

The symfony Reference Guide

symfony 1.2

Fabien Potencier

The symfony Reference Guide | symfony 1.2 | version *reference-1.2-en-2009-05-25*

© 2009 Fabien Potencier

ISBN-13: 978-2-918390-05-3

Editor: Fabien Potencier

Proofreader: Kris Wallsmith

Cover Design: Franck Bodiot

Indexer: Fabien Potencier

Icons: DocBook XSL stylesheets

Sensio SA

92-98, boulevard Victor Hugo

92 115 Clichy

France

info@sensio.com

This work is licensed under the “Attribution-Share Alike 3.0 Unported” license (<http://creativecommons.org/licenses/by-sa/3.0/>).

You are free **to share** (to copy, distribute and transmit the work), and **to remix** (to adapt the work) under the following conditions:

- **Attribution:** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **Share Alike:** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license. For any reuse or distribution, you must make clear to others the license terms of this work.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Sensio shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

If you find typos or errors, feel free to report them by creating a ticket on the symfony ticketing system (<http://trac.symfony-project.org/register>). Based on tickets and users feedback, this book is continuously updated.

You can contact the author about this book, symfony and Open-Source at fabien.potencier@symfony-project.com or for training, consulting, application development, or business related questions at fabien.potencier@sensio.com.

Events

The symfony core components are decoupled thanks to an `SfEventDispatcher` object. The event dispatcher manages the communication between core components.

Any object can notify an event to the dispatcher, and any other object can connect to the dispatcher to listen to a specific event.

An event is just a name composed of a namespace and a name separated by a dot (.).

Usage

You can notify an event by first creating an event object:

```
$event = new SfEvent($this, 'user.change_culture', array('culture' => $culture));
```

*Listing
15-1*

And notify it:

```
$dispatcher->notify($event);
```

*Listing
15-2*

The `SfEvent` constructor takes three arguments:

- The “subject” of the event (most of the time, this is the object notifying the event, but it can also be null)
- The event name
- An array of parameters to pass to the listeners

To listen for an event, connect to that event name:

```
$dispatcher->connect('user.change_culture', array($this, 'listenToChangeCultureEvent'));
```

*Listing
15-3*

The connect method takes two arguments:

- The event name
- A PHP callable to call when the event is notified

Here is an implementation example of a listener:

```
Listing public function listenToChangeCultureEvent(sfEvent $event)
15-4 {
    // change the message format object with the new culture
    $this->setCulture($event['culture']);
}
```

The listener gets the event as the first argument. The event object has several methods to get event information:

- `getSubject()`: Gets the subject object attached to the event
- `getParameters()`: Returns the event parameters

The event object can also be accessed as an array to get its parameters.

Event Types

Events can be triggered by three different methods:

- `notify()`
- `notifyUntil()`
- `filter()`

notify

The `notify()` method notifies all listeners. The listeners cannot return a value and all listeners are guaranteed to be executed.

notifyUntil

The `notifyUntil()` method notifies all listeners until one stops the chain by returning a true value.

The listener that stops the chain may also call the `setReturnValue()` method.

The notifier can check if a listener has processed the event by calling the `isProcessed()` method:

```
Listing if ($event->isProcessed())
15-5 {
```

```
// ...  
}
```

filter

The `filter()` method notifies all listeners that they can filter the given value, passed as a second argument by the notifier, and retrieved by the listener callable as the second argument. All listeners are passed the value and they must return the filtered value. All listeners are guaranteed to be executed.

The notifier can get the filtered value by calling the `getReturnValue()` method:

```
$ret = $event->getReturnValue();
```

*Listing
15-6*

Events

- application (*page 118*)
 - application.log (*page 118*)
- command (*page 118*)
 - command.log (*page 118*)
 - command.pre_command (*page 119*)
 - command.post_command (*page 119*)
 - command.filter_options (*page 119*)
- configuration (*page 119*)
 - configuration.method_not_found (*page 119*)
- component (*page 120*)
 - component.method_not_found (*page 120*)
- context (*page 120*)
 - context.load_factories (*page 120*)
- controller (*page 121*)
 - controller.change_action (*page 121*)
 - controller.method_not_found (*page 121*)
 - controller.page_not_found (*page 121*)
- plugin (*page 122*)
 - plugin.pre_install (*page 122*)
 - plugin.post_install (*page 122*)
 - plugin.pre_uninstall (*page 122*)
 - plugin.post_uninstall (*page 123*)
- request (*page 123*)
 - request.filter_parameters (*page 123*)
 - request.method_not_found (*page 123*)
- response (*page 124*)
 - response.method_not_found (*page 124*)
 - response.filter_content (*page 124*)

- routing (*page 124*)
 - routing.load_configuration (*page 124*)
- task (*page 124*)
 - task.cache.clear (*page 124*)
- template (*page 125*)
 - template.filter_parameters (*page 125*)
- user (*page 125*)
 - user.change_culture (*page 125*)
 - user.method_not_found (*page 125*)
 - user.change_authentication (*page 126*)
- view (*page 126*)
 - view.configure_format (*page 126*)
 - view.method_not_found (*page 127*)
- view.cache (*page 127*)
 - view.cache.filter_content (*page 127*)

application

application.log

Notify method: notify

Default notifiers: lot of classes

Parameter	Description
-----------	-------------

priority	The priority level (sfLogger::EMERG, sfLogger::ALERT, sfLogger::CRIT, sfLogger::ERR, sfLogger::WARNING, sfLogger::NOTICE, sfLogger::INFO, or sfLogger::DEBUG)
----------	---

The `application.log` event is the mechanism used by symfony to do the logging for web request (see the logger factory). The event is notified by most symfony core components.

application.throw_exception

Notify method: notifyUntil

Default notifiers: sfException

The `application.throw_exception` event is notified when an uncaught exception is thrown during the handling of a request.

You can listen to this event to do something special whenever an uncaught exception is thrown(like sending an email, or logging the error). You can also override the default exception management mechanism of symfony by processing the event.

command

command.log

Notify method: notify

Default notifiers: sfCommand* classes

Parameter	Description
-----------	-------------

priority	The priority level (sfLogger::EMERG, sfLogger::ALERT, sfLogger::CRIT, sfLogger::ERR, sfLogger::WARNING, sfLogger::NOTICE, sfLogger::INFO, or sfLogger::DEBUG)
----------	---

The `command.log` event is the mechanism used by symfony to do the logging for the symfony CLI utility (see the logger factory).

`command.pre_command`

Notify method: `notifyUntil`

Default notifiers: `sfTask`

Parameter	Description
------------------	--------------------

<code>arguments</code>	An array of arguments passed on the CLI
------------------------	---

<code>options</code>	An array of options passed on the CLI
----------------------	---------------------------------------

The `command.pre_command` event is notified just before a task is executed.

`command.post_command`

Notify method: `notify`

Default notifiers: `sfTask`

The `command.post_command` event is notified just after a task is executed.

`command.filter_options`

Notify method: `filter`

Default notifiers: `sfTask`

Parameter	Description
------------------	--------------------

<code>command_manager</code>	The <code>sfCommandManager</code> instance
------------------------------	--

The `command.filter_options` event is notified before the task CLI options are parsed. This event can be used to filter the options passed by the user.

configuration

`configuration.method_not_found`

Notify method: `notifyUntil`

Default notifiers: `sfProjectConfiguration`

Parameter Description

method	The name of the called missing method
arguments	The arguments passed to the method

The `configuration.method_not_found` event is notified when a method does not defined in the `sfProjectConfiguration` class. By listening to this event, a method can be added to the class, without using inheritance.

component

`component.method_not_found`

Notify method: `notifyUntil`

Default notifiers: `sfComponent`

Parameter Description

method	The name of the called missing method
arguments	The arguments passed to the method

The `component.method_not_found` event is notified when a method does not defined in the `sfComponent` class. By listening to this event, a method can be added to the class, without using inheritance.

context

`context.load_factories`

Notify method: `notify`

Default notifiers: `sfContext`

The `context.load_factories` event is notified once per request by the `sfContext` object just after all factories have been initialized. This is the first event to be notified with all core classes initialized.

controller

controller.change_action

Notify method: notify

Default notifiers: sfController

Parameter	Description
-----------	-------------

module	The module name to be executed
action	The action name to be executed

The controller.change_action is notified just before an action is executed.

controller.method_not_found

Notify method: notifyUntil

Default notifiers: sfController

Parameter	Description
-----------	-------------

method	The name of the called missing method
arguments	The arguments passed to the method

The controller.method_not_found event is notified when a method does not defined in the sfController class. By listening to this event, a method can be added to the class, without using inheritance.

controller.page_not_found

Notify method: notify

Default notifiers: sfController

Parameter	Description
-----------	-------------

module	The module name that generated the 404 error
action	The action name that generated the 404 error

The controller.page_not_found is notified whenever a 404 error is generated during the handling of a request.

You can listen to this event to do something special whenever a 404 page occurs, like sending an email, or logging the error. the event.

plugin

plugin.pre_install

Notify method: notify

Default notifiers: sfPluginManager

Parameter	Description
channel	The plugin channel
plugin	The plugin name
is_package	Whether the plugin to install is a local package (true), or a web package (false)

The `plugin.pre_install` event is notified just before a plugin will be installed.

plugin.post_install

Notify method: notify

Default notifiers: sfPluginManager

Parameter	Description
channel	The plugin channel
plugin	The plugin name

The `plugin.post_install` event is notified just after a plugin has been installed.

plugin.pre_uninstall

Notify method: notify

Default notifiers: sfPluginManager

Parameter	Description
channel	The plugin channel
plugin	The plugin name

The `plugin.pre_uninstall` event is notified just before a plugin will be uninstalled.

`plugin.post_uninstall`

Notify method: `notify`

Default notifiers: `sfPluginManager`

Parameter	Description
<code>channel</code>	The plugin channel
<code>plugin</code>	The plugin name

The `plugin.post_uninstall` event is notified just after a plugin has been uninstalled.

`request`

`request.filter_parameters`

Notify method: `filter`

Default notifiers: `sfWebRequest`

Parameter	Description
<code>path_info</code>	The request path

The `request.filter_parameters` event is notified when the request parameters are initialized.

`request.method_not_found`

Notify method: `notifyUntil`

Default notifiers: `sfRequest`

Parameter	Description
<code>method</code>	The name of the called missing method
<code>arguments</code>	The arguments passed to the method

The `request.method_not_found` event is notified when a method does not defined in the `sfRequest` class. By listening to this event, a method can be added to the class, without using inheritance.

response

`response.method_not_found`

Notify method: `notifyUntil`

Default notifiers: `sfResponse`

Parameter	Description
<code>method</code>	The name of the called missing method
<code>arguments</code>	The arguments passed to the method

The `response.method_not_found` event is notified when a method does not defined in the `sfResponse` class. By listening to this event, a method can be added to the class, without using inheritance.

`response.filter_content`

Notify method: `filter`

Default notifiers: `sfResponse`

The `response.filter_content` event is notified before a response is sent. By listening to this event, you can manipulate the content of the response before it is sent.

routing

`routing.load_configuration`

Notify method: `notify`

Default notifiers: `sfRouting`

The `routing.load_configuration` event is notified when the routing factory loads the routing configuration.

task

`task.cache.clear`

Notify method: `notifyUntil`

Default notifiers: sfCacheClearTask

Parameter	Description
app	The application name
type	The type of cache (all, config, i18n, routing, module, and template)
env	The environment

The `task.cache.clear` event is notified whenever the user clears the cache from the CLI with the `cache:clear` task.

template

template.filter_parameters

Notify method: filter

Default notifiers: sfViewParameterHolder

The `template.filter_parameters` event is notified before a view file is rendered. By listening to this event you can access and manipulate variables passed to a template.

user

user.change_culture

Notify method: notify

Default notifiers: sfUser

Parameter	Description
culture	The user culture

The `user.change_culture` event is notified when the user culture is changed during a request.

user.method_not_found

Notify method: notifyUntil

Default notifiers: sfUser

Parameter Description

<code>method</code>	The name of the called missing method
<code>arguments</code>	The arguments passed to the method

The `user.method_not_found` event is notified when a method does not defined in the `sfUser` class. By listening to this event, a method can be added to the class, without using inheritance.

`user.change_authentication`

Notify method: `notify`

Default notifiers: `sfBasicSecurityUser`

Parameter Description

<code>authenticated</code>	Whether the user is authenticated or not
----------------------------	--

The `user.change_authentication` event is notified whenever the user authentication status changes.

view

`view.configure_format`

Notify method: `notify`

Default notifiers: `sfView`

Parameter Description

<code>format</code>	The requested format
<code>response</code>	The response object
<code>request</code>	The request object

The `view.configure_format` event is notified by the view when the request has the `sf_format` parameter set. The event is notified after symfony has done simple things like changing setting or unsetting the layout. This event allows the view and the response object to be changed according to the requested format.

view.method_not_found

Notify method: notifyUntil

Default notifiers: sfView

Parameter	Description
method	The name of the called missing method
arguments	The arguments passed to the method

The `view.method_not_found` event is notified when a method is not defined in the `sfView` class. By listening to this event, a method can be added to the class, without using inheritance.

view.cache

view.cache.filter_content

Notify method: filter

Default notifiers: sfViewCacheManager

Parameter	Description
response	The response object
uri	The URI of the cached content
new	Whether the content is new in cache or not

The `view.cache.filter_content` event is notified whenever a content is retrieved from the cache.

