Parameters

- **$recordType** (*string*) –
- **$recordId** (*int*) –
- **$text** (*string*) –
- **$title** (*string*) –
- **$public** (*int*) –

Mixin_Tag

Package: *Record\Mixin*

class Mixin_Tag

extends *Omeka_Record_Mixin_AbstractMixin*

Mixin_Tag::**__construct**(*Omeka_Record_AbstractRecord $record*)

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Mixin_Tag::**beforeDelete**()
Fires whenever deleting a record that is taggable This will actually delete all the references to a specific tag for a specific record

Mixin_Tag::**afterSave**(*$args*)

Parameters

- **$args** –

Mixin_Tag::**deleteTaggings**()

Mixin_Tag::**getTaggings**()
Retrieve all the Taggings objects that represent between a specific tag and the current record Called by whatever record has enabled this module

Returns array of Taggings

Mixin_Tag::**getTags**(*$order = array()*)
Get all the Tag records associated with this record

Parameters

- **$order** –

Returns array of Tag

Mixin_Tag::**deleteTags**(*$tags*, *$delimiter = null*)
Delete a tag from the record

Parameters

- **$tags** (*string|array*) – The tag name or array of tag names to delete from the record

- **$delimiter** (*string*) – The delimiter of the tags. Not applicable if $tags is an array

Returns bool Returns whether a tag in $tags was deleted. Returns false if $tags is empty. Returns true if at least one tag in $tags is deleted.

Mixin_Tag::**hasTag**(*$tag*)

If the $tag were a string and the keys of Tags were just the names of the tags, this would be: in_array(array_keys($this->Tags))

Parameters

- **$tag** –

Returns bool

Mixin_Tag::**_getTagsFromString**(*$string*, *$delimiter = null*)

Converts a delimited string of tags into an array of tag strings

Parameters

- **$string** (*string*) – A delimited string of tags
- **$delimiter** –

Returns array An array of tag strings

Mixin_Tag::**addTags**(*$tags*, *$delimiter = null*)

Set tags to be saved to the record.

Parameters

- **$tags** (*array|string*) – Either an array of tags or a delimited string
- **$delimiter** –

Mixin_Tag::**applyTags**(*$inputTags*)

Apply tags

Parameters

- **$inputTags** (*array*) –

Mixin_Tag::**diffTags**(*$inputTags*, *$tags = null*)

Calculate the difference between a tag string and a set of tags

Parameters

- **$inputTags** –
- **$tags** –

Returns array Keys('removed','added')

Mixin_Tag::**applyTagString**(*$string*, *$delimiter = null*)

This will add tags that are in the tag string and remove those that are no longer in the tag string

Parameters

- **$string** (*string*) – A string of tags delimited by $delimiter
- **$delimiter** (*string|null*) –

Mixin_Timestamp

Package: *Record\Mixin*

class Mixin_Timestamp

extends *Omeka_Record_Mixin_AbstractMixin*

Mixin for models that keep added and/or modified timestamps.

property Mixin_Timestamp::$**_record**
protected

property Mixin_Timestamp::$**_addedColumn**
protected

property Mixin_Timestamp::$**_modifiedColumn**
protected

Mixin_Timestamp::**__construct**(*$record*, *$addedColumn = 'added'*, *$modifiedColumn = 'modified'*)
Initialize the mixin.

Setting either of the column parameters to null will skip updating that timestamp. The default column names are 'updated' and 'added'.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$addedColumn** (*string*) – Name of the column holding the "added" timestamp.
- **$modifiedColumn** –

Mixin_Timestamp::**beforeSave**(*$args*)
Before saving a record, set the "updated" timestamp.

Parameters

- **$args** –

Mixin_Timestamp::**_setTimestamp**(*$column*)
Update a timestamp column for the underlying record.

Parameters

- **$column** (*string*) – Column to update.

3.4.15 Option

Package: *Record*

class Option

extends *Omeka_Record_AbstractRecord*

A site option saved in the database.

Options are stored and accessed by string keys.

property Option::$**name**
string

Option name.

property Option::$**value**
string

Option value.

Option::**__toString**()
Use the option's value when treating it as a string.

Returns string

Option::**_validate**()
Validate the Option.

An option must have a non-empty and unique name.

3.4.16 models/Output

Output_CollectionOmekaXml

Package: *Output*

class Output_CollectionOmekaXml

extends *Omeka_Output_OmekaXml_AbstractOmekaXml*

Generates the omeka-xml output for a collection.

Output_CollectionOmekaXml::**_buildNode**()
Create a node representing a collection.

Output_FileOmekaXml

Package: *Output*

class Output_FileOmekaXml

extends *Omeka_Output_OmekaXml_AbstractOmekaXml*

Generates the omeka-xml output for File records.

Output_FileOmekaXml::**_buildNode**()
Create a node repesenting a File record.

Output_ItemAtom

Package: *Output*

class Output_ItemAtom
Model class for an Atom feed for a list of items.

__construct(*$items*)
Build the Atom feed using DOM.

Parameters

- **$items** (*array*) – An array of Item records.

_getFeedLinks(*$items*)
Returns the URLs, if any, for rel=self|next|previous links.

Parameters

- **$items** (*array*) –

Returns array

getFeed()
Returns the XML feed.

Returns string

Output_ItemContainerOmekaXml

Package: *Output*

class Output_ItemContainerOmekaXml

extends *Omeka_Output_OmekaXml_AbstractOmekaXml*

Generates the container element for items in the omeka-xml output format.

Output_ItemContainerOmekaXml::**_buildNode**()
Create a node to contain Item nodes.

Output_ItemDcmesXml

Package: *Output*

class Output_ItemDcmesXml

recordToDcmesXml(*$item*)

Parameters

- **$item** –

Output_ItemOmekaXml

Package: *Output*

class Output_ItemOmekaXml

extends *Omeka_Output_OmekaXml_AbstractOmekaXml*

Generates the omeka-xml output for Item records.

Output_ItemOmekaXml::**_buildNode**()
Create a node representing an Item record.

Output_ItemRss2

Package: *Output*

class Output_ItemRss2

render(*$records*)

Parameters

- **$records** –

buildRSSHeaders()

buildDescription(*$item*)

Parameters

- **$item** –

itemToRSS(*$item*)

Parameters

- **$item** –

Output_OmekaJson

Package: *Output*

class Output_OmekaJson

Generates JSON version of the omeka-xml output, as dictated by the JsonML XSLT.

constant JSONML_XSLT_FILENAME

JsonML XML stylesheet filename

toJson(*Omeka_Output_OmekaXml_AbstractOmekaXml $omekaXml*)

Convert omeka-xml output to JSON.

Parameters

- **$omekaXml** (Omeka_Output_OmekaXml_AbstractOmekaXml) –

Returns string

3.4.17 Plugin

Package: *Record*

class Plugin

extends *Omeka_Record_AbstractRecord*

implements Zend_Acl_Resource_Interface

A plugin and its metadata.

This record represents the data Omeka stores about each plugin and uses to manage the plugins, it is not a part of any plugin itself.

property Plugin::$**name**

string

Directory name for the plugin.

property Plugin::$**active**

int

Whether this plugin is active.

property Plugin::$**version**

string

Version string for the currently-installed plugin.

property Plugin::$**_displayName**
protected string

Human-readable display name of the plugin.

property Plugin::$**_author**
protected string

The plugin's author.

property Plugin::$**_description**
protected string

Description of the plugin.

property Plugin::$**_link**
protected string

URL for documentation or further information about the plugin.

property Plugin::$**_loaded**
protected bool

Whether the plugin has been loaded.

property Plugin::$**_hasConfig**
protected bool

Whether the plugin has a custom configuration form.

property Plugin::$**_requiredPlugins**
protected array

Directory names of required plugins.

property Plugin::$**_optionalPlugins**
protected array

Directory names of optional plugins.

property Plugin::$**_minimumOmekaVersion**
protected string

Minimum Omeka version requirement for the plugin.

property Plugin::$**_testedUpToVersion**
protected string

Maximum version of Omeka that the plugin has been tested on.

property Plugin::$**_iniVersion**
protected string

Version of the plugin that is stored in the INI.

property Plugin::$**_iniTags**
protected array

List of tags associated with this plugin, as retrieved from the ini file.

Plugin::**_validate**()
Validate the plugin.

The directory name must be set.

Plugin::**getDirectoryName**()
Get the name of the directory containing the plugin.

Returns string

Plugin::**setDirectoryName**(*$name*)
Set the name of the directory containing the plugin.

Parameters

- **$name** (*string*) –

Returns Plugin

Plugin::**getDisplayName**()
Get the human-readable name of the plugin.

If there is no human-readable name available, returns the directory name instead.

Returns string

Plugin::**setDisplayName**(*$name*)
Set the human-readable name of the plugin.

Parameters

- **$name** (*string*) –

Returns Plugin

Plugin::**getAuthor**()
Get the plugin's author.

Returns string

Plugin::**setAuthor**(*$author*)
Set the author's name.

Parameters

- **$author** (*string*) –

Returns Plugin

Plugin::**getDescription**()
Get the description of the plugin.

Returns string

Plugin::**setDescription**(*$description*)
Set the description of the plugin.

Parameters

- **$description** (*string*) –

Returns Plugin

Plugin::**getMinimumOmekaVersion**()
Get the minimum version of Omeka that this plugin requires to work.

Returns string

Plugin::**setMinimumOmekaVersion**(*$version*)
Set the minimum required version of Omeka.

Parameters

- **$version** (*string*) –

Returns Plugin

Plugin::**getTestedUpToOmekaVersion**()
Get the version of Omeka that this plugin is tested up to.

Returns string

Plugin::**setTestedUpToOmekaVersion**(*$version*)
Set the version of Omeka that this plugin is tested up to.

Parameters

- **$version** (*string*) –

Returns Plugin

Plugin::**getRequiredPlugins**()
Get the list of plugins that are required for this plugin to work.

Returns array

Plugin::**setRequiredPlugins**(*$plugins*)
Set the list of plugins that are required for this plugin to work.

Parameters

- **$plugins** –

Returns Plugin

Plugin::**getOptionalPlugins**()
Get the list of plugins that can be used, but are not required by, this plugin.

Returns array

Plugin::**setOptionalPlugins**(*$plugins*)
Set the list of optional plugins.

Parameters

- **$plugins** –

Returns Plugin

Plugin::**getIniTags**()
Get the list of tags for this plugin (from the ini file).

Returns array

Plugin::**setIniTags**(*$tags*)
Set the list of tags for this plugin.

Parameters

- **$tags** –

Returns Plugin

Plugin::**getSupportLinkUrl**()
Get the support link url from plugin.ini

Returns string

Plugin::**setSupportLinkUrl**(*$link*)
Set the support link url from plugin.ini

Parameters

- **$link** –

Returns Plugin

`Plugin::getLinkUrl()`
Get the URL link from the plugin.ini.

Returns string

`Plugin::setLinkUrl(*$link*)`
Set the link from the plugin.ini.

Parameters

- **$link** (*string*) –

Returns Plugin

`Plugin::isInstalled()`
Determine whether the Plugin has been installed.

Returns bool

`Plugin::isLoaded()`
Determine whether the Plugin has been loaded.

Returns bool

`Plugin::setLoaded(*$flag*)`
Set whether the plugin has been loaded.

Parameters

- **$flag** (*bool*) –

Returns Plugin

`Plugin::isActive()`
Determine whether the plugin is active.

Returns bool

`Plugin::setActive(*$flag*)`
Set whether the plugin is active.

Parameters

- **$flag** (*bool*) –

Returns Plugin

`Plugin::hasConfig()`
Determine whether the plugin has a custom configuration form.

Returns bool

`Plugin::setHasConfig(*$flag*)`
Set whether the plugin has a custom configuration form.

Parameters

- **$flag** (*bool*) –

Returns Plugin

`Plugin::getIniVersion()`
Get the version of the plugin stored in the INI file.

Returns string

Plugin::**setIniVersion**(*$version*)
Set the version of the plugin that is indicated by the INI file.

Parameters

- **$version** (*string*) –

Returns Plugin

Plugin::**getDbVersion**()
Get the version of the plugin that is stored in the database.

Returns string

Plugin::**setDbVersion**(*$version*)
Set the version of the plugin that is stored in the database.

Parameters

- **$version** (*string*) –

Returns Plugin

Plugin::**hasNewVersion**()
Determine whether there is a new version of the plugin available.

Returns bool

Plugin::**meetsOmekaMinimumVersion**()
Determine whether this Omeka install meets the plugin's minimum version requirements.

If the field is not set, assume that it meets the requirements. If the field is set, it must be greater than the current version of Omeka.

Returns bool

Plugin::**meetsOmekaTestedUpToVersion**()
Determine whether this Omeka version has been tested for use with the plugin.

Returns bool

Plugin::**getResourceId**()
Declare the Plugin model as relating to the Plugins ACL resource.

Returns string

3.4.18 Process

Package: *Record*

class Process

extends *Omeka_Record_AbstractRecord*

A process and its metadata.

property Process::$**pid**

property Process::$**class**

property Process::$**user_id**

property Process::$**status**

property Process::$**args**

property Process::$**started**

property Process::$**stopped**

Process::**beforeSave**(*$args*)

Parameters

- **$args** –

Process::**getArguments**()

Process::**setArguments**(*$args*)

Parameters

- **$args** –

Process::**_isSerialized**(*$s*)

Parameters

- **$s** –

3.4.19 RecordsTags

Package: *Record*

class RecordsTags

extends *Omeka_Record_AbstractRecord*

Linkage between a record and a tag.

property RecordsTags::$**record_id**
int

ID of the record being linked.

property RecordsTags::$**record_type**
int

Type of the record being linked.

property RecordsTags::$**tag_id**
int

ID of the tag being linked.

property RecordsTags::$**time**
string

Timestamp when this linkage was created.

RecordsTags::**_initializeMixins**()

3.4.20 SearchText

Package: *Record*

class SearchText

extends *Omeka_Record_AbstractRecord*

An entry in the site-wide fulltext search index for a record.

property SearchText::$**record_type**
int

Type of this text's associated record.

property SearchText::$**record_id**
int

ID of this text's associated record.

property SearchText::$**public**
int

Whether this text is publicly accessible.

property SearchText::$**title**
string

Display title for the record in search results.

property SearchText::$**text**
string

Searchable text for the record.

3.4.21 models/Table

Table_Collection

Package: *Db\Table*

class Table_Collection

extends *Omeka_Db_Table*

Table_Collection::**applySearchFilters**(*$select*, *$params*)

Parameters

- **$select** –
- **$params** –

Table_Collection::**findPairsForSelectForm**(*$options = array()*)

Parameters

- **$options** –

Table_Collection::**getSelect**()
Apply permissions checks to all SQL statements retrieving collections from the table

Table_Collection::**findRandomFeatured**()

Table_Collection::**applySorting**(*$select*, *$sortField*, *$sortDir*)
Enables sorting based on ElementSet,Element field strings.

Parameters

- **$select** (Omeka_Db_Select) –
- **$sortField** (*string*) – Field to sort on
- **$sortDir** (*string*) – Sorting direction (ASC or DESC)

Table_Element

Package: *Db\Table*

class Table_Element

extends *Omeka_Db_Table*

Table_Element::**findByRecordType**(*$recordTypeName*)
Find all the Element records that have a specific record type or the record type 'All', indicating that these elements would apply to any record type.

Parameters

- **$recordTypeName** –

Returns array

Table_Element::**getSelect**()
Overriding getSelect() to always return the type_name and type_regex for retrieved elements.

Returns Omeka_Db_Select

Table_Element::**_getColumnPairs**()
Return the element's name and id for <select> tags on it.

Table_Element::**orderElements**(*$select*)

Parameters

- **$select** –

Table_Element::**findBySet**(*$elementSet*)
Retrieve all elements for a set.

Parameters

- **$elementSet** –

Returns Element

Table_Element::**findByItemType**(*$itemTypeId*)
Retrieve a set of Element records that belong to a specific Item Type.

Parameters

- **$itemTypeId** –

Returns array Set of element records.

Table_Element::**findByElementSetNameAndElementName**(*$elementSetName*, *$elementName*)

Parameters

- **$elementSetName** –
- **$elementName** –

Table_Element::**applySearchFilters**(*$select*, *$params*)
Manipulate a Select object based on a set of criteria.

Parameters

- **$select** (Omeka_Db_Select) –

- **$params** (*array*) – Possible parameters include: <ul> record_types - array - Usually one or more of the following: All, Item, File sort - string - One of the following values: alpha element_set_name - string - Name of the element set to which results should belong.

Table_Element::**findPairsForSelectForm**(*$options = array()*)
Override parent class method to retrieve a multidimensional array of elements, organized by element set, to be used in Zend's FormSelect view helper.

Parameters

- **$options** (*array*) – Set of parameters for searching/filtering results.

Returns array

Table_ElementSet

Package: *Db\Table*

class Table_ElementSet

extends *Omeka_Db_Table*

Table_ElementSet::**getSelect**()

Table_ElementSet::**findByRecordType**(*$recordTypeName*, *$includeAll = true*)
Find all the element sets that correspond to a particular record type. If the second param is set, this will include all element sets that belong

to the 'All' record type.

Parameters

- **$recordTypeName** –
- **$includeAll** –

Returns array

Table_ElementSet::**findByName**(*$name*)

Parameters

- **$name** –

Table_ElementText

Package: *Db\Table*

class Table_ElementText

extends *Omeka_Db_Table*

Table_ElementText::**getSelectForRecord**(*$recordId*, *$recordType*)

Parameters

- **$recordId** –
- **$recordType** –

Returns Omeka_Db_Select

Table_ElementText::**findByRecord**(*Omeka_Record_AbstractRecord $record*)
Find all ElementText records for a given database record (Item, File, etc).

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Table_ElementText::**findByElement**(*$elementId*)

Parameters

- **$elementId** –

Table_File

Package: *Db\Table*

class Table_File

extends *Omeka_Db_Table*

property Table_File::$**_target**
protected

Table_File::**applySearchFilters**(*$select*, *$params*)

Parameters

- **$select** –
- **$params** –

Table_File::**filterByHasDerivativeImage**(*$select*, *$hasDerivative*)

Parameters

- **$select** –
- **$hasDerivative** –

Table_File::**getSelect**()
All files should only be retrieved if they join properly on the items table.

Returns Omeka_Db_Select

Table_File::**getRandomFileWithImage**(*$itemId*)
Retrieve a random file with an image associated with an item.

Parameters

- **$itemId** (*int*) –

Returns File

Table_File::**findByItem**(*$itemId*, *$fileIds = array()*, *$sort = 'order'*)
Retrieve files associated with an item.

Parameters

- **$itemId** (*int*) –
- **$fileIds** (*array*) – Optional If given, this will only retrieve files with these specific IDs.
- **$sort** (*string*) – The manner by which to order the files. For example: 'id': file id, 'filename' = alphabetical by filename. The default is 'order', following the user's specified order.

Returns array

`Table_File::`**`findOneByItem`**(*$itemId*, *$index = 0*, *$sort = 'order'*)
Get a single file associated with an item, by index.

Parameters

- **`$itemId`** (*`int`*) –
- **`$index`** (*`int`*) –
- **`$sort`** (*`string`*) – The manner by which to order the files. For example: 'id': file id, 'filename' = alphabetical by filename. The default is 'order', following the user's specified order.

Returns File|null

`Table_File::`**`findWithImages`**(*$itemId*, *$index = null*, *$sort = 'order'*)
Retrieve files for an item that has derivative images.

Parameters

- **`$itemId`** (*`int`*) – The ID of the item to get images for.
- **`$index`** (*`int|null`*) – Optional If given, this specifies the file to retrieve for an item, based upon the ordering of its files.
- **`$sort`** (*`string`*) – The manner by which to order the files. For example: 'id': file id, 'filename': alphabetical by filename. The default is 'order', following the user's specified order.

Returns File|array

`Table_File::`**`_orderFilesBy`**(*$select*, *$sort*)
Orders select results for files.

Parameters

- **`$select`** –
- **`$sort`** (*`string`*) – The manner in which to order the files by. For example: 'id' = file id 'filename' = alphabetical by filename

Table_Item

Package: *Db\Table*

`class Table_Item`

extends *`Omeka_Db_Table`*

`Table_Item::`**`filterBySearch`**(*$select*, *$params*)
Run the search filter on the SELECT statement

Parameters

- **`$select`** –
- **`$params`** –

`Table_Item::`**`_simpleSearch`**(*$select*, *$terms*)
Build the simple search.

The search query consists of a derived table that is INNER JOINed to the main SQL query. That derived table is a union of two SELECT queries. The first query searches the FULLTEXT index on

the items_elements table, and the second query searches the tags table for every word in the search terms and assigns each found result a rank of '1'. That should make tagged items show up higher on the found results list for a given search.

Parameters

- **$select** (*Zend_Db_Select*) –
- **$terms** –

Table_Item::**_advancedSearch**(*$select*, *$terms*)
Build the advanced search.

Parameters

- **$select** (*Zend_Db_Select*) –
- **$terms** –

Table_Item::**filterByCollection**(*$select*, *$collections*)
Filter the SELECT statement based on an item's collection

Parameters

- **$select** (*Zend_Db_Select*) –
- **$collections** (*Collection|int|array*) – Either a Collection object, or the collection id or an array of collection object or id.

Table_Item::**filterByItemType**(*$select*, *$types*)
Filter the SELECT statement based on the item Type

Parameters

- **$select** (*Zend_Db_Select*) –
- **$types** (*Type|int|string|array*) – One or multiple Item Type object, Item Type ID or Item Type name.

Table_Item::**filterByTags**(*$select*, *$tags*)
Query must look like the following in order to correctly retrieve items that have all the tags provided (in this example, all items that are tagged both 'foo' and 'bar'):

SELECT i.id FROM omeka_items i WHERE (i.id IN (SELECT tg.record_id as id FROM omeka_records_tags tg INNER JOIN omeka_tags t ON t.id = tg.tag_id WHERE t.name = 'foo' AND tg.record_type = 'Item') AND i.id IN (SELECT tg.record_id as id FROM omeka_records_tags tg INNER JOIN omeka_tags t ON t.id = tg.tag_id WHERE t.name = 'bar' AND tg.record_type = 'Item')) . . .

Parameters

- **$select** –
- **$tags** –

Table_Item::**filterByExcludedTags**(*$select*, *$tags*)
Filter SELECT statement based on items that are not tagged with a specific set of tags

Parameters

- **$select** –
- **$tags** –

Table_Item::**filterByHasDerivativeImage**(*$select*, *$hasDerivativeImage = true*)
Filter SELECT statement based on whether items have a derivative image file.

Parameters

- **$select** –
- **$hasDerivativeImage** (*bool*) – Whether items should have a derivative image file.

Table_Item::**applySearchFilters**(*$select*, *$params*)

Parameters

- **$select** –
- **$params** –

Table_Item::**applySorting**(*$select*, *$sortField*, *$sortDir*)

Enables sorting based on ElementSet,Element field strings.

Parameters

- **$select** (Omeka_Db_Select) –
- **$sortField** (*string*) – Field to sort on
- **$sortDir** (*string*) – Sorting direction (ASC or DESC)

Table_Item::**getSelect**()

This is a kind of simple factory that spits out proper beginnings of SQL statements when retrieving items

Returns Omeka_Db_Select

Table_Item::**findFirst**()

Return the first item accessible to the current user.

Returns Item|null

Table_Item::**findLast**()

Return the last item accessible to the current user.

Returns Item|null

Table_Item::**findPrevious**(*$item*)

Parameters

- **$item** –

Table_Item::**findNext**(*$item*)

Parameters

- **$item** –

Table_Item::**findNearby**(*$item*, *$position = 'next'*)

Parameters

- **$item** –
- **$position** –

Table_ItemType

Package: *Db\Table*

class Table_ItemType

extends *Omeka_Db_Table*

Table_ItemType::**_getColumnPairs**()

Table_ItemType::**findByName**(*$itemTypeName*)

Parameters

- **$itemTypeName** –

Table_ItemTypesElements

Package: *Db\Table*

class Table_ItemTypesElements

extends *Omeka_Db_Table*

Table_ItemTypesElements::**findByElement**(*$elementId*)

Parameters

- **$elementId** –

Table_Key

Package: *Db\Table*

class Table_Key

extends *Omeka_Db_Table*

Table_Plugin

Package: *Db\Table*

class Table_Plugin

extends *Omeka_Db_Table*

property Table_Plugin::$**_target**
protected

Table_Plugin::**applySearchFilters**(*$select*, *$params*)

Parameters

- **$select** –
- **$params** –

Table_Plugin::**findAllWithIniFiles**()

Table_Plugin::**findByDirectoryName**(*$pluginDirName*)

Parameters

- **$pluginDirName** –

Table_Process

Package: *Db\Table*

class Table_Process

extends *Omeka_Db_Table*

property Table_Process::$**_target**
protected

Table_Process::**findByClass**(*$className*)

Parameters

- **$className** –

Table_Process::**findByStatus**(*$status*)

Parameters

- **$status** –

Table_RecordsTags

Package: *Db\Table*

class Table_RecordsTags

extends *Omeka_Db_Table*

Table_RecordsTags::**applySearchFilters**(*$select*, *$params = array()*)

Parameters

- **$select** –
- **$params** –

Table_RecordsTags::**findForRecordAndTag**(*$record*, *$tag*)

Parameters

- **$record** –
- **$tag** –

Table_SearchText

Package: *Db\Table*

class Table_SearchText

extends *Omeka_Db_Table*

Table_SearchText::**findByRecord**(*$recordType*, *$recordId*)
Find search text by record.

Parameters

- **$recordType** (*string*) –
- **$recordId** (*int*) –

Returns SearchText|null

Table_SearchText::**applySearchFilters**(*$select*, *$params*)

Parameters

- **$select** –
- **$params** –

Table_Tag

Package: *Db\Table*

class Table_Tag

extends *Omeka_Db_Table*

Table_Tag::**findOrNew**(*$name*)

Parameters

- **$name** –

Table_Tag::**filterByRecord**(*$select*, *$record*)
Filter a SELECT statement based on an Omeka_Record_AbstractRecord instance

Parameters

- **$select** –
- **$record** –

Table_Tag::**applySorting**(*$select*, *$sortField*, *$sortDir*)
Apply custom sorting for tags.

This also applies the normal, built-in sorting.

Parameters

- **$select** (Omeka_Db_Select) –
- **$sortField** (*string*) – Sorting field.
- **$sortDir** (*string*) – Sorting direction, suitable for direct inclusion in SQL (ASC or DESC).

Table_Tag::**filterByTagType**(*$select*, *$type*)
Filter SELECT statement based on the type of tags to view (Item, Exhibit, etc.)

Parameters

- **$select** –
- **$type** –

Table_Tag::**filterByTagNameLike**(*$select*, *$partialTagName*)
Filter SELECT statement based on whether the tag contains the partial tag name

Parameters

- **$select** –
- **$partialTagName** –

Table_Tag::**applySearchFilters**(*$select*, *$params = array()*)
Retrieve a certain number of tags

Parameters

- **$select** –
- **$params** (*array*) – ‘limit’ => integer ‘record’ => instanceof Omeka_Record_AbstractRecord ‘like’ => partial_tag_name ‘type’ => tag_type

Table_Tag::**getSelect**()

Returns Omeka_Db_Select

Table_Tag::**getSelectForCount**(*$params = array()*)

Parameters

- **$params** –

Table_Tag::**findTagNamesLike**(*$partialName*, *$limit = 10*)

Parameters

- **$partialName** –
- **$limit** –

Table_User

Package: *Db\Table*

class Table_User

extends *Omeka_Db_Table*

Table_User::**findActiveById**(*$id*)
Find an active User given that user’s ID.

Returns null if the user being requested is not active.

Parameters

- **$id** –

Returns User|null

Table_User::**_getColumnPairs**()

Table_User::**findByEmail**(*$email*)

Parameters

- **$email** –

Table_User::**applySearchFilters**(*$select*, *$params*)

Parameters

- **$select** –
- **$params** –

Table_UsersActivations

Package: *Db\Table*

class Table_UsersActivations

extends *Omeka_Db_Table*

Table_UsersActivations::**findByUrl**(*$url*)

Parameters

- **$url** –

Table_UsersActivations::**findByUser**(*$user*)

Parameters

- **$user** –

3.4.22 Tag

Package: *Record*

class Tag

extends *Omeka_Record_AbstractRecord*

A tag and its metadata.

property Tag::$**name**
string

The tag text.

Tag::**__toString**()
Use the tag text when using this record as a string.

Returns string

Tag::**_delete**()
Delete handling for a tag.

Delete the taggings associated with this tag.

Tag::**_validate**()
Validate this tag.

The tag "name" must be non-empty and unique.

Tag::**rename**(*$new_names*)
Rename a tag.

Any records tagged with the "old" tag will be tagged with each of the tags given in $new_names. The original tag will be deleted (unless it is given as one of the $new_names).

Parameters

- **$new_names** (*array*) – Names of the tags this one should be renamed to.

3.4.23 Theme

Package: *Record*

class Theme
A theme and its metadata.

Unlike most other models, Themes are not stored in the database. This model relies only on INI data, but acts like an Omeka_Record_AbstractRecord model.

constant THEME_IMAGE_FILE_NAME
Filename for the theme screenshot.

constant THEME_INI_FILE_NAME
Filename for the theme INI file.

constant THEME_CONFIG_FILE_NAME
Filename for the theme config form INI file.

constant PUBLIC_THEME_OPTION
Option name for the current public theme.

constant ADMIN_THEME_OPTION
Option name for the current admin theme.

property path
string

Absolute path to the theme.

property directory
string

Directory name of the theme.

property image
string

Web path to the theme screenshot.

property author
string

The theme's author.

property title
string

The theme's title.

property description
string

The theme's description.

property license
string

The software license for the theme.

property website
string

A link to the theme's website.

property omeka_minimum_version
string

The minimum Omeka version the theme will run on.

__construct (*$themeName*)
Set the INI and file data for the theme, given its directory name.

Parameters

- **$themeName** (*string*) – Directory name.

setDirectoryName (*$dir*)
Set the theme's directory name and path.

Parameters

- **$dir** (*string*) – Directory name.

getScriptPath()
Get the physical path to the theme's scripts.

Returns string Physical path.

getAssetPath()
Get the web path to the theme's assets.

Returns string Web path.

getScriptPathForPlugin(*$pluginModuleName*)
Get the physical path to the theme's override scripts for the given plugin.

Parameters

- **$pluginModuleName** (*string*) – (i.e., 'exhibit-builder')

Returns string Physical path.

getAssetPathForPlugin(*$pluginModuleName*)
Get the web path to the theme's override assets for the given plugin.

Parameters

- **$pluginModuleName** (*string*) – (i.e., 'exhibit-builder')

Returns string Web path.

setImage(*$fileName*)
Set the web path to the screenshot, if it exists.

Parameters

- **$fileName** (*string*) – Relative filename of the image to check.

setIni(*$fileName*)
Load data from the INI file at the given path.

Parameters

- **$fileName** (*string*) – Relative filename of the INI file.

setConfig(*$fileName*)
Check for a theme config file at the given location.

Parameters

- **$fileName** (*string*) – Relative filename of the theme config.ini.

getCurrentThemeName(*$type = null*)
Get the directory name of the current theme.

Parameters

- **$type** (*string*) – 'admin' or 'public', defaults to current type

Returns string

getAllThemes()
Retrieve all themes

Returns array An array of theme objects

getAllAdminThemes()
Retrieve all admin themes

Returns array An array of theme objects

getTheme(*$themeName*)
Retrieve a theme.

Parameters

- **$themeName** (*string*) – The name of the theme.

Returns Theme A theme object

setOptions(*$themeName*, *$themeConfigOptions*)
Set theme configuration options.

Parameters

- **$themeName** (*string*) – The name of the theme
- **$themeConfigOptions** (*array*) – An associative array of configuration options, where each key is a configuration form input name and each value is a string value of that configuration form input

getOptions(*$themeName*)
Get theme configuration options.

Parameters

- **$themeName** (*string*) – The name of the theme

Returns array An associative array of configuration options, where each key is a configuration form input name and each value is a string value of that configuration form input

getOption(*$themeName*, *$themeOptionName*)
Get the value of a theme configuration option.

Parameters

- **$themeName** (*string*) – The name of the theme
- **$themeOptionName** (*string*) – The name of the theme option

Returns string The value of the theme option

setOption(*$themeName*, *$themeOptionName*, *$themeOptionValue*)
Set the value of a theme configuration option.

Parameters

- **$themeName** (*string*) – The name of the theme
- **$themeOptionName** (*string*) – The name of the theme option
- **$themeOptionValue** –

getOptionName(*$themeName*)
Get the name of a specific theme's option. Each theme has a single option in the option's table, which stores all of the configuration options for that theme

Parameters

- **$themeName** (*string*) – The name of the theme

Returns string The name of a specific theme's option.

getUploadedFileName (*$themeName*, *$optionName*, *$fileName*)
Get the name of a file uploaded as a theme configuration option. This is the name of the file after it has been uploaded and renamed.

Parameters

- **$themeName** (*string*) – The name of the theme
- **$optionName** (*string*) – The name of the theme option associated with the uploaded file
- **$fileName** (*string*) – The name of the uploaded file

Returns string The name of an uploaded file for the theme.

_parseWebsite (*$website*)
Parses the website string to confirm whether it has a scheme.

Parameters

- **$website** (*string*) – The website given in the theme's INI file.

Returns string The website URL with a prepended scheme.

3.4.24 User

Package: *Record*

class User

extends *Omeka_Record_AbstractRecord*

implements Zend_Acl_Resource_Interface implements Zend_Acl_Role_Interface

A user and its metadata.

constant User::USERNAME_MIN_LENGTH
Minimum username length.

constant User::USERNAME_MAX_LENGTH
Maximum username length.

constant User::PASSWORD_MIN_LENGTH
Minimum password length.

constant User::INVALID_EMAIL_ERROR_MSG
Error message for an invalid email address.

constant User::CLAIMED_EMAIL_ERROR_MSG
Error message for an already-taken email address.

property User::$username
string

This User's username.

property User::$password
string

The hashed password.

This field should never contain the plain-text password. Always use setPassword() to change the user password.

property User::$**salt**
string

The salt for the hashed password.

property User::$**active**
int

Whether this user is active and can log in.

property User::$**role**
string

This user's role.

property User::$**name**
string

This user's full or display name.

property User::$**email**
string

This user's email address.

User::**beforeSave**(*$args*)
Before-save hook.

Check the current user's privileges to change user roles before saving.

Parameters

- **$args** –

User::**filterPostData**(*$post*)
Filter form POST input.

Transform usernames to lowercase alphanumeric.

Parameters

- **$post** (*array*) –

Returns array Cleaned POST data.

User::**setPostData**(*$post*)
Set data from POST to the record.

Removes the 'password' and 'salt' entries, if passed.

Parameters

- **$post** –

User::**_validate**()
Validate this User.

User::**upgradeHashedPassword**(*$username*, *$password*)
Upgrade the hashed password.

Does nothing if the user/password is incorrect, or if same has been upgraded already.

Parameters

- **$username** (*string*) –
- **$password** (*string*) –

Returns bool False if incorrect username/password given, otherwise true when password can be or has been upgraded.

`User::`**`getRoleId`**`()`
Get this User's role.

Required by Zend_Acl_Role_Interface.

Returns string

`User::`**`getResourceId`**`()`
Get the Resource ID for the User model.

Required by Zend_Acl_Resource_Interface.

Returns string

`User::`**`generateSalt`**`()`
Generate a simple 16 character salt for the user.

`User::`**`setPassword`**`(`*$password*`)`
Set a new password for the user.

Always use this method to set a password, do not directly set the password or salt properties.

Parameters

- **`$password`** (*`string`*) – Plain-text password.

`User::`**`hashPassword`**`(`*$password*`)`
SHA-1 hash the given password with the current salt.

Parameters

- **`$password`** (*`string`*) – Plain-text password.

Returns string Salted and hashed password.

3.4.25 UsersActivations

Package: *Record*

`class UsersActivations`

extends *`Omeka_Record_AbstractRecord`*

An activation code for a User.

`property` `UsersActivations::$`**`user_id`**
int

The ID of the User this activation code is for.

`property` `UsersActivations::$`**`url`**
string

Random activation key.

`property` `UsersActivations::$`**`added`**
string

Date this activation key was created.

property UsersActivations::$**_related**
protected array

Related records.

UsersActivations::**factory**(*User $user*)
Get a new UsersActivations for a User.

Parameters

- **$user** (User) –

Returns UsersActivations

UsersActivations::**beforeSave**(*$args*)
Before-save hook.

Set the timestamp and create a random key.

Parameters

- **$args** –

UsersActivations::**getUser**()
Get the User for this Activation.

Returns User

3.5 views/helpers

3.5.1 Omeka_View_Helper_AllElementTexts

Package: *View\Helper*

class Omeka_View_Helper_AllElementTexts

extends Zend_View_Helper_Abstract

View helper for retrieving lists of metadata for any record that uses Mixin_ElementText.

property Omeka_View_Helper_AllElementTexts::$**_record**
protected Omeka_Record_AbstractRecord

The record being printed.

property Omeka_View_Helper_AllElementTexts::$**_showEmptyElements**
protected bool

Flag to indicate whether to show elements that do not have text.

property Omeka_View_Helper_AllElementTexts::$**_showElementSetHeadings**
protected bool

Whether to include a heading for each Element Set.

property Omeka_View_Helper_AllElementTexts::$**_emptyElementString**
protected string

String to display if elements without text are shown.

property Omeka_View_Helper_AllElementTexts::$**_elementSetsToShow**
protected array

Element sets to list.

property Omeka_View_Helper_AllElementTexts::$**_returnType**
protected string

Type of data to return.

property Omeka_View_Helper_AllElementTexts::$**_partial**
protected string

Path for the view partial.

Omeka_View_Helper_AllElementTexts::**allElementTexts**(*$record*, *$options = array()*)
Get the record metadata list.

Parameters

- **$record** (*Omeka_Record_AbstractRecord|string*) – Record to retrieve metadata from.
- **$options** (*array*) – Available options: - show_empty_elements' => boolstring Whether to show elements that do not contain text. A string will set self::$_showEmptyElements to true and set self::$_emptyElementString to the provided string. - 'show_element_sets' => array List of names of element sets to display. - 'return_type' => string 'array', 'html'. Defaults to 'html'.

Returns string|array

Omeka_View_Helper_AllElementTexts::**_setOptions**(*$options*)
Set the options.

Parameters

- **$options** (*array*) –

Omeka_View_Helper_AllElementTexts::**_getElementsBySet**()
Get an array of all element sets containing their respective elements.

Returns array

Omeka_View_Helper_AllElementTexts::**_filterItemTypeElements**(*$elementsBySet*)
Filter the display of the Item Type element set, if present.

Parameters

- **$elementsBySet** (*array*) –

Returns array

Omeka_View_Helper_AllElementTexts::**_elementIsShowable**(*Element $element*, *$texts*)
Determine if an element is allowed to be shown.

Parameters

- **$element** (Element) –
- **$texts** (*array*) –

Returns bool

Omeka_View_Helper_AllElementTexts::**_getFormattedElementTexts**(*$record*, *$metadata*)
Return a formatted version of all the texts for the requested element.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$metadata** (*array*) –

Returns array

Omeka_View_Helper_AllElementTexts::**_getOutputAsHtml**()
Output the default HTML format for displaying record metadata.

Returns string

Omeka_View_Helper_AllElementTexts::**_getOutputAsArray**()
Get the metadata list as a PHP array.

Returns array

Omeka_View_Helper_AllElementTexts::**_getOutput**()
Get the metadata list.

Returns string|array

Omeka_View_Helper_AllElementTexts::**_loadViewPartial**(*$vars = array()*)
Load a view partial to display the data.

Parameters

- **$vars** (*array*) – Variables to pass to the partial.

Returns string

3.5.2 Omeka_View_Helper_ElementForm

Package: *View\Helper*

class Omeka_View_Helper_ElementForm

extends Zend_View_Helper_Abstract

Generate the form markup for entering element text metadata.

property Omeka_View_Helper_ElementForm::$**_element**
protected Element

Displays a form for the record's element.

The function applies filters that allow plugins to customize the display of element form components. Here is an example of how a plugin may add and implement an element form filter:

add_filter(array('ElementForm', 'Item', 'Dublin Core', 'Title')), 'form_item_title'); function form_item_title(array $components, $args) {

// Where $components would looks like: // array(// 'label' => [...], // 'inputs' => [...], // 'description' => [...], // 'comment' => [...], // 'add_input' => [...], //) // and $args looks like: // array(// 'record' => [...], // 'element' => [...], // 'options' => [...], //) }

property Omeka_View_Helper_ElementForm::$**_record**
protected

Omeka_View_Helper_ElementForm::**elementForm**(*Element $element*, *Omeka_Record_AbstractRecord $record*, *$options = array()*)

Parameters

- **$element** (Element) –

- **$record** (Omeka_Record_AbstractRecord) –
- **$options** –

Omeka_View_Helper_ElementForm::**_getFieldLabel**()

Omeka_View_Helper_ElementForm::**_getFieldDescription**()

Omeka_View_Helper_ElementForm::**_getFieldComment**()

Omeka_View_Helper_ElementForm::**_isPosted**()

Omeka_View_Helper_ElementForm::**_getPostArray**()

Omeka_View_Helper_ElementForm::**_getFormFieldCount**()
How many form inputs to display for a given element.

Returns int

Omeka_View_Helper_ElementForm::**_getPostValueForField**(*$index*)

Parameters

- **$index** –

Returns mixed

Omeka_View_Helper_ElementForm::**_getHtmlFlagForField**(*$index*)

Parameters

- **$index** –

Omeka_View_Helper_ElementForm::**_getValueForField**(*$index*)
Retrieve the form value for the field.

Parameters

- **$index** –

Returns string

Omeka_View_Helper_ElementForm::**getElementTexts**(*$index = null*)
If index is not given, return all texts.

Parameters

- **$index** –

Omeka_View_Helper_ElementForm::**_getInputsComponent**(*$extraFieldCount = null*)

Parameters

- **$extraFieldCount** –

Omeka_View_Helper_ElementForm::**_getDescriptionComponent**()

Omeka_View_Helper_ElementForm::**_getCommentComponent**()

Omeka_View_Helper_ElementForm::**_getLabelComponent**()

3.5.3 Omeka_View_Helper_ElementInput

Package: *View\Helper*

class Omeka_View_Helper_ElementInput

extends Zend_View_Helper_Abstract

Generate the form markup for entering one HTML input for an Element.

property Omeka_View_Helper_ElementInput::$**_element**
protected Element

Element record to display the input for.

property Omeka_View_Helper_ElementInput::$**_record**
protected Omeka_Record_AbstractRecord

Omeka_Record_AbstractRecord to display the input for.

Omeka_View_Helper_ElementInput::**elementInput**(*Element $element*, *Omeka_Record_AbstractRecord $record*, *$index = 0*, *$value = ''*, *$isHtml = false*)

Display one form input for an Element.

Parameters

- **$element** (Element) – The Element to display the input for.
- **$record** (Omeka_Record_AbstractRecord) – The record to display the input for.
- **$index** (*int*) – The index of this input. (starting at zero).
- **$value** (*string*) – The default value of this input.
- **$isHtml** (*bool*) – Whether this input's value is HTML.

Returns string

Omeka_View_Helper_ElementInput::**_getInputComponent**(*$inputNameStem*, *$value*)

Get the actual HTML input for this Element.

Parameters

- **$inputNameStem** (*string*) –
- **$value** (*string*) –

Returns string

Omeka_View_Helper_ElementInput::**_getControlsComponent**()

Get the button that will allow a user to remove this form input. The submit input has a class of 'add-element', which is used by the Javascript to do stuff.

Returns string

Omeka_View_Helper_ElementInput::**_getHtmlCheckboxComponent**(*$inputNameStem*, *$isHtml*)

Get the HTML checkbox that lets users toggle the editor.

Parameters

- **$inputNameStem** (*string*) –
- **$isHtml** (*bool*) –

Returns string

3.5.4 Omeka_View_Helper_FileId3Metadata

Package: *View\Helper*

class Omeka_View_Helper_FileId3Metadata

extends Zend_View_Helper_Abstract

Helper used to retrieve file metadata for display.

Omeka_View_Helper_FileId3Metadata::**fileId3Metadata**(*$file*, *$options*)

Parameters

- **$file** –
- **$options** –

Omeka_View_Helper_FileId3Metadata::**_arrayToList**(*$array*)

Parameters

- **$array** –

3.5.5 Omeka_View_Helper_FileMarkup

Package: *View\Helper*

class Omeka_View_Helper_FileMarkup

extends Zend_View_Helper_Abstract

View Helper for displaying files through Omeka.

This will determine how to display any given file based on the MIME type (Internet media type) of that file. Individual rendering agents are defined by callbacks that are either contained within this class or defined by plugins. Callbacks defined by plugins will override native class methods if defined for existing MIME types. In order to define a rendering callback that should be in the core of Omeka, define a method in this class and then make sure that it responds to all the correct MIME types by modifying other properties in this class.

constant Omeka_View_Helper_FileMarkup::**GENERIC_FALLBACK_IMAGE**
Fallback image used when no other fallbacks are appropriate.

property Omeka_View_Helper_FileMarkup::$**_callbacks**
protected array

Array of MIME types and the callbacks that can process it.

Example: array('video/avi'=>'wmv');

property Omeka_View_Helper_FileMarkup::$**_callbackOptions**
protected array

The array consists of the default options which are passed to the callback.

property Omeka_View_Helper_FileMarkup::$**_fallbackImages**
protected array

Images to show when a file has no derivative.

Omeka_View_Helper_FileMarkup::**addMimeTypes**(*$fileIdentifiers*, *$callback*, *$defaultOptions = array()*)
Add MIME types and/or file extensions and associated callbacks to the list.

This allows plugins to override/define ways of displaying specific files. The most obvious example of where this would come in handy is to define ways of displaying uncommon files, such as QTVR, or novel ways of displaying more common files, such as using iPaper to display PDFs.

Parameters

- **$fileIdentifiers** (*array|string*) – Set of MIME types (Internet media types) and/or file extensions that this specific callback will respond to. Accepts the following: <ul> A string containing one MIME type: <code>'application/msword'</code> A simple array containing MIME types: <code>array('application/msword', 'application/doc')</code> A keyed array containing MIME types: <code>array('mimeTypes' => array('application/msword', 'application/doc'))</code> A keyed array containing file extensions: <code>array('fileExtensions' => array('doc', 'docx''DOC', 'DOCX'))</code> A keyed array containing MIME types and file extensions: <code> array('mimeTypes' => array('application/msword', 'application/doc', 'application/vnd.openxmlformats-officedocument.wordprocessingml.document',), 'fileExtensions' => array('doc', 'docx', 'DOC', 'DOCX'),) </code> Note that file extensions are case sensitive.
- **$callback** –
- **$defaultOptions** (*array*) –

Omeka_View_Helper_FileMarkup::**addFallbackImage**(*$mimeType*, *$image*)

Add a fallback image for the given mime type or type family.

Parameters

- **$mimeType** (*string*) – The mime type this fallback is for, or the mime "prefix" it is for (video, audio, etc.)
- **$image** (*string*) – The name of the image to use, as would be passed to img()

Omeka_View_Helper_FileMarkup::**defaultDisplay**(*$file*, *$options = array()*)

Default display for MIME types that do not have a valid rendering callback.

This wraps the original filename in a link to download that file, with a class of "download-file". Any behavior more complex than that should be processed with a valid callback.

Parameters

- **$file** (File) –
- **$options** (*array*) –

Returns string HTML

Omeka_View_Helper_FileMarkup::**_linkToFile**(*$file*, *$options*, *$html = null*)

Add a link for the file based on the given set of options.

If the 'linkToMetadata' option is true, then link to the file metadata page (files/show). If 'linkToFile' is true, link to the original file, and if 'linkToFile' is a string, try to link to that specific derivative. Otherwise just return the $html without wrapping in a link.

The attributes for the link will be based off the 'linkAttributes' option, which should be an array.

If $html is null, it defaults to original filename of the file.

Parameters

- **$file** (File) –

- **$options** (*array*) –
- **$html** (*string*) –

Returns string

Omeka_View_Helper_FileMarkup::**video**(*$file*, *$options = array()*)
Retrieve valid XHTML for displaying Quicktime video files

Parameters

- **$file** (File) –
- **$options** (*array*) – The set of default options for this includes: width, height, autoplay, controller, loop

Returns string

Omeka_View_Helper_FileMarkup::**audio**(*$file*, *$options*)
Default display of audio files via <audio> tag.

Parameters

- **$file** (File) –
- **$options** (*array*) – The set of default options for this includes: width, height, autoplay, controller, loop

Returns string

Omeka_View_Helper_FileMarkup::**_media**(*$type*, *$file*, *$options*)

Parameters

- **$type** –
- **$file** –
- **$options** –

Omeka_View_Helper_FileMarkup::**icon**(*$file*, *$options = array()*)
Default display of an icon to represent a file.

Example usage:

echo files_for_item(array('showFilename'=>false, 'linkToFile'=>false, 'linkAttributes'=>array('rel'=>'lightbox'), 'filenameAttributes'=>array('class'=>'error'), 'imgAttributes'=>array('id'=>'foobar'), 'icons' => array('audio/mpeg'=>img('audio.gif'))));

Parameters

- **$file** –
- **$options** (*array*) – Available options include: 'showFilename' => boolean, 'linkToFile' => boolean, 'linkAttributes' => array, 'filenameAttributes' => array (for the filename div), 'imgAttributes' => array, 'icons' => array.

Returns string

Omeka_View_Helper_FileMarkup::**derivativeImage**(*$file*, *$options = array()*)
Returns valid XHTML markup for displaying an image that has been stored in Omeka.

Parameters

- **$file** (File) – Options for customizing the display of images. Current options include: 'imageSize'
- **$options** –

Returns string HTML for display

Omeka_View_Helper_FileMarkup::**getCallback**(*$file*, *$options*)

Parameters

- **$file** –
- **$options** –

Omeka_View_Helper_FileMarkup::**getDefaultOptions**(*$callback*)

Parameters

- **$callback** (*mixed*) –

Returns array

Omeka_View_Helper_FileMarkup::**getHtml**(*$file*, *$renderer*, *$options*)

Retrieve the HTML for a given file from the callback.

Parameters

- **$file** (File) –
- **$renderer** (*callback*) – Any valid callback that will display the HTML.
- **$options** (*array*) – Set of options passed to the rendering callback.

Returns string HTML for displaying the file.

Omeka_View_Helper_FileMarkup::**fileMarkup**(*$file*, *$props = array()*, *$wrapperAttributes = array()*)

Bootstrap for the helper class. This will retrieve the HTML for displaying the file and by default wrap it in a .

Parameters

- **$file** (File) –
- **$props** (*array*) – Set of options passed by a theme writer to the customize the display of any given callback.
- **$wrapperAttributes** (*array*) –

Returns string HTML

Omeka_View_Helper_FileMarkup::**image_tag**(*$record*, *$props*, *$format*)

Return a valid img tag for an image.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$props** (*array*) – Image tag attributes
- **$format** (*string*) – Derivative image type (thumbnail, etc.)

Returns string

Omeka_View_Helper_FileMarkup::**_getFallbackImage**(*$file*)

Get the name of a fallback image to use for this file.

The fallback used depends on the file's mime type.

Parameters

- **$file** (File) – The file to get a fallback for.

Returns string Name of the image to use.

Omeka_View_Helper_FileMarkup::**_getCallbackKey**(*$callback*)
Get a string key to represent a given callback.

This key can be used to store and retrieve data about the callback, like default options.

Parameters

- **$callback** (*callback*) –

Returns string

3.5.6 Omeka_View_Helper_Flash

Package: *View\Helper*

class Omeka_View_Helper_Flash

extends Zend_View_Helper_Abstract

View helper to display messages from FlashMessenger.

Omeka_View_Helper_Flash::**__construct**()

Omeka_View_Helper_Flash::**flash**()
Display messages from the FlashMessenger.

Returns string HTML for messages.

Omeka_View_Helper_Flash::**_getListHtml**(*$status*, *$message*)
Get the HTML for a message.

Parameters

- **$status** (*string*) –
- **$message** (*string*) –

Returns string

3.5.7 Omeka_View_Helper_FormInput

Package: *View\Helper*

class Omeka_View_Helper_FormInput

extends Zend_View_Helper_FormElement

Helper for generic HTML5 input with settable type.

Omeka_View_Helper_FormInput::**formInput**(*$name*, *$value = null*, *$attribs = null*)
Generate an input element.

Parameters

- **$name** (*string|array*) – If a string, the element name. If an array, all other parameters are ignored, and the array elements are used in place of added parameters.
- **$value** (*mixed*) – The element value.
- **$attribs** (*array*) – Attributes for the element tag.

Returns string The element XHTML.

3.5.8 Omeka_View_Helper_GetCurrentRecord

Package: *View\Helper*

class Omeka_View_Helper_GetCurrentRecord

extends Zend_View_Helper_Abstract

Omeka_View_Helper_GetCurrentRecord**::getCurrentRecord**(*$recordVar*, *$throwException = true*)

Get the current record from the view.

Parameters

- **$recordVar** (*string*) –
- **$throwException** (*bool*) –

Returns Omeka_Record_AbstractRecord|false

3.5.9 Omeka_View_Helper_GetLoopRecords

Package: *View\Helper*

class Omeka_View_Helper_GetLoopRecords

extends Zend_View_Helper_Abstract

Omeka_View_Helper_GetLoopRecords**::getLoopRecords**(*$recordsVar*, *$throwException = true*)

Get records from the view for iteration.

Note that this method will return an empty array if it is set to the records variable. Use Omeka_View_Helper_HasLoopRecords::hasLoopRecords() to check if records exist.

Parameters

- **$recordsVar** (*string*) –
- **$throwException** –

Returns array|bool

3.5.10 Omeka_View_Helper_HasLoopRecords

Package: *View\Helper*

class Omeka_View_Helper_HasLoopRecords

extends Zend_View_Helper_Abstract

Omeka_View_Helper_HasLoopRecords**::hasLoopRecords**(*$recordsVar*)

Check if records have been set to the view for iteration.

Note that this method will return false if the records variable is set but is an empty array, unlike Omeka_View_Helper_GetLoopRecords::getLoopRecords(), which will return the empty array.

Parameters

- **$recordsVar** (*string*) –

Returns bool

3.5.11 Omeka_View_Helper_ItemSearchFilters

Package: *View\Helper*

class Omeka_View_Helper_ItemSearchFilters

extends Zend_View_Helper_Abstract

Show the currently-active filters for a search/browse.

Omeka_View_Helper_ItemSearchFilters::**itemSearchFilters**(*$params = null*)
Get a list of the currently-active filters for item browse/search.

Parameters

- **$params** (*array*) – Optional array of key-value pairs to use instead of reading the current params from the request.

Returns string HTML output

3.5.12 Omeka_View_Helper_Loop

Package: *View\Helper*

class Omeka_View_Helper_Loop

extends Zend_View_Helper_Abstract

Omeka_View_Helper_Loop::**loop**(*$recordsVar*, *$records = null*)
Return an iterator used for looping an array of records.

Parameters

- **$recordsVar** (*string*) –
- **$records** (*array|null*) –

Returns Omeka_Record_Iterator

3.5.13 Omeka_View_Helper_MaxFileSize

Package: *View\Helper*

class Omeka_View_Helper_MaxFileSize

extends Zend_View_Helper_Abstract

property Omeka_View_Helper_MaxFileSize::$**_maxFileSize**
protected string

Omeka_View_Helper_MaxFileSize::**__construct**()
Set the maximum file size.

The maximum file size is the least of the configurations that affect maximum file size.

Omeka_View_Helper_MaxFileSize::**maxFileSize**()
Return the maximum file size.

Returns string

Omeka_View_Helper_MaxFileSize::**_parseSize**(*$sizeString*)
Get the size in bytes represented by the given php ini config string

Parameters

- **$sizeString** (*string*) –

Returns int Size in bytes

3.5.14 Omeka_View_Helper_Metadata

Package: *View\Helper*

class Omeka_View_Helper_Metadata

extends Zend_View_Helper_Abstract

Helper used to retrieve record metadata for for display.

Omeka_View_Helper_Metadata::**metadata**(*$record*, *$metadata*, *$options = array()*)
Retrieve a specific piece of a record's metadata for display.

Parameters

- **$record** (Omeka_Record_AbstractRecord) – Database record representing the item from which to retrieve field data.
- **$metadata** (*string|array*) – The metadata field to retrieve. If a string, refers to a property of the record itself. If an array, refers to an Element: the first entry is the set name, the second is the element name.
- **$options** (*array|string|int*) – Options for formatting the metadata for display. - Array options: - 'all': If true, return an array containing all values for the field. - 'delimiter': Return the entire set of metadata as a string, where entries are separated by the given delimiter. - 'index': Return the metadata entry at the given zero-based index. - 'no_escape' => If true, do not escape the resulting values for HTML entities. - 'no_filter': If true, return the set of metadata without running any of the filters. - 'snippet': Trim the length of each piece of text to the given length in characters. - Passing simply the string 'all' is equivalent to array('all' => true) - Passing simply an integer is equivalent to array('index' => [the integer])

Returns string|array|null Null if field does not exist for item. Array if certain options are passed. String otherwise.

Omeka_View_Helper_Metadata::**_getOptions**(*$options*)
Options can sometimes be an integer or a string instead of an array, which functions as a handy shortcut for theme writers. This converts the short form of the options into its proper array form.

Parameters

- **$options** (*string|int|array*) –

Returns array

Omeka_View_Helper_Metadata::**_getText**(*$record*, *$metadata*)
Retrieve the text associated with a given element or field of the record.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$metadata** (*string|array*) –

Returns string|array Either an array of ElementText records or a string.

Omeka_View_Helper_Metadata::**_getRecordMetadata**(*$record*, *$specialValue*)
Retrieve record metadata that is not stored as ElementTexts.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$specialValue** (*string*) – Field name.

Returns mixed

Omeka_View_Helper_Metadata::**_getElementText**(*$record*, *$elementSetName*, *$elementName*)

Retrieve the set of ElementText records that correspond to a given element set and element.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$elementSetName** (*string*) –
- **$elementName** (*string*) –

Returns array Set of ElementText records.

Omeka_View_Helper_Metadata::**_process**(*$record*, *$metadata*, *$text*, *$snippet*, *$escape*, *$filter*)

Process an individual piece of text.

If given an ElementText record, the actual text string will be extracted automatically.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$metadata** (*string|array*) –
- **$text** (*string|ElementText*) – Text to process.
- **$snippet** (*int|bool*) – Snippet length, or false if no snippet.
- **$escape** (*bool*) – Whether to HTML escape the text.
- **$filter** (*bool*) – Whether to pass the output through plugin filters.

Returns string

Omeka_View_Helper_Metadata::**_filterText**(*$record*, *$metadata*, *$text*, *$elementText*)

Apply filters to a text value.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$metadata** (*string|array*) –
- **$text** (*string*) –
- **$elementText** (*ElementText|bool*) –

Returns string

3.5.15 Omeka_View_Helper_Pluralize

Package: *View\Helper*

class Omeka_View_Helper_Pluralize

extends Zend_View_Helper_Abstract

Omeka_View_Helper_Pluralize::**pluralize**(*$var*)

Parameters

- **$var** –

3.5.16 Omeka_View_Helper_RecordUrl

Package: *View\Helper*

class Omeka_View_Helper_RecordUrl

extends Zend_View_Helper_Abstract

Omeka_View_Helper_RecordUrl::**recordUrl**(*$record*, *$action = null*, *$getAbsoluteUrl = false*, *$queryParams = array()*)

Return a URL to a record.

Parameters

- **$record** (*Omeka_Record_AbstractRecord|string*) –
- **$action** (*string|null*) –
- **$getAbsoluteUrl** (*bool*) –
- **$queryParams** (*array*) –

Returns string

3.5.17 Omeka_View_Helper_SearchFilters

Package: *View\Helper*

class Omeka_View_Helper_SearchFilters

extends Zend_View_Helper_Abstract

Return a list of search filters in the current request.

Omeka_View_Helper_SearchFilters::**searchFilters**(*$options = array()*)

Return a list of current search filters in use.

Parameters

- **$options** (*array*) – Valid options are as follows: - id (string): the ID of the filter wrapping div.

Returns string

3.5.18 Omeka_View_Helper_SearchForm

Package: *View\Helper*

class Omeka_View_Helper_SearchForm

extends Zend_View_Helper_Abstract

Return the site-wide search form.

Omeka_View_Helper_SearchForm::**searchForm**(*$options = array()*)

Return the site-wide search form.

Parameters

- **$options** (*array*) – Valid options are as follows: - show_advanced: whether to show the advanced search; default is false. - submit_value: the value of the submit button; default "Submit". - form_attributes: an array containing form tag attributes.

Returns string The search form markup.

3.5.19 Omeka_View_Helper_SetCurrentRecord

Package: *View\Helper*

class Omeka_View_Helper_SetCurrentRecord

extends Zend_View_Helper_Abstract

Omeka_View_Helper_SetCurrentRecord::**setCurrentRecord**(*$recordVar*, *Omeka_Record_AbstractRecord $record*, *$setPreviousRecord = false*)

Set a record to the view as the current record.

Parameters

- **$recordVar** (*string*) –
- **$record** (Omeka_Record_AbstractRecord) –
- **$setPreviousRecord** (*bool*) –

3.5.20 Omeka_View_Helper_SetLoopRecords

Package: *View\Helper*

class Omeka_View_Helper_SetLoopRecords

extends Zend_View_Helper_Abstract

Omeka_View_Helper_SetLoopRecords::**setLoopRecords**(*$recordsVar*, *$records*)

Set records to the view for iteration.

Parameters

- **$recordsVar** (*string*) –
- **$records** (*array*) –

3.5.21 Omeka_View_Helper_Shortcodes

Package: *View\Helper*

class Omeka_View_Helper_Shortcodes

extends Zend_View_Helper_Abstract

View helper for processing shortcodes in text.

property Omeka_View_Helper_Shortcodes::$**shortcodeCallbacks**

protected array

List of predefined shortcodes.

Omeka_View_Helper_Shortcodes::**addShortcode**(*$shortcodeName*, *$callback*)
Add a new shortcode.

Parameters

- **$shortcodeName** (*string*) – Name of the shortcode
- **$callback** (*callback*) – Callback function that will return the shortcode content

Omeka_View_Helper_Shortcodes::**shortcodes**(*$content*)
Process any shortcodes in the given text.

Parameters

- **$content** (*string*) –

Returns string

Omeka_View_Helper_Shortcodes::**handleShortcode**(*$matches*)
Parse a detected shortcode and replace it with its actual content.

Parameters

- **$matches** (*array*) –

Returns string

Omeka_View_Helper_Shortcodes::**parseShortcodeAttributes**(*$text*)
Parse attributes section of a shortcode.

Parameters

- **$text** (*string*) –

Returns array

Omeka_View_Helper_Shortcodes::**shortcodeRecentItems**(*$args*, *$view*)
Shortcode for printing recently added items.

Parameters

- **$args** (*array*) –
- **$view** (Omeka_View) –

Returns string

Omeka_View_Helper_Shortcodes::**shortcodeFeaturedItems**(*$args*, *$view*)
Shortcode for printing featured items.

Parameters

- **$args** (*array*) –
- **$view** (Omeka_View) –

Returns string

Omeka_View_Helper_Shortcodes::**shortcodeItems**(*$args*, *$view*)
Shortcode for printing one or more items

Parameters

- **$args** (*array*) –
- **$view** (Omeka_View) –

Returns string

Omeka_View_Helper_Shortcodes::**shortcodeCollections**(*$args*, *$view*)
Shortcode for printing one or more collections

Parameters

- **$args** (*array*) –
- **$view** (Omeka_View) –

Returns string

Omeka_View_Helper_Shortcodes::**shortcodeRecentCollections**(*$args*, *$view*)
Shortcode for printing recent collections

Parameters

- **$args** (*array*) –
- **$view** (Omeka_View) –

Returns string

Omeka_View_Helper_Shortcodes::**shortcodeFeaturedCollections**(*$args*, *$view*)
Shortcode for printing featured collections

Parameters

- **$args** (*array*) –
- **$view** (Omeka_View) –

Returns string

Omeka_View_Helper_Shortcodes::**shortcodeFile**(*$args*, *$view*)
Shortcode for displaying a single file.

Parameters

- **$args** (*array*) –
- **$view** (Omeka_View) –

Returns string

3.5.22 Omeka_View_Helper_Singularize

Package: *View\Helper*

class Omeka_View_Helper_Singularize

extends Zend_View_Helper_Abstract

Omeka_View_Helper_Singularize::**singularize**(*$var*)

Parameters

- **$var** –

3.5.23 Omeka_View_Helper_Url

Package: *View\Helper*

class Omeka_View_Helper_Url

extends Zend_View_Helper_Abstract

Omeka_View_Helper_Url::**url**(*$options = array()*, *$name = null*, *$queryParams = array()*, *$reset = false*, *$encode = true*)

Generate a URL for use in one of Omeka's view templates.

There are two ways to use this method. The first way is for backwards compatibility with older versions of Omeka as well as ease of use for theme writers.

Here is an example of what URLs are generated by calling the function in different ways. The output from these examples assume that Omeka is running on the root of a particular domain, though that is of no importance to how the function works.

<code> echo $this->url('items/browse'); // outputs "/items/browse"

echo $this->url('items/browse', array('tags'=>'foo')); // outputs "/items/browse?tags=foo"

echo $this->url(array('controller'=>'items', 'action'=>'browse')); // outputs "/items/browse"

echo $this->url(array('controller'=>'items', 'action'=>'browse'), 'otherRoute', array('tags'=>'foo'),); // outputs "/miscellaneous?tags=foo" </code>

The first example takes a URL string exactly as one would expect it to be. This primarily exists for ease of use by theme writers. The second example appends a query string to the URL by passing it as an array. Note that in both examples, the first string must be properly URL-encoded in order to work. url('foo bar') would not work because of the space.

In the third example, the URL is being built directly from parameters passed to it. For more details on this, please see the Zend Framework's documentation.

In the last example, 'otherRoute' is the name of the route being used, as defined either in the routes.ini file or via a plugin. For examples of how to add routes via a plugin, please see Omeka's documentation.

Parameters

- **$options** (*string|array*) –
- **$name** (*string|null|array*) – Optional If $options is an array, $name should be the route name (string) or null. If $options is a string, $name should be the set of query string parameters (array) or null.
- **$queryParams** (*array*) – Optional Set of query string parameters.
- **$reset** (*bool*) – Optional Whether or not to reset the route parameters implied by the current request, e.g. if the current controller is 'items', then 'controller'=>'items' will be inferred when assembling the route.
- **$encode** (*bool*) –

Returns string

3.6 controllers

3.6.1 ApiController

Package: *Controller*

class ApiController

extends *Omeka_Controller_AbstractActionController*

The default controller for API resources.

ApiController::**init**()
Initialize this controller.

ApiController::**indexAction**()
Handle GET request without ID.

ApiController::**getAction**()
Handle GET request with ID.

ApiController::**postAction**()
Handle POST requests.

ApiController::**putAction**()
Handle PUT requests.

ApiController::**deleteAction**()
Handle DELETE requests.

ApiController::**_validateRecordType**(*$recordType*)
Validate a record type.

Parameters

- **$recordType** (*string*) –

ApiController::**_validateUser**(*Omeka_Record_AbstractRecord $record*, *$privilege*)
Validate a user against a privilege.

For GET requests, assume that records without an ACL resource do not require a permission check. Note that for POST, PUT, and DELETE, all records must define an ACL resource.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$privilege** (*string*) –

ApiController::**_getRecordAdapter**(*$recordType*)
Get the adapter for a record type.

Parameters

- **$recordType** (*string*) –

Returns Omeka_Record_Api_AbstractRecordAdapter

ApiController::**_setLinkHeader**(*$perPage*, *$page*, *$totalResults*, *$resource*)
Set the Link header for pagination.

Parameters

- **$perPage** (*int*) –
- **$page** (*int*) –
- **$totalResults** (*int*) –
- **$resource** (*string*) –

ApiController::**_getRepresentation**(*Omeka_Record_Api_AbstractRecordAdapter $recordAdapter*, *Omeka_Record_AbstractRecord $record*, *$resource*)
Get the representation of a record.

Parameters

- **$recordAdapter** (Omeka_Record_Api_AbstractRecordAdapter) –
- **$record** (Omeka_Record_AbstractRecord) –
- **$resource** (*string*) –

3.6.2 AppearanceController

Package: *Controller*

class AppearanceController

extends *Omeka_Controller_AbstractActionController*

AppearanceController::**indexAction**()

AppearanceController::**browseAction**()

AppearanceController::**editSettingsAction**()

AppearanceController::**editNavigationAction**()

AppearanceController::**resetNavigationConfirmAction**()

AppearanceController::**resetNavigationAction**()

AppearanceController::**_getResetForm**()

3.6.3 CollectionsController

Package: *Controller*

class CollectionsController

extends *Omeka_Controller_AbstractActionController*

property CollectionsController::$**_autoCsrfProtection**
protected

property CollectionsController::$**contexts**

property CollectionsController::$**_browseRecordsPerPage**
protected

CollectionsController::**init**()

CollectionsController::**showAction**()
The show collection action

CollectionsController::**addAction**()
The add collection action

CollectionsController::**editAction**()
The edit collection action

CollectionsController::**_getAddSuccessMessage**(*$collection*)

Parameters

- **$collection** –

CollectionsController::**_getEditSuccessMessage**(*$collection*)

Parameters

- **$collection** –

CollectionsController::**_getDeleteSuccessMessage**(*$collection*)

Parameters

- **$collection** –

CollectionsController::**_getDeleteConfirmMessage**(*$collection*)

Parameters

- **$collection** –

CollectionsController::**_getElementMetadata**(*$collection*, *$elementSetName*, *$elementName*)

Parameters

- **$collection** –
- **$elementSetName** –
- **$elementName** –

CollectionsController::**_getCollectionElementSets**()
Gets the element sets for the 'Collection' record type.

Returns array The element sets for the 'Collection' record type

CollectionsController::**_getBrowseDefaultSort**()

3.6.4 ElementSetsController

Package: *Controller*

class ElementSetsController

extends *Omeka_Controller_AbstractActionController*

ElementSetsController::**init**()

ElementSetsController::**_getDeleteConfirmMessage**(*$record*)

Parameters

- **$record** –

ElementSetsController::**addAction**()
Can't add element sets via the admin interface, so disable these actions from being POST'ed to.

ElementSetsController::**editAction**()

ElementSetsController::**_redirectAfterEdit**(*$record*)

Parameters

- **$record** –

3.6.5 ElementsController

Package: *Controller*

class ElementsController

extends *Omeka_Controller_AbstractActionController*

ElementsController::**elementFormAction**()

3.6.6 ErrorController

Package: *Controller*

class ErrorController

extends *Omeka_Controller_AbstractActionController*

ErrorController::**errorAction**()

ErrorController::**_getException**()

ErrorController::**notFoundAction**()
Generic action to render a 404 page.

ErrorController::**forbiddenAction**()

ErrorController::**methodNotAllowedAction**()

ErrorController::**logException**(*$e*, *$priority*)

Parameters

- **$e** –
- **$priority** –

ErrorController::**is404**(*Exception $e*, *$handler*)
Check to see whether the error qualifies as a 404 error

Parameters

- **$e** (*Exception*) –
- **$handler** –

Returns bool

ErrorController::**is403**(*Exception $e*)

Parameters

- **$e** (*Exception*) –

ErrorController::**renderException**(*Exception $e*)

Parameters

- **$e** (*Exception*) –

ErrorController::**isInDebugMode**()

3.6.7 FilesController

Package: *Controller*

class FilesController

extends *Omeka_Controller_AbstractActionController*

property FilesController::$**_autoCsrfProtection**
protected

property FilesController::$**contexts**

FilesController::**init**()

FilesController::**indexAction**()

FilesController::**browseAction**()

FilesController::**addAction**()

FilesController::**editAction**()

FilesController::**_getFileElementSets**()

FilesController::**_getDeleteConfirmMessage**(*$record*)

Parameters

- **$record** –

FilesController::**_redirectAfterDelete**(*$record*)

Parameters

- **$record** –

3.6.8 IndexController

Package: *Controller*

class IndexController

extends *Omeka_Controller_AbstractActionController*

IndexController::**indexAction**()

3.6.9 ItemTypesController

Package: *Controller*

class ItemTypesController

extends *Omeka_Controller_AbstractActionController*

ItemTypesController::**init**()

ItemTypesController::**addAction**()

ItemTypesController::**editAction**()

ItemTypesController::**addNewElementAction**()

ItemTypesController::**addExistingElementAction**()

ItemTypesController::**changeExistingElementAction**()

ItemTypesController::**_redirectAfterAdd**(*$itemType*)

Parameters

- **$itemType** –

ItemTypesController::**_getDeleteConfirmMessage**(*$itemType*)

Parameters

- **$itemType** –

ItemTypesController::**_getAddSuccessMessage**(*$itemType*)

Parameters

- **$itemType** –

ItemTypesController::**_getForm**(*$itemType*)

Parameters

- **$itemType** –

3.6.10 ItemsController

Package: *Controller*

class ItemsController

extends *Omeka_Controller_AbstractActionController*

property ItemsController::$**_autoCsrfProtection**
protected

property ItemsController::$**_browseRecordsPerPage**
protected

property ItemsController::$**contexts**

ItemsController::**init**()

ItemsController::**preDispatch**()

ItemsController::**searchAction**()
This shows the search form for items by going to the correct URI.

This form can be loaded as a partial by calling items_search_form().

ItemsController::**_getItemElementSets**()
Gets the element sets for the 'Item' record type.

Returns array The element sets for the 'Item' record type

ItemsController::**editAction**()
Adds an additional permissions check to the built-in edit action.

ItemsController::**_getAddSuccessMessage**(*$item*)

Parameters

- **$item** –

ItemsController::**_getEditSuccessMessage**(*$item*)

Parameters

- **$item** –

ItemsController::**_getDeleteSuccessMessage**(*$item*)

Parameters

- **$item** –

ItemsController::**_getDeleteConfirmMessage**(*$item*)

Parameters

- **$item** –

ItemsController::**_getElementMetadata**(*$item*, *$elementSetName*, *$elementName*)

Parameters

- **$item** –
- **$elementSetName** –
- **$elementName** –

ItemsController::**addAction**()

ItemsController::**tagsAction**()
Finds all tags associated with items (used for tag cloud)

ItemsController::**browseAction**()
Browse the items. Encompasses search, pagination, and filtering of request parameters. Should perhaps be split into a separate mechanism.

ItemsController::**_getBrowseDefaultSort**()

ItemsController::**changeTypeAction**()
Find or create an item for this mini-form

ItemsController::**batchEditAction**()
Batch editing of Items. If this is an AJAX request, it will render the 'batch-edit' as a partial.

ItemsController::**batchEditSaveAction**()
Processes batch edit information. Only accessible via POST.

ItemsController::**_batchEditAllSave**()
Processes batch edit all information. Only accessible via POST.

3.6.11 PluginsController

Package: *Controller*

class PluginsController

extends *Omeka_Controller_AbstractActionController*

PluginsController::**init**()

PluginsController::**configAction**()
Load the configuration form for a specific plugin. That configuration form will be POSTed back to this URL and processed by the plugin.

PluginsController::**installAction**()

PluginsController::**activateAction**()
Action to activate a plugin

PluginsController::**deactivateAction**()
Action to deactivate a plugin

PluginsController::**upgradeAction**()

PluginsController::**browseAction**()
Action to browse plugins

PluginsController::**uninstallAction**()
Action to uninstall a plugin

PluginsController::**deleteAction**()

PluginsController::**addAction**()

PluginsController::**_getPluginByName**(*$create = false*)
Retrieve the Plugin record based on the name passed via the request.

Parameters

- **$create** (*bool*) – Whether or not the plugin object should be created if it has not already been loaded.

3.6.12 RedirectorController

Package: *Controller*

class RedirectorController

extends *Omeka_Controller_AbstractActionController*

RedirectorController::**indexAction**()

3.6.13 ResourcesController

Package: *Controller*

class ResourcesController

extends *Omeka_Controller_AbstractActionController*

The controller for API /resources.

ResourcesController::**indexAction**()
Handle GET request without ID.

3.6.14 SearchController

Package: *Controller*

class SearchController

extends *Omeka_Controller_AbstractActionController*

property SearchController::$**_browseRecordsPerPage**
protected

SearchController::**init**()

SearchController::**indexAction**()

3.6.15 SettingsController

Package: *Controller*

class SettingsController

extends *Omeka_Controller_AbstractActionController*

SettingsController::**indexAction**()

SettingsController::**browseAction**()

SettingsController::**editSettingsAction**()

SettingsController::**editSecurityAction**()

SettingsController::**editSearchAction**()

SettingsController::**editItemTypeElementsAction**()

SettingsController::**editApiAction**()

SettingsController::**checkImagemagickAction**()
Determine whether or not ImageMagick has been correctly installed and configured.

In a few cases, this will indicate failure even though the ImageMagick program works properly. In those cases, users may ignore the results of this test. This is because the 'convert' command may have returned a non-zero status code for some reason. Keep in mind that a 0 status code always indicates success.

Returns bool True if the command line return status is 0 when attempting to run ImageMagick's convert utility, false otherwise.

SettingsController::**getFileExtensionWhitelistAction**()

SettingsController::**getFileMimeTypeWhitelistAction**()

SettingsController::**getHtmlPurifierAllowedHtmlElementsAction**()

SettingsController::**getHtmlPurifierAllowedHtmlAttributesAction**()

3.6.16 SiteController

Package: *Controller*

class SiteController

extends *Omeka_Controller_AbstractActionController*

The controller for API /site.

SiteController::**indexAction**()
Handle GET request without ID.

3.6.17 SystemInfoController

Package: *Controller*

class SystemInfoController

extends *Omeka_Controller_AbstractActionController*

SystemInfoController::**preDispatch**()

SystemInfoController::**indexAction**()

SystemInfoController::**_getInfoArray**()

SystemInfoController::**_addExtensionInfo**(*$info*)

Parameters

- **$info** –

SystemInfoController::**_addPluginInfo**(*$info*)

Parameters

- **$info** –

SystemInfoController::**_addThemeInfo**(*$info*)

Parameters

- **$info** –

3.6.18 TagsController

Package: *Controller*

class TagsController

extends *Omeka_Controller_AbstractActionController*

property TagsController::$**_browseRecordsPerPage**
protected

TagsController::**init**()

TagsController::**addAction**()

TagsController::**editAction**()

TagsController::**browseAction**()
Browse, filter and search tags

TagsController::**_getBrowseDefaultSort**()
Return the default sorting parameters to use when none are specified.

Returns array|null Array of parameters, with the first element being the sort_field parameter, and the second (optionally) the sort_dir.

TagsController::**autocompleteAction**()

TagsController::**renameAjaxAction**()

3.6.19 ThemesController

Package: *Controller*

class ThemesController

extends *Omeka_Controller_AbstractActionController*

ThemesController::**browseAction**()

ThemesController::**switchAction**()

ThemesController::**configAction**()
Load the configuration form for a specific theme. That configuration form will be POSTed back to this URL.

3.6.20 UpgradeController

Package: *Controller*

class UpgradeController

extends Zend_Controller_Action

UpgradeController::**__construct**(*Zend_Controller_Request_Abstract $request*, *Zend_Controller_Response_Abstract $response*, *$invokeArgs = array()*)

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –
- **$response** (*Zend_Controller_Response_Abstract*) –
- **$invokeArgs** –

UpgradeController::**indexAction**()

UpgradeController::**migrateAction**()
Run the migration script, obtain any success/error output and display it in a pretty way

UpgradeController::**_satisfiesPhpRequirement**()

UpgradeController::**_displayPhpRequirementMessage**()

3.6.21 UsersController

Package: *Controller*

class UsersController

extends *Omeka_Controller_AbstractActionController*

property UsersController::$**_publicActions**
protected array

Actions that are accessible by anonymous users.

property UsersController::$**_browseRecordsPerPage**
protected

UsersController::**init**()

UsersController::**_handlePublicActions**()
Peform common processing for the publicly accessible actions.

Set a view script variable for header and footer view scripts and don't allow logged-in users access.

The script variables are set for actions in $_publicActions, so the scripts for those actions should use these variables.

UsersController::**forgotPasswordAction**()
Send an email providing a link that allows the user to reset their password.

UsersController::**_sendResetPasswordEmail**(*$toEmail*, *$activationCode*)

Parameters

- **$toEmail** –
- **$activationCode** –

UsersController::**activateAction**()

UsersController::**addAction**()

UsersController::**editAction**()

Similar to 'add' action, except this requires a pre-existing record.

The ID For this record must be passed via the 'id' parameter.

UsersController::**changePasswordAction**()

UsersController::**apiKeysAction**()

UsersController::**browseAction**()

UsersController::**deleteAction**()

UsersController::**_getDeleteSuccessMessage**(*$record*)

Parameters

- **$record** –

UsersController::**_getDeleteConfirmMessage**(*$record*)

Parameters

- **$record** –

UsersController::**sendActivationEmail**(*$user*)

Send an activation email to a new user telling them how to activate their account.

Parameters

- **$user** (User) –

Returns bool True if the email was successfully sent, false otherwise.

UsersController::**loginAction**()

UsersController::**getLoginErrorMessages**(*Zend_Auth_Result $result*)

This exists to customize the messages that people see when their attempt to login fails. ZF has some built-in default messages, but it seems like those messages may not make sense to a majority of people using the software.

Parameters

- **$result** (*Zend_Auth_Result*) –

Returns string

UsersController::**logoutAction**()

UsersController::**_getUserForm**(*User $user*, *$ua = null*)

Parameters

- **$user** (User) –
- **$ua** –

UsersController::**_getLog**()

3.6.22 controllers/api

ApiController

Package: *Controller*

class ApiController

extends *Omeka_Controller_AbstractActionController*

The default controller for API resources.

ApiController::**init**()
Initialize this controller.

ApiController::**indexAction**()
Handle GET request without ID.

ApiController::**getAction**()
Handle GET request with ID.

ApiController::**postAction**()
Handle POST requests.

ApiController::**putAction**()
Handle PUT requests.

ApiController::**deleteAction**()
Handle DELETE requests.

ApiController::**_validateRecordType**(*$recordType*)
Validate a record type.

Parameters

- **$recordType** (*string*) –

ApiController::**_validateUser**(*Omeka_Record_AbstractRecord $record*, *$privilege*)
Validate a user against a privilege.

For GET requests, assume that records without an ACL resource do not require a permission check. Note that for POST, PUT, and DELETE, all records must define an ACL resource.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$privilege** (*string*) –

ApiController::**_getRecordAdapter**(*$recordType*)
Get the adapter for a record type.

Parameters

- **$recordType** (*string*) –

Returns Omeka_Record_Api_AbstractRecordAdapter

ApiController::**_setLinkHeader**(*$perPage*, *$page*, *$totalResults*, *$resource*)
Set the Link header for pagination.

Parameters

- **$perPage** (*int*) –
- **$page** (*int*) –

- **$totalResults** (*int*) –
- **$resource** (*string*) –

ApiController::**_getRepresentation**(*Omeka_Record_Api_AbstractRecordAdapter $recordAdapter*, *Omeka_Record_AbstractRecord $record*, *$resource*)

Get the representation of a record.

Parameters

- **$recordAdapter** (Omeka_Record_Api_AbstractRecordAdapter) –
- **$record** (Omeka_Record_AbstractRecord) –
- **$resource** (*string*) –

ElementSetsController

Package: *Controller*

class ElementSetsController

extends *ApiController*

The controller for API /element_sets.

ElementSetsController::**deleteAction**()
Handle DELETE requests.

ElementSetsController::**init**()
Initialize this controller.

ElementSetsController::**indexAction**()
Handle GET request without ID.

ElementSetsController::**getAction**()
Handle GET request with ID.

ElementSetsController::**postAction**()
Handle POST requests.

ElementSetsController::**putAction**()
Handle PUT requests.

ElementSetsController::**_validateRecordType**(*$recordType*)
Validate a record type.

Parameters

- **$recordType** (*string*) –

ElementSetsController::**_validateUser**(*Omeka_Record_AbstractRecord $record*, *$privilege*)

Validate a user against a privilege.

For GET requests, assume that records without an ACL resource do not require a permission check. Note that for POST, PUT, and DELETE, all records must define an ACL resource.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$privilege** (*string*) –

ElementSetsController::**_getRecordAdapter**(*$recordType*)
Get the adapter for a record type.

Parameters

- **$recordType** (*string*) –

Returns Omeka_Record_Api_AbstractRecordAdapter

ElementSetsController::**_setLinkHeader**(*$perPage*, *$page*, *$totalResults*, *$resource*)
Set the Link header for pagination.

Parameters

- **$perPage** (*int*) –
- **$page** (*int*) –
- **$totalResults** (*int*) –
- **$resource** (*string*) –

ElementSetsController::**_getRepresentation**(*Omeka_Record_Api_AbstractRecordAdapter $recordAdapter*, *Omeka_Record_AbstractRecord $record*, *$resource*)
Get the representation of a record.

Parameters

- **$recordAdapter** (Omeka_Record_Api_AbstractRecordAdapter) –
- **$record** (Omeka_Record_AbstractRecord) –
- **$resource** (*string*) –

ElementsController

Package: *Controller*

class ElementsController

extends *ApiController*

The controller for API /elements.

ElementsController::**deleteAction**()
Handle DELETE requests.

ElementsController::**init**()
Initialize this controller.

ElementsController::**indexAction**()
Handle GET request without ID.

ElementsController::**getAction**()
Handle GET request with ID.

ElementsController::**postAction**()
Handle POST requests.

ElementsController::**putAction**()
Handle PUT requests.

ElementsController::**_validateRecordType**(*$recordType*)
Validate a record type.

Parameters

- **$recordType** (*string*) –

ElementsController::**_validateUser**(*Omeka_Record_AbstractRecord $record*, *$privilege*)
Validate a user against a privilege.

For GET requests, assume that records without an ACL resource do not require a permission check. Note that for POST, PUT, and DELETE, all records must define an ACL resource.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$privilege** (*string*) –

ElementsController::**_getRecordAdapter**(*$recordType*)
Get the adapter for a record type.

Parameters

- **$recordType** (*string*) –

Returns Omeka_Record_Api_AbstractRecordAdapter

ElementsController::**_setLinkHeader**(*$perPage*, *$page*, *$totalResults*, *$resource*)
Set the Link header for pagination.

Parameters

- **$perPage** (*int*) –
- **$page** (*int*) –
- **$totalResults** (*int*) –
- **$resource** (*string*) –

ElementsController::**_getRepresentation**(*Omeka_Record_Api_AbstractRecordAdapter $recordAdapter*, *Omeka_Record_AbstractRecord $record*, *$resource*)
Get the representation of a record.

Parameters

- **$recordAdapter** (Omeka_Record_Api_AbstractRecordAdapter) –
- **$record** (Omeka_Record_AbstractRecord) –
- **$resource** (*string*) –

ErrorController

Package: *Controller*

class ErrorController

extends *Omeka_Controller_AbstractActionController*

constant ErrorController::**DEFAULT_HTTP_RESPONSE_CODE**
500 Internal Server Error

ErrorController::**errorAction**()
Handle all API errors.

FilesController

Package: *Controller*

class FilesController

extends *ApiController*

The controller for API /files.

constant FilesController::**FILE_NAME**
Name for file on a multipart/form-data request.

constant FilesController::**DATA_NAME**
Name for JSON data on a multipart/form-data request.

FilesController::**postAction**()
Handle POST requests.

FilesController::**init**()
Initialize this controller.

FilesController::**indexAction**()
Handle GET request without ID.

FilesController::**getAction**()
Handle GET request with ID.

FilesController::**putAction**()
Handle PUT requests.

FilesController::**deleteAction**()
Handle DELETE requests.

FilesController::**_validateRecordType**(*$recordType*)
Validate a record type.

Parameters

- **$recordType** (*string*) –

FilesController::**_validateUser**(*Omeka_Record_AbstractRecord $record*, *$privilege*)
Validate a user against a privilege.

For GET requests, assume that records without an ACL resource do not require a permission check. Note that for POST, PUT, and DELETE, all records must define an ACL resource.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$privilege** (*string*) –

FilesController::**_getRecordAdapter**(*$recordType*)
Get the adapter for a record type.

Parameters

- **$recordType** (*string*) –

Returns Omeka_Record_Api_AbstractRecordAdapter

FilesController::**_setLinkHeader**(*$perPage*, *$page*, *$totalResults*, *$resource*)
Set the Link header for pagination.

Parameters

- **$perPage** (*int*) –
- **$page** (*int*) –
- **$totalResults** (*int*) –
- **$resource** (*string*) –

FilesController::**_getRepresentation**(*Omeka_Record_Api_AbstractRecordAdapter $recordAdapter*, *Omeka_Record_AbstractRecord $record*, *$resource*)
Get the representation of a record.

Parameters

- **$recordAdapter** (Omeka_Record_Api_AbstractRecordAdapter) –
- **$record** (Omeka_Record_AbstractRecord) –
- **$resource** (*string*) –

ResourcesController

Package: *Controller*

class ResourcesController

extends *Omeka_Controller_AbstractActionController*

The controller for API /resources.

ResourcesController::**indexAction**()
Handle GET request without ID.

SiteController

Package: *Controller*

class SiteController

extends *Omeka_Controller_AbstractActionController*

The controller for API /site.

SiteController::**indexAction**()
Handle GET request without ID.

3.6.23 controllers/helpers

Omeka_Controller_Action_Helper_Acl

Package: *Controller\ActionHelper*

class Omeka_Controller_Action_Helper_Acl

extends Zend_Controller_Action_Helper_Abstract

Leverages the ACL to automatically check permissions for the current controller/action combo.

property Omeka_Controller_Action_Helper_Acl::$**_acl**
protected Zend_Acl

ACL object.

property Omeka_Controller_Action_Helper_Acl::$**_currentUser**
protected User|null

User record corresponding to the logged-in user, otherwise null.

property Omeka_Controller_Action_Helper_Acl::$**_allowed**
protected array

Temporarily allowed permissions.

property Omeka_Controller_Action_Helper_Acl::$**_autoloadResourceObject**
protected bool

Whether we should automatically try to set the resource object.

Omeka_Controller_Action_Helper_Acl::**__construct**(*$acl*, *$currentUser*)
Instantiated with the ACL permissions which will be used to verify permission levels.

Parameters

- **$acl** (*Zend_Acl*) –
- **$currentUser** –

Omeka_Controller_Action_Helper_Acl::**preDispatch**()
Determine whether or not access is granted to a specific controller/action.

If the user has been authenticated, display the Access Forbidden error page. Otherwise, give the user an opportunity to login before trying again.

Omeka_Controller_Action_Helper_Acl::**isAllowed**(*$privilege*, *$resource = null*)
Notifies whether the logged-in user has permission for a given resource/ privilege combination.

If an ACL resource being checked has not been defined, access to that resource should not be controlled. This allows plugin writers to implement controllers without also requiring them to be aware of the ACL.

Conversely, in the event that an ACL resource has been defined, all access permissions for that controller must be properly defined.

The names of resources should correspond to the name of the controller class minus 'Controller', e.g. Geolocation_IndexController -> 'Geolocation_Index' CollectionsController -> 'Collections'

Parameters

- **$privilege** (*string*) –
- **$resource** –

Returns bool

Omeka_Controller_Action_Helper_Acl::**getResourceName**()
Retrieve the name of the ACL resource based on the name of the controller and, if not the default module, the name of the module.

Returns string

Omeka_Controller_Action_Helper_Acl::**setCurrentUser**(*$currentUser*)

Parameters

- **$currentUser** (*User|null*) –

Omeka_Controller_Action_Helper_Acl::**setAllowed**(*$rule*, *$isAllowed = true*)
Temporarily override the ACL's permissions for this controller

Parameters

- **$rule** (*string*) –
- **$isAllowed** (*bool*) –

Omeka_Controller_Action_Helper_Acl::**setAutoloadResourceObject**(*$autoload*)
Set whether the ACL helper should try to automatically load a resource object from the request.

Parameters

- **$autoload** (*bool*) –

Omeka_Controller_Action_Helper_Acl::**_getResourceObjectFromRequest**()
Try to get the current resource object for the request.

Returns Zend_Acl_Resource_Interface|null

Omeka_Controller_Action_Helper_Acl::**_isLoginRequest**()

Omeka_Controller_Action_Helper_ContextSwitch

Package: *Controller\ActionHelper*

class Omeka_Controller_Action_Helper_ContextSwitch

extends Zend_Controller_Action_Helper_ContextSwitch

Extends the default ContextSwitch action helper to enable JSONP.

Omeka_Controller_Action_Helper_ContextSwitch::**postJsonContext**()
This extends the default ZF JSON serialization to work with JSONP, which enables cross-site AJAX using JSON.

Omeka_Controller_Action_Helper_Db

Package: *Controller\ActionHelper*

class Omeka_Controller_Action_Helper_Db

extends Zend_Controller_Action_Helper_Abstract

An action helper replacement for the database-oriented methods that were baked into Omeka_Controller_AbstractActionController in v1.x.

Omeka_Controller_Action_Helper_Db::**__construct**(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

Omeka_Controller_Action_Helper_Db::**init**()

Omeka_Controller_Action_Helper_Db::**__call**(*$method*, *$args*)
Delegate to the default table object for all other method calls.

Parameters

- **$method** –

- **$args** –

Omeka_Controller_Action_Helper_Db::**setDefaultModelName**(*$modelName*)
Set the class name corresponding to the default model.

Parameters

- **$modelName** –

Omeka_Controller_Action_Helper_Db::**getDefaultModelName**()

Omeka_Controller_Action_Helper_Db::**setDefaultTable**(*Omeka_Db_Table $table*)

Parameters

- **$table** (Omeka_Db_Table) –

Omeka_Controller_Action_Helper_Db::**getDb**()

Omeka_Controller_Action_Helper_Db::**getTable**(*$tableName = null*)

Parameters

- **$tableName** (*string|null*) –

Returns Omeka_Db_Table

Omeka_Controller_Action_Helper_Db::**findById**(*$id = null*, *$table = null*)
Find a particular record given its unique ID # and (optionally) its class name.

Parameters

- **$id** –
- **$table** –

Returns Omeka_Record_AbstractRecord

Omeka_Controller_Action_Helper_FlashMessenger

Package: *Controller\ActionHelper*

class Omeka_Controller_Action_Helper_FlashMessenger

extends Zend_Controller_Action_Helper_Abstract

FlashMessenger action helper.

Omeka_Controller_Action_Helper_FlashMessenger::**__construct**()

Omeka_Controller_Action_Helper_FlashMessenger::**addMessage**(*$message*, *$status = null*)
addMessage() - Add a message to flash message

Parameters

- **$message** –
- **$status** –

Returns Mu_Controller_Action_Helper_FlashMessenger Provides a fluent interface

Omeka_Controller_Action_Helper_FlashMessenger::**getMessages**()
getMessages() - Get messages

Returns array

Omeka_Controller_Action_Helper_FlashMessenger::**_filterMessages**(*$messages*)

Parameters

- **$messages** –

Omeka_Controller_Action_Helper_FlashMessenger::**clearMessages**()
Clear all messages from the previous request & specified status

Returns bool True if messages were cleared, false if none existed

Omeka_Controller_Action_Helper_FlashMessenger::**clearCurrentMessages**()
Clear all current messages with specified status

Returns bool True if messages were cleared, false if none existed

Omeka_Controller_Action_Helper_FlashMessenger::**hasMessages**()
Whether has messages with a specific status (or any messages, if null).

Omeka_Controller_Action_Helper_FlashMessenger::**hasCurrentMessages**()

Omeka_Controller_Action_Helper_FlashMessenger::**getCurrentMessages**(*$status = null*)

Parameters

- **$status** –

Omeka_Controller_Action_Helper_FlashMessenger::**direct**(*$message*, *$status = null*)
Strategy pattern: proxy to addMessage()

Parameters

- **$message** –
- **$status** –

Omeka_Controller_Action_Helper_JsonApi

Package: *Controller\ActionHelper*

class Omeka_Controller_Action_Helper_JsonApi

extends Zend_Controller_Action_Helper_Abstract

Output API JSON.

Omeka_Controller_Action_Helper_JsonApi::**direct**(*$data*)
Prepare the JSON, send it, and exit the application.

Parameters

- **$data** (*mixed*) –

Omeka_Controller_Action_Helper_Mail

Package: *Controller\ActionHelper*

class Omeka_Controller_Action_Helper_Mail

extends Zend_Controller_Action_Helper_Abstract

Action helper for sending email.

Omeka_Controller_Action_Helper_Mail::**__construct**(*Zend_View $view*)

Parameters

- **$view** (*Zend_View*) – View to render as the message body.

Omeka_Controller_Action_Helper_Mail::**__call**(*$method*, *$args*)

Delegate to the Zend_Mail instance.

Parameters

- **$method** (*string*) – Method called.
- **$args** (*array*) – Arguments to method.

Omeka_Controller_Action_Helper_Mail::**setSubjectPrefix**(*$prefix*)

Set the prefix for the subject header. Typically takes the form "[Site Name] ".

Parameters

- **$prefix** (*string*) – Subject prefix.

Omeka_Controller_Action_Helper_Mail::**setSubject**(*$subject*)

Set the subject of the email.

Parameters

- **$subject** (*string*) – Email subject.

Omeka_Controller_Action_Helper_Mail::**setBodyFromView**(*$viewScript*, *$html = false*)

Render the given view and use it as the body of the email.

Parameters

- **$viewScript** (*string*) – View script path.
- **$html** (*bool*) – Whether or not the assigned view script will render as HTML. Defaults to false.

Omeka_Controller_Action_Helper_Mail::**send**(*$transport = null*)

Send the email.

Parameters

- **$transport** (*Zend_Mail_Transport_Abstract*) – Optional defaults to null.

Omeka_Controller_Action_Helper_ThemeConfiguration

Package: *Controller\ActionHelper*

class Omeka_Controller_Action_Helper_ThemeConfiguration

extends Zend_Controller_Action_Helper_Abstract

Action helper for handling theme configuration.

Omeka_Controller_Action_Helper_ThemeConfiguration::**processForm**(*Zend_Form $form*, *$data*, *$originalOptions = array()*)

Process the theme configuration form.

For file elements, this will save them using the storage system or remove them as is necessary.

Parameters

- **$form** (*Zend_Form*) – The form to save.
- **$data** (*array*) – The data to fill the form with.
- **$originalOptions** (*array*) – The previous options for the form.

Returns array|bool Array of options if the form was validly submitted, false otherwise.

Omeka_Controller_Action_Helper_ThemeConfiguration::**_configFileElement**(*Zend_Form_Element_Fi $element*)

Ignore a file element that has an associated hidden element, since this means that the user did not change the uploaded file.

Parameters

- **$element** (*Zend_Form_Element_File*) –

Omeka_Controller_Action_Helper_ThemeConfiguration::**_processFileElement**(*Zend_Form_Element_I $element*)

Store and/or delete a file for a file element.

Parameters

- **$element** (*Zend_Form_Element_File*) –

Omeka_Controller_Action_Helper_ThemeConfiguration::**_deleteOldFile**(*Zend_Form_Element_File $element*)

Delete a previously-stored theme file.

Parameters

- **$element** (*Zend_Form_Element_File*) –

3.7 Packages

3.7.1 Acl

Up to *Packages*

Omeka_Acl_Assert_Ownership

Package: *Acl*

class Omeka_Acl_Assert_Ownership

implements Zend_Acl_Assert_Interface

Assertion to take account of "All" and "Self" sub-permissions for records.

A common use is the "edit" and "delete" permissions for Items and other "ownable" records.

Omeka_Acl_Assert_Ownership::**assert**(*Zend_Acl $acl*, *Zend_Acl_Role_Interface $role = null*, *Zend_Acl_Resource_Interface $resource = null*, *$privilege = null*)

Assert whether or not the ACL should allow access.

Parameters

- **$acl** (*Zend_Acl*) –
- **$role** (*Zend_Acl_Role_Interface*) –
- **$resource** (*Zend_Acl_Resource_Interface*) –
- **$privilege** –

Omeka_Acl_Assert_Ownership::**_userOwnsRecord**(*$user*, *$record*)

Check whether the user owns this specific record.

Parameters

- **$user** –
- **$record** –

Omeka_Acl_Assert_User

Package: *Acl*

class Omeka_Acl_Assert_User

implements Zend_Acl_Assert_Interface

Assert whether or not a specific user is allowed access to that person's user account data.

Omeka_Acl_Assert_User::**assert**(*Zend_Acl $acl*, *Zend_Acl_Role_Interface $role = null*, *Zend_Acl_Resource_Interface $resource = null*, *$privilege = null*)

Assert whether or not the ACL should allow access.

Assertions follow this logic:

Non-authenticated users (null role) have no access.

There exists a set of privileges (A) that are always allowed, provided that the user role and user resource are the same (editing own info, changing own password, etc.).

There also exists a set of privileges (B) that are always denied when performed on one's own user account (deleting own account, changing own role, etc.)

The super user can do anything that isn't on (B), e.g. the super user account cannot modify its own role.

All other users are limited to (A).

Parameters

- **$acl** (*Zend_Acl*) –
- **$role** (*Zend_Acl_Role_Interface*) –
- **$resource** (*Zend_Acl_Resource_Interface*) –
- **$privilege** –

Omeka_Acl_Assert_User::**_isAllowedSelf**(*$privilege*)

Parameters

- **$privilege** –

Omeka_Acl_Assert_User::**_isDeniedSelf**(*$privilege*)

Parameters

- **$privilege** –

Omeka_Acl_Assert_User::**_isSelf**(*$role*, *$resource*)

Parameters

- **$role** –
- **$resource** –

Omeka_Acl_Assert_User::**_isSuperUser**(*$user*)

Parameters

- **$user** –

3.7.2 Application

Up to *Packages*

Omeka_Application

Package: *Application*

class Omeka_Application

extends Zend_Application

Core class used to bootstrap the Omeka environment.

Various duties include, but are not limited to setting up class autoload, database, configuration files, logging, plugins, front controller, etc.

When any core resource returns from init(), the result is stored in the bootstrap container. Other parts of the application can get the resources from the bootstrap when needed.

Omeka_Application::**__construct**(*$environment*, *$options = null*)

Initialize the application.

Parameters

- **$environment** (*string*) – The environment name.
- **$options** (*string|array|Zend_Config*) – Application configuration.

Omeka_Application::**initialize**()
Bootstrap the entire application.

Omeka_Application::**run**()
Display the generic error page for all otherwise-uncaught exceptions.

Omeka_Application::**_displayErrorPage**(*$e*, *$title = null*)
Print an HTML page to display errors when starting the application.

Parameters

- **$e** (*Exception*) –
- **$title** (*string*) – The title of the error page.

Application\Resource

Up to *Application*

Omeka_Application_Resource_Acl

Package: *Application\Resource*

class Omeka_Application_Resource_Acl

extends Zend_Application_Resource_ResourceAbstract

Initializes Omeka's ACL.

property Omeka_Application_Resource_Acl::$**_acl**
protected Zend_Acl

Access control list object.

Omeka_Application_Resource_Acl::**init**()
Load the hardcoded ACL definitions, then apply definitions from plugins.

Returns Zend_Acl

Omeka_Application_Resource_Acl::**getAcl**()

Omeka_Application_Resource_Auth

Package: *Application\Resource*

class Omeka_Application_Resource_Auth

extends Zend_Application_Resource_ResourceAbstract

Authentication resource.

Omeka_Application_Resource_Auth::**init**()

Returns Zend_Auth

Omeka_Application_Resource_Cachemanager

Package: *Application\Resource*

class Omeka_Application_Resource_Cachemanager

extends Zend_Application_Resource_Cachemanager

Core resource for configuring caches for use by other components.

Omeka_Application_Resource_Cachemanager**::init**()

Omeka_Application_Resource_Config

Package: *Application\Resource*

class Omeka_Application_Resource_Config

extends Zend_Application_Resource_ResourceAbstract

Load the default configuration file for Omeka.

Omeka_Application_Resource_Config**::init**()

Returns Zend_Config_Ini

Omeka_Application_Resource_Currentuser

Package: *Application\Resource*

class Omeka_Application_Resource_Currentuser

extends Zend_Application_Resource_ResourceAbstract

Retrive the User record corresponding to the authenticated user.

If the user record is not retrievable (invalid ID), then the authentication ID will be cleared.

Omeka_Application_Resource_Currentuser**::init**()
Retrieve the User record associated with the authenticated user.

Returns User|null

Omeka_Application_Resource_Db

Package: *Application\Resource*

class Omeka_Application_Resource_Db

extends Zend_Application_Resource_Db

Set up the default database connection for Omeka.

constant Omeka_Application_Resource_Db**::INIT_COMMAND**
Command to run at connect time; currently sets the SQL mode

Omeka_Application_Resource_Db**::init**()

Returns Omeka_Db

Omeka_Application_Resource_Db::**setinipath**(*$path*)
Set the path to the database configuration file.

Allows {@link $_iniPath} to be set by the app configuration.

Parameters

- **$path** (*string*) –

Omeka_Application_Resource_Debug

Package: *Application\Resource*

class Omeka_Application_Resource_Debug

extends Zend_Application_Resource_ResourceAbstract

Sets up debugging output for web requests (if enabled).

Omeka_Application_Resource_Debug::**init**()

Omeka_Application_Resource_Exception

Package: *Application\Resource*

interface Omeka_Application_Resource_Exception
Marker interface.

For future exceptions thrown by Omeka_Application_Resource classes. This provides a pattern for differentiating setup/configuration errors.

Omeka_Application_Resource_Filederivatives

Package: *Application\Resource*

class Omeka_Application_Resource_Filederivatives

extends Zend_Application_Resource_ResourceAbstract

Bootstrap resource for configuring the derivative creator.

Omeka_Application_Resource_Filederivatives::**init**()

Omeka_Application_Resource_Frontcontroller

Package: *Application\Resource*

class Omeka_Application_Resource_Frontcontroller

extends Zend_Application_Resource_Frontcontroller

Front controller resource.

Omeka_Application_Resource_Frontcontroller::**init**()

Returns Zend_Controller_Front

Omeka_Application_Resource_Helpers

Package: *Application\Resource*

class Omeka_Application_Resource_Helpers

extends Zend_Application_Resource_ResourceAbstract

Initializes controller action helpers.

Omeka_Application_Resource_Helpers::**init**()

Omeka_Application_Resource_Helpers::**_initDbHelper**()

Omeka_Application_Resource_Helpers::**_initViewRenderer**()

Omeka_Application_Resource_Helpers::**_initResponseContexts**()
Define the custom response format contexts for Omeka.

Plugin writers should use the 'response_contexts' filter to modify or expand the list of formats that existing controllers may respond to.

Example of a definition of a response context through the ZF API:

$contexts->addContext('dc', array('suffix' => 'dc', 'headers' => array('Content-Type' => 'text/xml'), 'callbacks' => array('init' => 'atBeginningDoThis', 'post' => 'afterwardsDoThis')));

Omeka_Application_Resource_Helpers::**getDefaultResponseContexts**()
Returns the default response contexts for Omeka.

Returns array

Omeka_Application_Resource_Helpers::**_initAclHelper**()

Omeka_Application_Resource_Jobs

Package: *Application\Resource*

class Omeka_Application_Resource_Jobs

extends Zend_Application_Resource_ResourceAbstract

Bootstrap resource for configuring the job dispatcher.

Omeka_Application_Resource_Jobs::**init**()

Omeka_Application_Resource_Jobs_InvalidAdapterException

Package: *Application\Resource*

class Omeka_Application_Resource_Jobs_InvalidAdapterException

extends InvalidArgumentException

implements Throwable implements *Omeka_Application_Resource_Exception*

Exception thrown when an invalid job dispatcher has been configured.

property Omeka_Application_Resource_Jobs_InvalidAdapterException::$**message**
protected

property Omeka_Application_Resource_Jobs_InvalidAdapterException::$**code**
protected

property Omeka_Application_Resource_Jobs_InvalidAdapterException::$**file**
protected

property Omeka_Application_Resource_Jobs_InvalidAdapterException::$**line**
protected

Omeka_Application_Resource_Jobs_InvalidAdapterException::**__clone**()

Omeka_Application_Resource_Jobs_InvalidAdapterException::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Application_Resource_Jobs_InvalidAdapterException::**__wakeup**()

Omeka_Application_Resource_Jobs_InvalidAdapterException::**getMessage**()

Omeka_Application_Resource_Jobs_InvalidAdapterException::**getCode**()

Omeka_Application_Resource_Jobs_InvalidAdapterException::**getFile**()

Omeka_Application_Resource_Jobs_InvalidAdapterException::**getLine**()

Omeka_Application_Resource_Jobs_InvalidAdapterException::**getTrace**()

Omeka_Application_Resource_Jobs_InvalidAdapterException::**getPrevious**()

Omeka_Application_Resource_Jobs_InvalidAdapterException::**getTraceAsString**()

Omeka_Application_Resource_Jobs_InvalidAdapterException::**__toString**()

Omeka_Application_Resource_Locale

Package: *Application\Resource*

class Omeka_Application_Resource_Locale

extends Zend_Application_Resource_Locale

Core resource for configuring and loading the translation and locale components.

Omeka_Application_Resource_Locale::**init**()

Omeka_Application_Resource_Locale::**_setTranslate**(*$locale*, *$cache*)
Retrieve translation configuration options.

Parameters

- **$locale** –
- **$cache** –

Returns string

Omeka_Application_Resource_Logger

Package: *Application\Resource*

class Omeka_Application_Resource_Logger

extends Zend_Application_Resource_ResourceAbstract

If logging has been enabled in the config file, then set up Zend's logging mechanism.

Omeka_Application_Resource_Logger::**init**()

Returns Zend_Log

Omeka_Application_Resource_Logger::**_addMailWriter**(*Zend_Log $log*, *$toEmail*, *$filter = null*)

Set up debugging emails.

Parameters

- **$log** (*Zend_Log*) –
- **$toEmail** (*string*) – Email address of debug message recipient.
- **$filter** –

Omeka_Application_Resource_Mail

Package: *Application\Resource*

class Omeka_Application_Resource_Mail

extends Zend_Application_Resource_ResourceAbstract

Set up the mail transport that Omeka uses to send mail.

This makes use of Zend_Application_Resource_Mail for configuring the mail resource. config.ini can be set up using either the Zend Framework way or using the older Omeka configuration style (for backwards-compatibility), though the newer style is recommended.

Omeka_Application_Resource_Mail::**__construct**(*$options = null*)

Parameters

- **$options** –

Omeka_Application_Resource_Mail::**init**()

Returns Zend_Mail

Omeka_Application_Resource_Options

Package: *Application\Resource*

class Omeka_Application_Resource_Options

extends Zend_Application_Resource_ResourceAbstract

Retrieve all the options from the database.

Options are essentially site-wide variables that are stored in the database, for example the title of the site. Failure to load this resource currently indicates that Omeka needs to be installed.

Omeka_Application_Resource_Options::**init**()

Returns array

Omeka_Application_Resource_Options::**setInstallerRedirect**(*$flag*)

Parameters

- **$flag** –

Omeka_Application_Resource_Options::**_convertMigrationSchema**(*$options*)
If necessary, convert from the old sequentially-numbered migration scheme to the new timestamped migrations.

Parameters

- **$options** –

Omeka_Application_Resource_Pluginbroker

Package: *Application\Resource*

class Omeka_Application_Resource_Pluginbroker

extends Zend_Application_Resource_ResourceAbstract

Set up the plugin broker.

Omeka_Application_Resource_Pluginbroker::**init**()

Returns Omeka_Plugin_Broker

Omeka_Application_Resource_Plugins

Package: *Application\Resource*

class Omeka_Application_Resource_Plugins

extends Zend_Application_Resource_ResourceAbstract

Fire the 'initialize' hook for all installed plugins.

Note that this hook fires before the front controller has been initialized or dispatched.

Omeka_Application_Resource_Plugins::**init**()

Omeka_Application_Resource_Router

Package: *Application\Resource*

class Omeka_Application_Resource_Router

extends Zend_Application_Resource_Router

Set up the router and the built-in routes.

Omeka_Application_Resource_Router::**init**()

Returns Zend_Controller_Router_Rewrite

Omeka_Application_Resource_Router::**_addHomepageRoute**(*$router*)
Adds the homepage route to the router (as specified by the navigation settings page) The route will not be added if the user is currently on the admin theme.

Parameters

- **$router** (*Zend_Controller_Router_Rewrite*) – The router

Omeka_Application_Resource_Router::**_addHomepageRedirect**(*$uri*, *$router*)
Adds a redirect route for the homepage.

A redirect is required to make a "homepage" that is an external URL, an admin URL, or a URL with a query string.

Parameters

- **$uri** (*string*) – The absolute uri to redirect to the default route to
- **$router** (*Zend_Controller_Router_Rewrite*) – The router

Returns bool True if the route was successfully added, else false.

Omeka_Application_Resource_Router::**_leftTrim**(*$s*, *$n*)
Left trims the first occurrence of a string within a string. Note: it will only trim the first occurrence of the string.

Parameters

- **$s** (*string*) – The base string
- **$n** (*string*) – The string to remove from the left side of the base string

Returns string

Omeka_Application_Resource_Session

Package: *Application\Resource*

class Omeka_Application_Resource_Session

extends Zend_Application_Resource_Session

Initialize the session.

Customizes the session name to prevent session overlap between different applications that operate on the same server.

Omeka_Application_Resource_Session::**init**()

Omeka_Application_Resource_Session::**_getSessionConfig**()
Retrieve global session configuration options.

Returns array An array containing all the global configuration options for sessions. This array contains at least one key, 'name', corresponding to the name of the session, which is generated automatically if not provided.

Omeka_Application_Resource_Session::**_buildSessionName**()
Create a unique session name.

Hashes the base directory, this ensures that session names differ between Omeka instances on the same server.

Returns string

Omeka_Application_Resource_Session::**_setOptionsFromConfig**()

Omeka_Application_Resource_Session::**_canUseDbSessions**(*$options*)
Check if the DB is recent enough to use DB sessions by default.

Recent enough means that the DB version is 2.0 or higher. We can't use the DB sessions until the upgrade is complete to 2.0+.

Parameters

- **$options** (*array*) –

Returns bool

Omeka_Application_Resource_Storage

Package: *Application\Resource*

class Omeka_Application_Resource_Storage

extends Zend_Application_Resource_ResourceAbstract

Bootstrap resource for configuring the file storage layer.

Omeka_Application_Resource_Storage::**init**()

Omeka_Application_Resource_Theme

Package: *Application\Resource*

class Omeka_Application_Resource_Theme

extends Zend_Application_Resource_ResourceAbstract

Set up the controller plugin that determines theme view script paths.

property Omeka_Application_Resource_Theme::$**_basePath**
protected string

Theme base path. Set by application config.

property Omeka_Application_Resource_Theme::$**_webBasePath**
protected string

Theme base URI.

Set by application config.

Omeka_Application_Resource_Theme::**init**()

Omeka_Application_Resource_Theme::**setbasepath**(*$basePath*)
Set the base path for themes. Used to allow {@link $_basePath} to be set by application config.

Parameters

- **$basePath** (*string*) –

Omeka_Application_Resource_Theme::**setwebbasepath**(*$webBasePath*)
Set the base URI for themes. Used to allow {@link $_webBasePath} to be set by application config.

Parameters

- **$webBasePath** (*string*) –

Omeka_Application_Resource_View

Package: *Application\Resource*

class Omeka_Application_Resource_View

extends Zend_Application_Resource_ResourceAbstract

Initialize the view object.

Omeka_Application_Resource_View::**init**()
Register the view object so that it can be called by the view helpers.

3.7.3 Auth

Up to *Packages*

Omeka_Auth_Adapter_KeyTable

Package: *Auth*

class Omeka_Auth_Adapter_KeyTable

implements Zend_Auth_Adapter_Interface

Authenticate against an API key.

property Omeka_Auth_Adapter_KeyTable::$**_key**
protected

Omeka_Auth_Adapter_KeyTable::**__construct**(*$key = null*)

Parameters

- **$key** –

Omeka_Auth_Adapter_KeyTable::**authenticate**()
Authenticate against an API key.

Returns Zend_Auth_Result|null

Omeka_Auth_Adapter_UserTable

Package: *Auth*

class Omeka_Auth_Adapter_UserTable

extends Zend_Auth_Adapter_DbTable

Auth adapter that uses Omeka's users table for authentication.

Omeka_Auth_Adapter_UserTable::**__construct**(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) – Database object.

Omeka_Auth_Adapter_UserTable::**_authenticateValidateResult**(*$resultIdentity*)
Validate the identity returned from the database.

Overrides the Zend implementation to provide user IDs, not usernames upon successful validation.

Parameters

- **$resultIdentity** (*array*) –

3.7.4 Captcha

Up to *Packages*

Omeka_Captcha

Package: *Captcha*

class Omeka_Captcha

Factory for creating a captcha for use when soliciting public input.

getCaptcha()

Get a captcha object implementing Zend's captcha API.

Returns Zend_Captcha_Adapter|null

isConfigured()

Return whether the captcha is configured. If this returns true, getCaptcha will not return null.

Returns bool

3.7.5 Controller

Up to *Packages*

Omeka_Controller_AbstractActionController

Package: *Controller*

class Omeka_Controller_AbstractActionController

extends Zend_Controller_Action

Base class for Omeka controllers.

Provides basic create, read, update, and delete (CRUD) operations.

property Omeka_Controller_AbstractActionController::$**_browseRecordsPerPage**

protected string

The number of records to browse per page.

If this is left null, then results will not paginate. This is partially because not every controller will want to paginate records and also to avoid BC breaks for plugins.

Setting this to self::RECORDS_PER_PAGE_SETTING will cause the admin-configured page limits to be used (which is often what you want).

property Omeka_Controller_AbstractActionController::$**_autoCsrfProtection**

protected bool

Whether to automatically generate and check for a CSRF token on add and edit.

If set to true, a variable $csrf will be assigned to the add and edit views, you must echo it inside the form on those pages, or else the requests will fail.

Note: default deletion always uses a token, regardless of this setting.

Omeka_Controller_AbstractActionController::**__construct**(*Zend_Controller_Request_Abstract $request*, *Zend_Controller_Response_Abstract $response*, *$invokeArgs = array()*)

Base controller constructor.

Does the following things:

- Aliases the redirector helper to clean up the syntax
- Sets the table object automatically if given the class of the model to

use for CRUD. - Sets all the built-in action contexts for the CRUD actions.

Instead of overriding this constructor, controller subclasses should implement the init() method for initial setup.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) – Current request object.
- **$response** (*Zend_Controller_Response_Abstract*) – Response object.
- **$invokeArgs** (*array*) – Arguments passed to Zend_Controller_Action.

Omeka_Controller_AbstractActionController::**indexAction**()

Forward to the 'browse' action

Omeka_Controller_AbstractActionController::**browseAction**()

Retrieve and render a set of records for the controller's model.

Using this action requires some setup:

- In your controller's `init()`, set the default model name

`$this->_helper->db->setDefaultModelName('YourRecord');` - In your controller, set the records per page and return them using: `protected function _getBrowseRecordsPerPage();` - In your table record, filter the select object using the provided parameters using: `public function applySearchFilters($select, $params);`

Omeka_Controller_AbstractActionController::**showAction**()

Retrieve a single record and render it.

Every request to this action must pass a record ID in the 'id' parameter.

Omeka_Controller_AbstractActionController::**addAction**()

Add an instance of a record to the database.

This behaves differently based on the contents of the $_POST superglobal. If the $_POST is empty or invalid, it will render the form used for data entry. Otherwise, if the $_POST exists and is valid, it will save the new record and redirect to the 'browse' action.

Omeka_Controller_AbstractActionController::**editAction**()

Similar to 'add' action, except this requires a pre-existing record.

Every request to this action must pass a record ID in the 'id' parameter.

Omeka_Controller_AbstractActionController::**deleteConfirmAction**()
Ask for user confirmation before deleting a record.

Omeka_Controller_AbstractActionController::**deleteAction**()
Delete a record from the database.

Every request to this action must pass a record ID in the 'id' parameter.

Omeka_Controller_AbstractActionController::**getCurrentUser**()
Return the record for the current user.

Returns User|bool User object if a user is logged in, false otherwise.

Omeka_Controller_AbstractActionController::**_getBrowseRecordsPerPage**(*$pluralName = null*)
Return the number of records to display per page.

By default this will read from the _browseRecordsPerPage property, which in turn defaults to null, disabling pagination. This can be overridden in subclasses by redefining the property or this method.

Setting the property to self::RECORDS_PER_PAGE_SETTING will enable pagination using the admin-configued page limits.

Parameters

- **$pluralName** (*string|null*) –

Returns int|null

Omeka_Controller_AbstractActionController::**_getBrowseDefaultSort**()
Return the default sorting parameters to use when none are specified.

Returns array|null Array of parameters, with the first element being the sort_field parameter, and the second (optionally) the sort_dir.

Omeka_Controller_AbstractActionController::**_getAddSuccessMessage**(*$record*)
Return the success message for adding a record.

Default is empty string. Subclasses should override it.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns string

Omeka_Controller_AbstractActionController::**_getEditSuccessMessage**(*$record*)
Return the success message for editing a record.

Default is empty string. Subclasses should override it.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns string

Omeka_Controller_AbstractActionController::**_getDeleteSuccessMessage**(*$record*)
Return the success message for deleting a record.

Default is empty string. Subclasses should override it.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns string

Omeka_Controller_AbstractActionController::**_getDeleteConfirmMessage**(*$record*)
Return the delete confirm message for deleting a record.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns string

Omeka_Controller_AbstractActionController::**_redirectAfterAdd**(*$record*)
Redirect to another page after a record is successfully added.

The default is to reidrect to this controller's browse page.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Omeka_Controller_AbstractActionController::**_redirectAfterEdit**(*$record*)
Redirect to another page after a record is successfully edited.

The default is to redirect to this record's show page.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Omeka_Controller_AbstractActionController::**_redirectAfterDelete**(*$record*)
Redirect to another page after a record is successfully deleted.

The default is to redirect to this controller's browse page.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Omeka_Controller_AbstractActionController::**_setActionContexts**()
Augment Zend's default action contexts.

Passes Omeka's default additional contexts through the 'action_contexts' filter to allow plugins to add contexts.

Omeka_Controller_AbstractActionController::**_getDeleteForm**()
Get the form used for confirming deletions.

Returns Zend_Form

Omeka_Controller_Exception_403

Package: *Controller*

class Omeka_Controller_Exception_403

extends Exception

implements Throwable

If thrown by a controller, this exception will be caught within the ErrorController, which will then render a 403 Forbidden page.

property Omeka_Controller_Exception_403::$**message**
protected

property Omeka_Controller_Exception_403::$**code**
protected

property Omeka_Controller_Exception_403::$**file**
protected

property Omeka_Controller_Exception_403::$**line**
protected

Omeka_Controller_Exception_403::**__clone**()

Omeka_Controller_Exception_403::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Controller_Exception_403::**__wakeup**()

Omeka_Controller_Exception_403::**getMessage**()

Omeka_Controller_Exception_403::**getCode**()

Omeka_Controller_Exception_403::**getFile**()

Omeka_Controller_Exception_403::**getLine**()

Omeka_Controller_Exception_403::**getTrace**()

Omeka_Controller_Exception_403::**getPrevious**()

Omeka_Controller_Exception_403::**getTraceAsString**()

Omeka_Controller_Exception_403::**__toString**()

Omeka_Controller_Exception_404

Package: *Controller*

class Omeka_Controller_Exception_404

extends Exception

implements Throwable

If thrown within a controller, this will be caught in the ErrorController, which will render a 404 Not Found page.

property Omeka_Controller_Exception_404::$**message**
protected

property Omeka_Controller_Exception_404::$**code**
protected

property Omeka_Controller_Exception_404::$**file**
protected

property Omeka_Controller_Exception_404::$**line**
protected

Omeka_Controller_Exception_404::**__clone**()

Omeka_Controller_Exception_404::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Controller_Exception_404::**__wakeup**()

Omeka_Controller_Exception_404::**getMessage**()

Omeka_Controller_Exception_404::**getCode**()

Omeka_Controller_Exception_404::**getFile**()

Omeka_Controller_Exception_404::**getLine**()

Omeka_Controller_Exception_404::**getTrace**()

Omeka_Controller_Exception_404::**getPrevious**()

Omeka_Controller_Exception_404::**getTraceAsString**()

Omeka_Controller_Exception_404::**__toString**()

Omeka_Controller_Exception_Api

Package: *Controller*

class Omeka_Controller_Exception_Api

extends Exception

implements Throwable

API exception.

API implementers should throw this exception for controller errors.

property Omeka_Controller_Exception_Api::$**_errors**
protected array

property Omeka_Controller_Exception_Api::$**message**
protected

property Omeka_Controller_Exception_Api::$**code**
protected

property Omeka_Controller_Exception_Api::$**file**
protected

property Omeka_Controller_Exception_Api::$**line**
protected

Omeka_Controller_Exception_Api::**__construct**(*$message*, *$code*, *$errors = array()*)

Parameters

- **$message** (*string*) –
- **$code** (*int*) –
- **$errors** (*array*) – Custom errors

Omeka_Controller_Exception_Api::**getErrors**()

Returns array

Omeka_Controller_Exception_Api::**__clone**()

Omeka_Controller_Exception_Api::**__wakeup**()

Omeka_Controller_Exception_Api::**getMessage**()

Omeka_Controller_Exception_Api::**getCode**()

Omeka_Controller_Exception_Api::**getFile**()

Omeka_Controller_Exception_Api::**getLine**()

Omeka_Controller_Exception_Api::**getTrace**()

Omeka_Controller_Exception_Api::**getPrevious**()

Omeka_Controller_Exception_Api::**getTraceAsString**()

Omeka_Controller_Exception_Api::**__toString**()

Controller\ActionHelper

Up to *Controller*

Controller\Plugin

Up to *Controller*

Omeka_Controller_Plugin_Admin

Package: *Controller\Plugin*

class Omeka_Controller_Plugin_Admin

extends Zend_Controller_Plugin_Abstract

This controller plugin allows for all functionality that is specific to the Admin theme.

For now, all this includes is preventing unauthenticated access to all admin pages, with the exception of a few white-listed URLs, which are stored in this plugin.

This controller plugin should be loaded only in the admin bootstrap.

property Omeka_Controller_Plugin_Admin::$**_adminWhitelist**
protected string

Controller/Action list for admin actions that do not require being logged-in

Omeka_Controller_Plugin_Admin::**routeStartup**(*Zend_Controller_Request_Abstract $request*)
Direct requests to the admin interface. Called upon router startup, before the request is routed.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –

Omeka_Controller_Plugin_Admin::**preDispatch**(*Zend_Controller_Request_Abstract $request*)
Require login when attempting to access the admin interface. Whitelisted controller/action combinations are exempt from this requirement. Called before dispatching.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –

Omeka_Controller_Plugin_Admin::**getRedirector**()
Return the redirector action helper.

Returns Zend_Controller_Action_Helper_Redirector

Omeka_Controller_Plugin_Admin::**getAuth**()
Return the auth object.

Returns Zend_Auth

Omeka_Controller_Plugin_Admin::**_requireLogin**(*$request*)
Determine whether or not the request requires an authenticated user.

Parameters

- **$request** –

Returns bool

Omeka_Controller_Plugin_Api

Package: *Controller\Plugin*

class Omeka_Controller_Plugin_Api

extends Zend_Controller_Plugin_Abstract

property Omeka_Controller_Plugin_Api::$**_apiResources**
protected The

Use the "api_resources" filter to add resources, following this format:

<code> // For the path: /api/your_resources/:id 'your_resources' => array(// Module associated with your resource. 'module' => 'your-plugin-name', // Controller associated with your resource. 'controller' => 'your-resource-controller', // Type of record associated with your resource. 'record_type' => 'YourResourceRecord', // List of actions available for your resource. 'actions' => array('index', // GET request without ID 'get', // GET request with ID 'post', // POST request 'put', // PUT request (ID is required) 'delete', // DELETE request (ID is required)), // List of GET parameters available for your index action. 'index_params' => array('foo', 'bar'),) </code>

If not given, "module" and "controller" fall back to their defaults, "default" and "api". Resources using the default controller MUST include a "record_type". Remove "actions" that are not wanted or not implemented.

Omeka_Controller_Plugin_Api::**routeStartup**(*Zend_Controller_Request_Abstract $request*)
Handle API-specific controller logic.

Via Omeka_Application_Resource_Frontcontroller, this plugin is only registered during an API request.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –

Omeka_Controller_Plugin_Api::**getApiResources**()
Get the filtered API resources.

Returns array

Omeka_Controller_Plugin_Debug

Package: *Controller\Plugin*

class Omeka_Controller_Plugin_Debug

extends Zend_Controller_Plugin_Abstract

This controller plugin allows for debugging Request objects without inserting debugging code into the Zend Framework code files.

Debugging web requests is enabled by setting 'debug.request = true' in the config.ini file.

Omeka_Controller_Plugin_Debug**::preDispatch**(*Zend_Controller_Request_Abstract $request*)

Print request debugging info for every request.

Has no effect if request debugging is not enabled in config.ini.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) – Request object.

Omeka_Controller_Plugin_Debug**::postDispatch**(*Zend_Controller_Request_Abstract $request*)

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –

Omeka_Controller_Plugin_Debug**::dispatchLoopShutdown**()

Print database profiling info.

Enabled conditionally when debug.profileDb = true in config.ini.

Omeka_Controller_Plugin_Debug**::_getProfilerMarkup**(*Zend_Db_Profiler $profiler*)

Parameters

- **$profiler** (*Zend_Db_Profiler*) –

Omeka_Controller_Plugin_Debug**::_getRequestMarkup**(*$request*, *$router*)

Create HTML markup for request debugging.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) – Request object.
- **$router** (*Zend_Controller_Router_Interface*) – Router object.

Returns string HTML markup.

Omeka_Controller_Plugin_DefaultContentType

Package: *Controller\Plugin*

class Omeka_Controller_Plugin_DefaultContentType

extends Zend_Controller_Plugin_Abstract

This controller plugin sets the default Content-Type header when one hasn't been set at the end of the controller processing.

This has to be done here because Zend allows header duplication, the output contexts don't overwrite headers of the same name, and some servers (FastCGI) choke when they see two Content-Type headers.

Omeka_Controller_Plugin_DefaultContentType::**dispatchLoopShutdown**()
Add a default Content-Type to the response if none is already set.

Omeka_Controller_Plugin_HtmlPurifier

Package: *Controller\Plugin*

class Omeka_Controller_Plugin_HtmlPurifier

extends Zend_Controller_Plugin_Abstract

This ZF controller plugin allows the HtmlPurifier to filter the existing forms (items, collections, users, etc.) so that fields that are allowed to contain HTML are properly filtered.

Note that this will not operate on any of the plugins.

Omeka_Controller_Plugin_HtmlPurifier::**routeStartup**(*Zend_Controller_Request_Abstract $request*)
Add the HtmlPurifier options if needed.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –

Omeka_Controller_Plugin_HtmlPurifier::**preDispatch**(*Zend_Controller_Request_Abstract $request*)
Determine whether or not to filter form submissions for various controllers.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –

Omeka_Controller_Plugin_HtmlPurifier::**isFormSubmission**(*$request*)
Determine whether or not the request contains a form submission to either the 'add', 'edit', or 'config' actions.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –

Returns bool

Omeka_Controller_Plugin_HtmlPurifier::**filterCollectionsForm**(*$request*, *$htmlPurifierFilter = null*)
Filter the Collections form post, including the 'Elements' array of the POST.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –
- **$htmlPurifierFilter** (Omeka_Filter_HtmlPurifier) –

Omeka_Controller_Plugin_HtmlPurifier::**filterThemesForm**(*$request*, *$htmlPurifierFilter = null*)
Purify all of the data in the theme settings

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –

- **$htmlPurifierFilter** (Omeka_Filter_HtmlPurifier) –

Omeka_Controller_Plugin_HtmlPurifier::**_purifyArray**(*$dataArray = array()*, *$htmlPurifierFilter = null*)

Recurisvely purify an array

Parameters

- **$dataArray** –
- **$htmlPurifierFilter** (Omeka_Filter_HtmlPurifier) –

Returns array A purified array of string or array values

Omeka_Controller_Plugin_HtmlPurifier::**filterItemsForm**(*$request*, *$htmlPurifierFilter = null*)

Filter the Items form post, including the 'Elements' array of the POST.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –
- **$htmlPurifierFilter** (Omeka_Filter_HtmlPurifier) –

Omeka_Controller_Plugin_HtmlPurifier::**_filterElementsFromPost**(*$post*, *$htmlPurifierFilter = null*)

Filter the 'Elements' array of the POST.

Parameters

- **$post** (*Zend_Controller_Request_Abstract*) –
- **$htmlPurifierFilter** (Omeka_Filter_HtmlPurifier) –

Omeka_Controller_Plugin_HtmlPurifier::**_setupHtmlPurifierOptions**()

Omeka_Controller_Plugin_Ssl

Package: *Controller\Plugin*

class Omeka_Controller_Plugin_Ssl

extends Zend_Controller_Plugin_Abstract

Handle SSL configuration for Omeka sites.

Omeka_Controller_Plugin_Ssl::**__construct**(*$sslConfig*, *$redirector*, *Zend_Auth $auth*)

Parameters

- **$sslConfig** –
- **$redirector** –
- **$auth** (*Zend_Auth*) –

Omeka_Controller_Plugin_Ssl::**routeStartup**(*Zend_Controller_Request_Abstract $request*)

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –

Omeka_Controller_Plugin_Ssl::**preDispatch**(*Zend_Controller_Request_Abstract $request*)

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) –

Omeka_Controller_Plugin_Ssl::**_isLoginRequest**(*$request*)

Parameters

- **$request** –

Omeka_Controller_Plugin_Ssl::**_secureAuthenticatedSession**()

Unauthenticated sessions are not as valuable to attackers, so we only really need to check if an authenticated session is being used.

Omeka_Controller_Plugin_Ssl::**_isSslRequest**(*$request*)

Parameters

- **$request** –

Omeka_Controller_Plugin_Ssl::**_redirect**(*$request*)

Parameters

- **$request** –

Omeka_Controller_Plugin_Ssl::**_secureAllRequests**()

Omeka_Controller_Plugin_Upgrade

Package: *Controller\Plugin*

class Omeka_Controller_Plugin_Upgrade

extends Zend_Controller_Plugin_Abstract

Overrides Omeka's normal routing when the database needs to be upgraded.

Omeka_Controller_Plugin_Upgrade::**dispatchLoopStartup**(*Zend_Controller_Request_Abstract $request*)

Set up routing for the upgrade controller.

Only allows authorized users to upgrade, and blocks the public site when an upgrade is needed.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) – Request object.

Omeka_Controller_Plugin_Upgrade::**_dbNeedsUpgrade**()

Omeka_Controller_Plugin_Upgrade::**_dbCanUpgrade**()

Omeka_Controller_Plugin_Upgrade::**_upgrade**(*$request*)

Redirect to the upgrade controller.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) – Request object (not used).

Omeka_Controller_Plugin_ViewScripts

Package: *Controller\Plugin*

class Omeka_Controller_Plugin_ViewScripts

extends Zend_Controller_Plugin_Abstract

Sets up view script search paths on a per-request basis.

property Omeka_Controller_Plugin_ViewScripts::$**_view**
protected Zend_View

Registered view object.

property Omeka_Controller_Plugin_ViewScripts::$**_dbOptions**
protected array

List of options from the database.

property Omeka_Controller_Plugin_ViewScripts::$**_baseThemePath**
protected string

Base path to themes directory.

property Omeka_Controller_Plugin_ViewScripts::$**_webBaseThemePath**
protected string

Base web-accesible path to themes.

property Omeka_Controller_Plugin_ViewScripts::$**_pluginMvc**
protected Omeka_Plugin_Mvc

MVC plugin behaviors class.

Omeka_Controller_Plugin_ViewScripts::**__construct**(*$options*, *Omeka_Plugin_Mvc $pluginMvc*)

Parameters

- **$options** (*array*) – List of options.
- **$pluginMvc** (Omeka_Plugin_Mvc) – Plugin MVC class.

Omeka_Controller_Plugin_ViewScripts::**preDispatch**(*Zend_Controller_Request_Abstract $request*)
Add the appropriate view scripts directories for a given request. This is pretty much the glue between the plugin broker and the View object, since it uses data from the plugin broker to determine what script paths will be available to the view.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) – Request object.

Omeka_Controller_Plugin_ViewScripts::**_addPathsToView**(*$paths*)
Add multiple script paths.

Parameters

- **$paths** (*array*) – The paths to add.

Omeka_Controller_Plugin_ViewScripts::**_addPathToView**(*$scriptPath*)
Add a new script path for a plugin to the view.

Parameters

- **$scriptPath** (*string*) – Path from plugins dir to script dir.

Omeka_Controller_Plugin_ViewScripts::**_getView**()
Gets the view from the registry.

The initial call to the registry caches the view in this class.

Returns Zend_View

Omeka_Controller_Plugin_ViewScripts::**_addSharedViewsDir**()
Add the global views from the view scripts directory to the view.

Omeka_Controller_Plugin_ViewScripts::**_addThemePaths**(*$theme*)
Add script and asset paths for a theme to the view.

Parameters

- **$theme** (*string*) – Theme type; either 'public' or 'admin'.

Omeka_Controller_Plugin_ViewScripts::**_addOverridePathForPlugin**(*$theme*, *$pluginModuleName*)
Add theme view path for override views for a given plugin.

Parameters

- **$theme** (*string*) – Theme type; 'public' or 'admin'
- **$pluginModuleName** (*string*) –

Omeka_Controller_Plugin_ViewScripts::**getThemeOption**(*$type*)
Retrieve the option from the database that contains the directory of the theme to render.

Parameters

- **$type** (*string*) – Currently either 'admin' or 'public'.

Returns string

Controller\Router

Up to *Controller*

Omeka_Controller_Router_Api

Package: *Controller\Router*

class Omeka_Controller_Router_Api

extends Zend_Controller_Router_Route_Abstract

Router for the Omeka API.

constant Omeka_Controller_Router_Api::**DEFAULT_MODULE**
The default controller name.

constant Omeka_Controller_Router_Api::**DEFAULT_CONTROLLER**
The default controller name.

property Omeka_Controller_Router_Api::$**_legalActions**
protected All

property Omeka_Controller_Router_Api::$**_legalParams**
protected GET

property Omeka_Controller_Router_Api::$**_legalIndexParams**
protected GET

Omeka_Controller_Router_Api::**getInstance**(*Zend_Config $config*)

Parameters

- **$config** (*Zend_Config*) –

Omeka_Controller_Router_Api::**match**(*$request*)
Match the user submitted path.

Via Omeka_Application_Resource_Router, this is the only available route for API requests.

Parameters

- **$request** (*Zend_Controller_Request_Http*) –

Returns array|false

Omeka_Controller_Router_Api::**assemble**(*$data = array()*, *$reset = false*, *$encode = false*)

Parameters

- **$data** –
- **$reset** –
- **$encode** –

Omeka_Controller_Router_Api::**_getResource**(*$resource*, *$apiResources*)
Return this route's resource.

Parameters

- **$resource** (*string*) –
- **$apiResources** (*array*) –

Returns string

Omeka_Controller_Router_Api::**_getRecordType**(*$resource*, *$apiResources*)
Return this route's record type.

Parameters

- **$resource** (*string*) –
- **$apiResources** (*array*) –

Returns string|null

Omeka_Controller_Router_Api::**_getModule**(*$resource*, *$apiResources*)
Return this route's module.

Parameters

- **$resource** (*string*) –
- **$apiResources** (*array*) –

Returns string

Omeka_Controller_Router_Api::**_getController**(*$resource*, *$apiResources*)
Return this route's controller.

Parameters

- **$resource** (*string*) –
- **$apiResources** (*array*) –

Returns string

Omeka_Controller_Router_Api::**_getAction**(*$method*, *$params*, *$resource*, *$apiResources*)
Return this route's action.

Parameters

- **$method** (*string*) –
- **$params** (*array*) –
- **$resource** (*string*) –
- **$apiResources** (*array*) –

Returns string

Omeka_Controller_Router_Api::**_validateParams**(*$action*, *$resource*, *$apiResources*)
Validate the GET parameters against the whitelist.

Parameters

- **$action** (*string*) –
- **$resource** (*string*) –
- **$apiResources** (*array*) –

3.7.6 Db

Up to *Packages*

Omeka_Db

Package: *Db*

class Omeka_Db
Database manager object for Omeka

While mostly a wrapper for a Zend_Db_Adapter instance, this also provides shortcuts for retrieving table objects and table names for use in SQL.

property prefix
string|null

The prefix that every table in the omeka database will use.

property _adapter
protected Zend_Db_Adapter_Abstract

The database adapter.

property _tables
protected array

All the tables that are currently managed by this database object.

__construct (*$adapter*, *$prefix* = *null*)

Parameters

- **$adapter** (*Zend_Db_Adapter_Abstract*) – A Zend Framework connection object.
- **$prefix** (*string*) – The prefix for the database tables, if applicable.

__call (*$m*, *$a*)
Delegate to the database adapter.

Parameters

- **$m** (*string*) – Method name.
- **$a** (*array*) – Method arguments.

Returns mixed

__get (*$name*)
Magic getter is a synonym for Omeka_Db::getTableName().

Example: $db->Item is equivalent to $db->getTableName('Item').

Parameters

- **$name** (*string*) – Property name; table model class name in this case.

Returns string|null

setLogger (*$logger*)
Set logger for SQL queries.

Parameters

- **$logger** (*Zend_Log*) –

getAdapter ()
Retrieve the database adapter.

Returns Zend_Db_Adapter_Abstract

getTableName (*$class*)
Retrieve the name of the table (including the prefix).

Parameters

- **$class** –

Returns string

hasPrefix ()
Check whether the database tables have a prefix.

Returns bool

getTable (*$class*)
Retrieve a table object corresponding to the model class.

Table classes can be extended by inheriting off of Omeka_Db_Table and then calling your table Table_ModelName, e.g. Table_Item or Table_Collection. For backwards compatibility you may call your table ModelNameTable, i.e. ItemTable or CollectionTable. The latter naming pattern is deprecated.

This will cache every table object so that tables are not instantiated multiple times for complicated web requests.

Parameters

- **$class** (*string*) – Model class name.

Returns Omeka_Db_Table

setTable (*$alias*, *Omeka_Db_Table $table*)
Cache a table object.

Prevents the creation of unnecessary instances.

Parameters

- **$alias** (*string*) –
- **$table** (Omeka_Db_Table) –

insert (*$table*, *$values = array()*)
Every query ends up looking like: INSERT INTO table (field, field2, field3, ...) VALUES (?, ?, ?, ...) ON DUPLICATE KEY UPDATE field = ?, field2 = ?, ...

Note on portability: ON DUPLICATE KEY UPDATE is a MySQL extension. The advantage to using this is that it doesn't care whether a row exists already. Basically it combines what would be insert() and update() methods in other ORMs into a single method

Parameters

- **$table** (*string*) – Table model class name.
- **$values** (*array*) – Rows to insert (or update).

Returns int The ID for the row that got inserted (or updated).

log (*$sql*)
Log SQL query if logging is configured.

This logs the query before variable substitution from bind params.

Parameters

- **$sql** (*string|Zend_Db_Select*) –

queryBlock (*$sql*, *$delimiter = ';'*)
Execute more than one SQL query at once.

Parameters

- **$sql** (*string*) – String containing SQL queries.
- **$delimiter** (*string*) – Character that delimits each SQL query.

loadSqlFile (*$filePath*)
Read the contents of an SQL file and execute all the queries therein.

In addition to reading the file, this will make substitutions based on specific naming conventions. Currently makes the following substitutions: %PREFIX% will be replaced by the table prefix.

Parameters

- **$filePath** (*string*) – Path to the SQL file to load

Omeka_Db_Select

Package: *Db*

class Omeka_Db_Select

extends Zend_Db_Select

Class for SQL SELECT generation and results.

Omeka_Db_Select::**__construct**(*$adapter = null*)

Parameters

- **$adapter** (*Zend_Db_Adapter*) – (optional) Adapter to use instead of the one set up by Omeka.

Omeka_Db_Select::**hasJoin**(*$name*)

Detect if this SELECT joins with the given table.

Parameters

- **$name** (*string*) – Table name.

Returns bool

Omeka_Db_Select_PublicPermissions

Package: *Db*

class Omeka_Db_Select_PublicPermissions

Encapsulates the permissions check for a record that can be public or private.

property _allPermission

protected

property _selfPermission

protected

property _currentUser

protected

__construct(*$resource*)

Create the permissions object and perform the ACL checks.

The permissions check relies on 'showNotPublic' and (optionally) 'showSelfNotPublic' privileges on the give resource.

Parameters

- **$resource** (*string*) – ACL resource name to check.

apply(*Omeka_Db_Select $select*, *$alias*, *$ownerColumn = 'owner_id'*, *$publicColumn = 'public'*)

Apply the permissions to an SQL select object.

Parameters

- **$select** (Omeka_Db_Select) –
- **$alias** (*string*) – Table alias to query against
- **$ownerColumn** (*string*) – Optional column for checking for ownership. If falsy, the ownership check is skipped.

- **$publicColumn** (*string*) – Optional column for storing public status. The column must represent "public" status as the value 1.

Db\Migration

Up to *Db*

Omeka_Db_Migration_AbstractMigration

Package: *Db\Migration*

class Omeka_Db_Migration_AbstractMigration

implements *Omeka_Db_Migration_MigrationInterface*

Database migration classes may inherit from this one.

property Omeka_Db_Migration_AbstractMigration::$**db**
protected

Omeka_Db_Migration_AbstractMigration::**setDb**(*Omeka_Db $db*)
Set the database to migrate.

Parameters

- **$db** (Omeka_Db) –

Omeka_Db_Migration_AbstractMigration::**getDb**()

Returns Omeka_Db

Omeka_Db_Migration_AbstractMigration::**down**()
Template method for reversing the migration.

This is defined as a template method instead of leaving it abstract because pre-existing implementations of Omeka_Db_Migration were not required to implement the down() method. This ensures backwards compatibility for those migrations.

Omeka_Db_Migration_AbstractMigration::**__call**(*$m*, *$a*)
Proxy calls to Omeka_Db.

Allows migration writers to call db methods directly on $this.

Parameters

- **$m** (*string*) – Method name.
- **$a** (*array*) – Method arguments.

Omeka_Db_Migration_AbstractMigration::**form**()
If the migration requires a form submission, here's where to handle display of it

Omeka_Db_Migration_AbstractMigration::**up**()

Omeka_Db_Migration_Exception

Package: *Db\Migration*

class Omeka_Db_Migration_Exception

extends `Exception`

implements `Throwable`

Indicates an error during the database migration process.

property `Omeka_Db_Migration_Exception::$`**message**
protected

property `Omeka_Db_Migration_Exception::$`**code**
protected

property `Omeka_Db_Migration_Exception::$`**file**
protected

property `Omeka_Db_Migration_Exception::$`**line**
protected

`Omeka_Db_Migration_Exception::`**__clone**`()`

`Omeka_Db_Migration_Exception::`**__construct**`(`*$message*, *$code*, *$previous*`)`

Parameters

- **$message** –
- **$code** –
- **$previous** –

`Omeka_Db_Migration_Exception::`**__wakeup**`()`

`Omeka_Db_Migration_Exception::`**getMessage**`()`

`Omeka_Db_Migration_Exception::`**getCode**`()`

`Omeka_Db_Migration_Exception::`**getFile**`()`

`Omeka_Db_Migration_Exception::`**getLine**`()`

`Omeka_Db_Migration_Exception::`**getTrace**`()`

`Omeka_Db_Migration_Exception::`**getPrevious**`()`

`Omeka_Db_Migration_Exception::`**getTraceAsString**`()`

`Omeka_Db_Migration_Exception::`**__toString**`()`

Omeka_Db_Migration_Manager

Package: *Db\Migration*

class Omeka_Db_Migration_Manager
Manages database migrations (both upgrades and downgrades).

constant MIGRATION_TABLE_NAME
Name of the migrations table.

constant MIGRATION_DATE_FORMAT
Formatting string to convert dates into YYYYMMDDHHMMSS pattern.

constant ORIG_MIGRATION_OPTION_NAME
Name of the original database option storing the integer migration number.

constant VERSION_OPTION_NAME
Name of the new database option storing the core software version number.

__construct(*Omeka_Db $db*, *$migrationsDir*)

Parameters

- **$db** (Omeka_Db) –
- **$migrationsDir** (*string*) –

setupTimestampMigrations()
Set up Omeka to use timestamped database migrations.

This creates the 'schema_migrations' table, drops the 'migration' option and adds the 'omeka_version' option to the database.

markAllAsMigrated()
Mark all of the migrations as having been run. Used by the installer as a way of indicating that the database is entirely up to date.

migrate(*$endTimestamp = null*)
Migrate the database schema.

Parameters

- **$endTimestamp** (*string*) – (optional) Timestamp corresponding to the stop point for the migration. If older than the current time, database will migrate down to that point. If newer, the opposite. Defaults to the current timestamp.

canUpgrade()
Determine whether or not it is possible to migrate the Omeka database up.

This is based entirely on whether there exist any migrations that have not yet been applied.

dbNeedsUpgrade()
Determine whether the database must be upgraded.

In order to return true, this requires that canUprade() == true, and also that Omeka's code has recently been upgraded.

finalizeDbUpgrade()
Finalize the database upgrade by setting the most up-to-date version of Omeka.

getDefault(*$db = null*)
Return the default configuration of the database migration manager.

Parameters

- **$db** (*Omeka_Db|null*) –

Returns Omeka_Db_Migration_Manager

_getAllMigratedVersions()
Retrieve all the versions that have been migrated.

Returns array

_getMigrationTableName()
Return the name of the table associated with schema migrations.

Returns string

_getMigrationFileList()
Return a list of migration files in the migration directory.

Returns array An associative array where key = timestamp of migration, value = full filename of the migration.

_migrateUp(*$stopAt*)
Migrate upwards to a specific timestamp.

Parameters

- **$stopAt** (*DateTime*) –

_loadMigration(*$filename*)
Require the migration file and return an instance of the class associated with it.

Parameters

- **$filename** (*string*) – Migration script filename.

Returns Omeka_Db_Migration_AbstractMigration

_getPendingMigrations(*DateTime $until*)
Retrieve a list of all migrations that have not been run yet, ending at the latest time given by $until.

Parameters

- **$until** (*DateTime*) –

Returns array

_recordMigration(*$time*)
Record the migration timestamp in the schema_migrations table.

Parameters

- **$time** (*string*) –

Omeka_Db_Migration_MigrationInterface

Package: *Db\Migration*

interface Omeka_Db_Migration_MigrationInterface
Migration interface.

up()

down()

setDb(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

Db\Table

Up to *Db*

Omeka_Db_Table

Package: *Db\Table*

class Omeka_Db_Table
Database table classes.

Subclasses attached to models must follow the naming convention: Table_TableName, e.g. Table_ElementSet in models/Table/ElementSet.php.

property _target
protected string

The name of the model for which this table will retrieve objects.

property _name
protected string

The name of the table (sans prefix).

If this is not given, it will be inflected.

property _tablePrefix
protected string

The table prefix.

Generally used to differentiate Omeka installations sharing a database.

property _db
protected Omeka_Db

The Omeka database object.

__construct(*$targetModel*, *$db*)
Construct the database table object.

Do not instantiate this by itself. Access instances only via Omeka_Db::getTable().

Parameters

- **$targetModel** (*string*) – Class name of the table's model.
- **$db** (Omeka_Db) – Database object to use for queries.

__call(*$m*, *$a*)
Delegate to the database adapter.

Used primarily as a convenience method. For example, you can call fetchOne() and fetchAll() directly from this object.

Parameters

- **$m** (*string*) – Method name.
- **$a** (*array*) – Method arguments.

Returns mixed

getTableAlias()
Retrieve the alias for this table (the name without the prefix).

Returns string

getDb()
Retrieve the Omeka_Db instance.

Returns Omeka_Db

hasColumn(*$field*)
Determine whether a model has a given column.

Parameters

- **$field** (*string*) – Field name.

Returns bool

getColumns()
Retrieve a list of all the columns for a given model.

This should be here and not in the model class because get_class_vars() returns private/protected properties when called from within the class. Will only return public properties when called in this fashion.

Returns array

getTableName()
Retrieve the name of the table for the current table (used in SQL statements).

If the table name has not been set, it will inflect the table name.

Returns string

setTableName(*$name = null*)
Set the name of the database table accessed by this class.

If no name is provided, it will inflect the table name from the name of the model defined in the constructor. For example, Item -> items.

Parameters

- **$name** (*string*) – (optional) Table name.

getTablePrefix()
Retrieve the table prefix for this table instance.

Returns string

setTablePrefix(*$tablePrefix = null*)
Set the table prefix.

Defaults to the table prefix defined by the Omeka_Db instance. This should remain the default in most cases. However, edge cases may require customization, e.g. creating wrappers for tables generated by other applications.

Parameters

- **$tablePrefix** (*string|null*) –

find(*$id*)
Retrieve a single record given an ID.

Parameters

- **$id** (*int*) –

Returns Omeka_Record_AbstractRecord|false

findAll()
Get a set of objects corresponding to all the rows in the table

WARNING: This will be memory intensive and is thus not recommended for large data sets.

Returns array Array of {@link Omeka_Record_AbstractRecord}s.

findPairsForSelectForm(*$options = array()*)
Retrieve an array of key=>value pairs that can be used as options in a <select> form input.

Parameters

- **$options** (*array*) – (optional) Set of parameters for searching/ filtering results.

Returns array

_getColumnPairs()
Retrieve the array of columns that are used by findPairsForSelectForm().

This is a template method because these columns are different for every table, but the underlying logic that retrieves the pairs from the database is the same in every instance.

Returns array

findBy(*$params = array()*, *$limit = null*, *$page = null*)
Retrieve a set of model objects based on a given number of parameters

Parameters

- **$params** (*array*) – A set of parameters by which to filter the objects that get returned from the database.
- **$limit** (*int*) – Number of objects to return per "page".
- **$page** (*int*) – Page to retrieve.

Returns array|null The set of objects that is returned

getSelect()
Retrieve a select object for this table.

Returns Omeka_Db_Select

getSelectForFindBy(*$params = array()*)
Retrieve a select object that has had search filters applied to it.

Parameters

- **$params** (*array*) – optional Set of named search parameters.

Returns Omeka_Db_Select

getSelectForFind(*$recordId*)
Retrieve a select object that is used for retrieving a single record from the database.

Parameters

- **$recordId** (*int*) –

Returns Omeka_Db_Select

applySearchFilters(*$select*, *$params*)
Apply a set of filters to a Select object based on the parameters given.

By default, this simply checks the params for keys corresponding to database column names. For more complex filtering (e.g., when other tables are involved), or to use keys other than column names, override this method and optionally call this parent method.

Parameters

- **$select** (Omeka_Db_Select) –
- **$params** (*array*) –

applySorting(*$select*, *$sortField*, *$sortDir*)
Apply default column-based sorting for a table.

Parameters

- **$select** (Omeka_Db_Select) –
- **$sortField** (*string*) – Field to sort on.
- **$sortDir** (*string*) – Direction to sort.

applyPagination(*$select*, *$limit*, *$page = null*)
Apply pagination to a select object via the LIMIT and OFFSET clauses.

Parameters

- **$select** (*Zend_Db_Select*) –
- **$limit** (*int*) – Number of results per "page".
- **$page** (*int|null*) – Page to retrieve, first if omitted.

Returns Zend_Db_Select

findBySql(*$sqlWhereClause*, *$params = array()*, *$findOne = false*)
Retrieve an object or set of objects based on an SQL WHERE predicate.

Parameters

- **$sqlWhereClause** (*string*) –
- **$params** (*array*) – optional Set of parameters to bind to the WHERE clause. Used to prevent security flaws.
- **$findOne** (*bool*) – optional Whether or not to retrieve a single record or the whole set (retrieve all by default).

Returns array|Omeka_Record_AbstractRecord|false

count(*$params = array()*)
Retrieve a count of all the rows in the table.

Parameters

- **$params** (*array*) – optional Set of search filters upon which to base the count.

Returns int

exists(*$id*)
Check whether a row exists in the table.

Parameters

- **$id** (*int*) –

Returns bool

filterByPublic(*Omeka_Db_Select $select*, *$isPublic*)
Apply a public/not public filter to the select object.

A convenience function than derivative table classes may use while applying search filters.

Parameters

- **$select** (Omeka_Db_Select) –
- **$isPublic** (*bool*) –

filterByFeatured(*Omeka_Db_Select $select*, *$isFeatured*)
Apply a featured/not featured filter to the select object.

A convenience function than derivative table classes may use while applying search filters.

Parameters

- **$select** (Omeka_Db_Select) –
- **$isFeatured** (*bool*) –

filterBySince (*Omeka_Db_Select $select*, *$dateSince*, *$dateField*)
Apply a date since filter to the select object.

A convenience function than derivative table classes may use while applying search filters.

Parameters

- **$select** (Omeka_Db_Select) –
- **$dateSince** (*string*) – ISO 8601 formatted date
- **$dateField** (*string*) – "added" or "modified"

filterByUser (*Omeka_Db_Select $select*, *$userId*, *$userField*)
Apply a user filter to the select object.

A convenience function than derivative table classes may use while applying search filters.

Parameters

- **$select** (Omeka_Db_Select) –
- **$userId** (*int*) –
- **$userField** –

filterByRange (*$select*, *$range*)
Filter returned records by ID.

Can specify a range of valid record IDs or an individual ID

Parameters

- **$select** (Omeka_Db_Select) –
- **$range** (*string*) – Example: 1-4, 75, 89

getSelectForCount (*$params = array()*)
Retrieve a select object used to retrieve a count of all the table rows.

Parameters

- **$params** (*array*) – optional Set of search filters.

Returns Omeka_Db_Select

checkExists (*$id*)
Check whether a given row exists in the database.

Currently used to verify that a row exists even though the current user may not have permissions to access it.

Parameters

- **$id** (*int*) – The ID of the row.

Returns bool

fetchObjects (*$sql*, *$params = array()*)
Retrieve a set of record objects based on an SQL SELECT statement.

Parameters

- **$sql** (*string*) – This could be either a string or any object that can be cast to a string (commonly Omeka_Db_Select).
- **$params** (*array*) – Set of parameters to bind to the SQL statement.

Returns array|null Set of Omeka_Record_AbstractRecord instances, or null if none can be found.

fetchObject (*$sql*, *$params = array()*)
Retrieve a single record object from the database.

Parameters

- **$sql** (*string*) –
- **$params** (*string*) – Parameters to substitute into SQL query.

Returns Omeka_Record_AbstractRecord or null if no record

recordFromData (*$data*)
Populate a record object with data retrieved from the database.

Parameters

- **$data** (*array*) – A keyed array representing a row from the database.

Returns Omeka_Record_AbstractRecord

_getSortParams (*$params*)
Get and parse sorting parameters to pass to applySorting.

A sorting direction of 'ASC' will be used if no direction parameter is passed.

Parameters

- **$params** (*array*) –

Returns array|null Array of sort field, sort dir if params exist, null otherwise.

_getHookName (*$suffix*)
Get the name for a model-specific hook or filter..

Parameters

- **$suffix** (*string*) – The hook-specific part of the hook name.

Returns string

3.7.7 File\Derivative

Up to ../index

Omeka_File_Derivative_Creator

Package: *File\Derivative*

class Omeka_File_Derivative_Creator
Create derivative images for a file in Omeka.

create (*$sourcePath*, *$derivFilename*, *$mimeType*)
Create all the derivatives requested with addDerivative().

Parameters

- **$sourcePath** (*string*) –
- **$derivFilename** (*string*) –
- **$mimeType** (*string*) –

Returns bool

addDerivative(*$storageType*, *$size*)
Add a derivative image to be created.

Parameters

- **$storageType** (*string*) –
- **$size** (*int*) – The size constraint for the image, meaning it will have that maximum width or height, depending on whether the image is landscape or portrait.

setStrategy(*Omeka_File_Derivative_StrategyInterface $strategy*)
Set the strategy for creating derivatives.

Parameters

- **$strategy** (Omeka_File_Derivative_StrategyInterface) –

getStrategy()
Get the strategy for creating derivatives.

Returns Omeka_File_Derivative_StrategyInterface

setTypeBlacklist(*$blacklist*)
Set the type blacklist.

Parameters

- **$blacklist** (*array|null*) – An array of mime types to blacklist.

setTypeWhitelist(*$whitelist*)
Set the type whitelist.

Parameters

- **$whitelist** (*array|null*) – An array of mime types to whitelist.

_isDerivable(*$filePath*, *$mimeType*)
Returns whether Omeka can make derivatives of the given file.

The file must be readable and pass the mime whitelist/blacklist.

Parameters

- **$filePath** (*string*) –
- **$mimeType** (*string*) –

Returns bool

_passesBlacklist(*$mimeType*)
Return whether the given type is allowed by the blacklist.

If no blacklist is specified all types will pass.

Parameters

- **$mimeType** (*string*) –

Returns bool

_passesWhitelist(*$mimeType*)
Return whether the given type is allowed by the whitelist.

If no whitelist is specified all types will pass, but an empty whitelist will reject all types.

Parameters

- **$mimeType** (*string*) –

Returns bool

Omeka_File_Derivative_Exception

Package: *File\Derivative*

class Omeka_File_Derivative_Exception

extends `Exception`

implements `Throwable`

Exception to throw when something goes wrong in the process of creating derivative images.

property `Omeka_File_Derivative_Exception::$`**message**
protected

property `Omeka_File_Derivative_Exception::$`**code**
protected

property `Omeka_File_Derivative_Exception::$`**file**
protected

property `Omeka_File_Derivative_Exception::$`**line**
protected

`Omeka_File_Derivative_Exception::`**__clone**()

`Omeka_File_Derivative_Exception::`**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

`Omeka_File_Derivative_Exception::`**__wakeup**()

`Omeka_File_Derivative_Exception::`**getMessage**()

`Omeka_File_Derivative_Exception::`**getCode**()

`Omeka_File_Derivative_Exception::`**getFile**()

`Omeka_File_Derivative_Exception::`**getLine**()

`Omeka_File_Derivative_Exception::`**getTrace**()

`Omeka_File_Derivative_Exception::`**getPrevious**()

`Omeka_File_Derivative_Exception::`**getTraceAsString**()

`Omeka_File_Derivative_Exception::`**__toString**()

File\Derivative\Strategy

Up to *File\Derivative*

Omeka_File_Derivative_AbstractStrategy

Package: *File\Derivative\Strategy*

class Omeka_File_Derivative_AbstractStrategy

implements *Omeka_File_Derivative_StrategyInterface*

Abstract class for pluggable file derivative creation strategies.

property Omeka_File_Derivative_AbstractStrategy::$**_options**
protected

Omeka_File_Derivative_AbstractStrategy::**setOptions**(*$options*)
Set options for the derivative strategy.

Parameters

- **$options** (*array*) –

Omeka_File_Derivative_AbstractStrategy::**getOptions**()
Get the options for the strategy.

Returns array

Omeka_File_Derivative_AbstractStrategy::**getOption**(*$name*, *$default = null*)
Get the value for the specified option.

Parameters

- **$name** (*string*) – Name of the option to get
- **$default** (*mixed*) – Default value to return if the option is missing. Defaults to null.

Returns mixed

Omeka_File_Derivative_AbstractStrategy::**createImage**(*$sourcePath*, *$destPath*, *$type*, *$sizeConstraint*, *$mimeType*)
Create an derivative of the given image.

Parameters

- **$sourcePath** (*string*) – Local path to the source file.
- **$destPath** (*string*) – Local path to write the derivative to.
- **$type** (*string*) – The type of derivative being created.
- **$sizeConstraint** (*int*) – Size limitation on the derivative.
- **$mimeType** (*string*) – MIME type of the original file.

Returns bool

Omeka_File_Derivative_StrategyInterface

Package: *File\Derivative\Strategy*

interface Omeka_File_Derivative_StrategyInterface
Interface for pluggable file derivative creation strategies.

createImage(*$sourcePath*, *$destPath*, *$type*, *$sizeConstraint*, *$mimeType*)
Create an derivative of the given image.

Parameters

- **$sourcePath** (*string*) – Local path to the source file.
- **$destPath** (*string*) – Local path to write the derivative to.
- **$type** (*string*) – The type of derivative being created.
- **$sizeConstraint** (*int*) – Size limitation on the derivative.
- **$mimeType** (*string*) – MIME type of the original file.

Returns bool

setOptions(*$options*)
Set options for the derivative strategy.

Parameters

- **$options** (*array*) –

getOptions()
Get the options for the strategy.

Returns array

Omeka_File_Derivative_Strategy_ExternalImageMagick

Package: *File\Derivative\Strategy*

class Omeka_File_Derivative_Strategy_ExternalImageMagick

extends *Omeka_File_Derivative_AbstractStrategy*

Strategy for making derivatives with ImageMagick on the command line.

Omeka_File_Derivative_Strategy_ExternalImageMagick::**createImage**(*$sourcePath*, *$destPath*, *$type*, *$sizeConstraint*, *$mimeType*)
Generate a derivative image from an existing file stored in Omeka.

This image will be generated based on a constraint given in pixels. For example, if the constraint is 500, the resulting image file will be scaled so that the largest side is 500px. If the image is less than 500px on both sides, the image will not be resized.

Derivative images will only be generated for files with mime types that pass any configured blacklist and/or whitelist and can be processed by the convert binary.

Parameters

- **$sourcePath** –
- **$destPath** –
- **$type** –

- **$sizeConstraint** –
- **$mimeType** –

Omeka_File_Derivative_Strategy_ExternalImageMagick::**_getConvertPath**()
Get the full path to the ImageMagick 'convert' command.

Returns string

Omeka_File_Derivative_Strategy_ExternalImageMagick::**_getConvertArgs**(*$type*, *$constraint*)
Get the ImageMagick command line for resizing to the given constraints.

Parameters

- **$type** (*string*) – Type of derivative being made.
- **$constraint** (*int*) – Maximum side length in pixels.

Returns string

Omeka_File_Derivative_Strategy_ExternalImageMagick::**isValidImageMagickPath**(*$dirToIm*)
Determine whether or not the path given to ImageMagick is valid. The convert binary must be within the directory and executable.

Parameters

- **$dirToIm** –

Returns bool

Omeka_File_Derivative_Strategy_ExternalImageMagick::**getDefaultImageMagickDir**()
Retrieve the path to the directory containing ImageMagick's convert utility.

Uses the 'which' command-line utility to detect the path to 'convert'. Note that this will only work if the convert utility is in PHP's PATH and thus can be located by 'which'.

Returns string The path to the directory if it can be found. Otherwise returns an empty string.

Omeka_File_Derivative_Strategy_ExternalImageMagick::**executeCommand**(*$cmd*, *$status*, *$output*, *$errors*)

Parameters

- **$cmd** –
- **$status** –
- **$output** –
- **$errors** –

Omeka_File_Derivative_Strategy_GD

Package: *File\Derivative\Strategy*

class Omeka_File_Derivative_Strategy_GD

extends *Omeka_File_Derivative_AbstractStrategy*

Strategy for making derivatives with the GD PHP library (default since 4.3).

Omeka_File_Derivative_Strategy_GD::**__construct**()
Check for the imagick extension at creation.

Omeka_File_Derivative_Strategy_GD::**createImage**(*$sourcePath*, *$destPath*, *$type*, *$sizeConstraint*, *$mimeType*)
Generate a derivative image with GD.

Parameters

- **$sourcePath** –
- **$destPath** –
- **$type** –
- **$sizeConstraint** –
- **$mimeType** –

Returns bool Returns true on success or false on failure.

Omeka_File_Derivative_Strategy_GD::**_loadImageResource**(*$source*)
GD uses multiple functions to load an image, so this one manages all.

Parameters

- **$source** (*string*) – Path of the managed image file

Returns false|GD image ressource

Omeka_File_Derivative_Strategy_GD::**_makeThumbnail**(*$source*, *$destination*, *$sizeConstraint*)
Make a thumbnail from source and save it at destination.

Parameters

- **$source** (*string*) – Path of the source.
- **$destination** (*string*) – Path of the destination.
- **$sizeConstraint** (*int*) – Maximum size in pixels.

Returns bool Returns true on success or false on failure.

Omeka_File_Derivative_Strategy_GD::**_makeSquareThumbnail**(*$source*, *$destination*, *$sizeConstraint*)
Make a square thumbnail from source and save it at destination.

Parameters

- **$source** (*string*) – Path of the source.
- **$destination** (*string*) – Path of the destination.
- **$sizeConstraint** (*int*) – Maximum size in pixels.

Returns bool Returns true on success or false on failure.

Omeka_File_Derivative_Strategy_GD::**_getOffsetX**(*$width*, *$size*)
Get the required offset on the X axis.

This respects the 'gravity' setting.

Parameters

- **$width** (*int*) – Original image width
- **$size** (*int*) – Side size of the square region being selected

Returns int

Omeka_File_Derivative_Strategy_GD::**_getOffsetY**(*$height*, *$size*)

Get the required offset on the Y axis.

This respects the 'gravity' setting.

Parameters

- **$height** (*int*) – Original image height
- **$size** (*int*) – Side size of square region being selected

Returns int

Omeka_File_Derivative_Strategy_Imagick

Package: *File\Derivative\Strategy*

class Omeka_File_Derivative_Strategy_Imagick

extends *Omeka_File_Derivative_AbstractStrategy*

Strategy for making derivatives with the Imagick PHP extension.

The strategy requires ext/imagick.

Omeka_File_Derivative_Strategy_Imagick::**__construct**()

Check for the imagick extension at creation.

Omeka_File_Derivative_Strategy_Imagick::**createImage**(*$sourcePath*, *$destPath*, *$type*, *$sizeConstraint*, *$mimeType*)

Generate a derivative image with Imagick.

Parameters

- **$sourcePath** –
- **$destPath** –
- **$type** –
- **$sizeConstraint** –
- **$mimeType** –

Omeka_File_Derivative_Strategy_Imagick::**_getCropOffsetX**(*$resizedX*, *$sizeConstraint*)

Get the required crop offset on the X axis.

This respects the 'gravity' setting.

Parameters

- **$resizedX** (*int*) – Pre-crop image width
- **$sizeConstraint** (*int*) –

Returns int

`Omeka_File_Derivative_Strategy_Imagick::`**`_getCropOffsetY`**(*$resizedY*, *$sizeConstraint*)

Get the required crop offset on the Y axis.

This respects the 'gravity' setting.

Parameters

- **`$resizedY`** (*int*) – Pre-crop image height
- **`$sizeConstraint`** (*int*) –

Returns int

`Omeka_File_Derivative_Strategy_Imagick::`**`_autoOrient`**(*$imagick*)

Parameters

- **`$imagick`** –

3.7.8 Filter

Up to *Packages*

Omeka_Filter_Boolean

Package: *Filter*

`class Omeka_Filter_Boolean`

implements `Zend_Filter_Interface`

A Zend_Filter implementation that converts any boolean value passed to it to an integer: 1 or 0.

`Omeka_Filter_Boolean::`**`filter`**(*$value*)

Filter the value

Parameters

- **`$value`** –

Returns int 1 or 0

Omeka_Filter_Filename

Package: *Filter*

`class Omeka_Filter_Filename`

implements `Zend_Filter_Interface`

Rename a file to make it suitable for inclusion in the Omeka repository.

`Omeka_Filter_Filename::`**`filter`**(*$value*)

Grab the original path to the file, rename it according to our convention, and return the new path to the file.

Parameters

- **`$value`** (*string*) – Path to the file.

Returns string Path to the (renamed) file.

`Omeka_Filter_Filename::`**`renameFile`**`(`*`$name`*`)`
Creates a new, random filename for storage in Omeka.

Parameters

- **`$name`** (*`string`*) –

Returns string

Omeka_Filter_ForeignKey

Package: *Filter*

`class Omeka_Filter_ForeignKey`

implements `Zend_Filter_Interface`

Converts input into values suitable for use as Omeka 'id' key values.

`Omeka_Filter_ForeignKey::`**`filter`**`(`*`$value`*`)`
Convert any value into an unsigned integer that would be valid if stored as a foreign key in a database table.

This will return null for any value that falls outside the range of an unsigned integer (string, negative numbers, etc.)

Parameters

- **`$value`** (*`mixed`*) – Input value.

Returns int

Omeka_Filter_HtmlPurifier

Package: *Filter*

`class Omeka_Filter_HtmlPurifier`

implements `Zend_Filter_Interface`

A Zend_Filter implementation that uses HtmlPurifier to purify a string.

`Omeka_Filter_HtmlPurifier::`**`filter`**`(`*`$value`*`)`
Filter the value

Parameters

- **`$value`** –

Returns string An html purified string

`Omeka_Filter_HtmlPurifier::`**`getDefaultAllowedHtmlElements`**`()`
Get the default allowed html elements.

Returns array An array of strings corresponding to the allowed html elements

`Omeka_Filter_HtmlPurifier::`**`getDefaultAllowedHtmlAttributes`**`()`
Get the default allowed html attributes.

Returns array An array of strings corresponding to the allowed html attributes

`Omeka_Filter_HtmlPurifier::`**`getHtmlPurifier`**`()`
Gets the html purifier singleton

Returns HTMLPurifier $purifier

`Omeka_Filter_HtmlPurifier::`**`setHtmlPurifier`**(*$purifier*)
Sets the html purifier singleton

Parameters

- **`$purifier`** (*`HTMLPurifier`*) –

`Omeka_Filter_HtmlPurifier::`**`createHtmlPurifier`**(*$allowedHtmlElements = null*, *$allowedHtmlAttributes = null*)

Parameters

- **`$allowedHtmlElements`** (*`array`*) – An array of strings representing allowed HTML elements
- **`$allowedHtmlAttributes`** (*`array`*) – An array of strings representing allowed HTML attributes

Returns HTMLPurifier

`Omeka_Filter_HtmlPurifier::`**`filterAttributesWithMissingElements`**(*$htmlAttributes = array()*, *$htmlElements = array()*)

Parameters

- **`$htmlAttributes`** –
- **`$htmlElements`** –

3.7.9 Form

Up to *Packages*

Omeka_Form

Package: *Form*

`class Omeka_Form`

extends `Zend_Form`

A Zend_Form subclass that sets up forms to be properly displayed in Omeka.

`property` `Omeka_Form::$`**`_defaultDisplayGroupClass`**
protected string

Class name of Omeka DisplayGroup subclass.

`property` `Omeka_Form::$`**`_autoApplyOmekaStyles`**
protected bool

Whether or not to automatically apply Omeka-specific decorators and styling information to form elements prior to rendering.

Omeka_Form::**init**()
Set up Omeka-specific form elements and decorators.

Omeka_Form::**loadDefaultDecorators**()
Set up base form decorators.

Omeka_Form::**getDefaultElementDecorators**()
Return default decorators for form elements.

Makes form output conform to Omeka conventions.

Returns array

Omeka_Form::**applyOmekaStyles**()
Configure element styles / decorators based on the type of element.

This may be called after elements to the form, as the decorator configuration in init() runs before elements can be added.

Omeka_Form::**getMessagesAsString**(*$messageDelimiter = ' '*, *$elementDelimiter = ', '*)
Retrieve all of the form error messages as a nicely formatted string.

Useful for displaying all form errors at the top of a form, or for flashing form errors after redirects.

Parameters

- **$messageDelimiter** (*string*) – The string to display between different error messages for an element.
- **$elementDelimiter** (*string*) – The string to display between different elements.

Returns string

Omeka_Form::**setAutoApplyOmekaStyles**(*$flag*)
Specify whether or not to automatically apply Omeka-specific decorators and styles prior to rendering the form.

Parameters

- **$flag** (*mixed*) – A boolean or boolean-equivalent.

Omeka_Form::**render**(*Zend_View_Interface $view = null*)
Apply Omeka default styles (if requested) just before rendering.

Parameters

- **$view** (*Zend_View_Interface*) –

Returns string

Omeka_Form::**_addClassNameToElement**(*Zend_Form_Element $element*, *$className*)
Add a specific class name to an element.

Parameters

- **$element** (*Zend_Form_Element*) –
- **$className** (*string*) –

Omeka_Form_Admin

Package: *Form*

class Omeka_Form_Admin

extends *Omeka_Form*

A Zend_Form subclass to set up a record editing form for the Omeka 2.0 admin user interface

property Omeka_Form_Admin::$**_editDisplayGroup**
protected

property Omeka_Form_Admin::$**_saveDisplayGroup**
protected

property Omeka_Form_Admin::$**_saveDisplayGroupActionDecorator**
protected

property Omeka_Form_Admin::$**_record**
protected

property Omeka_Form_Admin::$**_type**
protected

property Omeka_Form_Admin::$**_hasPublicPage**
protected

property Omeka_Form_Admin::$**_editGroupCssClass**
protected

property Omeka_Form_Admin::$**_saveGroupCssClass**
protected

Omeka_Form_Admin::**init**()

Omeka_Form_Admin::**addElementToEditGroup**(*$element*, *$name*, *$options = null*)
Add an element to the edit area

Parameters

- **$element** (*Zend_Form_Element|string*) –
- **$name** (*string|null*) –
- **$options** (*array|null*) –

Omeka_Form_Admin::**addElementToSaveGroup**(*$element*, *$name = null*, *$options = null*)
Add an element to the save panel

Parameters

- **$element** (*Zend_Form_Element|string*) –
- **$name** (*string|null*) –
- **$options** (*array|null*) –

Omeka_Form_Admin::**addElementToDisplayGroup**(*$group*, *$element*, *$name = null*, *$options = null*)
Generalizes creating and adding new elements to one of the display groups

You can pass in either an Zend_Form_Element you have already created, or pass parameters as you would to Zend_Form::addElement

Parameters

- **$group** (*string*) – Either 'save' or 'edit'
- **$element** (*Zend_Form_Element*) – The element to add to the display group
- **$name** (*string*) –
- **$options** (*array*) –

Returns Omeka_Form_Admin

Omeka_Form_Admin::**getSaveGroupDefaultElementDecorators**()
Get the decorators for the save display group

Returns array The default decorators for the save display group

Omeka_Form_Admin::**setEditGroupCssClass**(*$cssClass*)
Set the class for the edit display group.

You can alter the default css class for the edit group panel by passing in an option for 'editGroupCssClass' when you create an instance of Omeka_Form_Admin. This should be done very sparingly, as the default class is the best match to existing admin theme look and feel

Parameters

- **$cssClass** (*string*) –

Omeka_Form_Admin::**setSaveGroupCssClass**(*$cssClass*)
Set the class for the save display group.

You can alter the default css class for the save group panel by passing in an option for 'editGroupCssClass' when you create an instance of Omeka_Form_Admin. This should be done very sparingly, as the default class is the best match to existing admin theme look and feel

Parameters

- **$cssClass** (*string*) –

Omeka_Form_Admin::**setType**(*$type*)
Set the record type of the object being edited (e.g., 'item')

Pass in the recordType as part of the options array when you create an instance

Parameters

- **$type** (*string*) –

Omeka_Form_Admin::**setRecord**(*$record*)
Set the record (if one exists) for the object being edited

Passing the record object as part of the options when you create the form will automatically add 'Edit' and 'Delete' buttons to the save panel

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Omeka_Form_Admin::**setHasPublicPage**(*$value = false*)
Set whether the save panel should display a link to the record's public page if it exists

By default, a link to a record's public page is available if it exists. Pass false as the value of hasPublicPage in the options array to suppress this behavior.

Parameters

- **$value** (*bool*) – true

Omeka_Form_DisplayGroup

Package: *Form*

class Omeka_Form_DisplayGroup

extends Zend_Form_DisplayGroup

Subclass of Zend_Form_DisplayGroup that exist to override the default decorators associated with display groups.

Omeka_Form_DisplayGroup::**loadDefaultDecorators**()
Cause display groups to render as HTML fieldset elements.

Omeka_Form_Element_Input

Package: *Form*

class Omeka_Form_Element_Input

extends Zend_Form_Element_Xhtml

HTML5 input form element

property Omeka_Form_Element_Input::$**helper**
string

Default form view helper to use for rendering

Omeka_Form_Element_SessionCsrfToken

Package: *Form*

class Omeka_Form_Element_SessionCsrfToken

extends Zend_Form_Element_Xhtml

CSRF form protection

This class is an adaptation of ZF's Hash element that uses a per-session token.

property Omeka_Form_Element_SessionCsrfToken::$**helper**
string

Use formHidden view helper by default

property Omeka_Form_Element_SessionCsrfToken::$**_disableLoadDefaultDecorators**
protected bool

Should we disable loading the default decorators?

property Omeka_Form_Element_SessionCsrfToken::$**_token**
protected mixed

Actual token used.

property Omeka_Form_Element_SessionCsrfToken::$**_session**
protected Zend_Session_Namespace

Omeka_Form_Element_SessionCsrfToken::**init**()
Constructor

Creates session namespace for CSRF token, and adds validator for CSRF token.

Omeka_Form_Element_SessionCsrfToken::**setSession**(*$session*)
Set session object

Parameters

- **$session** –

Returns self

Omeka_Form_Element_SessionCsrfToken::**getSession**()
Get session object

Instantiate session object if none currently exists

Returns Zend_Session_Namespace

Omeka_Form_Element_SessionCsrfToken::**getToken**()
Retrieve CSRF token

Returns string

Omeka_Form_Element_SessionCsrfToken::**render**(*Zend_View_Interface $view = null*)
Render CSRF token in form

Parameters

- **$view** (*Zend_View_Interface*) –

Returns string

Omeka_Form_Element_SessionCsrfToken::**getLabel**()
Override getLabel() to always be empty

Omeka_Form_Element_SessionCsrfToken::**_initToken**()
Set the CSRF token

If a session token exists, it is used. Otherwise, a new token is generated and saved in the session.

Returns self

Omeka_Form_Element_SessionCsrfToken::**_initCsrfValidator**()
Initialize CSRF validator

Returns self

Omeka_Form_Element_SessionCsrfToken::**_generateToken**()
Generate CSRF token

Form\Decorator

Up to *Form*

Omeka_Form_Decorator_SavePanelAction

Package: *Form\Decorator*

class Omeka_Form_Decorator_SavePanelAction

extends Zend_Form_Decorator_Abstract

Decorator for buttons (usually actions) for the save panel in Omeka_Form_Admin

property Omeka_Form_Decorator_SavePanelAction::$**content**
protected

Omeka_Form_Decorator_SavePanelAction::**getContent**()

Omeka_Form_Decorator_SavePanelAction::**getRecord**()
Checks if a record was passed to the Omeka_Form_Admin form. returns it if it has been

Returns mixed false or the record

Omeka_Form_Decorator_SavePanelAction::**hasPublicPage**()
Checks if the Omeka_Form_Admin should display a link to a record's public page

Omeka_Form_Decorator_SavePanelAction::**render**(*$content*)
Render html for the save panel buttons

Parameters

- **$content** (*string*) –

Returns string

Omeka_Form_Decorator_SavePanelHook

Package: *Form\Decorator*

class Omeka_Form_Decorator_SavePanelHook

extends Zend_Form_Decorator_Abstract

Decorator to add hooks into the admin save panel created via Omeka_Form_Admin

Omeka_Form_Decorator_SavePanelHook::**render**(*$content*)

Parameters

- **$content** –

Omeka_Form_Decorator_SavePanelHook::**getType**()
Get the record type if the Omeka_Form_Admin was created with that option set

Returns mixed false or the record

Omeka_Form_Decorator_SavePanelHook::**getRecord**()
Get the record if the Omeka_Form_Admin was created with that option set

Returns mixed false or the record

3.7.10 Http

Up to *Packages*

Omeka_Http_Client

Package: *Http*

class Omeka_Http_Client

extends Zend_Http_Client

Wrapper for Zend_Http_Client.

Adds the following functionality: retries on timeouts.

`Omeka_Http_Client::request`(*$method = null*)
Wraps Zend_Http_Client to automatically retry timed out requests.

Parameters

- **$method** –

`Omeka_Http_Client::setMaxRetries`(*$count*)
Set the maximum number of retries to make when a request times out.

Parameters

- **$count** (*int*) –

3.7.11 Install

Up to *Packages*

Install\Task

Up to *Install*

3.7.12 Job

Up to *Packages*

Omeka_Job_AbstractJob

Package: *Job*

class Omeka_Job_AbstractJob

implements *Omeka_Job_JobInterface*

Abstract implementation of an Omeka job.

Most plugin implementations of jobs will extend this class to gain convenient access to the database and other potentially important resources.

property `Omeka_Job_AbstractJob::$_db`
protected Omeka_Db

property `Omeka_Job_AbstractJob::$_dispatcher`
protected Omeka_Job_Dispatcher_DispatcherInterface

property `Omeka_Job_AbstractJob::$_user`
protected User

property `Omeka_Job_AbstractJob::$_options`
protected

`Omeka_Job_AbstractJob::__construct`(*$options*)

Parameters

- **$options** –

Omeka_Job_AbstractJob::**_setOptions**(*$options*)
Set all the options associated with this task.

This is a convenience method that calls setter methods for the options given in the array. If an element in the array does not have an associated setter method, it will be passed into the options array.

Parameters

- **$options** –

Omeka_Job_AbstractJob::**setDb**(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

Omeka_Job_AbstractJob::**setJobDispatcher**(*Omeka_Job_Dispatcher_DispatcherInterface $dispatcher*)

Parameters

- **$dispatcher** (Omeka_Job_Dispatcher_DispatcherInterface) –

Omeka_Job_AbstractJob::**setUser**(*User $user*)
Set the given User object on the Job object.

Parameters

- **$user** (User) –

Omeka_Job_AbstractJob::**getUser**()
Get the User currently set on this Job, if any.

Returns User|null

Omeka_Job_AbstractJob::**resend**()
Resend the job using the same options that were passed to the current job.

Omeka_Job_AbstractJob::**perform**()

Omeka_Job_JobInterface

Package: *Job*

interface Omeka_Job_JobInterface
Interface for jobs.

__construct(*$options*)

Parameters

- **$options** –

perform()

Omeka_Job_Mock

Package: *Job*

class Omeka_Job_Mock

extends *Omeka_Job_AbstractJob*

implements *Omeka_Job_JobInterface*

Mock job class for unit tests.

property Omeka_Job_Mock::$**options**

property Omeka_Job_Mock::$**performed**

property Omeka_Job_Mock::$**_db**
protected Omeka_Db

property Omeka_Job_Mock::$**_dispatcher**
protected Omeka_Job_Dispatcher_DispatcherInterface

property Omeka_Job_Mock::$**_user**
protected User

property Omeka_Job_Mock::$**_options**
protected

Omeka_Job_Mock::**__construct**(*$options*)

Parameters

- **$options** –

Omeka_Job_Mock::**perform**()

Omeka_Job_Mock::**getDb**()
Getter method to expose protected properties.

Omeka_Job_Mock::**getDispatcher**()
Getter method to expose protected properties.

Omeka_Job_Mock::**getMiscOptions**()

Omeka_Job_Mock::**_setOptions**(*$options*)
Set all the options associated with this task.

This is a convenience method that calls setter methods for the options given in the array. If an element in the array does not have an associated setter method, it will be passed into the options array.

Parameters

- **$options** –

Omeka_Job_Mock::**setDb**(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

Omeka_Job_Mock::**setJobDispatcher**(*Omeka_Job_Dispatcher_DispatcherInterface $dispatcher*)

Parameters

- **$dispatcher** (Omeka_Job_Dispatcher_DispatcherInterface) –

Omeka_Job_Mock::**setUser**(*User $user*)
Set the given User object on the Job object.

Parameters

- **$user** (User) –

Omeka_Job_Mock::**getUser**()
Get the User currently set on this Job, if any.

Returns User|null

`Omeka_Job_Mock::resend()`
Resend the job using the same options that were passed to the current job.

Job\Dispatcher

Up to *Job*

Omeka_Job_Dispatcher_Default

Package: *Job\Dispatcher*

class Omeka_Job_Dispatcher_Default

implements *Omeka_Job_Dispatcher_DispatcherInterface*

Dispatches jobs in Omeka.

This provides a clean interface to adapter classes that deal with the details of how to dispatch jobs. It is initialized in the Jobs bootstrap resource and can be accessed via the registry.

Standard usage, where Job_Class_Name corresponds to a valid class name for a class implementing Omeka_JobInterface:

<code> $dispatcher = Zend_Registry::get('job_dispatcher'); $dispatcher->send('Job_Class_Name', array(

'firstOption' => 'text', 'secondOption' => 2

)); </code>

`Omeka_Job_Dispatcher_Default::__construct(`*Omeka_Job_Dispatcher_Adapter_AdapterInterface $defaultAdapter*, *Omeka_Job_Dispatcher_Adapter_AdapterInterface $longRunningAdapter*, *$user*`)`

Parameters

- **$defaultAdapter** (Omeka_Job_Dispatcher_Adapter_AdapterInterface) –
- **$longRunningAdapter** (Omeka_Job_Dispatcher_Adapter_AdapterInterface) –
- **$user** (*User|null*) – The user account associated with the request, i.e. the user account associated with jobs sent by the dispatcher.

`Omeka_Job_Dispatcher_Default::setUser(`*$user*`)`
Set the user.

Parameters

- **$user** (*User|null*) –

`Omeka_Job_Dispatcher_Default::getUser()`
Get the user.

Returns User|null

`Omeka_Job_Dispatcher_Default::setDefaultAdapter(`*Omeka_Job_Dispatcher_Adapter_AdapterInterface $defaultAdapter*`)`

Set the default adapter.

Parameters

- **$defaultAdapter** (Omeka_Job_Dispatcher_Adapter_AdapterInterface) –

Omeka_Job_Dispatcher_Default::**setLongRunningAdapter**(*Omeka_Job_Dispatcher_Adapter_AdapterInterface $longRunningAdapter*)

Set the long running adapter.

Parameters

- **$longRunningAdapter** (Omeka_Job_Dispatcher_Adapter_AdapterInterface) –

Omeka_Job_Dispatcher_Default::**setQueueName**(*$name*)

Set the name of the queue to which default jobs will be sent.

NOTE: This may be ignored by adapters that do not understand the notion of named queues (or queues in general).

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Default::**setQueueNameLongRunning**(*$name*)

Set the name of the queue to which long-running jobs will be sent.

NOTE: This may be ignored by adapters that do not understand the notion of named queues (or queues in general).

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Default::**send**(*$jobClass*, *$options = array()*)

Dispatch a job using the default dispatcher.

Parameters

- **$jobClass** (*string*) – Class name that implements Omeka_JobInterface.
- **$options** (*array*) – Optional associative array containing options that the task needs in order to do its job. Note that all options should be primitive data types (or arrays containing primitive data types).

Omeka_Job_Dispatcher_Default::**sendLongRunning**(*$jobClass*, *$options = array()*)

Dispatch a job using the long-running dispatcher.

Parameters

- **$jobClass** (*string*) – Name of a class that implements Omeka_JobInterface.
- **$options** (*array*) – Optional associative array containing options that the task needs in order to do its job. Note that all options should be primitive data types (or arrays containing primitive data types).

Omeka_Job_Dispatcher_Default::**_getJobMetadata**(*$class*, *$options*)

Parameters

- **$class** –
- **$options** –

Omeka_Job_Dispatcher_Default::**_toJson**(*$metadata*)

Parameters

- **$metadata** –

Omeka_Job_Dispatcher_DispatcherInterface

Package: *Job\Dispatcher*

interface Omeka_Job_Dispatcher_DispatcherInterface

Interface for job dispatchers in Omeka.

setQueueName(*$name*)

Set the name of the queue to which jobs will be sent.

NOTE: This may be ignored by adapters that do not understand the notion of named queues (or queues in general).

Parameters

- **$name** (*string*) –

send(*$jobClass*, *$options = array()*)

Parameters

- **$jobClass** (*string*) – Name of a class that implements Omeka_JobInterface.
- **$options** (*array*) – Optional Associative array containing options that the task needs in order to do its job. Note that all options should be primitive data types (or arrays containing primitive data types).

Job\Dispatcher\Adapter

Up to *Job\Dispatcher*

Omeka_Job_Dispatcher_Adapter_AbstractAdapter

Package: *Job\Dispatcher\Adapter*

class Omeka_Job_Dispatcher_Adapter_AbstractAdapter

implements *Omeka_Job_Dispatcher_Adapter_AdapterInterface*

Abstract class for job dispatcher adapters.

Omeka_Job_Dispatcher_Adapter_AbstractAdapter::**__construct**(*$options = null*)

Parameters

- **$options** (*array|null*) – Optional Options to instantiate in the adapter.

Omeka_Job_Dispatcher_Adapter_AbstractAdapter::**_setOptions**(*$options*)

Parameters

- **$options** –

Omeka_Job_Dispatcher_Adapter_AbstractAdapter::**getOption**(*$name*)

Retrieve an option by name as it was passed to the constructor of the adapter.

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Adapter_AbstractAdapter::**hasOption**(*$name*)
Whether or not the given option has been set.

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Adapter_AbstractAdapter::**setQueueName**(*$name*)
Adapter implementations do not understand named queues by default, so this default implementation returns false. Override this in subclasses to specify the correct behavior.

Parameters

- **$name** –

Omeka_Job_Dispatcher_Adapter_AbstractAdapter::**send**(*$encodedJob*, *$metadata*)
Send the job to whatever underlying system is used by the adapter.

Parameters

- **$encodedJob** (*string*) – The job encoded as a string. In most cases, this will be passed directly into whatever client or queue the adapter uses.
- **$metadata** (*array*) – An array containing all the metadata for the job. This is the unencoded version of the first argument and exists as a convenience so that adapter writers do not have to attempt to decode the first argument manually. This array contains the following keys: <ul> className - Corresponds to the class name of the job. options - Options that are passed to the job when it is instantiated. createdBy - User object (or null) corresponding to the user who created this job. createdAt - Zend_Date corresponding to the date/time at which this job was created.

Omeka_Job_Dispatcher_Adapter_AdapterInterface

Package: *Job\Dispatcher\Adapter*

interface Omeka_Job_Dispatcher_Adapter_AdapterInterface
Interface for job dispatcher adapters.

setQueueName(*$name*)
Set the name of the queue that the adapter will use for incoming jobs.

Note that this will not be used by some adapters and should be implemented to return false in those cases.

Parameters

- **$name** (*string*) –

send(*$encodedJob*, *$metadata*)
Send the job to whatever underlying system is used by the adapter.

Parameters

- **$encodedJob** (*string*) – The job encoded as a string. In most cases, this will be passed directly into whatever client or queue the adapter uses.
- **$metadata** (*array*) – An array containing all the metadata for the job. This is the unencoded version of the first argument and exists as a convenience so that adapter writers do not have to attempt to decode the first argument manually. This array contains the following keys: <ul> className - Corresponds to the class name of the job.

options - Options that are passed to the job when it is instantiated. createdBy - User object (or null) corresponding to the user who created this job. createdAt - Zend_Date corresponding to the date/time at which this job was created.

Omeka_Job_Dispatcher_Adapter_Array

Package: *Job\Dispatcher\Adapter*

class Omeka_Job_Dispatcher_Adapter_Array

implements *Omeka_Job_Dispatcher_Adapter_AdapterInterface*

Store dispatched jobs in an array.

This is used primarily by unit tests and should not be used in production code.

Omeka_Job_Dispatcher_Adapter_Array::**setQueueName**(*$name*)

Parameters

- **$name** –

Omeka_Job_Dispatcher_Adapter_Array::**send**(*$encodedJob*, *$metadata*)

Parameters

- **$encodedJob** –
- **$metadata** –

Omeka_Job_Dispatcher_Adapter_Array::**getJobs**()

Omeka_Job_Dispatcher_Adapter_Array::**getJob**(*$index = 0*)

Parameters

- **$index** –

Omeka_Job_Dispatcher_Adapter_BackgroundProcess

Package: *Job\Dispatcher\Adapter*

class Omeka_Job_Dispatcher_Adapter_BackgroundProcess

extends *Omeka_Job_Dispatcher_Adapter_AbstractAdapter*

implements *Omeka_Job_Dispatcher_Adapter_AdapterInterface*

Job dispatcher that uses Omeka's existing background process API.

Omeka_Job_Dispatcher_Adapter_BackgroundProcess::**send**(*$encodedJob*, *$metadata*)

Dispatches a background process that executes the given job.

NOTE: No user account is bootstrapped when background.php runs (since it is CLI), so if a process triggers its own subprocesses, those will be listed as belonging to no user (ID = 0).

Parameters

- **$encodedJob** –
- **$metadata** –

Omeka_Job_Dispatcher_Adapter_BackgroundProcess::**setProcessDispatcher**(*Omeka_Job_Process_Dis*
$dis-
patcher)

For test purposes.

Parameters

- **$dispatcher** (Omeka_Job_Process_Dispatcher) –

Omeka_Job_Dispatcher_Adapter_BackgroundProcess::**getProcessDispatcher**()

Omeka_Job_Dispatcher_Adapter_BackgroundProcess::**__construct**(*$options*
= null)

Parameters

- **$options** (*array|null*) – Optional Options to instantiate in the adapter.

Omeka_Job_Dispatcher_Adapter_BackgroundProcess::**_setOptions**(*$options*)

Parameters

- **$options** –

Omeka_Job_Dispatcher_Adapter_BackgroundProcess::**getOption**(*$name*)
Retrieve an option by name as it was passed to the constructor of the adapter.

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Adapter_BackgroundProcess::**hasOption**(*$name*)
Whether or not the given option has been set.

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Adapter_BackgroundProcess::**setQueueName**(*$name*)
Adapter implementations do not understand named queues by default, so this default implementation returns false. Override this in subclasses to specify the correct behavior.

Parameters

- **$name** –

Omeka_Job_Dispatcher_Adapter_Beanstalk

Package: *Job\Dispatcher\Adapter*

class Omeka_Job_Dispatcher_Adapter_Beanstalk

extends *Omeka_Job_Dispatcher_Adapter_AbstractAdapter*

implements *Omeka_Job_Dispatcher_Adapter_AdapterInterface*

Job dispatcher for Beanstalk.

Requires Pheanstalk library (Beanstalk client) in order to work properly.

This adapter must be instantiated with the 'host' option (IP address of beanstalk daemon) in order to work properly.

constant Omeka_Job_Dispatcher_Adapter_Beanstalk::**DEFAULT_TTR**
Because of the potential for long-running imports (and the fact that jobs are not idemopotent), TTR should be pretty high by default.

Omeka_Job_Dispatcher_Adapter_Beanstalk::**setQueueName**(*$name*)
Beanstalk understands the concept of 'tubes' instead of named queues, so set the appropriate 'tube' to dispatch jobs.

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Adapter_Beanstalk::**send**(*$encodedJob*, *$metadata*)

Parameters

- **$encodedJob** –
- **$metadata** –

Omeka_Job_Dispatcher_Adapter_Beanstalk::**_pheanstalk**()

Omeka_Job_Dispatcher_Adapter_Beanstalk::**getOption**(*$name*)

Parameters

- **$name** –

Omeka_Job_Dispatcher_Adapter_Beanstalk::**__construct**(*$options* = *null*)

Parameters

- **$options** (*array|null*) – Optional Options to instantiate in the adapter.

Omeka_Job_Dispatcher_Adapter_Beanstalk::**_setOptions**(*$options*)

Parameters

- **$options** –

Omeka_Job_Dispatcher_Adapter_Beanstalk::**hasOption**(*$name*)
Whether or not the given option has been set.

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Adapter_RequiredOptionException

Package: *Job\Dispatcher\Adapter*

class Omeka_Job_Dispatcher_Adapter_RequiredOptionException

extends LogicException

implements Throwable

Exception thrown when required options have not been passed to the Omeka_Job_Dispatcher_Adapter_AdapterInterface's constructor.

property Omeka_Job_Dispatcher_Adapter_RequiredOptionException::$**message**
protected

property Omeka_Job_Dispatcher_Adapter_RequiredOptionException::$**code**
protected

property Omeka_Job_Dispatcher_Adapter_RequiredOptionException::$**file**
protected

property Omeka_Job_Dispatcher_Adapter_RequiredOptionException::$**line**
protected

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**__clone**()

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**__wakeup**()

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**getMessage**()

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**getCode**()

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**getFile**()

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**getLine**()

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**getTrace**()

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**getPrevious**()

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**getTraceAsString**()

Omeka_Job_Dispatcher_Adapter_RequiredOptionException::**__toString**()

Omeka_Job_Dispatcher_Adapter_Synchronous

Package: *Job\Dispatcher\Adapter*

class Omeka_Job_Dispatcher_Adapter_Synchronous

extends *Omeka_Job_Dispatcher_Adapter_AbstractAdapter*

implements *Omeka_Job_Dispatcher_Adapter_AdapterInterface*

Dispatcher for executing jobs in real-time, i.e. executing within the browser request.

WARNING: While especially useful for simple jobs or instances where it is not possible to use one of the other adapters, keep in mind that long jobs may lead to request timeouts or open the possibility of DoS attacks by malicious users.

Omeka_Job_Dispatcher_Adapter_Synchronous::**send**(*$encodedJob*, *$metadata*)

Parameters

- **$encodedJob** –
- **$metadata** –

Omeka_Job_Dispatcher_Adapter_Synchronous::**__construct**(*$options* = *null*)

Parameters

- **$options** (*array|null*) – Optional Options to instantiate in the adapter.

Omeka_Job_Dispatcher_Adapter_Synchronous::**_setOptions**(*$options*)

Parameters

- **$options** –

Omeka_Job_Dispatcher_Adapter_Synchronous::**getOption**(*$name*)

Retrieve an option by name as it was passed to the constructor of the adapter.

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Adapter_Synchronous::**hasOption**(*$name*)

Whether or not the given option has been set.

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Adapter_Synchronous::**setQueueName**(*$name*)

Adapter implementations do not understand named queues by default, so this default implementation returns false. Override this in subclasses to specify the correct behavior.

Parameters

- **$name** –

Omeka_Job_Dispatcher_Adapter_ZendQueue

Package: *Job\Dispatcher\Adapter*

class Omeka_Job_Dispatcher_Adapter_ZendQueue

extends *Omeka_Job_Dispatcher_Adapter_AbstractAdapter*

implements *Omeka_Job_Dispatcher_Adapter_AdapterInterface*

Dispatcher for Zend_Queue.

This would be particularly useful for installations that want to interface with ActiveMQ or Zend Server's Job Queue via Zend_Queue. Note that using the 'Array' adapter should only be used for testing, as all jobs passed to it will be thrown away.

Required options include 'adapter' and 'options', which correspond to the first and second arguments to Zend_Queue's constructor respectively.

For example, it would be configured like so in config.ini: <code> jobs.dispatcher = "Omeka_Job_Dispatcher_ZendQueue" jobs.adapterOptions.adapter = "PlatformJobQueue" jobs.adapterOptions.options.host = "127.0.0.1" jobs.adapterOptions.options.password = "foobar" </code>

Omeka_Job_Dispatcher_Adapter_ZendQueue::**setQueueName**(*$name*)

Note that some Zend_Queue implementations understand the concept of named queues, while others do not.

Parameters

- **$name** –

Omeka_Job_Dispatcher_Adapter_ZendQueue::**send**(*$encodedJob*, *$metadata*)

Parameters

- **$encodedJob** –

- **$metadata** –

Omeka_Job_Dispatcher_Adapter_ZendQueue::**_queue**()

Omeka_Job_Dispatcher_Adapter_ZendQueue::**__construct**(*$options = null*)

Parameters

- **$options** (*array|null*) – Optional Options to instantiate in the adapter.

Omeka_Job_Dispatcher_Adapter_ZendQueue::**_setOptions**(*$options*)

Parameters

- **$options** –

Omeka_Job_Dispatcher_Adapter_ZendQueue::**getOption**(*$name*)
Retrieve an option by name as it was passed to the constructor of the adapter.

Parameters

- **$name** (*string*) –

Omeka_Job_Dispatcher_Adapter_ZendQueue::**hasOption**(*$name*)
Whether or not the given option has been set.

Parameters

- **$name** (*string*) –

Job\Factory

Up to *Job*

Omeka_Job_Factory

Package: *Job\Factory*

class Omeka_Job_Factory
Factory for instantiating Omeka_Job instances.

__construct(*$options = array()*)

Parameters

- **$options** –

from(*$json*)
Decode a message from JSON and use the results to instantiate a new job instance.

Parameters

- **$json** (*string*) –

build(*$data*)
Instantiate a new job instance from the arguments given.

Parameters

- **$data** –

Omeka_Job_Factory_MalformedJobException

Package: *Job\Factory*

class Omeka_Job_Factory_MalformedJobException

extends `InvalidArgumentException`

implements `Throwable`

Exception thrown when the message could not be decoded into valid job metadata.

property `Omeka_Job_Factory_MalformedJobException::$`**message**
protected

property `Omeka_Job_Factory_MalformedJobException::$`**code**
protected

property `Omeka_Job_Factory_MalformedJobException::$`**file**
protected

property `Omeka_Job_Factory_MalformedJobException::$`**line**
protected

`Omeka_Job_Factory_MalformedJobException::`**__clone**`()`

`Omeka_Job_Factory_MalformedJobException::`**__construct**`(`*$message*, *$code*, *$previous*`)`

Parameters

- **$message** –
- **$code** –
- **$previous** –

`Omeka_Job_Factory_MalformedJobException::`**__wakeup**`()`

`Omeka_Job_Factory_MalformedJobException::`**getMessage**`()`

`Omeka_Job_Factory_MalformedJobException::`**getCode**`()`

`Omeka_Job_Factory_MalformedJobException::`**getFile**`()`

`Omeka_Job_Factory_MalformedJobException::`**getLine**`()`

`Omeka_Job_Factory_MalformedJobException::`**getTrace**`()`

`Omeka_Job_Factory_MalformedJobException::`**getPrevious**`()`

`Omeka_Job_Factory_MalformedJobException::`**getTraceAsString**`()`

`Omeka_Job_Factory_MalformedJobException::`**__toString**`()`

Omeka_Job_Factory_MissingClassException

Package: *Job\Factory*

class Omeka_Job_Factory_MissingClassException

extends `InvalidArgumentException`

implements `Throwable`

Exception thrown when the type of job could not be inferred from the message passed to the factory.

property Omeka_Job_Factory_MissingClassException::$**message**
protected

property Omeka_Job_Factory_MissingClassException::$**code**
protected

property Omeka_Job_Factory_MissingClassException::$**file**
protected

property Omeka_Job_Factory_MissingClassException::$**line**
protected

Omeka_Job_Factory_MissingClassException::**__clone**()

Omeka_Job_Factory_MissingClassException::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Job_Factory_MissingClassException::**__wakeup**()

Omeka_Job_Factory_MissingClassException::**getMessage**()

Omeka_Job_Factory_MissingClassException::**getCode**()

Omeka_Job_Factory_MissingClassException::**getFile**()

Omeka_Job_Factory_MissingClassException::**getLine**()

Omeka_Job_Factory_MissingClassException::**getTrace**()

Omeka_Job_Factory_MissingClassException::**getPrevious**()

Omeka_Job_Factory_MissingClassException::**getTraceAsString**()

Omeka_Job_Factory_MissingClassException::**__toString**()

Job\Process

Up to *Job*

Omeka_Job_Process_AbstractProcess

Package: *Job\Process*

class Omeka_Job_Process_AbstractProcess
Base class background processes descend from.

property _process
protected

__construct(*Process $process*)

Parameters

- **$process** (Process) –

__destruct()

run(*$args*)

Parameters

- **$args** –

Omeka_Job_Process_Dispatcher

Package: *Job\Process*

class Omeka_Job_Process_Dispatcher

Spawns and manages background processes.

startProcess(*$className*, *$user = null*, *$args = null*)

Create a table entry for a new background process and spawn it.

Parameters

- **$className** (*string*) – Omeka_Job_Process_AbstractProcess subclass name to spawn
- **$user** (User) – User to run process as, defaults to current user
- **$args** (*Array|null*) – Arguments specific to the child class process

Returns Process The model object for the background process

stopProcess(*Process $process*)

Stops a background process in progress.

Parameters

- **$process** (Process) – The process to stop.

Returns bool True if the process was stopped, false if not.

getPHPCliPath()

_checkCliPath(*$cliPath*)

Checks if the configured PHP-CLI path points to a valid PHP binary. Flash an appropriate error if the path is invalid.

Parameters

- **$cliPath** –

_autodetectCliPath()

_getBootstrapFilePath()

Returns the path to the background bootstrap script.

Returns string Path to bootstrap

_fork(*$command*)

Launch a background process, returning control to the foreground.

Parameters

- **$command** –

Omeka_Job_Process_Wrapper

Package: *Job\Process*

class Omeka_Job_Process_Wrapper

extends *Omeka_Job_Process_AbstractProcess*

Wrapper that allows Omeka_Job to work with the existing Process/ Omeka_Job_Process_Dispatcher API. Jobs are passed in as the 'job' argument, and this wrapper handles decoding and executing the job.

property Omeka_Job_Process_Wrapper::$**_process**
protected

Omeka_Job_Process_Wrapper::**_getJob**(*$str*)

Parameters

- **$str** –

Omeka_Job_Process_Wrapper::**run**(*$args*)
Args passed in will consist of the JSON-encoded task.

Parameters

- **$args** –

Omeka_Job_Process_Wrapper::**__construct**(*Process $process*)

Parameters

- **$process** (Process) –

Omeka_Job_Process_Wrapper::**__destruct**()

Job\Worker

Up to *Job*

Omeka_Job_Worker_Beanstalk

Package: *Job\Worker*

class Omeka_Job_Worker_Beanstalk

__construct(*Pheanstalk_Pheanstalk $pheanstalk*, *Omeka_Job_Factory $jobFactory*, *Omeka_Db $db*)

Parameters

- **$pheanstalk** (*Pheanstalk_Pheanstalk*) –
- **$jobFactory** (Omeka_Job_Factory) –
- **$db** (Omeka_Db) –

work(*Pheanstalk_Job $pJob*)

Parameters

- **$pJob** (*Pheanstalk_Job*) –

_interrupt(*$job = null*)

Parameters

- **$job** –

Omeka_Job_Worker_InterruptException

Package: *Job\Worker*

class Omeka_Job_Worker_InterruptException

extends `Exception`

implements `Throwable`

Exception thrown when the type of job could not be inferred from the message passed to the factory.

property Omeka_Job_Worker_InterruptException::$**message**
protected

property Omeka_Job_Worker_InterruptException::$**code**
protected

property Omeka_Job_Worker_InterruptException::$**file**
protected

property Omeka_Job_Worker_InterruptException::$**line**
protected

Omeka_Job_Worker_InterruptException::**__clone**()

Omeka_Job_Worker_InterruptException::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Job_Worker_InterruptException::**__wakeup**()

Omeka_Job_Worker_InterruptException::**getMessage**()

Omeka_Job_Worker_InterruptException::**getCode**()

Omeka_Job_Worker_InterruptException::**getFile**()

Omeka_Job_Worker_InterruptException::**getLine**()

Omeka_Job_Worker_InterruptException::**getTrace**()

Omeka_Job_Worker_InterruptException::**getPrevious**()

Omeka_Job_Worker_InterruptException::**getTraceAsString**()

Omeka_Job_Worker_InterruptException::**__toString**()

3.7.13 Navigation

Up to *Packages*

Omeka_Navigation

Package: *Navigation*

class Omeka_Navigation

extends Zend_Navigation

Customized subclass of Zend Framework's Zend_Navigation class.

Omeka_Navigation::**__construct**(*$pages = null*)
Creates a new navigation container

Parameters

- **$pages** (*array|Zend_Config*) – [optional] pages to add

Omeka_Navigation::**saveAsOption**(*$optionName*)
Saves the navigation in the global options table.

Parameters

- **$optionName** (*String*) – The name of the option

Omeka_Navigation::**loadAsOption**(*$optionName*)
Loads the navigation from the global options table

Parameters

- **$optionName** (*String*) – The name of the option

Omeka_Navigation::**addPage**(*$page*)
Adds a page to the container. If a page does not have a valid id, it will give it one. If a direct child page already has another page with the same uid then it will not add the page. However, it will add the page as a child of this navigation if one of its descendants already has the page.

This method will inject the container as the given page's parent by calling {@link Zend_Navigation_Page::setParent()}.

Parameters

- **$page** –

Returns Zend_Navigation_Container fluent interface, returns self

Omeka_Navigation::**getChildByUid**(*$uid*)
Returns an immediate child page that has a uid of $uid. If none exists, it returns null.

Parameters

- **$uid** (*string*) – The uid to search for in this navigation

Returns Zend_Navigation_Page The page

Omeka_Navigation::**addPageToContainer**(*$page*, *$container*)
Adds a page to a container after normalizing it and its subpages

Parameters

- **$page** (*Zend_Navigation_Page*) – The page to add
- **$container** (*Zend_Navigation_Container*) – The container to which to add the page

Returns Zend_Navigation_Container The container with the page added

Omeka_Navigation::**createNavigationFromFilter**(*$filterName = ''*)
Creates an Omeka Navigation object by adding pages generated by Omeka plugins and other contributors via a filter (e.x. 'public_navigation_main'). The filter should provide an array pages like they are added to Zend_Navigation_Container::addPages However, the page types should only be one of the following types: Omeka_Navigation_Page_Uri or Zend_Navigation_Page_Mvc. If the associated uri of any page is invalid, it will not add that page to the navigation. Also, it removes expired pages from formerly active plugins and other former handlers of the filter.

Parameters

- **$filterName** (*String*) – The name of the filter

Omeka_Navigation::**baseAddNormalizedPage**(*$normalizedPage*)
Add a normalized page to the navigation using parent::addPage() This needs to wrapped so that methods like createNavigationFromFilter() can add pages directly using the parent class method.

Parameters

- **$normalizedPage** –

Returns Zend_Navigation_Container fluent interface, returns self

Omeka_Navigation::**mergePage**(*Zend_Navigation_Page $page*, *Zend_Navigation_Container $parentContainer = null*)
Merges a page (and its subpages) into this navigation. If the page already exists in the navigation, then it attempts to add any new subpages of the page to it. If a subpages already exists in the navigation, then it it recursively attempts to add its new subpages to it, and so on.

Parameters

- **$page** (*Zend_Navigation_Page*) –
- **$parentContainer** (*Zend_Navigation_Container*) –

Returns Zend_Navigation_Container $parentContainer the suggested parentContainer for the page. The parentContainer must already be in the navigation and remain so throughout the merge.

Omeka_Navigation::**_getLastPageOrderInContainer**(*$container*)
Returns the page order of the last child page in the container. If no page exists in the container, it returns 0.

Parameters

- **$container** –

Returns int the last page order in the container

Omeka_Navigation::**mergeNavigation**(*Omeka_Navigation $nav*)
Merges a navigation object into this navigation.

Parameters

- **$nav** (Omeka_Navigation) –

Omeka_Navigation::**addPagesFromFilter**(*$filterName = ''*)
Adds pages generated by Omeka plugins and other contributors via a filter (e.x. 'public_navigation_main'). The filter should provide an array pages like they are added to Zend_Navigation_Container::addPages However, the page types should only be one of the following types: Omeka_Navigation_Page_Uri or Omeka_Navigation_Page_Mvc. If the associated uri of any page is invalid, it will not add that page to the navigation. Also, it removes expired pages from formerly active plugins and other former handlers of the filter.

Parameters

- **$filterName** (*String*) – The name of the filter

Omeka_Navigation::**getExpiredPagesFromNav**(*Omeka_Navigation $freshNav*)
Returns an array of expired pages from this navigation, where all pages in the $feshNav are considered non-expired.

Parameters

- **$freshNav** (Omeka_Navigation) –

Returns array The array of expired pages

Omeka_Navigation::**prunePage**(*$page*)
Prune page from this navigation. When a page is pruned its children pages are reattached to the first non-pruneable ancestor page.

Parameters

- **$page** (*Omeka_Navigation_Page_Mvc|Omeka_Navigation_Page_Uri*) – The page to prune

Omeka_Navigation::**getOtherPages**(*$excludePageUids = null*)
Returns an array of all pages from navigation that lack a uid in $excludePageUids

Parameters

- **$excludePageUids** (*array|null*) – The list uids for pages to exclude

Returns array The array of other pages.

Omeka_Navigation::**getPageByUid**(*$pageUid*, *$container = null*)
Returns the navigation page associated with uid. It searches all descendant pages of this navigation If not page is associated, then it returns null.

Parameters

- **$pageUid** (*String*) – The uid of the page
- **$container** (*Zend_Navigation_Container*) – The container within which to search for the page. By default, it uses this navigation.

Returns Omeka_Zend_Navigation_Page_Uri|Omeka_Navigation_Page_Mvc|null

Omeka_Navigation::**createPageUid**(*$href*)
Returns the unique id for the page, which can be used to determine whether it can be added to the navigation

Parameters

- **$href** (*String*) – The href of the page.

Returns String

Omeka_Navigation::**removePageRecursive**(*Zend_Navigation_Page $page*, *Zend_Navigation_Container $parentContainer = null*, *$reattach = false*)
Recursively removes the given page from the parent container, including all subpages

Parameters

- **$page** (*Zend_Navigation_Page*) – The page to remove from the parent container and all its subpages.

- **$parentContainer** (*Zend_Navigation_Container*) – The parent container (by default it is this navigation) from which to remove the page from its subpages
- **$reattach** (*bool*) – Whether the subpages of the $page should be reattached to $parentContainer

Returns bool Whether the page was removed

Omeka_Navigation::**getNavigationOptionValueForInstall**()

Returns the option value associated with the default navigation during installation

Returns String The option value associated with the default navigation during installation. If no option is found for the option name, then it returns an empty string.

Omeka_Navigation::**_normalizePageRecursive**(*$page*, *$pageOptions = array()*)

Normalizes a page and its subpages so it can be added

Parameters

- **$page** –
- **$pageOptions** – The options to set during normalization for every page and subpage

Returns Omeka_Navigation_Page_Uri|Omeka_Navigation_Page_Mvc|null The normalized page

Omeka_Navigation::**_convertZendToOmekaNavigationPage**(*Zend_Navigation_Page $page*, *$subclassPostfix*)

Converts a Zend_Navigation_Page subclass object to a corresponding Omeka object

Parameters

- **$page** (*Zend_Navigation_Page*) – The page to convert
- **$subclassPostfix** (*string*) – The postfix of the subclass. Must be 'Uri' or 'Mvc'

Returns Omeka_Navigation_Page_Uri|Omeka_Navigation_Page_Mvc The converted page

Omeka_Navigation::**_conditionalReplaceValueInArray**(*$array*, *$childKey*, *$targetKey*, *$oldValue*, *$newValue*)

Returns an nested associative array such that all array elements have replaced an key value to a new key value only if it is equal to a specific old key value.

Parameters

- **$array** (*array*) – The associative array
- **$childKey** (*string*) – The associative array
- **$targetKey** (*string*) – The target key whose value can be replaced
- **$oldValue** (*mixed*) – The old value of the element associated with the target key used to determine if the value should be changed
- **$newValue** (*mixed*) – The new value of the element associated with the target key

Returns array The replaced associative array

Omeka_Navigation_Page_Mvc

Package: *Navigation*

class Omeka_Navigation_Page_Mvc

extends Zend_Navigation_Page_Mvc

Customized subclass of Zend Framework's Zend_Navigation_Page_Mvc class.

property Omeka_Navigation_Page_Mvc::$**_theme**
protected string

Theme option to use when assembling URL

Omeka_Navigation_Page_Mvc::**getHref**()
Returns href for this page

This method uses {@link Zend_Controller_Action_Helper_Url} to assemble the href based on the page's properties.

Returns string page href

Omeka_Navigation_Page_Mvc::**getTheme**()
Returns theme option to use when assembling URL

Returns string|null theme option

Omeka_Navigation_Page_Mvc::**setTheme**(*$theme*)
Sets theme option to use when assembling URL

Parameters

- **$theme** –

Returns Omeka_Navigation_Page_Mvc fluent interface, returns self

Omeka_Navigation_Page_Uri

Package: *Navigation*

class Omeka_Navigation_Page_Uri

extends Zend_Navigation_Page_Uri

Customized subclass of Zend Framework's Zend_Navigation_Page_Uri class.

Omeka_Navigation_Page_Uri::**isActive**(*$recursive = false*)
Returns whether page should be considered active or not

Parameters

- **$recursive** –

Returns bool whether page should be considered active

Omeka_Navigation_Page_Uri::**setHref**(*$href*)
Sets page href. It will parse the href and update the uri and fragment properties.

Parameters

- **$href** –

Returns Omeka_Navigation_Page_Uri fluent interface, returns self

Omeka_Navigation_Page_Uri::**_normalizeHref**(*$href*)
Normalizes a string href for a navigation page and returns an array with the following keys: 'uri' => the uri of the href. 'fragment' => the fragment of the href If the $href is a relative path, then it must be a root path. If the $href is a relative path then the value for the 'uri' key will be a relative path. If $href is an invalid uri, then return null.

Parameters

- **$href** (*String*) –

Returns array

Omeka_Navigation_Page_Uri_Exception

Package: *Navigation*

class Omeka_Navigation_Page_Uri_Exception

extends Zend_Exception

Exception for the Omeka_Navigation_Page_Uri

3.7.14 Output

Up to *Packages*

Omeka_Output_OmekaXml_AbstractOmekaXml

Package: *Output*

class Omeka_Output_OmekaXml_AbstractOmekaXml
Abstract base class for creating omeka-xml output formats.

constant XMLNS_XSI
XML Schema instance namespace URI.

constant XMLNS
Omeka-XML namespace URI.

constant XMLNS_SCHEMALOCATION
Omeka-XML XML Schema URI.

property _record
protected array|Omeka_Record_AbstractRecord

This class' contextual record(s).

property _context
protected string

The context of this DOMDocument. Determines how buildNode() builds the elements. Valid contexts include: item, file.

property _doc
protected DOMDocument

The final document object.

property _node
protected DOMNode

The node built and set in child::_buildNode()

_buildNode()
Abstract method. child::_buildNode() should set self::$_node.

__construct(*$record*, *$context*)

Parameters

- **$record** (*Omeka_Record_AbstractRecord|array*) –
- **$context** (*string*) – The context of this DOM document.

getDoc()
Get the document object.

Returns DOMDocument

_setRootElement(*$rootElement*)
Set an element as root.

Parameters

- **$rootElement** (*DOMElement*) –

Returns DOMElement The root element, including required attributes.

_createElement(*$name*, *$value = null*, *$id = null*, *$parentElement = null*)
Create a DOM element.

Parameters

- **$name** (*string*) – The name of the element.
- **$value** –
- **$id** –
- **$parentElement** –

Returns DOMElement

_setContainerPagination(*DOMElement $parentElement*)
Set the pagination node for container elements

Parameters

- **$parentElement** (*DOMElement*) –

_getElemetSetsByElementTexts(*Omeka_Record_AbstractRecord $record*, *$getItemType = false*)
Get all element sets, elements, and element texts associated with the provided record.

Parameters

- **$record** (Omeka_Record_AbstractRecord) – The record from which to extract metadata.
- **$getItemType** (*bool*) – Whether to get the item type metadata.

Returns stdClass A list of element sets or an item type.

_buildElementSetContainerForRecord(*Omeka_Record_AbstractRecord $record*, *DOMElement $parentElement*)
Build an elementSetContainer element in a record (item or file) context.

Parameters

- **$record** (Omeka_Record_AbstractRecord) – The record from which to build element sets.
- **$parentElement** (*DOMElement*) – The element set container will append to this element.

Returns void|null

_buildItemTypeForItem(*Item $item*, *DOMElement $parentElement*)
Build an itemType element in an item context.

Parameters

- **$item** (Item) – The item from which to build the item type.
- **$parentElement** (*DOMElement*) – The item type will append to this element.

Returns void|null

_buildFileContainerForItem(*Item $item*, *DOMElement $parentElement*)
Build a fileContainer element in an item context.

Parameters

- **$item** (Item) – The item from which to build the file container.
- **$parentElement** (*DOMElement*) – The file container will append to this element.

Returns void|null

_buildCollectionForItem(*Item $item*, *DOMElement $parentElement*)
Build a collection element in an item context.

Parameters

- **$item** (Item) – The item from which to build the collection.
- **$parentElement** (*DOMElement*) – The collection will append to this element.

Returns void|null

_buildTagContainerForItem(*Item $item*, *DOMElement $parentElement*)
Build a tagContainer element in an item context.

Parameters

- **$item** (Item) – The item from which to build the tag container.
- **$parentElement** (*DOMElement*) – The tag container will append to this element.

Returns void|null

_buildItemContainerForCollection(*Collection $collection*, *DOMElement $parentElement*)
Build an itemContainer element in a collection context.

Parameters

- **$collection** (Collection) – The collection from which to build the item container.
- **$parentElement** (*DOMElement*) – The item container will append to this element.

Returns void|null

_buildTagUri()
Create a Tag URI to uniquely identify this Omeka XML instance.

Returns string

_buildUrl()
Create a absolute URI containing the current query string.

Returns string

3.7.15 Plugin

Up to *Packages*

Omeka_Plugin_AbstractPlugin

Package: *Plugin*

class Omeka_Plugin_AbstractPlugin
Abstract plugin class.

Plugin authors may inherit from this class to aid in building their plugin framework.

property _db
protected Omeka_Db

Database object accessible to plugin authors.

property _hooks
protected array

Plugin hooks.

In the child class plugin authors should set an array containing hook names as values and, optionally, callback names as keys. If a callback name is given, the child class should contain an identically named method. If no callback key is given, the child class should contain a corresponding hookCamelCased() method. E.g: the after_save_form_record filter should have a corresponding hookAfterSaveRecord() method.

For example: <code> array('install', 'uninstall', 'doSomething' => 'after_save_item') </code>

property _filters
protected array

Plugin filters.

In the child class plugin authors should set an array containing filter names as values and, optionally, callback names as keys. If a callback name is given, the child class should contain an identically named method. If no callback key is given, the child class should contain a corresponding filterCamelCased() method. E.g: the admin_navigation_main filter should have a corresponding filterAdminNavigationMain() method.

For example: <code> array('admin_navigation_main', 'public_navigation_main', 'changeSomething' => 'display_option_site_title', 'displayItemDublinCoreTitle' => array('Display', 'Item', 'Dublin Core', 'Title')) </code>

property _options
protected array

Plugin options.

Plugin authors should give an array containing option names as keys and their default values as values, if any.

For example: <code> array('option_name1' => 'option_default_value1', 'option_name2' => 'option_default_value2', 'option_name3', 'option_name4') </code>

__construct()
Construct the plugin object.

Sets the database object. Plugin authors must call parent::__construct() in the child class's constructor, if used.

setUp()
Set up the plugin to hook into Omeka.

Adds the plugin's hooks and filters. Plugin writers must call this method after instantiating their plugin class.

_installOptions()
Set options with default values.

Plugin authors may want to use this convenience method in their install hook callback.

_uninstallOptions()
Delete all options.

Plugin authors may want to use this convenience method in their uninstall hook callback.

_addHooks()
Validate and add hooks.

_addFilters()
Validate and add filters.

Omeka_Plugin_Exception

Package: *Plugin*

class Omeka_Plugin_Exception

extends `Exception`

implements `Throwable`

Exception type thrown by the Omeka_Plugin class.

property `Omeka_Plugin_Exception::$`**message**
protected

property `Omeka_Plugin_Exception::$`**code**
protected

property `Omeka_Plugin_Exception::$`**file**
protected

property `Omeka_Plugin_Exception::$`**line**
protected

`Omeka_Plugin_Exception::`**__clone**()

`Omeka_Plugin_Exception::`**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

`Omeka_Plugin_Exception::`**__wakeup**()

Omeka_Plugin_Exception::**getMessage**()

Omeka_Plugin_Exception::**getCode**()

Omeka_Plugin_Exception::**getFile**()

Omeka_Plugin_Exception::**getLine**()

Omeka_Plugin_Exception::**getTrace**()

Omeka_Plugin_Exception::**getPrevious**()

Omeka_Plugin_Exception::**getTraceAsString**()

Omeka_Plugin_Exception::**__toString**()

Omeka_Plugin_Factory

Package: *Plugin*

class Omeka_Plugin_Factory

Responsible for creating a set of Plugin records corresponding to plugins that have not been installed yet.

property _basePath

protected string

Base path for plugins; the plugin directory

__construct(*$basePath*)

Parameters

- **$basePath** (*string*) – Plugin base directory.

getNewPlugins(*$existingPlugins*)

Retrieve all new plugins in the plugin directory.

Parameters

- **$existingPlugins** (*array*) – An array of existing Plugin objects.

Returns array An array of Plugin objects for the new plugins.

_getDirectoryList()

Retrieve an array of all the plugins in the plugin directory. A plugin is considered to be present when a directory includes a plugin.php file or has a valid plugin class.

Returns array A list of valid plugin directory names.

Omeka_Plugin_Ini

Package: *Plugin*

class Omeka_Plugin_Ini

Responsible for parsing the plugin.ini file for any given plugin.

property _pluginsRootDir

protected string

Plugins directory.

property _configs

protected array

Set of Zend_Config_Ini objects corresponding to each plugin.

__construct (*$pluginsRootDir*)

Parameters

- **$pluginsRootDir** (*string*) – Plugins directory.

getPluginIniValue (*$pluginDirName*, *$iniKeyName*)
Retrieve a value in plugin.ini for a given key.

Will return a null value if no value can be found in the ini file for the key.

Parameters

- **$pluginDirName** (*string*) – Plugin name.
- **$iniKeyName** (*string*) – INI key to retrieve.

Returns string|null Retrieved INI value (null if not found).

hasPluginIniFile (*$pluginDirName*)
Return whether a plugin has a plugin.ini file

Parameters

- **$pluginDirName** (*string*) – Plugin name.

Returns bool

getPluginIniFilePath (*$pluginDirName*)
Return the path to the plugin.ini file

Parameters

- **$pluginDirName** (*string*) – Plugin name.

Returns string

load (*Plugin $plugin*)
Initialize a Plugin model object with the values from the INI file.

Parameters

- **$plugin** (Plugin) – The plugin model to initialize.

Omeka_Plugin_Mvc

Package: *Plugin*

class Omeka_Plugin_Mvc
Connects plugins with Omeka's model-view-controller system.

property _basePath
protected string

Path to the root plugins directory.

property _pluginViewDirs
protected array

View script directories that have been added by plugins.

property _pluginHelpersDirs
protected array

View helper directories from plugins.

__construct (*$basePath*)

Parameters

- **$basePath** (*string*) – Plugins directory path.

addThemeDir(*$pluginDirName*, *$path*, *$themeType*, *$moduleName*)
Add a theme directory to the list of plugin-added view directories.

Used by the add_theme_pages() helper to create a list of directories that can store static pages that integrate into the themes.

Parameters

- **$pluginDirName** (*string*) – Plugin name.
- **$path** (*string*) – Path to directory to add.
- **$themeType** (*string*) – Type of theme ('public', 'admin', or 'shared').
- **$moduleName** (*string*) – MVC module name.

getViewScriptDirs(*$themeType*)
Retrieve the list of plugin-added view script directories.

Parameters

- **$themeType** (*string*) – Type of theme (public or admin)

Returns array Module-name-indexed directory names.

getHelpersDirs()
Get all the existing plugin view helper dirs, indexed by plugin name.

Returns array

addControllerDir(*$pluginDirName*, *$moduleName*)
Make an entire directory of controllers available to the front controller.

This has to use addControllerDirectory() instead of addModuleDirectory() because module names are case-sensitive and module directories need to be lowercased to conform to Zend's weird naming conventions.

Parameters

- **$pluginDirName** (*string*) – Plugin name.
- **$moduleName** (*string*) – MVC module name.

addApplicationDirs(*$pluginDirName*)
Set up the following directory structure for plugins:

controllers/ models/ libraries/ views/ admin/ public/ shared/

This also adds these folders to the correct include paths.

Parameters

- **$pluginDirName** (*string*) – Plugin name.

_getModuleName(*$pluginDirName*)
Retrieve the module name for the plugin (based on the directory name of the plugin).

Parameters

- **$pluginDirName** (*string*) – Plugin name.

Returns string Plugin MVC module name.

_hasIncludePath(*$path*)
Check include path to see if it already contains a specific path.

Parameters

- **$path** (*string*) –

Returns bool

Plugin\Broker

Up to *Plugin*

Omeka_Plugin_Broker

Package: *Plugin\Broker*

class Omeka_Plugin_Broker

Plugin Broker for Omeka.

For example, $broker->callHook('add_action_contexts', array('controller' => $controller)) would call the 'add_action_contexts' on all plugins, and it would provide the controller object as the first argument to all implementations of that hook.

property _callbacks

protected array

Array of hooks that have been implemented for plugins.

property _filters

protected array

Stores all defined filters.

Storage in array where $_filters['filterName']['priority']['plugin'] = $hook;

property _current

protected string

The directory name of the current plugin (used for calling hooks)

addHook (*$hook*, *$callback*, *$plugin = null*)

Add a hook implementation for a plugin.

Parameters

- **$hook** (*string*) – Name of the hook being implemented.
- **$callback** (*string*) – PHP callback for the hook implementation.
- **$plugin** (*string|null*) – Optional name of the plugin for which to add the hook. If omitted, the current plugin is used.

getHook (*$pluginDirName*, *$hook*)

Get the hook implementation for a plugin.

Parameters

- **$pluginDirName** (*string*) – Name of the plugin to get the implementation from.
- **$hook** (*string*) – Name of the hook to get the implementation for.

Returns callback|null

setCurrentPluginDirName(*$pluginDirName*)
Set the currently-focused plugin by directory name.

The plugin helper functions do not have any way of determining what plugin to is currently in focus. These get/setCurrentPluginDirName methods allow the broker to know how to delegate to specific plugins if necessary.

Parameters

- **$pluginDirName** (*string*) – Plugin to set as current.

getCurrentPluginDirName()
Get the directory name of the currently-focused plugin.

Returns string

callHook(*$name*, *$args = array()*, *$plugin = null*)
Call a hook by name.

Hooks can either be called globally or for a specific plugin only.

Parameters

- **$name** (*string*) – The name of the hook.
- **$args** (*array*) – Arguments to be passed to the hook implementations.
- **$plugin** (*Plugin|string*) – Name of the plugin that will invoke the hook.

addFilter(*$name*, *$callback*, *$priority = 10*)
Add a filter implementation.

Parameters

- **$name** (*string|array*) – Name of filter being implemented.
- **$callback** (*callback*) – PHP callback for filter implementation.
- **$priority** –

_getFilterNamespace()
Retrieve the namespace to use for the filter to be added.

Returns string Name of the current plugin (if applicable). Otherwise, a magic constant that denotes globally applied filters.

_getFilterKey(*$name*)
Retrieve the key used for indexing the filter. The filter name should be either a string or an array of strings. If the filter name is an object, that might cause fiery death when using the serialized value for an array key.

Parameters

- **$name** (*string|array*) – Filter name.

Returns string Key for filter indexing.

getFilters(*$hookName*)
Return all the filters for a specific hook in the correct order of execution.

Parameters

- **$hookName** (*string|array*) – Filter name.

Returns array Indexed array of filter callbacks.

clearFilters(*$name = null*)
Clear all implementations for a filter (or all filters).

Parameters

- **$name** (*string|null*) – The name of the filter to clear. If null or omitted, all filters will be cleared.

applyFilters (*$name*, *$value*, *$args = array()*)
Run an arbitrary value through a set of filters.

Parameters

- **$name** (*mixed*) – The filter name.
- **$value** (*mixed*) – The value to filter.
- **$args** (*array*) – Additional arguments to pass to filter implementations.

Returns mixed Result of applying filters to $value.

register ()
Register the plugin broker so that plugin writers can use global functions like add_plugin_hook() to interact with the plugin API.

Omeka_Plugin_Broker_Factory

Package: *Plugin\Broker*

class Omeka_Plugin_Broker_Factory

__construct (*$basePluginDir*)

Parameters

- **$basePluginDir** –

getAll ()

_register (*$objs*)

Parameters

- **$objs** –

Plugin\Installer

Up to *Plugin*

Omeka_Plugin_Installer

Package: *Plugin\Installer*

class Omeka_Plugin_Installer
Changes the state of any given plugin (installed/uninstalled/activated/deactivated)

property _broker
protected Omeka_Plugin_Broker

Plugin broker object.

property _loader
protected Omeka_Plugin_Loader

Plugin loader object.

__construct(*Omeka_Plugin_Broker $broker*, *Omeka_Plugin_Loader $loader*)

Parameters

- **$broker** (Omeka_Plugin_Broker) – Plugin broker object.
- **$loader** (Omeka_Plugin_Loader) – Plugin loader object.

activate(*Plugin $plugin*)
Activate a plugin.

Parameters

- **$plugin** (Plugin) – Plugin to activate.

deactivate(*Plugin $plugin*)
Deactivate a plugin.

Parameters

- **$plugin** (Plugin) – Plugin to deactivate.

upgrade(*Plugin $plugin*)
Upgrade a plugin.

This will activate the plugin, then run the 'upgrade' hook.

Parameters

- **$plugin** (Plugin) – Plugin to upgrade.

install(*Plugin $plugin*)
Install a plugin.

This will activate the plugin, then run the 'install' hook.

Parameters

- **$plugin** (Plugin) – Plugin to install.

uninstall(*Plugin $plugin*)
Uninstall a plugin.

This will run the 'uninstall' hook for the given plugin, and then it will remove the entry in the DB corresponding to the plugin.

Parameters

- **$plugin** (Plugin) – Plugin to uninstall.

Omeka_Plugin_Installer_Exception

Package: *Plugin\Installer*

class Omeka_Plugin_Installer_Exception

extends Exception

implements Throwable

An exception thrown when the plugin installer is unable to install, uninstall, activate, deactivate, or upgrade a plugin.

property Omeka_Plugin_Installer_Exception::$**message**
protected

property Omeka_Plugin_Installer_Exception::$**code**
protected

property Omeka_Plugin_Installer_Exception::$**file**
protected

property Omeka_Plugin_Installer_Exception::$**line**
protected

Omeka_Plugin_Installer_Exception::**__clone**()

Omeka_Plugin_Installer_Exception::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Plugin_Installer_Exception::**__wakeup**()

Omeka_Plugin_Installer_Exception::**getMessage**()

Omeka_Plugin_Installer_Exception::**getCode**()

Omeka_Plugin_Installer_Exception::**getFile**()

Omeka_Plugin_Installer_Exception::**getLine**()

Omeka_Plugin_Installer_Exception::**getTrace**()

Omeka_Plugin_Installer_Exception::**getPrevious**()

Omeka_Plugin_Installer_Exception::**getTraceAsString**()

Omeka_Plugin_Installer_Exception::**__toString**()

Plugin\Loader

Up to *Plugin*

Omeka_Plugin_Loader

Package: *Plugin\Loader*

class Omeka_Plugin_Loader
Loads plugins for any given request.

This will iterate through the plugins root directory and load all plugin.php files by require()'ing them.

property _broker
protected Omeka_Plugin_Broker

Plugin broker object.

property _iniReader
protected Omeka_Plugin_Ini

Plugin INI reader object.

property _mvc
protected Omeka_Plugin_Mvc

Plugin MVC object.

property _basePath
protected string

Plugins directory.

property _plugins
protected array

An array of all plugins (installed or not) that are currently located in the plugins/ directory.

__construct(*Omeka_Plugin_Broker $broker*, *Omeka_Plugin_Ini $iniReader*, *Omeka_Plugin_Mvc $mvc*, *$pluginsBaseDir*)

Parameters

- **$broker** (Omeka_Plugin_Broker) – Plugin broker.
- **$iniReader** (Omeka_Plugin_Ini) – plugin.ini reader.
- **$mvc** (Omeka_Plugin_Mvc) – Plugin MVC object.
- **$pluginsBaseDir** (*string*) – Plugins directory.

loadPlugins(*$plugins*, *$force = false*)
Load a list of plugins.

Parameters

- **$plugins** (*array*) – List of Plugin records to load.
- **$force** (*bool*) – If true, throws exceptions for plugins that cannot be loaded for some reason.

registerPlugin(*Plugin $plugin*)
Register a plugin so that it can be accessed by other plugins (if necessary) during the load process.

There should only be a single instance of a plugin per directory name. Registering a plugin more than once, i.e. loading a plugin again after the first time failed, will not cause a problem as long as the same instance was registered.

Parameters

- **$plugin** (Plugin) – Record of plugin to register.

isRegistered(*Plugin $plugin*)
Return whether a plugin is registered or not.

Parameters

- **$plugin** (Plugin) –

Returns bool Whether the plugin is registered or not.

load(*Plugin $plugin*, *$force = false*, *$pluginsWaitingToLoad = array()*)
Load a plugin (and make sure the plugin API is available).

To be loaded, the plugin must be installed, active, and not have a newer version. If loaded, the plugin will attempt to first load all plugins, both required and optional, that the plugin uses. However, it will not load a plugin that it uses if that plugin is not installed and activated.

Parameters

- **$plugin** (Plugin) –
- **$force** (*bool*) – If true, throws exceptions if a plugin can't be loaded.
- **$pluginsWaitingToLoad** (*array*) – Plugins waiting to be loaded

_canLoad(*$plugin*, *$force*)

Determine whether or not a plugin can be loaded. To be loaded, it must meet the following criteria: - Has a plugin.php file. - Is installed. - Is active. - Meets the minimum required version of Omeka (in plugin.ini). - Is not already loaded. - Does not have a new version available.

Parameters

- **$plugin** (Plugin) – Plugin to test.
- **$force** (*bool*) – If true, throw an exception if the plugin can't be loaded.

Returns bool

hasPluginBootstrap(*$pluginDirName*)

Check whether a plugin has a bootstrap file.

Parameters

- **$pluginDirName** (*string|Plugin*) –

Returns bool

getPluginClassName(*$pluginDirName*)

Return the valid plugin class name.

Parameters

- **$pluginDirName** (*string*) –

Returns string

getPluginFilePath(*$pluginDirName*)

Return the path to the plugin.php file.

Parameters

- **$pluginDirName** (*string*) –

Returns string

getPluginClassFilePath(*$pluginDirName*)

Return the path to the plugin class file.

Parameters

- **$pluginDirName** (*string*) –

Returns string

getPlugins()

Return a list of all the plugins that have been loaded (or attempted to be loaded) thus far.

Returns array List of Plugin objects.

getPlugin(*$directoryName*)

Get a plugin object by name (plugin subdirectory name).

Parameters

- **$directoryName** (*string*) – Plugin name.

Returns Plugin|null

_loadPluginBootstrap(*Plugin $plugin*)
Loads the plugin bootstrap file for a plugin.

Parameters

- **$plugin** (Plugin) –

Omeka_Plugin_Loader_Exception

Package: *Plugin\Loader*

class Omeka_Plugin_Loader_Exception

extends Exception

implements Throwable

An exception thrown when the plugin loader is unable to load a plugin.

property Omeka_Plugin_Loader_Exception::$**message**
protected

property Omeka_Plugin_Loader_Exception::$**code**
protected

property Omeka_Plugin_Loader_Exception::$**file**
protected

property Omeka_Plugin_Loader_Exception::$**line**
protected

Omeka_Plugin_Loader_Exception::**__clone**()

Omeka_Plugin_Loader_Exception::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Plugin_Loader_Exception::**__wakeup**()

Omeka_Plugin_Loader_Exception::**getMessage**()

Omeka_Plugin_Loader_Exception::**getCode**()

Omeka_Plugin_Loader_Exception::**getFile**()

Omeka_Plugin_Loader_Exception::**getLine**()

Omeka_Plugin_Loader_Exception::**getTrace**()

Omeka_Plugin_Loader_Exception::**getPrevious**()

Omeka_Plugin_Loader_Exception::**getTraceAsString**()

Omeka_Plugin_Loader_Exception::**__toString**()

3.7.16 Record

Up to *Packages*

Omeka_Record_AbstractRecord

Package: *Record*

class Omeka_Record_AbstractRecord

implements `ArrayAccess`

A base class for domain objects, inspired by, though not strictly adherent to, the ActiveRecord pattern.

property `Omeka_Record_AbstractRecord::$`**id**
int

Unique ID for the record.

All implementations of Omeka_Record_AbstractRecord must have a table containing an 'id' column, preferably as the primary key.

property `Omeka_Record_AbstractRecord::$`**_cache**
protected array

An in-memory cache for related objects that have been retrieved via the magic __get() syntax.

property `Omeka_Record_AbstractRecord::$`**_mixins**
protected array

Set of Omeka_Record_Mixin_AbstractMixin objects that are designed to extend the behavior of Omeka_Record_AbstractRecord implementations.

Examples include {@link Taggable}, {@link Relatable}, {@link ActsAsElementText}, etc.

property `Omeka_Record_AbstractRecord::$`**_db**
protected Omeka_Db

property `Omeka_Record_AbstractRecord::$`**_related**
protected array

Key/value pairs indicating aliases for methods that retrieve related data objects. For example, a subclass might define the following: <code> protected $_related = array('Sections'=>'loadSections'); </code> This would allow the client to write code like: <code> $sections = $subclassInstance->Sections; </code> Which would be equivalent to: <code> $sections = $subclassInstance->loadSections(); </code> The difference being, the former is cached so as to avoid multiple trips to the database.

property `Omeka_Record_AbstractRecord::$`**_postData**
protected ArrayObject

Storage for the POST data when handling a form.

`Omeka_Record_AbstractRecord::`**__construct**(*$db = null*)

Parameters

- **$db** (*Omeka_Db|null*) – (optional) Defaults to the Omeka_Db instance from the bootstrap.

`Omeka_Record_AbstractRecord::`**construct**()
Subclass constructor behavior.

Subclasses of Omeka_Record_AbstractRecord can override this function to add behavior to the constructor without overriding __construct.

Omeka_Record_AbstractRecord::**__destruct**()
Unsets mixins, which contain circular references, upon record destruction

IMPORTANT: Solves a memory leak when retrieving/saving records.

Required because PHP 5.2 does not do garbage collection on circular references.

Omeka_Record_AbstractRecord::**__get**(*$prop*)
Retrieve database records that are associated with the current one.

Parameters

- **$prop** (*string*) – Related data to retrieve.

Returns mixed

Omeka_Record_AbstractRecord::**__call**(*$m*, *$a*)
Delegate unknown method calls to Omeka_Record_Mixin_AbstractMixin instances.

Parameters

- **$m** (*string*) – Method name.
- **$a** (*array*) – Method arguments.

Returns mixed

Omeka_Record_AbstractRecord::**_initializeMixins**()
Initialize the mixins for a record.

Any Omeka_Record_AbstractRecord subclass that uses mixins should initialize them here, since this is called on construction and when mixins need to be reinitialized.

Omeka_Record_AbstractRecord::**delegateToMixins**(*$method*, *$args = array()*, *$all = false*)
Delegate to the given method in one or more mixin instances.

Parameters

- **$method** (*string*) –
- **$args** (*array*) –
- **$all** (*bool*) – (optional) Whether or not to call the same method on every mixin instance that has that method. Defaults to false.

Returns mixed If $all is false, the return value from the invoked method. Otherwise there is no return value.

Omeka_Record_AbstractRecord::**runCallbacks**(*$event*, *$args = array()*)
Invoke all callbacks associated with a specific record event.

Callbacks execute in the following order: - Omeka_Record_AbstractRecord hooks like Omeka_Record_AbstractRecord::afterDelete() - Record mixin hooks like Taggable::afterSave() - Generic record plugin hooks like 'before_delete_record' - Specific record plugin hooks like 'before_delete_item'

Parameters

- **$event** –
- **$args** –

Omeka_Record_AbstractRecord::**_addToCache**(*$value*, *$key*)
Add a value to the record-specific cache.

Parameters

- **$value** (*mixed*) –
- **$key** (*string*) –

Omeka_Record_AbstractRecord::**_getCached**(*$name*)
Get a value from the record-specific cache.

Parameters

- **$name** (*string*) –

Returns mixed

Omeka_Record_AbstractRecord::**getProperty**(*$property*)
Get a property about the record for display purposes.

Parameters

- **$property** (*string*) – Property to get. Always lowercase.

Returns mixed

Omeka_Record_AbstractRecord::**exists**()
Determine whether or not this record is persistent in the database.

For simplicity, non-persistent records are indicated by the lack of a value for the 'id' column.

Returns bool

Omeka_Record_AbstractRecord::**_validate**()
Template method for defining record validation rules.

Should be overridden by subclasses.

Omeka_Record_AbstractRecord::**isValid**()
Determine whether or not the record is valid.

Returns bool

Omeka_Record_AbstractRecord::**getErrors**()
Retrieve validation errors associated with this record.

Returns Omeka_Validate_Errors

Omeka_Record_AbstractRecord::**hasErrors**()
Determine whether or not this record has any validation errors.

Returns bool

Omeka_Record_AbstractRecord::**addError**(*$field*, *$msg*)
Add a validation error for a specific field.

Currently limited to a single error per field, so multiple error messages must be concatenated together.

Parameters

- **$field** (*string|null*) – Name of the field. This can be null to indicate a general error not associated with a specific field.
- **$msg** (*string*) – The error message.

Omeka_Record_AbstractRecord::**addErrorsFrom**(*Omeka_Record_AbstractRecord $record*)
Combine errors from a different Omeka_Record_AbstractRecord instance with the errors already on this record.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Omeka_Record_AbstractRecord::**lock**()
Prevent a record from being modified.

Can be used to prevent accidentally saving/deleting a record if its state may change but saving would be undesirable, such as modifying a record for display purposes.

Omeka_Record_AbstractRecord::**getTable**(*$class = null*)
Retrieve the Omeka_Db_Table instance associated with this record, or with that of any given record class.

Parameters

- **$class** –

Returns Omeka_Db_Table

Omeka_Record_AbstractRecord::**getDb**()
Retrieve the Omeka_Db instance associated with this record.

Returns Omeka_Db

Omeka_Record_AbstractRecord::**toArray**()
Retrieve an associative array of all the record's columns and their values.

Returns array

Omeka_Record_AbstractRecord::**save**(*$throwIfInvalid = true*)
Save the record.

Parameters

- **$throwIfInvalid** (*bool*) –

Returns bool Whether the save was successful.

Omeka_Record_AbstractRecord::**__clone**()
Clone the record.

Unsets the ID so the cloned record can be saved on its own.

Omeka_Record_AbstractRecord::**delete**()
Delete the record.

Omeka_Record_AbstractRecord::**_delete**()
Template method for defining record deletion logic.

Subclasses can override this method to define additional logic for deleting records. Note that this is different from both the beforeDelete() and afterDelete() hooks in that it executes after beforeDelete(), but before the record is actually deleted.

Common use cases include emulating cascading deletes with other database rows.

Omeka_Record_AbstractRecord::**beforeSave**(*$args*)
Executes before the record is saved.

Parameters

- **$args** –

Omeka_Record_AbstractRecord::**afterSave**(*$args*)
Executes after the record is inserted.

Parameters

- **$args** –

Omeka_Record_AbstractRecord::**beforeDelete**()
Executes before the record is deleted.

Omeka_Record_AbstractRecord::**afterDelete**()
Executes after the record is deleted.

Omeka_Record_AbstractRecord::**setArray**(*$data*)
Set values for the record using an associative array or iterator.

Parameters

- **$data** (*array|Traversable*) –

Omeka_Record_AbstractRecord::**getPluginBroker**()

Omeka_Record_AbstractRecord::**setPluginBroker**(*$broker = null*)

Parameters

- **$broker** –

Omeka_Record_AbstractRecord::**offsetExists**(*$name*)
Determine whether or not the given field has a value associated with it.

Required by ArrayAccess.

Parameters

- **$name** (*string*) –

Returns bool

Omeka_Record_AbstractRecord::**offsetUnset**(*$name*)
Unset the given field.

Required by ArrayAccess.

Parameters

- **$name** (*string*) –

Omeka_Record_AbstractRecord::**offsetGet**(*$name*)
Retrieve the value of a given field.

Required by ArrayAccess.

Parameters

- **$name** (*string*) –

Returns mixed

Omeka_Record_AbstractRecord::**offsetSet**(*$name*, *$value*)
Set the value of a given field.

Required by ArrayAccess.

Parameters

- **$name** (*string*) –
- **$value** (*mixed*) –

Omeka_Record_AbstractRecord::**filterPostData**(*$post*)
Filter the form input according to some criteria.

Template method should be overridden by subclasses that wish to implement some sort of filtering criteria.

Parameters

- **$post** (*array*) –

Returns array Filtered post data.

Omeka_Record_AbstractRecord::**setPostData**(*$post*)
Set the POST data to the record.

Parameters

- **$post** (*array*) –

Omeka_Record_AbstractRecord::**fieldIsUnique**(*$field*, *$value = null*)
Check uniqueness of one of the record's fields.

Parameters

- **$field** (*string*) –
- **$value** (*mixed*) – Optional If null, this will check the value of the record's $field. Otherwise check the uniqueness of this value for the given field.

Returns bool

Omeka_Record_AbstractRecord::**getRecordUrl**(*$action = 'show'*)
Get the routing parameters or the URL string to this record.

The record_url() global uses this method to get routing parameters for non-standard records, e.g. records defined by plugins. Subclasses should override this method if the default route (as defined below) is incorrect.

Parameters

- **$action** (*string*) –

Returns string|array A URL string or a routing array.

Omeka_Record_AbstractRecord::**getFile**()
Get a representative file for this record.

Returns File|null

Omeka_Record_Exception

Package: *Record*

class Omeka_Record_Exception

extends Exception

implements Throwable

property Omeka_Record_Exception::$**message**
protected

property Omeka_Record_Exception::$**code**
protected

property Omeka_Record_Exception::$**file**
protected

property Omeka_Record_Exception::$**line**
protected

Omeka_Record_Exception::**__clone**()

Omeka_Record_Exception::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Record_Exception::**__wakeup**()

Omeka_Record_Exception::**getMessage**()

Omeka_Record_Exception::**getCode**()

Omeka_Record_Exception::**getFile**()

Omeka_Record_Exception::**getLine**()

Omeka_Record_Exception::**getTrace**()

Omeka_Record_Exception::**getPrevious**()

Omeka_Record_Exception::**getTraceAsString**()

Omeka_Record_Exception::**__toString**()

Omeka_Record_Iterator

Package: *Record*

class Omeka_Record_Iterator

implements Iterator implements Traversable

property Omeka_Record_Iterator::$**_records**
protected

property Omeka_Record_Iterator::$**_view**
protected

property Omeka_Record_Iterator::$**_currentRecordVar**
protected

Omeka_Record_Iterator::**__construct**(*$records*, *$view = null*, *$currentRecordVar = null*)

Construct the record iterator.

Parameters

- **$records** (*array*) –
- **$view** (*null|Zend_View_Abstract*) –
- **$currentRecordVar** (*null|string*) –

Omeka_Record_Iterator::**rewind**()

Omeka_Record_Iterator::**current**()
Return the current record, setting it to the view if applicable.

Omeka_Record_Iterator::**key**()

Omeka_Record_Iterator::**next**()
Release the previous record and advance the pointer to the next one.

Omeka_Record_Iterator::**valid**()

Record\Api

Up to *Record*

Omeka_Record_Api_AbstractRecordAdapter

Package: *Record\Api*

class Omeka_Record_Api_AbstractRecordAdapter

implements *Omeka_Record_Api_RecordAdapterInterface*

property Omeka_Record_Api_AbstractRecordAdapter::$**_elementsCache**
protected array

property Omeka_Record_Api_AbstractRecordAdapter::$**_elementSetsCache**
protected array

Omeka_Record_Api_AbstractRecordAdapter::**setPostData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set data to a record during a POST request.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Omeka_Record_Api_AbstractRecordAdapter::**setPutData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set data to a record during a PUT request.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Omeka_Record_Api_AbstractRecordAdapter::**getElementTextRepresentations**(*Omeka_Record_Abstrac $record*)
Get representations of element texts belonging to a record.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Omeka_Record_Api_AbstractRecordAdapter::**setElementTextData**(*Omeka_Record_AbstractRecord $record*, *$data*)
Set element text data to a record.

The record must initialize the ElementText mixin.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Omeka_Record_Api_AbstractRecordAdapter::**getTagRepresentations**(*Omeka_Record_AbstractRecord $record*)

Get representations of tags belonging to a record.

The record must initialize the Tag mixin.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Omeka_Record_Api_AbstractRecordAdapter::**setTagData**(*Omeka_Record_AbstractRecord $record*, *$data*)

Set tag data to a record.

The record must initialize the Tag mixin.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Omeka_Record_Api_AbstractRecordAdapter::**getResourceUrl**(*$uri*)

Get the absolute URL to the passed resource.

Parameters

- **$uri** (*string*) – The full resource URI

Returns string

Omeka_Record_Api_AbstractRecordAdapter::**getDate**(*$date*)

Format a date string as an ISO 8601 date, UTC timezone.

Parameters

- **$date** (*string*) –

Returns string

Omeka_Record_Api_AbstractRecordAdapter::**_getUnfilteredElementTextRepresentations**(*Omeka_ $record*

Get unfiltered representations of element texts belonging to a record.

Note the HTML flag in the representation. This indicates to the consumer that the representation is unfiltered.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Omeka_Record_Api_AbstractRecordAdapter::**_getFilteredElementTextRepresentations**(*Omeka_Re $record*)

Get filtered representations of element texts belonging to a record.

Note the lack of the HTML flag in the representation. This indicates to the consumer that the representation is filtered through the display_elements and array('Display',...) element texts filters.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Returns array

Omeka_Record_Api_AbstractRecordAdapter::**getRepresentation**(*Omeka_Record_AbstractRecord $record*)

Get the REST representation of a record.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Omeka_Record_Api_RecordAdapterInterface

Package: *Record\Api*

interface Omeka_Record_Api_RecordAdapterInterface

getRepresentation(*Omeka_Record_AbstractRecord $record*)

Get the REST representation of a record.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

setPostData(*Omeka_Record_AbstractRecord $record*, *$data*)

Set data to a record during a POST request.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

setPutData(*Omeka_Record_AbstractRecord $record*, *$data*)

Set data to a record during a PUT request.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –
- **$data** (*mixed*) –

Record\Builder

Up to *Record*

Omeka_Record_Builder_AbstractBuilder

Package: *Record\Builder*

class Omeka_Record_Builder_AbstractBuilder

Build or update an {@link Omeka_Record_AbstractRecord} as needed.

property _recordClass

protected string

Class of record that the builder will create.

property _settableProperties
protected array

String names denoting the properties of a specific record that can be set directly through the builder. This will not always be all of the fields for the record.

property _record
protected Omeka_Record_AbstractRecord

Record being built or updated.

property _db
protected Omeka_Db

__construct(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

build()
Build the actual record. If the record already exists, update it as necessary.

Returns Omeka_Record_AbstractRecord

setRecordMetadata(*$metadata*)
Set basic metadata for the record.

Note that the columns to be set must be specified in the $_settableProperties property of subclassed Builders.

Parameters

- **$metadata** (*array*) –

getRecordMetadata()
Get the metadata that will be saved to the record.

Returns array

getRecord()
Get the record that is being acted upon by the builder.

When an Omeka_Record_AbstractRecord instance has been provided via setRecord(), that will be returned. If a record ID has been provided, then the appropriate record will be returned.

Otherwise, a new instance of Omeka_Record_AbstractRecord will be returned.

Returns Omeka_Record_AbstractRecord

setRecord(*$record = null*)
Set the record upon which this builder will act.

Parameters

- **$record** (*Omeka_Record_AbstractRecord|int|null*) –

_beforeBuild(*Omeka_Record_AbstractRecord $record*)
All necessary tasks to take place before the record is inserted.

Exceptions may be thrown, validation errors may be added.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

_afterBuild(*Omeka_Record_AbstractRecord $record*)
All necessary tasks that take place after the record has been inserted into the database.

Should not throw exceptions in this method.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

_setRecordProperties(*$record*)
Set the properties for the record, taking care to filter based on the $_settableProperties array.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

Omeka_Record_Builder_Exception

Package: *Record\Builder*

class Omeka_Record_Builder_Exception

extends Exception

implements Throwable

Exception thrown when there is an error creating a Record using {@link Omeka_Record_Builder_AbstractBuilder}.

property Omeka_Record_Builder_Exception::$**message**
protected

property Omeka_Record_Builder_Exception::$**code**
protected

property Omeka_Record_Builder_Exception::$**file**
protected

property Omeka_Record_Builder_Exception::$**line**
protected

Omeka_Record_Builder_Exception::**__clone**()

Omeka_Record_Builder_Exception::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Record_Builder_Exception::**__wakeup**()

Omeka_Record_Builder_Exception::**getMessage**()

Omeka_Record_Builder_Exception::**getCode**()

Omeka_Record_Builder_Exception::**getFile**()

Omeka_Record_Builder_Exception::**getLine**()

Omeka_Record_Builder_Exception::**getTrace**()

Omeka_Record_Builder_Exception::**getPrevious**()

Omeka_Record_Builder_Exception::**getTraceAsString**()

Omeka_Record_Builder_Exception::**__toString**()

Record\Mixin

Up to *Record*

Omeka_Record_Mixin_AbstractMixin

Package: *Record\Mixin*

class Omeka_Record_Mixin_AbstractMixin

Represents a kind of mixin for Omeka_Record_AbstractRecord implementations.

Any methods declared for an implementation of this class can be called transparently by an Omeka_Record_AbstractRecord object that uses one of these modules.

For instance, the Item model does not have an addTags() method, but the Taggable class does. Since Item declares Taggable as one of its modules, an Item instance call all of Taggable's methods, so that adding tags would be as simple as calling $item->addTags('foo, bar');

Note that this is not a true mixin because it cannot override any existing methods on a Record object.

property _record

protected Omeka_Record_AbstractRecord

Underlying record object.

__construct(*$record*)

Base mixin constructor.

Store the underlying record for use in the mixin.

Parameters

- **$record** (Omeka_Record_AbstractRecord) –

beforeSave(*$args*)

Callback automatically called by Omeka_Record_AbstractRecord.

See the corresponding {@link Omeka_Record_AbstractRecord} method for definitions of call times.

Parameters

- **$args** –

Returns void

afterSave(*$args*)

Parameters

- **$args** –

beforeDelete()

afterDelete()

3.7.17 Session

Up to *Packages*

Omeka_Session_SaveHandler_DbTable

Package: *Session*

class Omeka_Session_SaveHandler_DbTable

extends Zend_Session_SaveHandler_DbTable

Wrapper for Zend_Session_SaveHandler_DbTable to hard code the table definition. This boosts performance by skipping the DESCRIBE query that retrieves this metadata by default.

Note that this must be updated meticulously after any changes to the sessions table schema.

Omeka_Session_SaveHandler_DbTable**::init**()

Omeka_Session_SaveHandler_DbTable**::write**(*$id*, *$data*)

Write session data

Parameters

- **$id** (*string*) –
- **$data** (*string*) –

Returns bool

3.7.18 Storage

Up to *Packages*

Omeka_Storage

Package: *Storage*

class Omeka_Storage

Top-level helper class for handling file storage.

__construct(*$options = null*)

Allows storage options to be set immediately at construction.

Parameters

- **$options** (*array*) – If set, this array will be passed to setOptions.

__call(*$name*, *$arguments*)

Delegates calls directly to Omeka_Storage to the currently-set storage adapter.

All of the methods of the Adapter interface are accessible in this way, as well as any other methods declared by the adapter.

Parameters

- **$name** (*string*) – Method name.
- **$arguments** (*string*) – Method arguments.

Returns mixed

setOptions(*$options*)

Set global options for the storage system, as well as any adapter-specific options.

Parameters

- **$options** (*array*) – Options to set. Valid options include: * 'adapter': (string) Name of the storage adapter to use. * 'adapterOptions': (array) Array of options to pass to the adapter; see the specific adapter classes for details. * 'temp_dir': (string) Local temporary directory where files stored before they are handled by the adapter.

setAdapter(*$adapter*, *$options = array()*)
Set the storage adapter to be used, as well as options for that adapter.

You can either pass an already-constructed adapter object to this method or use this method as a factory by passing the name of an adapter class and options to set on it.

Parameters

- **$adapter** (*Omeka_Storage_Adapter_AdapterInterface|string*) – Storage adapter to set. If an adapter object is passed, it is simply set as the current adapter. If a string is passed, an object of that class is created and set as the current adapter.
- **$options** (*array|null*) – If a string is passed to $adapter, this array of options is passed to the class' constructor.

getAdapter()
Get the current storage adapter.

You generally need to use the adapter object returned by this method to perform any storage actions.

Returns Omeka_Storage_Adapter_AdapterInterface

setTempDir(*$dir*)
Set the temporary file storage directory path.

Parameters

- **$dir** (*string*) – Local path to directory.

getTempDir()
Get the temporary file storage directory path.

If no directory has been explicitly selected, the system's temp directory is set as the temp dir and returned.

Returns string Local path to directory.

getPathByType(*$filename*, *$type = 'files'*)

Parameters

- **$filename** –
- **$type** –

Omeka_Storage_Exception

Package: *Storage*

class Omeka_Storage_Exception

extends `Exception`

implements `Throwable`

property `Omeka_Storage_Exception::$`**message**
protected

property `Omeka_Storage_Exception::$`**code**
protected

property Omeka_Storage_Exception::$**file**
protected

property Omeka_Storage_Exception::$**line**
protected

Omeka_Storage_Exception::**__clone**()

Omeka_Storage_Exception::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_Storage_Exception::**__wakeup**()

Omeka_Storage_Exception::**getMessage**()

Omeka_Storage_Exception::**getCode**()

Omeka_Storage_Exception::**getFile**()

Omeka_Storage_Exception::**getLine**()

Omeka_Storage_Exception::**getTrace**()

Omeka_Storage_Exception::**getPrevious**()

Omeka_Storage_Exception::**getTraceAsString**()

Omeka_Storage_Exception::**__toString**()

Storage\Adapter

Up to *Storage*

Omeka_Storage_Adapter_AdapterInterface

Package: *Storage\Adapter*

interface Omeka_Storage_Adapter_AdapterInterface
Interface for file storage adapters.

Classes that implement this interface handle the actual work ofstoring and retrieving files.

__construct(*$options = array()*)
Set options for the storage adapter.

Parameters

- **$options** (*array*) –

setUp()
Follow any necessary steps to set up storage prior to use.

E.g. for the filesystem adapter, this would include creating any directories that did not already exist. For S3, it might involve creating a new bucket if it did not exist.

canStore()
Check whether the adapter is set up correctly to be able to store files.

Returns bool

store (*$source*, *$dest*)
Move a local file to "storage."

Parameters

- **$source** (*string*) – Local filesystem path to file.
- **$dest** (*string*) – Destination path.

move (*$source*, *$dest*)
Move a file between two storage locations.

Parameters

- **$source** (*string*) – Original storage path.
- **$dest** (*string*) – Destination storage path.

delete (*$path*)
Remove a "stored" file.

Parameters

- **$path** (*string*) –

getUri (*$path*)
Get a URI for a "stored" file.

Parameters

- **$path** (*string*) –

Returns string URI

Omeka_Storage_Adapter_Filesystem

Package: *Storage\Adapter*

class Omeka_Storage_Adapter_Filesystem

implements *Omeka_Storage_Adapter_AdapterInterface*

Standard local filesystem storage adapter.

The default adapter; this stores files in the Omeka files directory by default, but can be set to point to a different path.

property Omeka_Storage_Adapter_Filesystem::$**_localDir**
protected string

Local directory where files are stored.

property Omeka_Storage_Adapter_Filesystem::$**_subDirs**
protected

property Omeka_Storage_Adapter_Filesystem::$**_webDir**
protected string

Web-accesible path that corresponds to $_localDir.

Omeka_Storage_Adapter_Filesystem::**__construct** (*$options = array()*)
Set options for the storage adapter.

Parameters

- **$options** (*array*) –

Omeka_Storage_Adapter_Filesystem::**setUp**()

Omeka_Storage_Adapter_Filesystem::**canStore**()
Check whether the adapter is set up correctly to be able to store files.

Specifically, this checks to see if the local storage directory is writable.

Returns bool

Omeka_Storage_Adapter_Filesystem::**store**(*$source*, *$dest*)
Move a local file to "storage."

Parameters

- **$source** (*string*) – Local filesystem path to file.
- **$dest** (*string*) – Destination path.

Omeka_Storage_Adapter_Filesystem::**move**(*$source*, *$dest*)
Move a file between two "storage" locations.

Parameters

- **$source** (*string*) – Original stored path.
- **$dest** (*string*) – Destination stored path.

Omeka_Storage_Adapter_Filesystem::**delete**(*$path*)
Remove a "stored" file.

Parameters

- **$path** (*string*) –

Omeka_Storage_Adapter_Filesystem::**getUri**(*$path*)
Get a URI for a "stored" file.

Parameters

- **$path** (*string*) –

Returns string URI

Omeka_Storage_Adapter_Filesystem::**getOptions**()
Return the options set by the adapter. Used primarily for testing.

Omeka_Storage_Adapter_Filesystem::**setLocalDir**(*$dir*)
Set the path of the local directory where files are stored.

Parameters

- **$dir** –

Omeka_Storage_Adapter_Filesystem::**setWebDir**(*$dir*)
Set the web URL that corresponds with the local dir.

Parameters

- **$dir** –

Omeka_Storage_Adapter_Filesystem::**_getAbsPath**(*$path*)
Convert a "storage" path to an absolute filesystem path.

Parameters

- **$path** (*string*) – Storage path.

Returns string Absolute local filesystem path.

`Omeka_Storage_Adapter_Filesystem::_rename`(*$source*, *$dest*)

Parameters

- **$source** –
- **$dest** –

Returns bool

Omeka_Storage_Adapter_TempFilesystem

Package: *Storage\Adapter*

class Omeka_Storage_Adapter_TempFilesystem

extends *Omeka_Storage_Adapter_Filesystem*

implements *Omeka_Storage_Adapter_AdapterInterface*

Storage adapter that uses the system temp directory for its filesystem.

After the adapter is no longer needed (__destruct()), all the files that were created during its lifetime are removed.

Used primarily by the test framework.

property `Omeka_Storage_Adapter_TempFilesystem::$_localDir`
protected string

Local directory where files are stored.

property `Omeka_Storage_Adapter_TempFilesystem::$_subDirs`
protected

property `Omeka_Storage_Adapter_TempFilesystem::$_webDir`
protected string

Web-accesible path that corresponds to $_localDir.

`Omeka_Storage_Adapter_TempFilesystem::canStore`()
No need to perform this check.

`Omeka_Storage_Adapter_TempFilesystem::store`(*$source*, *$dest*)
Move a local file to "storage."

Parameters

- **$source** (*string*) – Local filesystem path to file.
- **$dest** (*string*) – Destination path.

`Omeka_Storage_Adapter_TempFilesystem::move`(*$source*, *$dest*)
Move a file between two "storage" locations.

Parameters

- **$source** (*string*) – Original stored path.
- **$dest** (*string*) – Destination stored path.

`Omeka_Storage_Adapter_TempFilesystem::getUri`(*$path*)

Parameters

- **$path** –

Omeka_Storage_Adapter_TempFilesystem::**_mkdir**(*$filepath*)

Parameters

- **$filepath** –

Omeka_Storage_Adapter_TempFilesystem::**__construct**(*$options = array()*)
Set options for the storage adapter.

Parameters

- **$options** (*array*) –

Omeka_Storage_Adapter_TempFilesystem::**setUp**()

Omeka_Storage_Adapter_TempFilesystem::**delete**(*$path*)
Remove a "stored" file.

Parameters

- **$path** (*string*) –

Omeka_Storage_Adapter_TempFilesystem::**getOptions**()
Return the options set by the adapter. Used primarily for testing.

Omeka_Storage_Adapter_TempFilesystem::**setLocalDir**(*$dir*)
Set the path of the local directory where files are stored.

Parameters

- **$dir** –

Omeka_Storage_Adapter_TempFilesystem::**setWebDir**(*$dir*)
Set the web URL that corresponds with the local dir.

Parameters

- **$dir** –

Omeka_Storage_Adapter_TempFilesystem::**_getAbsPath**(*$path*)
Convert a "storage" path to an absolute filesystem path.

Parameters

- **$path** (*string*) – Storage path.

Returns string Absolute local filesystem path.

Omeka_Storage_Adapter_TempFilesystem::**_rename**(*$source*, *$dest*)

Parameters

- **$source** –
- **$dest** –

Returns bool

Omeka_Storage_Adapter_ZendS3

Package: *Storage\Adapter*

class Omeka_Storage_Adapter_ZendS3

implements *Omeka_Storage_Adapter_AdapterInterface*

Cloud storage adapter for Amazon S3, using Zend's built-in service.

Caveat: Zend's storage adapter currently does not function correctly with buckets that are validly-named, but use characters that cannot appear in domain names.

Omeka_Storage_Adapter_ZendS3::**__construct**(*$options = array()*)
Set options for the storage adapter.

Parameters

- **$options** (*array*) –

Omeka_Storage_Adapter_ZendS3::**setUp**()

Omeka_Storage_Adapter_ZendS3::**canStore**()

Omeka_Storage_Adapter_ZendS3::**store**(*$source*, *$dest*)
Move a local file to S3 storage.

Parameters

- **$source** (*string*) – Local filesystem path to file.
- **$dest** (*string*) – Destination path.

Omeka_Storage_Adapter_ZendS3::**move**(*$source*, *$dest*)
Move a file between two "storage" locations.

Parameters

- **$source** (*string*) – Original stored path.
- **$dest** (*string*) – Destination stored path.

Omeka_Storage_Adapter_ZendS3::**delete**(*$path*)
Remove a "stored" file.

Parameters

- **$path** (*string*) –

Omeka_Storage_Adapter_ZendS3::**getUri**(*$path*)
Get a URI for a "stored" file.

Parameters

- **$path** (*string*) –

Returns string URI

Omeka_Storage_Adapter_ZendS3::**getS3Service**()
Return the service object being used for S3 requests.

Returns Zend_Service_Amazon_S3

Omeka_Storage_Adapter_ZendS3::**_getBucketName**()
Get the name of the bucket files should be stored in.

Returns string Bucket name

Omeka_Storage_Adapter_ZendS3::**_getObjectName**(*$path*)
Get the object name. Zend's S3 service requires you to build the object name by prepending the name of the target bucket.

Parameters

- **$path** (*string*) –

Returns string Object name.

`Omeka_Storage_Adapter_ZendS3::_getExpiration()`
Normalizes and returns the expiration time.

Converts to integer and returns zero for all non-positive numbers.

Returns int

3.7.19 Test

Up to *Packages*

Omeka_Test_AppTestCase

Package: *Test*

class Omeka_Test_AppTestCase

extends `Zend_Test_PHPUnit_ControllerTestCase`

Abstract test case class that bootstraps the entire application.

property `Omeka_Test_AppTestCase::$_isAdminTest`
protected bool

Flag that determines whether the test should run against admin or public.

Defaults to true (for admin).

`Omeka_Test_AppTestCase::setUp()`
Bootstrap the application on each test run.

`Omeka_Test_AppTestCase::__get(`*$property*`)`
Proxy gets to properties to allow access to bootstrap container properties.

Parameters

- **$property** (*string*) –

Returns mixed

`Omeka_Test_AppTestCase::appBootstrap()`
Bootstrap the application.

`Omeka_Test_AppTestCase::setUpBootstrap(`*$bootstrap*`)`
Subclasses can override this to perform specialized setup on the Omeka application.

Parameters

- **$bootstrap** (*Zend_Application_Bootstrap*) –

`Omeka_Test_AppTestCase::tearDown()`
Reset objects that carry global state between test runs.

`Omeka_Test_AppTestCase::dispatch(`*$url = null*, *$throwExceptions = true*`)`

Parameters

- **$url** (*string*) –
- **$throwExceptions** (*bool*) –

Omeka_Test_AppTestCase::**_authenticateUser**(*User $user*)
Trick the environment into thinking that a user has been authenticated.

Parameters

- **$user** (User) –

Omeka_Test_AppTestCase::**_getDefaultUser**()
Get the user that is installed by default.

Returns User

Omeka_Test_AppTestCase::**_setUpThemeBootstrap**(*$themeType*)
Set up the bootstrap differently depending on whether the test is meant for the public or admin themes.

Parameters

- **$themeType** –

Omeka_Test_Bootstrap

Package: *Test*

class Omeka_Test_Bootstrap
Abstract test case class that bootstraps the entire application.

setContainer(*$container*)
Set resource container

By default, if a resource callback has a non-null return value, this value will be stored in a container using the resource name as the key.

Containers must be objects, and must allow setting public properties.

Parameters

- **$container** –

Returns Zend_Application_Bootstrap_BootstrapAbstract

getContainer()
Retrieve resource container

Returns object

hasResource(*$name*)
Determine if a resource has been stored in the container

During bootstrap resource initialization, you may return a value. If you do, it will be stored in the {@link setContainer() container}. You can use this method to determine if a value was stored.

Parameters

- **$name** –

Returns bool

getResource(*$name*)
Retrieve a resource from the container

During bootstrap resource initialization, you may return a value. If you do, it will be stored in the {@link setContainer() container}. You can use this method to retrieve that value.

If no value was returned, this will return a null value.

Parameters

- **$name** –

Returns null|mixed

Test\Helper

Up to *Test*

Omeka_Test_Helper_Db

Package: *Test\Helper*

class Omeka_Test_Helper_Db

Catch-all class for database helper methods that are shared across test cases.

__construct (*Zend_Db_Adapter_Abstract $dbAdapter*, *$prefix*)

Parameters

- **$dbAdapter** (*Zend_Db_Adapter_Abstract*) –
- **$prefix** –

__call (*$method*, *$args*)

Proxy to the db adapter object for all other requests.

Parameters

- **$method** (*string*) – Method name.
- **$args** (*array*) – Method arguments.

Returns array

factory (*$dbConfig*)

Create an instance of the helper that is configured for the correct database.

Parameters

- **$dbConfig** –

tableExists (*$tableName*)

Check whether a table exists in the database.

Parameters

- **$tableName** (*string*) –

Returns bool

getTableCount (*$prefix = null*)

Get the number of tables in the database.

Parameters

- **$prefix** (*string*) –

Returns int

dropTables (*$tables = null*)

Drop the tables from the database.

Parameters

- **$tables** –

truncateTables(*$tables = null*)
Truncate the tables from the database.

Parameters

- **$tables** –

install()

getTableNames()
Get the tables in the database.

Returns array

getRowCount(*$tableName*)
Get the number of rows in a table.

Parameters

- **$tableName** (*string*) –

Returns int

getAdapter()

getPrefix()

Omeka_Test_Helper_DbProfiler

Package: *Test\Helper*

class Omeka_Test_Helper_DbProfiler
Catch-all class for database helper methods that are shared across test cases.

__construct(*Zend_Db_Profiler $profiler*, *PHPUnit_Framework_Assert $test*)
Constructor.

Parameters

- **$profiler** (*Zend_Db_Profiler*) –
- **$test** (*PHPUnit_Framework_Assert*) –

assertDbQuery(*$sqlPart*, *$message = null*)

Parameters

- **$sqlPart** –
- **$message** –

assertTotalNumQueries(*$queryCount*, *$msg = null*)
Assert that the given number of SQL queries were made.

Parameters

- **$queryCount** (*int*) –
- **$msg** –

Omeka_Test_Helper_Mail

Package: *Test\Helper*

class Omeka_Test_Helper_Mail
Encapsulates testing functionality for email.

__construct (*$path*)

Parameters

- **$path** (*string*) – Real path to the mail storage directory.

factory ()
Configure an instance of the Mail helper using the registered test_config.

Returns Omeka_Test_Helper_Mail

_getIterator ()
Get an iterator over the fakemail directory.

Returns DirectoryIterator

_isMailFile (*SplFileInfo $file*)
Check if a directory entry is a mail message file.

Parameters

- **$file** (*SplFileInfo*) –

Returns bool

getMailText (*$index = 0*)
Return the text of the n'th email that was sent during the test.

Note that this will not return correct results if reset() was not invoked between test runs.

Parameters

- **$index** (*int*) –

Returns string

count ()
The number of mails that have been sent.

Omeka_Test_Helper_Plugin

Package: *Test\Helper*

class Omeka_Test_Helper_Plugin
Encapsulates testing functionality for Omeka plugins.

setUp (*$pluginName*)
Install and initialize a plugin.

Note: Normally used in the setUp() method of plugin tests.

Parameters

- **$pluginName** (*string*) –

install (*$pluginName*)
Install a plugin

Parameters

- **$pluginName** (*string*) – The name of the plugin to install.

Returns Plugin

initialize(*$pluginName = null*)
Initializes the plugin hooks and filters fired in the core resources for a plugin Note: Normally used in the setUp() function of the subclasses that test plugins.

Parameters

- **$pluginName** (*string*) – If omitted, initialize all installed plugins.

_defineResponseContexts()
Run the response_contexts filter.

setPluginLoader(*$pluginLoader*)
Set the plugin loader for the helper to use.

Parameters

- **$pluginLoader** (Omeka_Plugin_Loader) –

setPluginIniReader(*$pluginIniReader*)
Set the plugin INI reader for the helper to use.

Parameters

- **$pluginIniReader** (Omeka_Plugin_Ini) –

setPluginBroker(*$pluginBroker*)
Set the plugin broker for the helper to use.

Parameters

- **$pluginBroker** (Omeka_Plugin_Broker) –

setAcl(*$acl*)
Set the ACL for the helper to use.

Parameters

- **$acl** (*Zend_Acl*) –

setRouter(*$router*)
Set the router for the helper to use.

Parameters

- **$router** (*Zend_Controller_Router_Interface*) –

__get(*$name*)
Lazy-loading for helper properties.

When a property is not set, attempts to load a default through standard Omeka global state. If this state is unavailable or undesireable, use the set*() methods before calling any of the other public methods of this class.

Parameters

- **$name** –

Returns mixed

Test\Resource

Up to *Test*

Omeka_Test_Resource_Config

Package: *Test\Resource*

class Omeka_Test_Resource_Config

extends Zend_Application_Resource_ResourceAbstract

Load the default config for the application, but *also* load the test config into Zend_Registry.

Omeka_Test_Resource_Config::**__construct**(*$options = null*)

Parameters

- **$options** (*array*) – Options for resource.

Omeka_Test_Resource_Config::**init**()

Load both config files.

Returns Zend_Config

Omeka_Test_Resource_Currentuser

Package: *Test\Resource*

class Omeka_Test_Resource_Currentuser

extends Zend_Application_Resource_ResourceAbstract

Mask the behavior of Omeka_Application_Resource_Currentuser in tests.

Omeka_Test_Resource_Currentuser::**init**()

Omeka_Test_Resource_Db

Package: *Test\Resource*

class Omeka_Test_Resource_Db

extends Zend_Application_Resource_Db

Set up the database test environment by wiping and resetting the database to a recently-installed state.

property Omeka_Test_Resource_Db::$**dropTables**

Flag to determine whether the tables need to be dropped. This is a slow process, and really should only be happening once, when the tests are first run.

property Omeka_Test_Resource_Db::$**runInstaller**

Flag to determine whether the installer needs to be run.

Omeka_Test_Resource_Db::**init**()

Load and initialize the database.

Returns Omeka_Db

Omeka_Test_Resource_Db::**getDb**()

Returns Omeka_Db

Omeka_Test_Resource_Db::**useTestConfig**()

Omeka_Test_Resource_Db::**setInstall**(*$flag*)

Set the flag that indicates whether or not to run the installer during init().

Parameters

- **$flag** (*bool*) –

Omeka_Test_Resource_Db::**getDbAdapter**()

Omeka_Test_Resource_Db::**setDbAdapter**(*Zend_Db_Adapter_Abstract $dbAdapter*)

Parameters

- **$dbAdapter** (*Zend_Db_Adapter_Abstract*) –

Omeka_Test_Resource_Db::**_getOmekaDb**()

Create a DB instance with the **omeka_** prefix.

Returns Omeka_Db

Omeka_Test_Resource_Db::**_enableSqlLogging**(*Omeka_Db $db*)

Parameters

- **$db** (Omeka_Db) –

Omeka_Test_Resource_Debug

Package: *Test\Resource*

class Omeka_Test_Resource_Debug

extends Zend_Application_Resource_ResourceAbstract

Stub resource to mask the behavior of Omeka_Application_Resource_Debug.

Omeka_Test_Resource_Debug::**init**()

Omeka_Test_Resource_Mail

Package: *Test\Resource*

class Omeka_Test_Resource_Mail

extends Zend_Application_Resource_ResourceAbstract

Testing resource for saving mail to the filesystem.

Omeka_Test_Resource_Mail::**init**()

Returns Zend_Mail

Omeka_Test_Resource_Mail::**mailCallback**(*$transport*)

Makes mail output names contain both the timestamp and an incrementing counter. The timestamp ensures mails between test runs don't collide, the counter differentiates between messages sent during the same timestamp unit (common).

Parameters

- **$transport** –

Omeka_Test_Resource_Storage

Package: *Test\Resource*

class Omeka_Test_Resource_Storage

extends Zend_Application_Resource_ResourceAbstract

Bootstrap resource for storage in test environment.

Omeka_Test_Resource_Storage::**init**()

Omeka_Test_Resource_Tempdir

Package: *Test\Resource*

class Omeka_Test_Resource_Tempdir

extends Zend_Application_Resource_ResourceAbstract

Bootstrap resource for storage in test environment.

Omeka_Test_Resource_Tempdir::**init**()

Omeka_Test_Resource_Tempdir::**cleanDir**(*$dir*)

Parameters

- **$dir** –

3.7.20 Validate

Up to *Packages*

Omeka_Validate_Confirmation

Package: *Validate*

class Omeka_Validate_Confirmation

extends Zend_Validate_Abstract

Adapted from Zend Framework documentation on custom validators.

constant Omeka_Validate_Confirmation::**NOT_MATCH**
Error message for non-matching confirmation.

property Omeka_Validate_Confirmation::$**_field**
protected string

Field needing confirmation.

property Omeka_Validate_Confirmation::$**_messageTemplates**
protected array

Error messages.

property Omeka_Validate_Confirmation::$**_messageVariables**
protected array

Error message replace variables.

Omeka_Validate_Confirmation::**__construct**(*$field*)
Sets validator options

Parameters

- **$field** –

Omeka_Validate_Confirmation::**isValid**(*$value*, *$context* = *null*)
Check that the value is valid.

Parameters

- **$value** (*string*) –
- **$context** (*string|array*) –

Returns bool

Omeka_Validate_Confirmation::**getField**()
Get the name of the field that needs confirmation.

Returns string

Omeka_Validate_Confirmation::**setField**(*$field*)
Set the name of the field that needs confirmation.

Parameters

- **$field** (*string*) –

Omeka_Validate_Errors

Package: *Validate*

class Omeka_Validate_Errors

extends ArrayObject

implements Countable implements Serializable implements ArrayAccess implements Traversable implements IteratorAggregate

This is an object wrapper for validation errors. The primary advantage to having this class is that casting it to a string will convert the errors into a nicely formatted, human-readable string.

property Omeka_Validate_Errors::$**_errors**
protected array

List of validation errors.

Omeka_Validate_Errors::**__construct**(*$errors* = *null*)

Parameters

- **$errors** (*array|null*) – Initial errors to set.

Omeka_Validate_Errors::**offsetGet**(*$key*)
Get an error from the list. Required by ArrayObject.

Parameters

- **$key** (*mixed*) – Key into array.

Omeka_Validate_Errors::**offsetSet**(*$key*, *$val*)
Set an error into the list. Required by ArrayObject.

Parameters

- **$key** (*mixed*) – Key into array.
- **$val** (*mixed*) – Value to store.

Omeka_Validate_Errors::**get**()
Get the array of errors.

Returns array

Omeka_Validate_Errors::**count**()
Get the number of errors.

Returns int

Omeka_Validate_Errors::**__toString**()
Get a string representation of all the stored errors.

Returns string

Omeka_Validate_Errors::**offsetExists**(*$index*)

Parameters

- **$index** –

Omeka_Validate_Errors::**offsetUnset**(*$index*)

Parameters

- **$index** –

Omeka_Validate_Errors::**append**(*$value*)

Parameters

- **$value** –

Omeka_Validate_Errors::**getArrayCopy**()

Omeka_Validate_Errors::**getFlags**()

Omeka_Validate_Errors::**setFlags**(*$flags*)

Parameters

- **$flags** –

Omeka_Validate_Errors::**asort**()

Omeka_Validate_Errors::**ksort**()

Omeka_Validate_Errors::**uasort**(*$cmp_function*)

Parameters

- **$cmp_function** –

Omeka_Validate_Errors::**uksort**(*$cmp_function*)

Parameters

- **$cmp_function** –

Omeka_Validate_Errors::**natsort**()

Omeka_Validate_Errors::**natcasesort**()

Omeka_Validate_Errors::**unserialize**(*$serialized*)

Parameters

- **$serialized** –

Omeka_Validate_Errors::**serialize**()

Omeka_Validate_Errors::**getIterator**()

Omeka_Validate_Errors::**exchangeArray**(*$array*)

Parameters

- **$array** –

Omeka_Validate_Errors::**setIteratorClass**(*$iteratorClass*)

Parameters

- **$iteratorClass** –

Omeka_Validate_Errors::**getIteratorClass**()

Omeka_Validate_Exception

Package: *Validate*

class Omeka_Validate_Exception

extends Exception

implements Throwable

Exception that is thrown when a form could not be validated correctly.

property Omeka_Validate_Exception::$**_errors**
protected string

Message representing form errors.

property Omeka_Validate_Exception::$**message**
protected

property Omeka_Validate_Exception::$**code**
protected

property Omeka_Validate_Exception::$**file**
protected

property Omeka_Validate_Exception::$**line**
protected

Omeka_Validate_Exception::**__construct**(*$errors*)

Parameters

- **$errors** –

Omeka_Validate_Exception::**getErrors**()
Get the error message that caused this exception.

Returns string

Omeka_Validate_Exception::**__clone**()

Omeka_Validate_Exception::**__wakeup**()

Omeka_Validate_Exception::**getMessage**()

Omeka_Validate_Exception::**getCode**()

Omeka_Validate_Exception::**getFile**()

Omeka_Validate_Exception::**getLine**()

Omeka_Validate_Exception::**getTrace**()

Omeka_Validate_Exception::**getPrevious**()

Omeka_Validate_Exception::**getTraceAsString**()

Omeka_Validate_Exception::**__toString**()

Omeka_Validate_File_Extension

Package: *Validate*

class Omeka_Validate_File_Extension

extends Zend_Validate_File_Extension

Define custom behavior for the default whitelist file extension validator.

Baseline behavior of this class is to tweak the default error messages. Messages are intentionally as detailed as possible. Note that it is the responsibility of plugin writers to suppress or replace these messages if necessary for security reasons, e.g. if displaying it to the end user might expose the site to vulnerability probes.

property Omeka_Validate_File_Extension::$**_messageTemplates**
protected array

Overrides default error message templates.

property Omeka_Validate_File_Extension::$**_targetExtension**
protected string

The extension of the file being validated

Omeka_Validate_File_Extension::**__construct**(*$options = null*)
Constructor retrieves the whitelist from the database if no arguments are given.

Parameters

- **$options** (*mixed*) –

Omeka_Validate_File_Extension::**isValid**(*$value*, *$file = null*)
Returns true if and only if the fileextension of $value is included in the set extension list.

Parameters

- **$value** (*string*) – Real file to check for extension.
- **$file** (*array*) – File data from Zend_File_Transfer.

Returns bool

Omeka_Validate_File_MimeType

Package: *Validate*

class Omeka_Validate_File_MimeType

extends Zend_Validate_Abstract

Validates files against a MIME type whitelist.

property Omeka_Validate_File_MimeType::$**_messageTemplates**
protected array

property Omeka_Validate_File_MimeType::$**_messageVariables**
protected array

property Omeka_Validate_File_MimeType::$**_customWhitelist**
protected string

property Omeka_Validate_File_MimeType::$**_file**
protected string

property Omeka_Validate_File_MimeType::$**_mimeType**
protected string

Omeka_Validate_File_MimeType::**__construct**()
Construct the validator object.

Omeka_Validate_File_MimeType::**isValid**(*$file*)
Vaidate the file MIME type.

Parameters

- **$file** (*string*) –

Returns bool

Omeka_Validate_HexColor

Package: *Validate*

class Omeka_Validate_HexColor

extends Zend_Validate_Abstract

Validate an input as a hex color value as accepted by HTML5's color input.

property Omeka_Validate_HexColor::$**_messageTemplates**
protected

Omeka_Validate_HexColor::**isValid**(*$value*)

Parameters

- **$value** –

Omeka_Validate_Uri

Package: *Validate*

class Omeka_Validate_Uri

extends Zend_Validate_Abstract

Adapted from: http://www.techchorus.net/validate-uri-form-fields-zend-framework-custom-validator

property Omeka_Validate_Uri::$**_messageTemplates**
protected

Omeka_Validate_Uri::**isValid**(*$value*)

Parameters

- **$value** –

Omeka_Validate_UserPassword

Package: *Validate*

class Omeka_Validate_UserPassword

extends Zend_Validate_Abstract

Validate a password to see if it matches that of an existing user.

constant Omeka_Validate_UserPassword::**INVALID**
Invalid password error.

property Omeka_Validate_UserPassword::$**_messageTemplates**
protected array

Error message templates.

Omeka_Validate_UserPassword::**__construct**(*User $user*)

Parameters

- **$user** (User) –

Omeka_Validate_UserPassword::**isValid**(*$value*, *$context = null*)
Validate against a user's stored password.

Parameters

- **$value** (*string*) – Password to check.
- **$context** (*null*) – Not used.

3.7.21 View

Up to *Packages*

Omeka_View

Package: *View*

class Omeka_View

extends Zend_View_Abstract

Customized subclass of Zend Framework's View class.

This adds the correct script paths for themes and plugins so that controllers can render the appropriate scripts.

This will also inject directly into the view scripts all variables that have been assigned to the view, so that theme writers can access them as $item instead of $this->item, for example.

property Omeka_View::$**_asset_paths**
protected array

Maintains a key => value pairing corresponding to hard path => web path for possible assets for Omeka views.

Omeka_View::**__construct**(*$config = array()*)

Parameters

- **$config** (*array*) – View configuration.

Omeka_View::**getAssetPaths**()
Get the currently-configured asset paths.

Returns array

Omeka_View::**addAssetPath**(*$physical*, *$web*)
Add an asset path to the view.

Parameters

- **$physical** (*string*) – Local filesystem path.
- **$web** (*string*) – URL path.

Omeka_View::**setAssetPath**(*$physical*, *$web*)
Remove the existing asset paths and set a single new one.

Parameters

- **$physical** (*string*) – Local filesystem path.
- **$web** (*string*) – URL path.

Omeka_View::**_run**()
Allow for variables set to the view object to be referenced in the view script by their actual name.

Also allows access to theme helpers.

For example, in a controller you might do something like: $view->assign('themes', $themes); Normally in the view you would then reference $themes through: $this->themes;

Now you can reference it simply by using: $themes;

Omeka_View::**_loadCustomThemeScripts**()
Look for a 'custom.php' script in all script paths and include the file if it exists.

Omeka_View::**addScriptPath**(*$path*)
Add a script path to the view.

Parameters

- **$path** (*string*) – Local filesystem path.

Omeka_View_Exception

Package: *View*

class Omeka_View_Exception

extends Exception

implements Throwable

Exceptions thrown by Omeka view code and helpers.

property Omeka_View_Exception::$**message**
protected

property Omeka_View_Exception::$**code**
protected

property Omeka_View_Exception::$**file**
protected

property Omeka_View_Exception::$**line**
protected

Omeka_View_Exception::**__clone**()

Omeka_View_Exception::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_View_Exception::**__wakeup**()

Omeka_View_Exception::**getMessage**()

Omeka_View_Exception::**getCode**()

Omeka_View_Exception::**getFile**()

Omeka_View_Exception::**getLine**()

Omeka_View_Exception::**getTrace**()

Omeka_View_Exception::**getPrevious**()

Omeka_View_Exception::**getTraceAsString**()

Omeka_View_Exception::**__toString**()

View\Helper

Up to *View*

3.8 libraries/Omeka

3.8.1 libraries/Omeka/Acl

libraries/Omeka/Acl/Assert

3.8.2 libraries/Omeka/Application

libraries/Omeka/Application/Resource

libraries/Omeka/Application/Resource/Jobs

3.8.3 libraries/Omeka/Auth

libraries/Omeka/Auth/Adapter

3.8.4 libraries/Omeka/Controller

libraries/Omeka/Controller/Exception

libraries/Omeka/Controller/Plugin

Omeka_Controller_Plugin_Jsonp

Package: *Controller\Plugin*

class Omeka_Controller_Plugin_Jsonp

extends `Zend_Controller_Plugin_Abstract`

Sets the Content-Type header for all JSON-P requests.

constant `Omeka_Controller_Plugin_Jsonp::`**CALLBACK_KEY**
Callback parameter key.

`Omeka_Controller_Plugin_Jsonp::`**postDispatch**(*Zend_Controller_Request_Abstract $request*)
Set the 'Content-Type' HTTP header to 'application/x-javascript' for omeka-json requests.

Parameters

- **$request** (*Zend_Controller_Request_Abstract*) – Request object.

Returns void

libraries/Omeka/Controller/Router

3.8.5 libraries/Omeka/Db

libraries/Omeka/Db/Migration

libraries/Omeka/Db/Select

3.8.6 libraries/Omeka/File

libraries/Omeka/File/Derivative

libraries/Omeka/File/Derivative/Strategy

libraries/Omeka/File/Ingest

Omeka_File_Ingest_AbstractIngest

Package: File\Ingest

class Omeka_File_Ingest_AbstractIngest
An abstract class that handles ingesting files into Omeka and database.

Specific responsibilities handled by this class: - Parsing/validating arbitrary inputs that somehow identify the files to be ingested. - Iterating through the parsed file information, validating, and transferring each file to Omeka. - Inserting a new record into the files table that corresponds to the transferred file's metadata. - Returning a collection of the records associated with the ingested files.

Typical usage is via the factory() method:

<code> $ingest = Omeka_File_Ingest_AbstractIngest::factory('Url', $item); $fileRecords = $ingest->ingest('http://www.example.com'); </code>

property _item
protected Item

property _options
protected array

Set of arbitrary options to use when ingesting files.

property mimeType
string

The current validated file MIME type.

setItem(*Item $item*)
Set the item to use as a target when ingesting files.

Parameters

- **$item** (Item) –

factory(*$adapterName*, *$item*, *$options = array()*)
Factory to retrieve Omeka_File_Ingest_* instances.

Parameters

- **$adapterName** (*string*) – Ingest adapter.
- **$item** (Item) –
- **$options** (*array*) –

Returns Omeka_File_Ingest_AbstractIngest

_getOriginalFilename(*$fileInfo*)
Retrieve the original filename of the file.

Parameters

- **$fileInfo** (*array*) –

Returns string

_transferFile(*$fileInfo*, *$originalFilename*)
Transfer the file to Omeka.

To indicate validation errors, Omeka_File_Ingest_InvalidException can be thrown at any time. To indicate other types of non-recoverable errors related to file ingest, throw Omeka_File_Ingest_Exception.

Parameters

- **$fileInfo** (*array*) –
- **$originalFilename** (*string*) –

Returns string Real path to the transferred file.

_parseFileInfo(*$files*)
Ingest classes receive arbitrary information. This method needs to parse that information into an iterable array so that multiple files can be ingested from a single identifier.

Example use case is Omeka_File_Ingest_Upload.

Parameters

- **$files** (*mixed*) –

Returns array

setOptions(*$options*)
Set options for ingesting files.

Parameters

- **$options** (*array*) – Available options include: - 'ignore_invalid_files': boolean false by default. Determine whether or not to throw exceptions when a file is not valid. This can be based on a number of factors: whether or not the original identifier is valid (i.e. a valid URL), whether or not the file itself is valid (i.e. invalid file extension), or whether the basic algorithm for ingesting the file fails (i.e., files cannot be transferred because the files/ directory is not writeable). This option is primarily useful for skipping known invalid files when ingesting large data sets.

ingest(*$fileInfo*)

Ingest based on arbitrary file identifier info.

If this is an array that has a 'metadata' key, that should be an array representing element text metadata to assign to the file. See ActsAsElementText::addElementTextsByArray() for more details.

Parameters

- **$fileInfo** (*mixed*) – An arbitrary input (array, string, object, etc.) that corresponds to one or more files to be ingested into Omeka.

Returns array Ingested file records.

_ignoreIngestErrors()

Determine whether or not to ignore file ingest errors. Based on 'ignore_invalid_files', which is false by default.

Returns bool

_logException(*Exception $e*)

Log any exceptions that are thrown as a result of attempting to ingest invalid files.

These are logged as warnings because they are being ignored by the script, so they don't actually kill the file ingest process.

Parameters

- **$e** (*Exception*) –

_createFile(*$newFilePath*, *$oldFilename*, *$elementMetadata = array()*)

Insert a File record corresponding to an ingested file and its metadata.

Parameters

- **$newFilePath** (*string*) – Path to the file within Omeka.
- **$oldFilename** (*string*) – The original filename for the file. This will usually be displayed to the end user.
- **$elementMetadata** (*array*) – See ActsAsElementText::addElementTextsByArray() for more information about the format of this array.

Returns File

_getDestination(*$fromFilename*)

Retrieve the destination path for the file to be transferred.

This will generate an archival filename in order to prevent naming conflicts between ingested files.

This should be used as necessary by Omeka_File_Ingest_AbstractIngest implementations in order to determine where to transfer any given file.

Parameters

- **$fromFilename** (*string*) – The filename from which to derive the archival filename.

Returns string

addValidator(*Zend_Validate_Interface $validator*)
Add Zend Framework file validators.

Emulates the way Zend Framework adds validators.

Parameters

- **$validator** (*Zend_Validate_Interface*) –

Returns Omeka_File_Ingest_AbstractIngest

_validateFile(*$filePath*, *$fileInfo*)
Validate a file that has been transferred to Omeka.

Implementations of Omeka_File_Ingest_AbstractIngest should use this to validate the uploaded file based on user-defined security criteria.

Important: $fileInfo may need to contain the following keys in order to work with particular Zend_Validate_File_* validation classes:

- 'name': string filename (for Zend_Validate_File_Extension) If ZF is

unable to determine the file extension when validating, it will check the 'name' attribute instead. Current use cases involve saving the file to a temporary location before transferring to Omeka. Most temporary files do not maintain the original file extension. - 'type': string MIME type (for Zend_Validate_File_MimeType) If ZF is unable to determine the mime type from the transferred file. Unless the server running Omeka has a mime_magic file or has installed the FileInfo extension, this will be necessary.

Parameters

- **$filePath** (*string*) – Absolute path to the file. The file should be local and readable, which is required by most (if not all) of the Zend_Validate_File_* classes.
- **$fileInfo** (*array*) – Set of file info that describes a given file being ingested.

Returns bool True if valid, otherwise throws an exception.

Omeka_File_Ingest_AbstractSourceIngest

Package: File\Ingest

class Omeka_File_Ingest_AbstractSourceIngest

extends *Omeka_File_Ingest_AbstractIngest*

This abstract class encapsulates all the behavior that facilitates file ingest based on the assumption that each file can be retrieved via a string containing both the name and location of that file.

Applies to: URLs, file paths on a server. Does not apply to: direct HTTP uploads.

Also, if the original filename is not properly represented by the source identifier (incorrect file extension, etc.), a more accurate filename can be provided via the 'filename' attribute.

property Omeka_File_Ingest_AbstractSourceIngest::$**_item**
protected Item

property Omeka_File_Ingest_AbstractSourceIngest::$**_options**
protected array

Set of arbitrary options to use when ingesting files.

property Omeka_File_Ingest_AbstractSourceIngest::$**mimeType**
string

The current validated file MIME type.

Omeka_File_Ingest_AbstractSourceIngest::**_getFileSource**(*$fileInfo*)
The 'source' key of the file info is parsed out by default.

Parameters

- **$fileInfo** –

Returns string

Omeka_File_Ingest_AbstractSourceIngest::**_parseFileInfo**(*$files*)
Normalize a file info array.

Files can be represented as one of the following: - a string, representing the source identifier for a single file. - an array containing a 'source' key. - an array of strings. - an array of arrays that each contain a 'source' key.

Parameters

- **$files** (*string|array*) –

Returns array Formatted info array.

Omeka_File_Ingest_AbstractSourceIngest::**_addZendValidatorAttributes**(*$fileInfo*)
Modify the set of info about each file to ensure that it is compatible with the Zend_Validate_File_* validators.

Parameters

- **$fileInfo** (*array*) –

Returns array

Omeka_File_Ingest_AbstractSourceIngest::**_getOriginalFilename**(*$fileInfo*)
Retrieve the original filename.

By default, this is stored as the 'name' attribute in the array.

Parameters

- **$fileInfo** (*array*) –

Returns string

Omeka_File_Ingest_AbstractSourceIngest::**_transferFile**(*$fileInfo*, *$originalFilename*)
Transfer the file to Omeka.

Parameters

- **$fileInfo** (*array*) –
- **$originalFilename** (*string*) –

Returns string Path to file in Omeka.

Omeka_File_Ingest_AbstractSourceIngest::**_transfer**(*$source*, *$destination*, *$fileInfo*)
Transfer the file from the original location to its destination.

Examples would include transferring the file via wget, or making use of stream wrappers to copy the file.

Parameters

- **$source** (*string*) –
- **$destination** (*string*) –
- **$fileInfo** (*array*) –

Omeka_File_Ingest_AbstractSourceIngest::**_validateSource**(*$source*, *$info*)

Determine whether or not the file source is valid.

Examples of this would include determining whether a URL exists, or whether read access is available for a given file.

Parameters

- **$source** (*string*) –
- **$info** (*array*) –

Omeka_File_Ingest_AbstractSourceIngest::**setItem**(*Item $item*)

Set the item to use as a target when ingesting files.

Parameters

- **$item** (Item) –

Omeka_File_Ingest_AbstractSourceIngest::**factory**(*$adapterName*, *$item*, *$options = array()*)

Factory to retrieve Omeka_File_Ingest_* instances.

Parameters

- **$adapterName** (*string*) – Ingest adapter.
- **$item** (Item) –
- **$options** (*array*) –

Returns Omeka_File_Ingest_AbstractIngest

Omeka_File_Ingest_AbstractSourceIngest::**setOptions**(*$options*)

Set options for ingesting files.

Parameters

- **$options** (*array*) – Available options include: - 'ignore_invalid_files': boolean false by default. Determine whether or not to throw exceptions when a file is not valid. This can be based on a number of factors: whether or not the original identifier is valid (i.e. a valid URL), whether or not the file itself is valid (i.e. invalid file extension), or whether the basic algorithm for ingesting the file fails (i.e., files cannot be transferred because the files/ directory is not writeable). This option is primarily useful for skipping known invalid files when ingesting large data sets.

Omeka_File_Ingest_AbstractSourceIngest::**ingest**(*$fileInfo*)

Ingest based on arbitrary file identifier info.

If this is an array that has a 'metadata' key, that should be an array representing element text metadata to assign to the file. See ActsAsElementText::addElementTextsByArray() for more details.

Parameters

- **$fileInfo** (*mixed*) – An arbitrary input (array, string, object, etc.) that corresponds to one or more files to be ingested into Omeka.

Returns array Ingested file records.

Omeka_File_Ingest_AbstractSourceIngest::**_ignoreIngestErrors**()
Determine whether or not to ignore file ingest errors. Based on 'ignore_invalid_files', which is false by default.

Returns bool

Omeka_File_Ingest_AbstractSourceIngest::**_logException**(*Exception $e*)
Log any exceptions that are thrown as a result of attempting to ingest invalid files.

These are logged as warnings because they are being ignored by the script, so they don't actually kill the file ingest process.

Parameters

- **$e** (*Exception*) –

Omeka_File_Ingest_AbstractSourceIngest::**_createFile**(*$newFilePath*, *$oldFilename*, *$elementMetadata = array()*)
Insert a File record corresponding to an ingested file and its metadata.

Parameters

- **$newFilePath** (*string*) – Path to the file within Omeka.
- **$oldFilename** (*string*) – The original filename for the file. This will usually be displayed to the end user.
- **$elementMetadata** (*array*) – See ActsAsElementText::addElementTextsByArray() for more information about the format of this array.

Returns File

Omeka_File_Ingest_AbstractSourceIngest::**_getDestination**(*$fromFilename*)
Retrieve the destination path for the file to be transferred.

This will generate an archival filename in order to prevent naming conflicts between ingested files.

This should be used as necessary by Omeka_File_Ingest_AbstractIngest implementations in order to determine where to transfer any given file.

Parameters

- **$fromFilename** (*string*) – The filename from which to derive the archival filename.

Returns string

Omeka_File_Ingest_AbstractSourceIngest::**addValidator**(*Zend_Validate_Interface $validator*)
Add Zend Framework file validators.

Emulates the way Zend Framework adds validators.

Parameters

- **$validator** (*Zend_Validate_Interface*) –

Returns Omeka_File_Ingest_AbstractIngest

Omeka_File_Ingest_AbstractSourceIngest::**_validateFile**(*$filePath*, *$fileInfo*)
Validate a file that has been transferred to Omeka.

Implementations of Omeka_File_Ingest_AbstractIngest should use this to validate the uploaded file based on user-defined security criteria.

Important: $fileInfo may need to contain the following keys in order to work with particular Zend_Validate_File_* validation classes:

- 'name': string filename (for Zend_Validate_File_Extension) If ZF is

unable to determine the file extension when validating, it will check the 'name' attribute instead. Current use cases involve saving the file to a temporary location before transferring to Omeka. Most temporary files do not maintain the original file extension. - 'type': string MIME type (for Zend_Validate_File_MimeType) If ZF is unable to determine the mime type from the transferred file. Unless the server running Omeka has a mime_magic file or has installed the FileInfo extension, this will be necessary.

Parameters

- **$filePath** (*string*) – Absolute path to the file. The file should be local and readable, which is required by most (if not all) of the Zend_Validate_File_* classes.
- **$fileInfo** (*array*) – Set of file info that describes a given file being ingested.

Returns bool True if valid, otherwise throws an exception.

Omeka_File_Ingest_Exception

Package: File\Ingest

class Omeka_File_Ingest_Exception

extends `Exception`

implements `Throwable`

An exception to be thrown when something goes wrong during the file ingest process.

property `Omeka_File_Ingest_Exception::$`**message**
protected

property `Omeka_File_Ingest_Exception::$`**code**
protected

property `Omeka_File_Ingest_Exception::$`**file**
protected

property `Omeka_File_Ingest_Exception::$`**line**
protected

`Omeka_File_Ingest_Exception::`**__clone**`()`

`Omeka_File_Ingest_Exception::`**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

`Omeka_File_Ingest_Exception::`**__wakeup**`()`

`Omeka_File_Ingest_Exception::`**getMessage**`()`

`Omeka_File_Ingest_Exception::`**getCode**`()`

Omeka_File_Ingest_Exception::**getFile**()

Omeka_File_Ingest_Exception::**getLine**()

Omeka_File_Ingest_Exception::**getTrace**()

Omeka_File_Ingest_Exception::**getPrevious**()

Omeka_File_Ingest_Exception::**getTraceAsString**()

Omeka_File_Ingest_Exception::**__toString**()

Omeka_File_Ingest_Filesystem

Package: File\Ingest

class Omeka_File_Ingest_Filesystem

extends *Omeka_File_Ingest_AbstractSourceIngest*

Implements ingesting files from the local filesystem.

property Omeka_File_Ingest_Filesystem::$**_item**
protected Item

property Omeka_File_Ingest_Filesystem::$**_options**
protected array

Set of arbitrary options to use when ingesting files.

property Omeka_File_Ingest_Filesystem::$**mimeType**
string

The current validated file MIME type.

Omeka_File_Ingest_Filesystem::**_getOriginalFilename**(*$info*)
Retrieve the original filename of the file to be transferred.

Check for the 'name' attribute first, otherwise extract the basename() from the given file path.

Parameters

- **$info** (*array*) – File info array.

Returns string

Omeka_File_Ingest_Filesystem::**_transfer**(*$source*, *$destination*, *$info*)
Transfer a file.

Parameters

- **$source** (*string*) – Source path.
- **$destination** (*string*) – Destination path.
- **$info** (*array*) – File info array. If 'rename' is specified as true, move the file instead of copying.

Omeka_File_Ingest_Filesystem::**_validateSource**(*$source*, *$info*)
Validate file transfer.

Parameters

- **$source** (*string*) – Source path.
- **$info** (*array*) – File info array.

Omeka_File_Ingest_Filesystem::**_getFileSource**(*$fileInfo*)
The 'source' key of the file info is parsed out by default.

Parameters

- **$fileInfo** –

Returns string

Omeka_File_Ingest_Filesystem::**_parseFileInfo**(*$files*)
Normalize a file info array.

Files can be represented as one of the following: - a string, representing the source identifier for a single file. - an array containing a 'source' key. - an array of strings. - an array of arrays that each contain a 'source' key.

Parameters

- **$files** (*string|array*) –

Returns array Formatted info array.

Omeka_File_Ingest_Filesystem::**_addZendValidatorAttributes**(*$fileInfo*)
Modify the set of info about each file to ensure that it is compatible with the Zend_Validate_File_* validators.

Parameters

- **$fileInfo** (*array*) –

Returns array

Omeka_File_Ingest_Filesystem::**_transferFile**(*$fileInfo*, *$originalFilename*)
Transfer the file to Omeka.

Parameters

- **$fileInfo** (*array*) –
- **$originalFilename** (*string*) –

Returns string Path to file in Omeka.

Omeka_File_Ingest_Filesystem::**setItem**(*Item $item*)
Set the item to use as a target when ingesting files.

Parameters

- **$item** (Item) –

Omeka_File_Ingest_Filesystem::**factory**(*$adapterName*, *$item*, *$options = array()*)
Factory to retrieve Omeka_File_Ingest_* instances.

Parameters

- **$adapterName** (*string*) – Ingest adapter.
- **$item** (Item) –
- **$options** (*array*) –

Returns Omeka_File_Ingest_AbstractIngest

Omeka_File_Ingest_Filesystem::**setOptions**(*$options*)
Set options for ingesting files.

Parameters

- **$options** (*array*) – Available options include: - 'ignore_invalid_files': boolean false by default. Determine whether or not to throw exceptions when a file is not valid. This can be based on a number of factors: whether or not the original identifier is valid (i.e. a valid URL), whether or not the file itself is valid (i.e. invalid file extension), or whether the basic algorithm for ingesting the file fails (i.e., files cannot be transferred because the files/ directory is not writeable). This option is primarily useful for skipping known invalid files when ingesting large data sets.

Omeka_File_Ingest_Filesystem::**ingest**(*$fileInfo*)
Ingest based on arbitrary file identifier info.

If this is an array that has a 'metadata' key, that should be an array representing element text metadata to assign to the file. See ActsAsElementText::addElementTextsByArray() for more details.

Parameters

- **$fileInfo** (*mixed*) – An arbitrary input (array, string, object, etc.) that corresponds to one or more files to be ingested into Omeka.

Returns array Ingested file records.

Omeka_File_Ingest_Filesystem::**_ignoreIngestErrors**()
Determine whether or not to ignore file ingest errors. Based on 'ignore_invalid_files', which is false by default.

Returns bool

Omeka_File_Ingest_Filesystem::**_logException**(*Exception $e*)
Log any exceptions that are thrown as a result of attempting to ingest invalid files.

These are logged as warnings because they are being ignored by the script, so they don't actually kill the file ingest process.

Parameters

- **$e** (*Exception*) –

Omeka_File_Ingest_Filesystem::**_createFile**(*$newFilePath*, *$oldFilename*, *$elementMetadata = array()*)
Insert a File record corresponding to an ingested file and its metadata.

Parameters

- **$newFilePath** (*string*) – Path to the file within Omeka.
- **$oldFilename** (*string*) – The original filename for the file. This will usually be displayed to the end user.
- **$elementMetadata** (*array*) – See ActsAsElementText::addElementTextsByArray() for more information about the format of this array.

Returns File

Omeka_File_Ingest_Filesystem::**_getDestination**(*$fromFilename*)
Retrieve the destination path for the file to be transferred.

This will generate an archival filename in order to prevent naming conflicts between ingested files.

This should be used as necessary by Omeka_File_Ingest_AbstractIngest implementations in order to determine where to transfer any given file.

Parameters

- **$fromFilename** (*string*) – The filename from which to derive the archival filename.

Returns string

Omeka_File_Ingest_Filesystem::**addValidator**(*Zend_Validate_Interface $validator*)

Add Zend Framework file validators.

Emulates the way Zend Framework adds validators.

Parameters

- **$validator** (*Zend_Validate_Interface*) –

Returns Omeka_File_Ingest_AbstractIngest

Omeka_File_Ingest_Filesystem::**_validateFile**(*$filePath*, *$fileInfo*)

Validate a file that has been transferred to Omeka.

Implementations of Omeka_File_Ingest_AbstractIngest should use this to validate the uploaded file based on user-defined security criteria.

Important: $fileInfo may need to contain the following keys in order to work with particular Zend_Validate_File_* validation classes:

- 'name': string filename (for Zend_Validate_File_Extension) If ZF is

unable to determine the file extension when validating, it will check the 'name' attribute instead. Current use cases involve saving the file to a temporary location before transferring to Omeka. Most temporary files do not maintain the original file extension. - 'type': string MIME type (for Zend_Validate_File_MimeType) If ZF is unable to determine the mime type from the transferred file. Unless the server running Omeka has a mime_magic file or has installed the FileInfo extension, this will be necessary.

Parameters

- **$filePath** (*string*) – Absolute path to the file. The file should be local and readable, which is required by most (if not all) of the Zend_Validate_File_* classes.
- **$fileInfo** (*array*) – Set of file info that describes a given file being ingested.

Returns bool True if valid, otherwise throws an exception.

Omeka_File_Ingest_InvalidException

Package: File\Ingest

class Omeka_File_Ingest_InvalidException

extends Zend_Validate_Exception

Exception for indicating validation errors within an ingest.

Omeka_File_Ingest_Upload

Package: File\Ingest

class Omeka_File_Ingest_Upload

extends *Omeka_File_Ingest_AbstractIngest*

This class creates a bridge between the ZF File Transfer HTTP adapter and Omeka's file ingest classes.

property Omeka_File_Ingest_Upload::$**_adapter**
protected Zend_File_Transfer_Adapter_Http

property Omeka_File_Ingest_Upload::$**_adapterOptions**
protected array

property Omeka_File_Ingest_Upload::$**_item**
protected Item

property Omeka_File_Ingest_Upload::$**_options**
protected array

Set of arbitrary options to use when ingesting files.

property Omeka_File_Ingest_Upload::$**mimeType**
string

The current validated file MIME type.

Omeka_File_Ingest_Upload::**_buildAdapter**()
Create a ZF HTTP file transfer adapter.

Omeka_File_Ingest_Upload::**setOptions**(*$options*)
In addition to the default options available for Omeka_File_Ingest_AbstractIngest, this understands the following options: - 'ignoreNoFile' => boolean False by default. Whether or not to ignore - validation errors that occur when an uploaded file is missing. This - may occur when a file input is left empty on a form.

This option can be overridden by the 'ignore_invalid_files' option. For instance, if 'ignoreNoFile' is set to false but 'ignore_invalid_files' is set to true, any exceptions due to missing uploads will be suppressed and ignored.

Parameters

- **$options** (*array*) –

Omeka_File_Ingest_Upload::**_getOriginalFilename**(*$fileInfo*)
The 'name' attribute of the $_FILES array will always contain the original name of the file.

Parameters

- **$fileInfo** (*array*) –

Returns string

Omeka_File_Ingest_Upload::**_transferFile**(*$fileInfo*, *$originalFilename*)
Use the Zend_File_Transfer adapter to upload the file.

Parameters

- **$fileInfo** (*array*) –
- **$originalFilename** (*string*) –

Returns string Path to the file in Omeka.

Omeka_File_Ingest_Upload::**_parseFileInfo**(*$fileInfo*)
Use the adapter to extract the array of file information.

Parameters

- **$fileInfo** (*string|null*) – The name of the form input to ingest.

Returns array

Omeka_File_Ingest_Upload::**addValidator**(*Zend_Validate_Interface $validator*)
Use the Zend Framework adapter to handle validation instead of the built-in _validateFile() method.

Parameters

- **$validator** (*Zend_Validate_Interface*) –

Omeka_File_Ingest_Upload::**setItem**(*Item $item*)
Set the item to use as a target when ingesting files.

Parameters

- **$item** (Item) –

Omeka_File_Ingest_Upload::**factory**(*$adapterName*, *$item*, *$options = array()*)
Factory to retrieve Omeka_File_Ingest_* instances.

Parameters

- **$adapterName** (*string*) – Ingest adapter.
- **$item** (Item) –
- **$options** (*array*) –

Returns Omeka_File_Ingest_AbstractIngest

Omeka_File_Ingest_Upload::**ingest**(*$fileInfo*)
Ingest based on arbitrary file identifier info.

If this is an array that has a 'metadata' key, that should be an array representing element text metadata to assign to the file. See ActsAsElementText::addElementTextsByArray() for more details.

Parameters

- **$fileInfo** (*mixed*) – An arbitrary input (array, string, object, etc.) that corresponds to one or more files to be ingested into Omeka.

Returns array Ingested file records.

Omeka_File_Ingest_Upload::**_ignoreIngestErrors**()
Determine whether or not to ignore file ingest errors. Based on 'ignore_invalid_files', which is false by default.

Returns bool

Omeka_File_Ingest_Upload::**_logException**(*Exception $e*)
Log any exceptions that are thrown as a result of attempting to ingest invalid files.

These are logged as warnings because they are being ignored by the script, so they don't actually kill the file ingest process.

Parameters

- **$e** (*Exception*) –

Omeka_File_Ingest_Upload::**_createFile**(*$newFilePath*, *$oldFilename*, *$elementMetadata = array()*)
Insert a File record corresponding to an ingested file and its metadata.

Parameters

- **$newFilePath** (*string*) – Path to the file within Omeka.
- **$oldFilename** (*string*) – The original filename for the file. This will usually be displayed to the end user.

- **$elementMetadata** (*array*) – See ActsAsElementText::addElementTextsByArray() for more information about the format of this array.

Returns File

Omeka_File_Ingest_Upload::**_getDestination**(*$fromFilename*)

Retrieve the destination path for the file to be transferred.

This will generate an archival filename in order to prevent naming conflicts between ingested files.

This should be used as necessary by Omeka_File_Ingest_AbstractIngest implementations in order to determine where to transfer any given file.

Parameters

- **$fromFilename** (*string*) – The filename from which to derive the archival filename.

Returns string

Omeka_File_Ingest_Upload::**_validateFile**(*$filePath*, *$fileInfo*)

Validate a file that has been transferred to Omeka.

Implementations of Omeka_File_Ingest_AbstractIngest should use this to validate the uploaded file based on user-defined security criteria.

Important: $fileInfo may need to contain the following keys in order to work with particular Zend_Validate_File_* validation classes:

- 'name': string filename (for Zend_Validate_File_Extension) If ZF is

unable to determine the file extension when validating, it will check the 'name' attribute instead. Current use cases involve saving the file to a temporary location before transferring to Omeka. Most temporary files do not maintain the original file extension. - 'type': string MIME type (for Zend_Validate_File_MimeType) If ZF is unable to determine the mime type from the transferred file. Unless the server running Omeka has a mime_magic file or has installed the FileInfo extension, this will be necessary.

Parameters

- **$filePath** (*string*) – Absolute path to the file. The file should be local and readable, which is required by most (if not all) of the Zend_Validate_File_* classes.
- **$fileInfo** (*array*) – Set of file info that describes a given file being ingested.

Returns bool True if valid, otherwise throws an exception.

Omeka_File_Ingest_Url

Package: File\Ingest

class Omeka_File_Ingest_Url

extends *Omeka_File_Ingest_AbstractSourceIngest*

Ingest URLs into Omeka.

property Omeka_File_Ingest_Url::$**_item**

protected Item

property Omeka_File_Ingest_Url::$**_options**
protected array

Set of arbitrary options to use when ingesting files.

property Omeka_File_Ingest_Url::$**mimeType**
string

The current validated file MIME type.

Omeka_File_Ingest_Url::**_getOriginalFilename**(*$fileInfo*)
Return the original filename.

Parameters

- **$fileInfo** (*array*) –

Returns string

Omeka_File_Ingest_Url::**_getHttpClient**(*$source*)
Get a HTTP client for retrieving the given file.

Parameters

- **$source** (*string*) – Source URI.

Returns Zend_Http_Client

Omeka_File_Ingest_Url::**_transfer**(*$source*, *$destination*, *$fileInfo*)
Fetch a file from a URL.

Parameters

- **$source** (*string*) – Source URL.
- **$destination** (*string*) – Destination file path.
- **$fileInfo** (*array*) –

Omeka_File_Ingest_Url::**_validateSource**(*$source*, *$info*)
Ensure the source URL exists and can be read from.

Parameters

- **$source** (*string*) – Source URL.
- **$info** (*array*) – File info array (unused).

Omeka_File_Ingest_Url::**_getFileSource**(*$fileInfo*)
The 'source' key of the file info is parsed out by default.

Parameters

- **$fileInfo** –

Returns string

Omeka_File_Ingest_Url::**_parseFileInfo**(*$files*)
Normalize a file info array.

Files can be represented as one of the following: - a string, representing the source identifier for a single file. - an array containing a 'source' key. - an array of strings. - an array of arrays that each contain a 'source' key.

Parameters

- **$files** (*string|array*) –

Returns array Formatted info array.

Omeka_File_Ingest_Url::**_addZendValidatorAttributes**(*$fileInfo*)
Modify the set of info about each file to ensure that it is compatible with the Zend_Validate_File_* validators.

Parameters

- **$fileInfo** (*array*) –

Returns array

Omeka_File_Ingest_Url::**_transferFile**(*$fileInfo*, *$originalFilename*)
Transfer the file to Omeka.

Parameters

- **$fileInfo** (*array*) –
- **$originalFilename** (*string*) –

Returns string Path to file in Omeka.

Omeka_File_Ingest_Url::**setItem**(*Item $item*)
Set the item to use as a target when ingesting files.

Parameters

- **$item** (Item) –

Omeka_File_Ingest_Url::**factory**(*$adapterName*, *$item*, *$options = array()*)
Factory to retrieve Omeka_File_Ingest_* instances.

Parameters

- **$adapterName** (*string*) – Ingest adapter.
- **$item** (Item) –
- **$options** (*array*) –

Returns Omeka_File_Ingest_AbstractIngest

Omeka_File_Ingest_Url::**setOptions**(*$options*)
Set options for ingesting files.

Parameters

- **$options** (*array*) – Available options include: - 'ignore_invalid_files': boolean false by default. Determine whether or not to throw exceptions when a file is not valid. This can be based on a number of factors: whether or not the original identifier is valid (i.e. a valid URL), whether or not the file itself is valid (i.e. invalid file extension), or whether the basic algorithm for ingesting the file fails (i.e., files cannot be transferred because the files/ directory is not writeable). This option is primarily useful for skipping known invalid files when ingesting large data sets.

Omeka_File_Ingest_Url::**ingest**(*$fileInfo*)
Ingest based on arbitrary file identifier info.

If this is an array that has a 'metadata' key, that should be an array representing element text metadata to assign to the file. See ActsAsElementText::addElementTextsByArray() for more details.

Parameters

- **$fileInfo** (*mixed*) – An arbitrary input (array, string, object, etc.) that corresponds to one or more files to be ingested into Omeka.

Returns array Ingested file records.

`Omeka_File_Ingest_Url::_ignoreIngestErrors()`
Determine whether or not to ignore file ingest errors. Based on 'ignore_invalid_files', which is false by default.

Returns bool

`Omeka_File_Ingest_Url::_logException(Exception $e)`
Log any exceptions that are thrown as a result of attempting to ingest invalid files.

These are logged as warnings because they are being ignored by the script, so they don't actually kill the file ingest process.

Parameters

- **$e** (*Exception*) –

`Omeka_File_Ingest_Url::_createFile($newFilePath, $oldFilename, $elementMetadata = array())`
Insert a File record corresponding to an ingested file and its metadata.

Parameters

- **$newFilePath** (*string*) – Path to the file within Omeka.
- **$oldFilename** (*string*) – The original filename for the file. This will usually be displayed to the end user.
- **$elementMetadata** (*array*) – See ActsAsElementText::addElementTextsByArray() for more information about the format of this array.

Returns File

`Omeka_File_Ingest_Url::_getDestination($fromFilename)`
Retrieve the destination path for the file to be transferred.

This will generate an archival filename in order to prevent naming conflicts between ingested files.

This should be used as necessary by Omeka_File_Ingest_AbstractIngest implementations in order to determine where to transfer any given file.

Parameters

- **$fromFilename** (*string*) – The filename from which to derive the archival filename.

Returns string

`Omeka_File_Ingest_Url::addValidator(Zend_Validate_Interface $validator)`
Add Zend Framework file validators.

Emulates the way Zend Framework adds validators.

Parameters

- **$validator** (*Zend_Validate_Interface*) –

Returns Omeka_File_Ingest_AbstractIngest

`Omeka_File_Ingest_Url::_validateFile($filePath, $fileInfo)`
Validate a file that has been transferred to Omeka.

Implementations of Omeka_File_Ingest_AbstractIngest should use this to validate the uploaded file based on user-defined security criteria.

Important: $fileInfo may need to contain the following keys in order to work with particular Zend_Validate_File_* validation classes:

- 'name': string filename (for Zend_Validate_File_Extension) If ZF is

unable to determine the file extension when validating, it will check the 'name' attribute instead. Current use cases involve saving the file to a temporary location before transferring to Omeka. Most temporary files do not maintain the original file extension. - 'type': string MIME type (for Zend_Validate_File_MimeType) If ZF is unable to determine the mime type from the transferred file. Unless the server running Omeka has a mime_magic file or has installed the FileInfo extension, this will be necessary.

Parameters

- **$filePath** (*string*) – Absolute path to the file. The file should be local and readable, which is required by most (if not all) of the Zend_Validate_File_* classes.
- **$fileInfo** (*array*) – Set of file info that describes a given file being ingested.

Returns bool True if valid, otherwise throws an exception.

libraries/Omeka/File/MimeType

libraries/Omeka/File/MimeType/Detect

libraries/Omeka/File/MimeType/Detect/Strategy

Omeka_File_MimeType_Detect_Strategy_Browser

Package: File\MimeType\Detect\Strategy

class Omeka_File_MimeType_Detect_Strategy_Browser

implements *Omeka_File_MimeType_Detect_StrategyInterface*

Omeka_File_MimeType_Detect_Strategy_Browser::**detect**(*$file*)

Parameters

- **$file** –

Omeka_File_MimeType_Detect_Strategy_FileCommand

Package: File\MimeType\Detect\Strategy

class Omeka_File_MimeType_Detect_Strategy_FileCommand

implements *Omeka_File_MimeType_Detect_StrategyInterface*

Omeka_File_MimeType_Detect_Strategy_FileCommand::**detect**(*$file*)

Parameters

- **$file** –

Omeka_File_MimeType_Detect_Strategy_Fileinfo

Package: File\MimeType\Detect\Strategy

class Omeka_File_MimeType_Detect_Strategy_Fileinfo

implements *Omeka_File_MimeType_Detect_StrategyInterface*

Omeka_File_MimeType_Detect_Strategy_Fileinfo::**detect**(*$file*)

Parameters

- **$file** –

Omeka_File_MimeType_Detect_Strategy_GetId3

Package: File\MimeType\Detect\Strategy

class Omeka_File_MimeType_Detect_Strategy_GetId3

implements *Omeka_File_MimeType_Detect_StrategyInterface*

Omeka_File_MimeType_Detect_Strategy_GetId3::**detect**(*$file*)

Parameters

- **$file** –

Omeka_File_MimeType_Detect_Strategy_MimeContentType

Package: File\MimeType\Detect\Strategy

class Omeka_File_MimeType_Detect_Strategy_MimeContentType

implements *Omeka_File_MimeType_Detect_StrategyInterface*

Omeka_File_MimeType_Detect_Strategy_MimeContentType::**detect**(*$file*)

Parameters

- **$file** –

Omeka_File_MimeType_Detect_StrategyInterface

Package: File\MimeType\Detect\Strategy

interface Omeka_File_MimeType_Detect_StrategyInterface

detect(*$file*)

Parameters

- **$file** –

Omeka_File_MimeType_Detect

Package: File\MimeType\Detect

class Omeka_File_MimeType_Detect
Represents the detected MIME types of a file.

property _file
protected string

property _strategies
protected array

property _mimeType
protected string

property _mimeTypes
protected array

property _ambiguousMimeTypes
protected array

__construct (*$file*, *$strategies = array()*)
Set the required properties for detecting the MIME types of a file.

Parameters

- **$file** (*string|File*) – The full path to the file or a File record.
- **$strategies** (*array*) – An array of file detection strategies in priority order. If none are passed, a default list will be set. All strategies must implement Omeka_File_MimeType_Detect_StrategyInterface.

detect()
Detect the MIME type of this file.

Returns string The definitive MIME type.

getMimeType()
Get the definitive MIME type of this file.

Returns string

getMimeTypes()
Get the MIME types of this file, one for each detection strategy.

Returns array

Omeka_File_MimeType_Exception

Package: File\MimeType

class Omeka_File_MimeType_Exception

extends Exception

implements Throwable

property Omeka_File_MimeType_Exception::$**message**
protected

property Omeka_File_MimeType_Exception::$**code**
protected

property Omeka_File_MimeType_Exception::$**file**
protected

property Omeka_File_MimeType_Exception::$**line**
protected

Omeka_File_MimeType_Exception::**__clone**()

Omeka_File_MimeType_Exception::**__construct**(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

Omeka_File_MimeType_Exception::**__wakeup**()

Omeka_File_MimeType_Exception::**getMessage**()

Omeka_File_MimeType_Exception::**getCode**()

Omeka_File_MimeType_Exception::**getFile**()

Omeka_File_MimeType_Exception::**getLine**()

Omeka_File_MimeType_Exception::**getTrace**()

Omeka_File_MimeType_Exception::**getPrevious**()

Omeka_File_MimeType_Exception::**getTraceAsString**()

Omeka_File_MimeType_Exception::**__toString**()

3.8.7 libraries/Omeka/Filter

3.8.8 libraries/Omeka/Form

libraries/Omeka/Form/Decorator

libraries/Omeka/Form/Element

3.8.9 libraries/Omeka/Http

3.8.10 libraries/Omeka/Job

libraries/Omeka/Job/Dispatcher

libraries/Omeka/Job/Dispatcher/Adapter

libraries/Omeka/Job/Factory

libraries/Omeka/Job/Process

libraries/Omeka/Job/Worker

3.8.11 libraries/Omeka/Navigation

libraries/Omeka/Navigation/Page

libraries/Omeka/Navigation/Page/Uri

3.8.12 libraries/Omeka/Output

libraries/Omeka/Output/OmekaXml

3.8.13 libraries/Omeka/Plugin

libraries/Omeka/Plugin/Broker

libraries/Omeka/Plugin/Installer

libraries/Omeka/Plugin/Loader

3.8.14 libraries/Omeka/Record

libraries/Omeka/Record/Api

libraries/Omeka/Record/Builder

libraries/Omeka/Record/Mixin

3.8.15 libraries/Omeka/Session

libraries/Omeka/Session/SaveHandler

3.8.16 libraries/Omeka/Storage

libraries/Omeka/Storage/Adapter

3.8.17 libraries/Omeka/Test

libraries/Omeka/Test/Helper

libraries/Omeka/Test/Resource

3.8.18 libraries/Omeka/Validate

libraries/Omeka/Validate/File

3.8.19 libraries/Omeka/View

3.9 Omeka REST API

New in version 2.1.

Omeka sites each can expose a REST API for viewing and modifying the site's data.

3.9.1 Clients

See also the *Examples* using the API.

PHP

- ZendService_Omeka by Jim Safley

Javascript

- omeka-client-js by Jim Safley

Python

- omeka-client-py by Jim Safley

Ruby

- Omeka Client by Lincoln Mullen

3.9.2 Configuration

Global Configuration

Super users can configure the API using the Settings > API form.

- Enable/disable the API (default: enabled)
- Adjust the amount of results per page (default: 50)

API Keys

All users with log in credentials can create and rescind API keys using their Edit User form. API keys are tied to and have all the permissions of an individual user. Super users can create keys for other users. Treat keys as you would your password. Omeka provides the client's IP address and the time it was last accessed.

3.9.3 Examples

Adding Items

- omekadd (Python) by Caleb McDaniel. A Python script for adding new items to Omeka using simple YAML documents.

Working With Output

- OmekaApiToCsv (PHP) by Patrick Murray-John. Simple conversion tool for data from Omeka API to CSV file.
- omekacsv (Python) by Caleb McDaniel. Turn JSON from Omeka API items resource into CSV file.

3.9.4 Extending the API

Registering Your Resource

You can extend the API to include custom resources. Most resources are correlated to Omeka records, but you may include non-record resources as well. In your plugin class, register your resource using the *api_resources* filter, following this format:

```
<?php
protected $_filters = array('api_resources');

public function filterApiResources($apiResources)
{
    // For the resource URI: /api/your_resources/[:id]
    $apiResources['your_resources'] = array(
        // Module associated with your resource.
        'module' => 'your-plugin-name',
        // Controller associated with your resource.
        'controller' => 'your-resource-controller',
        // Type of record associated with your resource.
        'record_type' => 'YourResourceRecord',
        // List of actions available for your resource.
        'actions' => array(
            'index',  // GET request without ID
            'get',    // GET request with ID
            'post',   // POST request
            'put',    // PUT request (ID is required)
            'delete', // DELETE request (ID is required)
        ),
        // List of GET parameters available for your index action.
        'index_params' => array('foo', 'bar'),
    );
    return $apiResources;
}
```

If not given, `module` and `controller` fall back to their defaults, "default" and "api". Resources using the default controller MUST include a `record_type`. Remove `actions` that are not wanted or not implemented. For index actions, all GET parameters must be registered with the `index_params` whitelist.

Using the Default Controller

If your resource corresponds to an Omeka record, we highly recommend that you use the default controller. Doing so will ensure consistent behavior across resources. These records must extend `Omeka_Record_AbstractRecord`, implement `Zend_Acl_Resource_Interface`, and have a corresponding API adapter called `Api_{YourRecordType}` that must extend `Omeka_Record_Api_AbstractRecordAdapter` and be saved to your plugin's models directory in a subdirectory named Api. This adapter is responsible for building representations of, and setting properties to, your record.

For example, an API adapter for `YourRecordType` saved to YourPlugin/models/Api/YourRecordType.php:

```
<?php
class Api_YourRecordType extends Omeka_Record_Api_AbstractRecordAdapter
{
    // Get the REST representation of a record.
    public function getRepresentation(Omeka_Record_AbstractRecord $record)
    {
```

(continues on next page)

(continued from previous page)

```
        // Return a PHP array, representing the passed record.
    }

    // Set data to a record during a POST request.
    public function setPostData(Omeka_Record_AbstractRecord $record, $data)
    {
        // Set properties directly to a new record.
    }

    // Set data to a record during a PUT request.
    public function setPutData(Omeka_Record_AbstractRecord $record, $data)
    {
        // Set properties directly to an existing record.
    }
}
```

`Omeka_Record_Api_AbstractRecordAdapter` provides a few convenience methods that make representation and record building easier:

- `getResourceUrl($uri)`: Get the absolute URL to the passed resource. The convention is to provide an absolute URL to all resources in your representations, identified by the "url" key. This follows a hypermedia approach to API design, offering the client an easy way to consume and locate available resources. Note that this is a static method, which means you can use it in your plugin class when extending an existing resource (see "Extending Existing Resources" below).
- `getDate($date)`: Format a date string as an ISO 8601 date, UTC timezone. The convention is to convert all dates to this format. Note that this is a static method.
- `getElementTextRepresentations($record)`: Get representations of element texts belonging to a record. The record must initialize the ElementText mixin.
- `setElementTextData($record, $data)`: Set element text data to a record. The record must initialize the ElementText mixin.

By implementing `Zend_Acl_Resource_Interface`, you record class must include the `getResourceId()` method. This identifies your record as relating to a unique ACL resource ID, which is used during permission checks, often automatically.

```
<?php
public function getResourceId()
{
    // This is typically the name of the plugin, an underscore, and the pluralized␣
↪record type.
    return 'YourPlugin_YourRecords';
}
```

You may find this resource ID already defined in the plugin's `define_acl` hook. If not you'll need to add it yourself:

```
<?php
public function hookDefineAcl($args)
{
    $acl = $args['acl'];
    $acl->addResource('YourPlugin_YourRecords');
}
```

One last thing you may need to do is filter the select object in the record's table class by overriding `Omeka_Db_Table::getSelect()`. This should protect unauthorized API users from viewing non-public records:

```php
<?php
public function getSelect()
{
    $select = parent::getSelect();
    $permissions = new Omeka_Db_Select_PublicPermissions('YourPlugin_YourRecords');
    // Where "your_records" is the table alias, "owner_column" is the user column to␣
↪check against,
    // and "public_column" is the permissions column to check against.
    $permissions->apply($select, 'your_records', 'owner_column', 'public_column');
    return $select;
}
```

Extending Existing Resources

You can extend the representations of existing resources by using the *api_extend_<resource>* filter, where <resource> is the resource you want to extend.

```php
<?php
protected $_filters = array('api_extend_items');

public function filterApiExtendItems($extend, $args)
{
    $item = $args['record'];

    // For one resource:
    $resourceId = $this->_db->getTable('YourResource')->findByItemId($item->id);
    $extend['your_resources'] = array(
        'id' => 1,
        'url' => Omeka_Record_Api_AbstractRecordAdapter::getResourceUrl("/your_
↪resources/{$resourceId->id}"),
        'resource' => 'your_resources',
    );

    // Or, for multiple resources:
    $extend['your_resources'] = array(
        'count' => 10,
        'url' => Omeka_Record_Api_AbstractRecordAdapter::getResourceUrl("/your_
↪resources?item={$item->id}"),
        'resource' => 'your_resources',
    );

    return $extend;
}
```

Note that the API enforces a pattern when extending a resource:

- `id` and `url` for a one-to-one relationship
- `count` and `url` for a one-to-many relationship
- `resource` is recommeded but not required

All other keys pass through as custom data that may be used for the client's convenience.

3.9.5 APIs for beginners

What's a REST API?

Put simply, an API is a formal line of communication between two programs. A REST API is one that utilizes inherent features of the Web to enable communication between a client and a server. It's important to note that APIs aren't designed to be exposed to you, the end user, except through intermediary programs. Remember this when you're experimenting directly with the API; otherwise you may get frustrated.

Omeka's REST API

Omeka's REST API provides an *endpoint* against which you can query, create, update, and delete Omeka *resources*. Your Omeka endpoint is the URL to your Omeka website followed by `/api`, like this:

You can see which resources are available by appending `/resources?pretty_print` to your endpoint, like this:

Here you are getting the `/resources` resource as a JSON object containing all the available resources and other information about them. (`?pretty_print` is helpful to get back nicely formatted JSON, but is not necessary.)

HTTP Request Methods

In the above response you'll notice that all resources have at least one *action*. These actions, while not important to a beginner, correspond to HTTP request methods. These methods are central to REST, and you should know what they do:

- `GET` gets one or more representations of a resource;
- `POST` creates a new resource and returns the representation of the newly created resource;
- `PUT` modifies an existing resource and returns the representation of the newly modified resource;
- `DELETE` deletes an existing resource.

Until you start building your own API client, you won't need to make `POST`, `PUT`, or `DELETE` requests, but `GET` is well suited for beginners because you can run them directly from your Web browser, as you probably did above with `/resources`.

Omeka's Resources and Representations

A *representation*, in this case, is a JSON object that represents an Omeka resource. In other words, you are not getting a resource itself, but rather a representation of it. This is very much like viewing an item on a Omeka website: the item show page is not the item itself, but a representation of the item.

Every Omeka website comes with the following resources, most of which should already be familiar to you as an Omeka user:

- `/collections`
- `/items`
- `/files`
- `/item_types`
- `/elements`
- `/element_sets`
- `/tags`

One of the powerful features of the Omeka API is that plugins can add more resources if they wish. If they do, you'll see them on the `/resources` resource.

Using the Omeka API

Let's take a look at the `/items` resource by making a `GET` request to the following URL using your Web browser:

Assuming there are already items added to Omeka, you'll see a JSON array of item representations, each with an "id" (the item ID), a "url" (the direct URL to that item), and other *metadata* about the item. Copy that URL to your Web browser and make another `GET` request (you'll first need to remove backslashes from the URL, which were a necessary part of JSON formatting):

Where `:id` is the item ID. You'll see the same JSON representation of the item, but by itself. This URL is the item's *canonical* URL, that is, it is an unambiguous and permanent link that represents the item itself.

So far you've only been using the `GET` method to get representations of resources. To experiment with `POST`, `PUT`, and `DELETE` you'll need to use a program that is capable of sending those requests. Most modern browsers have plugins that do this in a user-friendly way:

- Google Chrome
- Mozilla Firefox

If you're comfortable adding, modifying, and deleting Omeka resources, read the API documentation and start making requests!

3.9.6 Requests

Endpoint URL

```
yourdomain.com/api/
```

Pagination

For index requests using the defualt controller, pagination is set in the response's `Link` header:

```
Link: <http://yourdomain.com/api/items?page=1&per_page=50>; rel="first",
      <http://localhost/omeka/api/items?page=10&per_page=50>; rel="last",
      <http://localhost/omeka/api/items?page=1&per_page=50>; rel="previous",
      <http://localhost/omeka/api/items?page=3&per_page=50>; rel="next"``
```

Global GET Parameters

- **key**: string, API key authentication
- **callback**: string, JSONP function name
- **pretty_print**: on/off, Pretty print the JSON output

Generic Requests

`POST` and `PUT` requests are designed to take a JSON payload that is essentially identical to a `GET` response of the same resource. This makes it possible to, for example, `GET` an item, modify the representation directly and `POST` or `PUT` it back to Omeka.

Some servers do not accept `PUT` or `DELETE` requests. For compatibility we've added support for the `X-HTTP-Method-Override` header, which you can use to declare an unsupported HTTP method.

```
X-HTTP-Method-Override: DELETE
```

GET a resource

Return data about the specified resource (most often a specific record):

```
GET /:resources/:id HTTP/1.1
```

GET one or more resources

Return data about resources (most often records):

```
GET /:resources HTTP/1.1
```

- **page**: integer
- **sort_field**: string
- **sort_dir**: string (`'a'` or `'d'`)

A response of a resource using the default controller will include a non-standard `Omeka-Total-Results` header set to the total count of request results.

POST a resource

Create a new resource:

```
POST /:resources HTTP/1.1
```

PUT a resource

Update an existing resource:

```
PUT /:resource/:id HTTP/1.1
```

DELETE a resource

Delete a resource

```
DELETE /:resource/:id HTTP/1.1
```

Errors

Router Errors

All requests may return the following errors:

- `400 Bad Request`

    - Invalid GET request parameter: “[parameter]”
- `403 Forbidden`
    - Invalid key.
    - API is disabled
- `404 Not Found`
    - The “[resource]” resource is unavailable.
- `405 Method Not Allowed`
    - This resource does not implement the “[action]” action.
    - POST requests must not include an ID.
    - PUT and DELETE requests must include an ID.
- `500 Internal Server Error`
    - Resources using the default controller must register a record type.

Default Controller Errors

Requests to the default controller may return the following errors:

- `400 Bad Request`
    - Invalid request. Request body must be a JSON object.
    - Error when saving record.
- `403 Forbidden`
    - Permission denied.
- `404 Not Found`
    - Invalid record. Record not found.
    - Invalid record. Record type “[record_type]” not found.
    - Invalid record adapter. Record adapter “[record_adapter_class]” not found.
- `500 Internal Server Error`
    - Invalid record adapter. Record adapter “[record_adapter_class]” is invalid
    - Invalid record. Record “[record_type]” must define an ACL resource.

Record Errors

Requests that invoke the abstract record adapter may return the following errors:

- `500 Internal Server Error`
    - The “[record_type]” API record adapter does not implement setPostData
    - The “[record_type]” API record adapter does not implement setPutData

3.9.7 API Requests

Collections

GET collection

Return data about the specified collection.

Request

```
GET /collections/:id HTTP/1.1
```

Response


```
{
  "id": 1,
  "url": "http://yourdomain.com/api/collections/1",
  "public": true,
  "featured": false,
  "added": "2013-03-27T08:17:37+00:00",
  "modified": "2013-04-21T15:05:07+00:00",
  "owner": {
    "id": 1,
    "url": "/users/1",
    "resource": "users"
  },
  "items": {
    "count": 100,
    "url": "http://yourdomain.com/api/items?collection=1",
    "resource": "items"
  },
  "element_texts": [
    {
      "html": false,
      "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
      "element_set": {
        "id": 1,
        "url": "http://yourdomain.com/api/element_sets/1",
        "resource": "element_sets",
        "name": "Dublin Core"
      },
      "element": {
        "id": 1,
        "url": "http://yourdomain.com/api/elements/1",
        "resource": "element",
        "name": "Title"
      }
    }
  ]
}
```

GET collections

Return data about collections.

Request

```
GET /collections HTTP/1.1
```

Parameters

- **public**: boolean
- **featured**: boolean
- **added_since**: string (ISO 8601)
- **modified_since**: string (ISO 8601)
- **owner** (user): integer

Response

An array of JSON collection representations (see above).

POST collection

Create a new collection.

Request

```
POST /collections HTTP/1.1
```

```
{
  "public": true,
  "featured": false,
  "element_texts": [
    {
      "html": false,
      "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
      "element": {"id": 1}
    }
  ]
}
```

Response

```
HTTP/1.1 201 Created
Location: http://yourdomain.com/api/collections/:id
```

An JSON representation of the newly created collection (see above).

PUT collection

Edit an existing collection.

Request

```
PUT /collections/:id HTTP/1.1
```

```
{
  "public": true,
  "featured": false,
  "element_texts": [
    {
      "html": false,
      "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
      "element": {"id": 1}
    }
  ]
}
```

Response

An JSON representation of the newly edited collection (see above).

DELETE collection

Delete a collection.

Request

```
DELETE /collections/:id HTTP/1.1
```

Response

```
HTTP/1.1 204 No Content
```

Element Sets

GET element set

Return data about the specified element set:

Request

```
GET /element_sets/:id HTTP/1.1
```

Response

```
{
  "id": 1,
  "url": "http://yourdomain.com/api/element_sets/1",
  "record_type": null,
  "name": "Dublin Core",
  "description": "The Dublin Core metadata element set is common to all Omeka records.
↪",
  "elements": {
    "count": 100,
    "url": "http://yourdomain.com/api/elements?element_set=1",
    "resource": "elements"
  }
}
```

GET element sets

Return data about element sets:

Request

```
GET /element_sets HTTP/1.1
```

Parameters

- **name**: string
- **record_type**: string

Response

An array of JSON element set representations (see above).

POST element set

Create a new element set.

Request

```
POST /element_sets HTTP/1.1
```

```
{
  "name": "Dublin Core",
  "description": "The Dublin Core metadata element set is common to all Omeka records.
↪"
}
```

Response

```
HTTP/1.1 201 Created
Location: http://yourdomain.com/api/element_sets/:id
```

An JSON representation of the newly created element set (see above).

PUT element set

Edit an existing element set.

Request

```
PUT /element_sets/:id HTTP/1.1
```

```
{
  "name": "Dublin Core",
  "description": "The Dublin Core metadata element set is common to all Omeka records.
↪"
}
```

Response

An JSON representation of the newly edited element set (see above).

DELETE element set

Delete an element set. “Dublin Core” and “Item Type Metadata” element sets may not be deleted.

Request

```
DELETE /element_sets/:id
```

Response

```
HTTP/1.1 204 No Content
```

Elements

GET element

Return data about the specified element:

Request

```
GET /elements/:id HTTP/1.1
```

Response

```
{
  "id": 1,
  "url": "http://yourdomain.com/api/elements/1",
  "name": "Text",
  "order": 1,
  "description": "Any textual data included in the document",
  "comment": null,
  "element_set": {
    "id": 1,
    "url": "http://yourdomain.com/api/element_sets/1",
    "resource": "element_sets"
  },
}
```

GET elements

Return data about elements:

Request

```
GET /elements HTTP/1.1
```

Parameters

- **element_set**: integer
- **name**: string
- **item_type**: integer

Response

An array of JSON element representations (see above).

POST element

Create a new element.

Request

```
POST /elements HTTP/1.1
```

```
{
  "order": 1,
  "name": "Foo",
  "description": "Foo description.",
  "comment": "Foo comment.",
  "element_set": {"id": 1}
}
```

Response

```
HTTP/1.1 201 Created
Location: http://yourdomain.com/api/elements/:id
```

An JSON representation of the newly created element (see above).

PUT element

Edit an existing element.

Request

```
PUT /elements/:id HTTP/1.1
```

```
{
  "order": 1,
  "name": "Foo",
  "description": "Foo description.",
  "comment": "Foo comment.",
  "element_set": {"id": 1}
}
```

Response

An JSON representation of the newly edited element (see above).

DELETE element

Delete an element. Only elements belonging to the "Item Type Metadata" element set may be deleted.

Request

```
DELETE /elements/:id HTTP/1.1
```

Response

```
HTTP/1.1 204 No Content
```

Errors

In addition to the general errors, requests to the elements resource my return the following errors:

- `400 Bad Request`
  - Invalid record. Only elements belonging to the "Item Type Metadata" element set may be deleted.

Files

GET file

Return data about the specified file.

Request

```
GET /files/:id HTTP/1.1
```

Response

```
{
  "id": 1,
  "url": "/files/1",
  "file_urls": {
    "original": "http://yourdomain.com/files/original/
↪2b42672de6e47a67698a52e1415bd2c0.pdf",
    "fullsize": "http://yourdomain.com/files/fullsize/
↪2b42672de6e47a67698a52e1415bd2c0.jpg",
    "thumbnail": "http://yourdomain.com/files/thumbnails/
↪2b42672de6e47a67698a52e1415bd2c0.jpg",
    "square_thumbnail": "http://yourdomain.com/files/square_thumbnails/
↪2b42672de6e47a67698a52e1415bd2c0.jpg"
  },
  "item": {
    "id": 1,
    "url": "http://yourdomain.com/api/items/1",
    "resource": "items"
  },
  "order": null,
  "size": 1953540,
```

(continues on next page)

(continued from previous page)


```
  "has_derivative_images": true,
  "authentication": "f6089bca39470e8c19410242061b1f78",
  "mime_type": "audio/mpeg",
  "type_os": "MPEG ADTS, v1, 128 kbps, 44.1 kHz, JntStereo",
  "filename": "54a21c1552feef92069d504fbaf145a8.mp3",
  "original_filename": "1ATwUDzA3SCU.128.mp3",
  "added": "2013-03-27T08:17:37+00:00",
  "modified": "2013-04-21T15:05:07+00:00",
  "stored": true,
  "metadata": {},
  "element_texts": [
    {
      "html": false,
      "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
      "element_set": {
        "id": 1,
        "url": "http://yourdomain.com/api/element_sets/1",
        "name": "Dublin Core",
        "resource": "element_sets"
      },
      "element": {
        "id": 1,
        "url": "http://yourdomain.com/api/elements/1",
        "name": "Title",
        "resource": "elements"
      }
    }
  ]
}
```


Note on "metadata"

The `metadata` object contains data extracted by the getId3 library, and so its structure and content will vary widely based on the file type, what data the library is able to extract, and what data is embedded in the file.

Possible properties within `metadata` (if it is not empty) are: mime_type, video, audio, comments, comments_html, iptc, jpg. `mime_type` within `metadata` is a string, and can differ from the `mime_type` property in the top level of the response. All other properties will be objects.

GET files

Return data about files.

Request

```
GET /files HTTP/1.1
```

Parameters

- **item**: integer

- **order**: integer
- **size_greater_than**: integer
- **has_derivative_image**: boolean
- **mime_type**: string
- **added_since**: string (ISO 8601)
- **modified_since**: string (ISO 8601)

Response

An array of JSON file representations (see above).

POST files

Create a new file. Only one file may be uploaded at a time.

Request

```
POST /files HTTP/1.1
```

Requests must use the multipart/form-data content type. The content disposition name for the file must be `file`. The content disposition name for the JSON data must be `data`. We highly recommend that you use a HTTP client that is capable of encoding form-data requests for you, but the typical request should look something like this:

```
POST /api/files HTTP/1.1
Content-Type: multipart/form-data; boundary=E19zNvXGzXaLvS5C

----E19zNvXGzXaLvS5C
Content-Disposition: form-data; name="data"

{
  "order": 2,
  "item": {
    "id": 1
  },
  "element_texts": [
    {
      "html": false,
      "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
      "element": {"id": 1}
    }
  ]
}
----E19zNvXGzXaLvS5C
Content-Disposition: form-data; name="file"; filename="example.pdf"
Content-Type: application/pdf

...contents of example.pdf...
----E19zNvXGzXaLvS5C
```

Response

```
HTTP/1.1 201 Created
Location: http://yourdomain.com/api/files/:id
```

An JSON representation of the newly created file (see above).

PUT file

Edit an existing file.

Request

```
PUT /files/:id HTTP/1.1
```

```
{
  "order": 2,
  "element_texts": [
    {
      "html": false,
      "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
      "element": {"id": 1}
    }
  ]
}
```

Response

An JSON representation of the newly edited file (see above).

DELETE file

Delete a file.

Request

```
DELETE /files/:id HTTP/1.1
```

Response

```
HTTP/1.1 204 No Content
```

Errors

In addition to the general errors, requests to the files resource may return the following errors:

- `400 Bad Request`
  - Invalid request. Request body must be a JSON object.
  - Invalid request. Exactly one file must be uploaded per request.
  - Invalid request. Missing JSON data.
  - Invalid item. File must belong to an existing item.
- `500 Internal Server Error`
  - Invalid request. The default job dispatcher must be synchronous.

Item Types

GET item type

Return data about the specified item type.

Request

```
GET /item_types/:id HTTP/1.1
```

Response

```
{
  "id": 1,
  "url": "/item_types/1",
  "name": "Text",
  "description": "A resource consisting primarily of words for reading.",
  "elements": [
    {
      "id": 1,
      "url": "http://yourdomain.com/api/elements/1",
      "resource": "elements"
    },
    {
      "id": 2,
      "url": "http://yourdomain.com/api/elements/2",
      "resource": "elements"
    },
    {
      "id": 3,
      "url": "http://yourdomain.com/api/elements/3",
      "resource": "elements"
    }
  ],
  "items": {
    "count": 100,
```

(continues on next page)

(continued from previous page)

```
    "url": "http://yourdomain.com/api/items?item_type=1",
    "resource": "items"
  }
}
```

GET item types

Return data about item types.

Request

```
GET /item_types HTTP/1.1
```

Parameters

- **name**: string

Response

An array of JSON item type representations (see above).

POST item type

Create a new item type.

Request

```
POST /item_types HTTP/1.1
```

```
{
  "name": "Text",
  "description": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
  "elements": [
    {"id": 1},
    {"id": 2},
    {"id": 3}
  ]
}
```

Response

```
HTTP/1.1 201 Created
Location: http://yourdomain.com/api/item_types/:id
```

An JSON representation of the newly created item type (see above).

PUT item type

Edit an existing item type.

Request

```
PUT /item_types/:id HTTP/1.1
```

```
{
  "name": "Text",
  "description": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
  "elements": [
    {"id": 1},
    {"id": 2},
    {"id": 3}
  ]
}
```

Response

An JSON representation of the newly edited item type (see above).

DELETE item type

Delete an item type.

Request

```
DELETE /item_types/:id HTTP/1.1
```

Response

```
HTTP/1.1 204 No Content
```

Items

GET item

Return data about the specified item.

Request

```
GET /items/:id HTTP/1.1
```

Response


```
{
  "id": 1,
  "url": "http://yourdomain.com/api/items/1",
  "item_type": {
    "id": 1,
    "url": "http://yourdomain.com/api/item_types/1",
    "name": "Text",
    "resource": "item_types"
  },
  "collection": {
    "id": 1,
    "url": "http://yourdomain.com/api/collections/1",
    "resource": "collections"
  },
  "owner": {
    "id": 1,
    "url": "http://yourdomain.com/api/users/1",
    "resource": "users"
  },
  "public": true,
  "featured": false,
  "added": "2013-03-27T08:17:37+00:00",
  "modified": "2013-04-21T15:05:07+00:00",
  "files": {
    "count": 100,
    "url": "http://yourdomain.com/api/files?item=1",
    "resource": "files"
  },
  "tags": [
    {
      "id": 1,
      "url": "http://yourdomain.com/api/tags/1",
      "name": "foo",
      "resource": "tags"
    }
  ],
  "element_texts": [
    {
      "html": false,
      "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
      "element_set": {
        "id": 1,
        "url": "http://yourdomain.com/api/element_sets/1",
        "name": "Dublin Core",
        "resource": "element_sets"
      },
      "element": {
        "id": 1,
        "url": "http://yourdomain.com/api/elements/1",
        "name": "Title",
        "resource": "elements"
      }
    }
  ]
}
```

GET items

Return data about items.

Request

```
GET /items HTTP/1.1
```

Parameters

- **collection**: integer
- **item_type**: integer
- **featured**: boolean
- **public**: boolean
- **added_since**: string (ISO 8601)
- **modified_since**: string (ISO 8601)
- **owner** (user): integer
- **tags**: string (since 2.3.1)
- **excludeTags**: string (since 2.3.1)
- **hasImage**: boolean (since 2.3.1)
- **range**: string (since 2.3.1)
- **search** (simple search): string (since 2.5)

Response

An array of JSON item representations (see above).

POST item

Create a new item.

Request

```
POST /items HTTP/1.1
```

```
{
  "item_type": {"id": 1},
  "collection": {"id": 1},
  "public": true,
  "featured": false,
  "tags": [
    {"name": "foo"},
```

(continues on next page)

(continued from previous page)

```
    {"name": "bar"}
  ],
  "element_texts": [
    {
      "html": false,
      "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
      "element": {"id": 1}
    }
  ]
}
```

Response

```
HTTP/1.1 201 Created
Location: http://yourdomain.com/api/items/:id
```

An JSON representation of the newly created item (see above).

PUT item

Edit an existing item.

Request

```
PUT /items/:id HTTP/1.1
```

```
{
  "item_type": {"id": 1},
  "collection": {"id": 1},
  "public": true,
  "featured": false,
  "tags": [
    {"name": "foo"},
    {"name": "bar"}
  ],
  "element_texts": [
    {
      "html": false,
      "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
      "element": {"id": 1}
    }
  ]
}
```

Response

An JSON representation of the newly edited item (see above).

DELETE item

Delete an item.

Request

```
DELETE /items/:id HTTP/1.1
```

Response

```
HTTP/1.1 204 No Content
```

Resources

GET resource

Individual resources cannot be shown.

GET resources

Return available API resources and information about them.

Request

```
GET /resources HTTP/1.1
```

Response

```
{
  "resources": {
    "controller": "resources",
    "actions": ["get"]
  },
  "items": {
    "record_type": "Item",
    "actions": ["index", "get", "post", "put", "delete"],
    "index_params": ["collection", "item_type", "featured", "public", "added_since",
↪"modified_since", "owner"]
  }
}
```

POST resource

Resources cannot be created. They must be registered via the `api_resources` filter.

Site

GET site

Return information about the Omeka installation.

Request

```
GET /site HTTP/1.1
```

Response

```
{
  "omeka_url":"http://yourdomain.com",
  "omeka_version":"2.1",
  "title":"My Omeka Site",
  "description":"Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
  "author":"Center for History and New Media",
  "copyright":"Creative Commons Attribution-ShareAlike 3.0 License"
}
```

Tags

GET tag

Return data about the specified tag.

Request

```
GET /tag HTTP/1.1
```

Response

```
{
  "id": 1,
  "url": "http://yourdomain.com/api/tags/1",
  "name": "foo"
}
```

GET tags

Return data about tags.

Request

```
GET /tags HTTP/1.1
```

Response

An array of JSON tag representations (see above).

POST tag

New tags must be created via the owner record.

DELETE tag

Delete a tag.

Request

```
DELETE /tags/:id HTTP/1.1
```

Response

```
HTTP/1.1 204 No Content
```

Users

GET user

Return data about the specified user.

Request

```
GET /users/:id HTTP/1.1
```

Response

```
{
  "id": 1,
  "url": "/users/1",
  "username": "janedoe",
  "name": "Jane Doe",
  "email": "janedoe@example.com",
```

(continues on next page)

(continued from previous page)

```
  "active": true,
  "role": "super"
}
```

GET users

Users cannot be browsed.

POST user

Users cannot be created.

CHAPTER 4

Helping With Documentation

We always appreciate help correcting and filling out stubs in our documentation. You can submit contribute and make pull requests by clicking the "Edit on GitHub" link on any of the documentation pages. Check the README on the documentation GitHub repo for details.

From our forums and a developer survey, we know that the biggest thing that will help the community of Omeka developers is example code, especially code that fills out the documentation for Global (Theming) Functions. Those files are located under `libraries/globals/examples`. Each file there corresponds to an Omeka global function that has automatically generated documentation.

4.1 Function Examples

If you would like to contribute how you have used or modified one of the Omeka globals functions, you can use the exampletemplate file as a starting point. Also, look at the Metadata file as a working sample. Just branch the corresponding file within GitHub (or create your own local copy), fill in the details of what you did, and submit a new pull request.

4.2 Recipe Examples

A second, broader, need is for recipes that fill out larger tasks. A recipe might combine several functions or plugin structures to solve a large task. These are found under `Tutorials/recipes`. New ones can also be added via pull request. There is also a recipetemplate for these, but the structure you use can follow whatever structures explain the process best. Here is an example for Retaining Search Order.

4.3 Docathon March 2017

Thanks to the CUNY Graduate Center, Omeka will be one of the projects tackled in Docathon the week of March 6, 2017.

You can find some guides to technical aspects above and in the Documentation README.

Here are a couple thoughts on quick and easy ways to help us all get the most out of Docathon.

Have you hacked or built an Omeka theme?

You've probably needed to modify the default use of one or more of Omeka's global functions. (Or maybe you just had to figure out how it works!) If so, share your example of what you did with the function so others can make more of the possibilities.

You might have rearranged pages, or incorporated other libraries to get your display just right. Those would make excellent recipes to show how your solved a particular problem – others will want to learn from it for similar needs.

Have you hacked or built an Omeka plugin?

Some of the best Omeka plugins are the smallest ones that take care of a small need. Or, an existing plugin might have been _close_ to what you needed, but called for some modification. Sharing what you did as a recipe is a great way to reinforce what you learned and help others in similar situations.

Do you just want to learn a little about coding (for Omeka)?

No time like the present. Tinkering with existing themes to make small changes to the display is a great way to get started. You'll learn your way around themes, see how the functions work, learn a little more about PHP, and help your fellow Omekans!

Start with an existing theme, and try your hand at modifying the output from one of the functions. Browse through the basic documentation for global functions, then find where they are used in an existing theme. Then, experiment with some of the options listed in the documentation, but don't have an example of the variety of options described. Then, share your example back for others to see and learn from.

4.3.1 Submitting Additions and Changes

A pull request is the best way to contribute. We'll give a quick look and provide feedback as needed. We want to help you get your contribution available for everyone working with Omeka as quickly as possible. You can do that either from a fork of the Documentation you work on locally, or directly from GitHub. We'll be watching over the Docathon and respond as quickly as we can.

4.3.2 Talking About Additions and Changes

For simplicity, we've created a couple of issues on the Documentation repo that reflect the guides above. Adding comments there will be the quickest way to ask us a question. We'll be watching, and will respond as quickly as we can.

Hannah has also created a #cunygc channel in the Docathon's Slack. That'll also be a good place to ask questions.

CHAPTER 5

Registering a new module or theme with omeka.org

After you register your GitHub repository as an Omeka addon we'll automatically identify and publish new versions of your addon if you follow these steps.

First, in your addon's GitHub repository, go to the releases page and click "Draft a new release". Enter a tag name (it does not have to be your addon version) and target branch. We recommend entering a release title (e.g. addon version) and description (e.g. release notes, changelog, etc.), but it's not required.

Then, on your own computer, create a ZIP archive of the packaged, ready-for-release addon; we recommend using Git's `archive` command:

```
$ git archive --output={AddonZipName}.zip --prefix={AddonDirName}/ {tagname}
```

Note that the `{AddonDirName}` must match the directory name you initially registered. The `{AddonZipName}` can be anything, but we recommend that it's the same as `{AddonDirName}`. Afterwards, attach this ZIP file to the release as a binary.

Then, to complete the process, click "Publish release". If everything checks out, we'll register the release shortly. After we have added it to the registry, updates following the same procedure will be automatically updated. If you subsequently set the release to be a prerelease or draft, we will remove the release from our registry.

Note that we pull in your GitHub repository's README and publish it alongside your addon versions. Make sure the README is ready for publication.

Before registring your releases we check that certain things are true:

- Your repository has releases
- The release is not a draft
- The release is not a prerelease
- The release has an attached binary (asset)
- The release/asset combination is not already registered
- The asset has the .zip extension
- The asset is a ZIP file

- The ZIP file contains only one top-level directory
- The ZIP file top-level directory is named the same as the one you registered
- The ZIP file contains a .jpg thumbnail (themes only)
- The ZIP file contains an INI file at the correct path:
  - for plugins: /plugin.ini
  - for themes: /theme.ini
- The INI file has no parsing errors
- The INI file has an `[info]` section
- The INI file has a `version`

If all these things are true we register and publish the release. Keep in mind that we do not check if the addon works as expected. It is your responsibility to set your release to a tag or commit that's in working condition.

CHAPTER 6

Indices and tables

- genindex
- search

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X