VIEW

The result of an action execution is a **View**. A View is the combination of a template described by a classic PHP file, and a configuration file describing the way this template will fit with other interface elements. It **contains** some **HTML code** and some **basic PHP** code, usually **calls to variables defined in the action** and **helpers**.

TEMPLATES

NAMING CONVENTIONS

A **template name** is made of two parts:

1st part: is related to the **action**

2nd part: to the **result**

So, an action called **index** that executed successfully is:

indexSuccess.php

This is an implicit rule: if the value returned by the action is **sfView::SUCCESS**, then the name of the template called will be: the **name of the action** concatenated with **Success.php**

With two actions index and list in a product module:

```
class productActions extends sfActions{
  public function executeIndex(){
    ...
    return sfView::SUCCESS;
  }
  public function executeList(){
    ...
    return sfView::SUCCESS;
  }
}
```

The templates called at the end of the execution of each action will be located in `myapp/modules/product/templates/` and named: `indexSuccess.php` `listSuccess.php`

ALTERNATE TEMPLATE

An action can set an alternate template:

```
$this->setTemplate('myCustomTemplate');
```

The template called is:

`myCustomTemplateSuccess.php`

VARIABLES

Variables called in the templates must be either one of the usual shortcuts (see below) or attributes of the action object defined in the related action file.
E.g.: to define a value for the **$name** variable, the action must contain a line:

```
$this->name = 'myvalue';
```

SHORTCUTS

Pre-defined variables or shortcuts:

$sf_context	//the whole context object
$sf_request	//the request
$sf_params	//parameters of the request
$sf_user	//the current sfUser object
$sf_view	//the calling sfView object

E.g.:
the template code:

```
echo $sf_params->get('total');
```

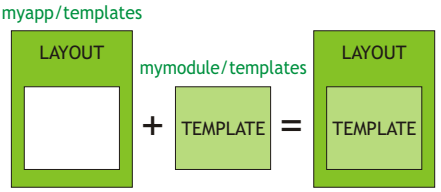
is equivalent to the following action code:

```
echo $this->getRequestParameter('total');
```

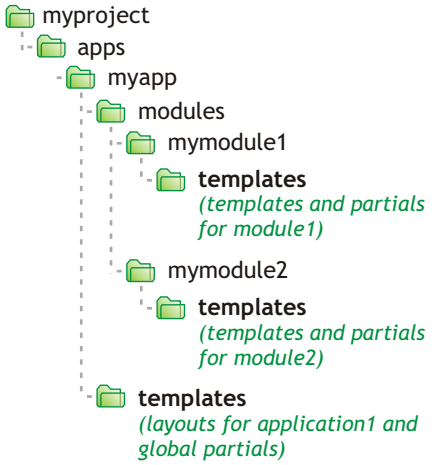
Access the action stack:

```
$sf_context->getModuleName()
//last module called
$sf_context->getActionName()
//last action called
```

DECORATOR DESIGN PATTERN



TEMPLATES LOCATION



HELPERS

Facilitate the process of writing templates and produce the best possible HTML code in terms of **performance** and **accessibility.**

Three types of helpers are available:

1) **Standard compulsory helpers**- must be used instead of the corresponding HTML code because they allow internal symfony mechanisms (routing, i18n, ...) to work.
2) **Standard optional helpers**, that use less code than classic HTML for the same purpose. Using these helpers, you make sure that your code will even work after any subsequent change in the file tree structure.
3) **Helpers defined specifically for an application** (custom helpers).

Default helpers - loaded for every app:

Helper:	defines the use_helper() helpers, needed for helper inclusion
Tag:	defines the basic tag operations
Url:	links and URL management
Asset:	head, include, images and js call
Partial:	allow inclusion of template fragments
Cache:	manipulation of cached code fragments
Form:	form input

Loading a non-default helper

```
echo use_helper('HelperName')
```

Using helpers outside a template

```
sfLoader::loadHelpers($helpers)
```

DEFAULT GLOBAL LAYOUT

`<myproject>/apps/<myapp>/templates/layout.php`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
   www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
   <?php echo include_http_metas() ?>
   <?php echo include_metas() ?>
   <?php echo include_title() ?>
   <link rel="shortcut icon" href="/favicon.ico" />
</head>
<body>

<?php echo $sf_data->getRaw('sf_content') ?>

</body>
</html>
```

LAYOUT CONFIGURATION

You may have several layouts for your application.
The default layout is **myproject/apps/myapp/templates/layout.php**. Additional layouts must be added in the same global **templates/** directory.

To use a custom layout file, you can set it in the **view.yml** file or in the **action.**

Some views don't need any layout at all (for instance, plain text pages or RSS feeds). In that case, set **has_layout** to **false**

LAYOUT DEFINITION IN VIEW.YML

```
indexSuccess:
  layout: my_layout
```

LAYOUT REMOVAL IN VIEW.YML

```
indexSuccess:
  has_layout: false
```

LAYOUT DEFINITION IN ACTION

```
$this->setLayout('my_layout');
```

LAYOUT REMOVAL IN ACTION

```
$this->setLayout(false);
```

Ajax actions views **have no layout** by **default.**