

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.

The factories.yml Configuration File

Factories are core objects needed by the framework during the life of any request. They are configured in the `factories.yml` configuration file and always accessible via the `sfContext` object:

```
// get the user factory
sfContext::getInstance()->getUser();
```

*Listing
5-1*

The main `factories.yml` configuration file for an application can be found in the `apps/APP_NAME/config/` directory.

As discussed in the introduction, the `factories.yml` file is **environment-aware** (page 30), benefits from the **configuration cascade mechanism** (page 30), and can include **constants** (page 28).

The `factories.yml` configuration file contains a list of named factories:

```
FACTORY_1:
  # definition of factory 1
```

*Listing
5-2*

```
FACTORY_2:
  # definition of factory 2
```

```
# ...
```

The supported factory names are: `controller`, `logger`, `i18n`, `request`, `response`, `routing`, `storage`, `user`, `view_cache`, and `view_cache_manager`.

When the `sfContext` initializes the factories, it reads the `factories.yml` file for the class name of the factory (`class`) and the parameters (`param`) used to configure the factory object:

```
FACTORY_NAME:
  class: CLASS_NAME
  param: { ARRAY OF PARAMETERS }
```

*Listing
5-3*

Being able to customize the factories means that you can use a custom class for symfony core objects instead of the default one. You can also change the default behavior of these classes by customizing the parameters sent to them.

If the factory class cannot be autoloaded, a file path can be defined and will be automatically included before the factory is created:

Listing
5-4

```
FACTORY_NAME:  
  class: CLASS_NAME  
  file:  ABSOLUTE_PATH_TO_FILE
```



The `factories.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfFactoryConfigHandler` class (*page 111*).

Factories

- request (page 47)
 - formats (page 47)
 - path_info_array (page 47)
 - path_info_key (page 47)
 - relative_url_root (page 48)
- response (page 48)
 - charset (page 48)
 - http_protocol (page 48)
 - send_http_headers (page 48)
- user (page 49)
 - default_culture (page 49)
 - timeout (page 49)
 - use_flash (page 49)
- storage (page 50)
 - auto_start (page 50)
 - database (page 51)
 - db_table (page 51)
 - db_id_col (page 51)
 - db_data_col (page 51)
 - db_time_col (page 51)
 - session_cache_limiter (page 51)
 - session_cookie_domain (page 50)
 - session_cookie_httponly (page 50)
 - session_cookie_lifetime (page 50)
 - session_cookie_path (page 50)
 - session_cookie_secure (page 50)
 - session_name (page 50)
- view_cache_manager (page 51)
- view_cache (page 52)
- i18n (page 52)
 - cache (page 53)
 - debug (page 53)
 - source (page 53)
 - untranslated_prefix (page 53)

- `untranslated_suffix` (*page 53*)
- `routing` (*page 53*)
 - `cache` (*page 53*)
 - `extra_parameters_as_query_string` (*page 54*)
 - `generate_shortest_url` (*page 54*)
 - `lazy_routes_deserialize` (*page 55*)
 - `lookup_cache_dedicated_keys` (*page 55*)
 - `load_configuration` (*page 55*)
 - `segment_separators` (*page 54*)
 - `suffix` (*page 55*)
 - `variable_prefixes` (*page 54*)
- `logger` (*page 56*)
 - `level` (*page 57*)
 - `loggers` (*page 57*)
- `controller` (*page 57*)

request

sfContext Accessor: `$context->getRequest()`

Default configuration:

```
request:
  class: sfWebRequest
  param:
    logging:          %SF_LOGGING_ENABLED%
    path_info_array:  SERVER
    path_info_key:    PATH_INFO
    relative_url_root: ~
  formats:
    txt: text/plain
    js:  [application/javascript, application/x-javascript, text/
javascript]
    css: text/css
    json: [application/json, application/x-json]
    xml:  [text/xml, application/xml, application/x-xml]
    rdf:  application/rdf+xml
    atom: application/atom+xml
```

Listing
5-5

path_info_array

The `path_info_array` option defines the global PHP array that will be used to retrieve information. On some configurations you may want to change the default `SERVER` value to `ENV`.

path_info_key

The `path_info_key` option defines the key under which the `PATH_INFO` information can be found.

If you use IIS with a rewriting module like IIFR or ISAPI, you may need to change this value to `HTTP_X_REWRITE_URL`.

formats

The `formats` option defines an array of file extensions and their corresponding Content-Types. It is used by the framework to automatically manage the Content-Type of the response, based on the request URI extension.

relative_url_root

The `relative_url_root` option defines the part of the URL before the front controller. Most of the time, this is automatically detected by the framework and does not need to be changed.

response

SfContext Accessor: `$context->getResponse()`

Default configuration:

Listing 5-6

```
response:
  class: sfWebResponse
  param:
    logging:          %SF_LOGGING_ENABLED%
    charset:          %SF_CHARSET%
    send_http_headers: true
```

Default configuration for the test environment:

Listing 5-7

```
response:
  class: sfWebResponse
  param:
    send_http_headers: false
```

send_http_headers

The `send_http_headers` option specifies whether the response should send HTTP response headers along with the response content. This setting is mostly useful for testing, as headers are sent with the `header()` PHP function which sends warnings if you try to send headers after some output.

charset

The `charset` option defines the charset to use for the response. By default, it uses the `charset` setting from `settings.yml`, which is what you want most of the time.

http_protocol

The `http_protocol` option defines the HTTP protocol version to use for the response. By default, it checks the `$_SERVER['SERVER_PROTOCOL']` value if available or defaults to HTTP/1.0.

user

sfContext Accessor: `$context->getUser()`

Default configuration:

```
user:
  class: myUser
  param:
    timeout:          1800
    logging:          %SF_LOGGING_ENABLED%
    use_flash:        true
    default_culture: %SF_DEFAULT_CULTURE%
```

Listing
5-8



By default, the `myUser` class inherits from `sfBasicSecurityUser`, which can be configured in the `security.yml` (page 81) configuration file.

timeout

The `timeout` option defines the timeout for user authentication. It is not related to the session timeout. The default setting automatically unauthenticates a user after 30 minutes of inactivity.

This setting is only used by user classes that inherit from the `sfBasicSecurityUser` base class, which is the case of the generated `myUser` class.



To avoid unexpected behavior, the user class automatically forces the maximum lifetime for the session garbage collector (`session.gc_maxlifetime`) to be greater than the timeout.

use_flash

The `use_flash` option enables or disables the flash component.

default_culture

The `default_culture` option defines the default culture to use for a user who comes to the site for the first time. By default, it uses the `default_culture` setting from `settings.yml`, which is what you want most of the time.



If you change the `default_culture` setting in `factories.yml` or `settings.yml`, you need to clear your cookies in your browser to check the result.

storage

The storage factory is used by the user factory to persist user data between HTTP requests.

sfContext Accessor: `$context->getStorage()`

Default configuration:

Listing 5-9

```
storage:
  class: sfSessionStorage
  param:
    session_name: symfony
```

Default configuration for the test environment:

Listing 5-10

```
storage:
  class: sfSessionTestStorage
  param:
    session_path: %SF_TEST_CACHE_DIR%/sessions
```

auto_start

The `auto_start` option enables or disables the session auto-starting feature of PHP (via the `session_start()` function).

session_name

The `session_name` option defines the name of the cookie used by symfony to store the user session. By default, the name is `symfony`, which means that all your applications share the same cookie (and as such the corresponding authentication and authorizations).

session_set_cookie_params() parameters

The storage factory calls the `session_set_cookie_params()`⁷ function with the value of the following options:

7. http://www.php.net/session_set_cookie_params

- `session_cookie_lifetime`: Lifetime of the session cookie, defined in seconds.
- `session_cookie_path`: Path on the domain where the cookie will work. Use a single slash (/) for all paths on the domain.
- `session_cookie_domain`: Cookie domain, for example `www.php.net`. To make cookies visible on all subdomains then the domain must be prefixed with a dot like `.php.net`.
- `session_cookie_secure`: If `true` cookie will only be sent over secure connections.
- `session_cookie_httponly`: If set to `true` then PHP will attempt to send the `httponly` flag when setting the session cookie.



The description of each option comes from the `session_set_cookie_params()` function description on the PHP website

`session_cache_limiter`

If the `session_cache_limiter` option is set, PHP's `session_cache_limiter()`⁸ function is called and the option value is passed as an argument.

Database Storage-specific Options

When using a storage that inherits from the `sfDatabaseSessionStorage` class, several additional options are available:

- `database`: The database name (required)
- `db_table`: The table name (required)
- `db_id_col`: The primary key column name (`sess_id` by default)
- `db_data_col`: The data column name (`sess_data` by default)
- `db_time_col`: The time column name (`sess_time` by default)

`view_cache_manager`

sfContext Accessor: `$context->getViewCacheManager()`

Default configuration:

```
view_cache_manager:  
  class: sfViewCacheManager
```

Listing
5-11

8. http://www.php.net/session_cache_limiter



This factory is only created if the `cache` (page 38) setting is set to on.

The view cache manager configuration does not include a `param` key. This configuration is done via the `view_cache` factory, which defines the underlying cache object used by the view cache manager.

view_cache

sfContext Accessor: none (used directly by the `view_cache_manager` factory)

Default configuration:

Listing
5-12

```
view_cache:
  class: sfFileCache
  param:
    automatic_cleaning_factor: 0
    cache_dir:                 %SF_TEMPLATE_CACHE_DIR%
    lifetime:                  86400
    prefix:                    %SF_APP_DIR%/template
```



This factory is only defined if the `cache` (page 38) setting is set to on.

The `view_cache` factory defines a cache class that must inherit from `sfCache` (see the Cache section for more information).

i18n

sfContext Accessor: `$context->getI18N()`

Default configuration:

Listing
5-13

```
i18n:
  class: sfI18N
  param:
    source:                    XLIFF
    debug:                     off
    untranslated_prefix:      "[T]"
    untranslated_suffix:      "[/T]"
    cache:
```

```
class: sfFileCache
param:
  automatic_cleaning_factor: 0
  cache_dir:                 %SF_I18N_CACHE_DIR%
  lifetime:                  31556926
  prefix:                    %SF_APP_DIR%/i18n
```



This factory is only defined if the `i18n` (page 38) setting is set to `on`.

source

The `source` option defines the container type for translations.

Built-in containers: XLIFF, SQLite, MySQL, and gettext.

debug

The `debug` option sets the debugging mode. If set to `on`, un-translated messages are decorated with a prefix and a suffix (see below).

untranslated_prefix

The `untranslated_prefix` defines a prefix to used for un-translated messages.

untranslated_suffix

The `untranslated_suffix` defines a suffix to used for un-translated messages.

cache

The `cache` option defines a anonymous cache factory to be used for caching `i18n` data (see the `Cache` section for more information).

routing

sfContext Accessor: `$context->getRouting()`

Default configuration:

```
routing:
  class: sfPatternRouting
```

*Listing
5-14*

```

param:
  load_configuration:      true
  suffix:                 ''
  default_module:         default
  default_action:         index
  debug:                  %SF_DEBUG%
  logging:                 %SF_LOGGING_ENABLED%
  generate_shortest_url:   true
  extra_parameters_as_query_string: true
  cache:
    class: sfFileCache
    param:
      automatic_cleaning_factor: 0
      cache_dir:                 %SF_CONFIG_CACHE_DIR%/routing
      lifetime:                  31556926
      prefix:                    %SF_APP_DIR%/routing

```

variable_prefixes

Default: :

The `variable_prefixes` option defines the list of characters that starts a variable name in a route pattern.

segment_separators

Default: / and .

The `segment_separators` option defines the list of route segment separators. Most of the time, you don't want to override this option for the whole routing, but for specific routes.

generate_shortest_url

Default: true for new projects, false for upgraded projects

If set to `true`, the `generate_shortest_url` option will tell the routing system to generate the shortest route possible. Set it to `false` if you want your routes to be backward compatible with symfony 1.0 and 1.1.

extra_parameters_as_query_string

Default: true for new projects, false for upgraded projects

When some parameters are not used in the generation of a route, the `extra_parameters_as_query_string` allows those extra parameters to be converted to a

query string. Set it to `false` to fallback to the behavior of symfony 1.0 or 1.1. In those versions, the extra parameters were just ignored by the routing system.

cache

The `cache` option defines an anonymous cache factory to be used for caching routing configuration and data (see the Cache section for more information).

suffix

Default: none

The default suffix to use for all routes. This option is deprecated and is not useful anymore.

load_configuration

Default: true

The `load_configuration` option defines whether the `routing.yml` files must be automatically loaded and parsed. Set it to `false` if you want to use the routing system of symfony outside of a symfony project.

lazy_routes_deserialize

Default: false

If set to `true`, the `lazy_routes_deserialize` setting enables lazy unserialization of the routing cache. It can improve the performance of your applications if you have a large number of routes and if most matching routes are among the first ones. It is strongly advised to test the setting before deploying to production, as it can harm your performance in certain circumstances.



This setting is only available for symfony 1.2.7 and up.

lookup_cache_dedicated_keys

Default: false

The `lookup_cache_dedicated_keys` setting determines how the routing cache is constructed. When set to `false`, the cache is stored as one big value; when set to `true`, each route has its own cache store. This setting is a performance optimization setting.

As a rule of thumb, setting this to `false` is better when using a file-based cache class (`sfFileCache` for instance), and setting it to `true` is better when using a memory-based cache class (`sfAPCCache` for instance).



This setting is only available for symfony 1.2.7 and up.

logger

sfContext Accessor: `$context->getLogger()`

Default configuration:

*Listing
5-15*

```
logger:
  class: sfAggregateLogger
  param:
    level: debug
    loggers:
      sf_web_debug:
        class: sfWebDebugLogger
        param:
          level: debug
          condition: %SF_WEB_DEBUG%
          xdebug_logging: true
          web_debug_class: sfWebDebug
      sf_file_debug:
        class: sfFileLogger
        param:
          level: debug
          file: %SF_LOG_DIR%/%SF_APP%_%SF_ENVIRONMENT%.log
```

Default configuration for the prod environment:

*Listing
5-16*

```
logger:
  class: sfNoLogger
  param:
    level: err
    loggers: ~
```



This factory is always defined, but the logging only occurs if the `logging_enabled` setting is set to on.

level

The `level` option defines the level of the logger.

Possible values: EMERG, ALERT, CRIT, ERR, WARNING, NOTICE, INFO, or DEBUG.

loggers

The `loggers` option defines a list of loggers to use. The list is an array of anonymous logger factories.

Built-in logger classes: `sfConsoleLogger`, `sfFileLogger`, `sfNoLogger`, `sfStreamLogger`, and `sfVarLogger`.

controller

sfContext Accessor: `$context->getController()`

Default configuration:

```
controller:  
  class: sfFrontWebController
```

*Listing
5-17*

Anonymous Cache Factories

Several factories (`view_cache`, `i18n`, and `routing`) can take advantage of a cache object if defined in their configuration. The configuration of the cache object is similar for all factories. The `cache` key defines an anonymous cache factory. Like any other factory, it takes a `class` and a `param` entries. The `param` entry can take any option available for the given cache class.

The `prefix` option is the most important one as it allows to share or separate a cache between different environments/applications/projects.

Built-in cache classes: `sfAPCCache`, `sfAcceleratorCache`, `sfFileCache`, `sfMemcacheCache`, `sfNoCache`, `sfSQLiteCache`, and `sfXCacheCache`.

