Recite CMS Page Render Hook Driver Development Guide

Recite CMS 2.0.5

Recite CMS Page Render Hook Driver Development Guide

Table of Contents

List of Examples

Chapter 1. Introduction

When a web site visitor requests a page on your Recite-powered web site, there is a sequence of bootstrapping events that takes place in order to fulfill that request. With Recite page render hooks, you can write custom functionality that is executed at specified points within this bootstrap process.

This document shows you how to write your own rendering hooks. It will instruct you on the different methods that can be implemented to take control of the boostrapping process.

To get started you will need a text editor that you can write PHP code with. Additionally, you will need a working copy of Recite CMS 2.0.0 (or newer), as well as administration access.

Chapter 2. Getting Started

Each page render hook is a Recite driver that lives in the `./lib/drivers/pages/renderhooks` directory.

Creating Your Driver File

The driver file is a PHP class with the filename `driver.php` which extends from the `Module_Pages_RenderHook_Abstract` class.

The class in this file must follow standard Recite driver naming conventions. For example, if your driver is called **publishrules**, your `driver.php` file would have the path `./lib/drivers/pages/renderhooks/`*`publishrules`*`/driver.php`.

In this example, the file would define a single class called `Driver_pages_renderhooks_`*`publishrules`*`_driver`

The only method you must define is the `__toString()`, which must return a brief description of what the render hook driver does.

Note

There are other methods you can also define which we'll cover shortly; if you only define the `__toString()` your driver won't do anything.

Example 2.1. A sample rendering hook (`driver.php`)

```php
<?php
    class Driver_pages_renderhooks_publishrules_driver extends Module_Pages_RenderHook_Abstract
    {
        public function __toString()
        {
            return 'Enforce page publishing rules';
        }
    }
?>
```

Assigning the Driver to Clients

Once your driver has been created you must assign it to each client you want to use it. This is done using the Recite administration section.

Caution

Once you assign the render hook to a client it takes instant effect. There are no controllable options in the Control Panel to enable or disable the driver.

Controlling the Order of Execution

It is possible to have any number of render hooks assigned to a client. At this stage it is not possible to control the order of execution. You must assume the hooks are executed in a random order.

Having said that, for a single request the hooks will be executed in the same order for every hook method.

Chapter 3. Triggering Errors

In several of the methods described in this document it is possible to trigger an error.

For example, if you were implementing the "page publish rules" driver used as an example earlier in this book, you might want to trigger a "404 Not Found" error if a user tries to access a page that is no longer published.

You can trigger an error simply by throwing an exception. You can throw either the standard PHP `Exception`, or you can throw any custom exceptions you may have previously defined.

In any case, the desired HTTP error should be set in the exception. This is achieved with the `Exception` by specifying it as the second argument to the constructor. For example, to trigger a "404 Not Found" error you would use code such as the following: `throw new Exception('Page not found', 404)`.

Recite will use the built-in error-handling mechanism for handling thrown exceptions. For instance, if you throw a 404 error then Recite will render the error template as chosen by the site administrator in the Recite Control Panel.

Note

If the code thrown in the exception is not a valid HTTP code then Recite will send a 500 code with the response.

Each method that supports exceptions has it noted in its description in this document.

Caution

In some cases when you throw an exception from one of the specified methods the `handleBootException()` method in your driver (and all other render hooks) will subsequently be executed.

Conversely, if another render hook triggers an exception then this will also result in your `handleBootException()` method being called.

Chapter 4. Building the Request Tree

When a page is requested on a Recite web site, the first stage of the bootstrapping process is to read the URL and determine exactly which content the requesting user is after. We refer to this as *building the page request tree*.

This chapter describes the methods that are called surrounding the building of the page request tree.

preBuildRequestTree

- `public function preBuildRequestTree()`
- Throws `Exception`

This method is called prior to Recite trying to read the requested URL and determine which page that user is loading.

As such, when this method is called you have no idea which page is being requested.

postBuildRequestTree

- `public function postBuildRequestTree(Module_Pages_ClientRenderer_PageRequestTree $pageRequestTree)`
- Throws `Exception`

This method is called after the page request tree has been built. At this stage you know which page has been requested, and you can retrieve details about the requested page in the `$pageRequestTree` variable passed to this method.

Note

If Recite was unable to build the page request tree (typically due to the user requesting an invalid URL), then this method is never executed, since Recite will bypass straight to the built-in error handler.

Instead, the `handleBootException()` method of each render hook will be called. This allows you to still process invalid requests if required.

Chapter 5. Checking Permissions

Once the page request tree has been built, the next stage in the bootstrapping process is to check permissions associated with the loaded page.

preAssertPermissions

- `public function preAssertPermissions()`
- Throws `Exception`

This method is called prior to the permissions being checked.

postAssertPermissions

- `public function postAssertPermissions()`
- Throws `Exception`

This method is called after permissions have been checked. If the permissions check failed (that is, the user is not allowed to access the page), then this method is not called. Instead, the `handleBootException()` method will be called with an error code of 401 (the HTTP code for "Unauthorized").

Chapter 6. Page Rendering

preRendererRun

- `public function preRendererRun()`

This method is run prior to the page being rendered.

outputFilter

- `public function outputFilter($output)`
- Returns `string`

This method is executed during the rendering process after the page content has been generated but not yet set in the HTTP repsonse.

This allows you modify the output of a page. For example, Recite comes with a render hook called "Powered by Recite". This hook modifies the output for each page using this method to insert a HTML meta tag indicating that Recite is being used.

postRendererRun

- `public function postRendererRun()`

This method is called after the rendering process is complete. This allows for any final processing required.

Chapter 7. Other Methods

In addition to the interface methods already outlined in this document, there are several other methods you can implement which control how pages are rendered.

handleBootException

- `public function handleBootException(Exception $ex)`
- Throws `Exception`

This method is called when an error occurs in the bootstrapping process. This could be triggered by a file not found error; a permissions error; or some other error that has resulted from another render hook.

After this method has been called Recite will then execute its built-in error handler.

Note

You can throw a different from this method if required, but doing so may result in other render hooks not being able to render their own error handler.

getCacheOptions

- `public function getCacheOptions()`
- Returns `Application_Cache_Options`

This method is used to control caching options for pages that run through you render hook. For instance, if you don't want any pages to be cached then you can control that by setting the appropriate option in the return object.

Recite gathers the cache options of all render hooks and uses the "least" of these. For example, if one render hook allows caching for 2 hours and another allows for caching of only 1 hour, then the lesser of these is used (1 hour).

Note

If a page is cached then when a page is subsequently re-requested and served from page, then the bootstrapping process is completely skipped. That is, your render hook script(s) won't be executed for a cached page.