Routing

Thelia uses the Symfony-cmf Routing component, so it's possible to declare as many routers as are needed and add them in this routing component. If you need to add a router you can do it in two different ways

THE DEFAULT BEHAVIOR

All you have to do is to create a file named routing.xml in your Config directory. Thelia will configure a new router and set a default priority (150) to it.

CUSTOM ROUTING

If you need a custom configuration for your routing, you can declare a new service and tag this service and put **router.register** for the name property and the priority you want.

Here is an example :

```
<service id="router.front" class="%router.class%">
    <argument type="service" id="router.module.xmlLoader"/>
    <argument>Front/Config/front.xml</argument>
    <argument type="collection">
      <argument key="cache_dir">%kernel.cache_dir%</argument>
      <argument key="debug">%kernel.debug%</argument>
    </argument>
    <argument type="service" id="request.context"/>
    <tag name="router.register" priority="128"/>
</service>
```

Model

Your module may need to create tables, generate model classes and interact with Thelia's model. How to do ?

1. Create the file schema.xml in your Config directory.
2. Fill schema.xml file, you can find all the information you need in Propel documentation.
3. Use the CLI tools to generate model and sql (php Thelia module:generate:model MyModule --generate-sql).

Note : it's better to put the namespace property on each table attribute instead of the database attribute.

Main class

The main class in your module is the most important file. This class is used when the module is activated or deactivated.

Most of the time this class will have the same name as your module directory. If my module directory is **Atos**, my main class will be **Atos** too and the full namespace will be **Atos\Atos**.

Thelia\Module\AbstractDeliveryModule : Use this class when you develop a delivery module

Thelia\Module\AbstractPaymentModule : Use this class when you develop a payment module

Thelia\Module\BaseModule : Use this class when you develop a classic module

AbstractDeliveryModule and **AbstractPaymentModule** classes extend the BaseModule class.

Some methods in BaseModule can be useful if you want to interact with Thelia during the installation or the removal process. You just have to overload the method you want and implement your code.

preActivation
This method is called before the module activation, and may prevent it by returning false.

postActivation
This method is called just after the module was successfully activated. If an exception is thrown the procedure will be stopped and a rollback of the current transaction will be performed.

preDeactivation
This method is called before the module deactivation, and may prevent it by returning false.

postDeactivation
This method is called just after the module was successfully deactivated. If an exception is thrown the procedure will be stopped and a rollback of the current transaction will be performed.

getCompilers
This method adds new compilers to Thelia container.

getHooks
This method must be used if your module defines hooks.

Specific methods for **AbstractDeliveryModule**

isValidDelivery
This method is called by the Delivery loop, to check if the current module has to be displayed to the customer. This method must be implemented in your module.

getPostage
This method calculates and returns the delivery price.
This method must be implemented in your module.

Specific methods for **AbstractPaymentModule**

pay
Method used by payment gateways.
This method must be implemented in your module.

isValidPayment
This method is called by the Payment loop, to check if the current module has to be displayed to the customer. This method must be implemented in your module.

generateGatewayFormResponse
This method renders the payment gateway template. The module should provide the gateway URL and the form fields names and values. This method is a helper.

getPaymentSuccessPageUrl
Return the order payment success page URL.

getPaymentFailurePageUrl
Redirect the customer to the failure payment page. If $message is null, a generic message is displayed.

Thelia templates use the Smarty template engine, enriched by many Thelia additions, such as loops, data access functions, internationalization function, etc

Structure

See Thelia structure for more information.

Every template should contain specific template files, which are the views invoked in the Front and Back Offices controllers. For a front-office template, these files are :

product.html : displays a product.
content.html : displays a content.
category.html : displays a category's content.
feed.html : the RSS product or content feed.
folder.html : displays a folder content.
404.html : is displayed if a page cannot be found.
order-delivery.html : is displayed during ordering process to choose a delivery method
order-invoice.html : is displayed during ordering process to choose a payment gateway.
order-failed.html : is displayed when a payment fails.
order-payment-gateway.html : filled by the payment gateway to send to the platform a specific form.
order-placed.html : is displayed once the payment is successfully performed.

Assets management

Template assets are managed in a sub-directory of the template directory.
For example, the default front-office template contains an 'assets' directory to store all template's assets.
To use this feature, you'll have to add some specific directives to your template files.

{declare_assets}

This directive tells Thelia's template system where your assets are located, e.g. the name of the root directory which contains all your assets.

Example :
```
{declare_assets directory="assets"}
```

{stylesheets}

This directive processes your CSS style sheets.

Example :
```
{stylesheets file="assets/css/*.less" filters="less"}
    <link href="{$asset_url}" rel="stylesheet" type="text/css" />
{/stylesheets}
```

This block returns only one parameter, $asset_url, which is the asset URL in the web directory, e.g. under the web/assets path.

file
This is the path to the file (or files, as jokers like '*' are allowed), relative to the template base path.

filters
Apply a filter to the source(s) files. Available filters are :
less : compiles CSS using the LESS compiler
sass : compiles CSS using the SASS compiler
compass : compiles CSS using the Compass compiler

source
When in the templates files of a module, use this parameter to specify that the source of the asset has to be searched within the module's path instead of the main template path.

template
You may want to use an asset located in another template of the same type (for example, another front office template). To do so, specify the name of this template in the template parameter.

{images}

This directive processes the static images used in your template.

Example :
```
{images file='assets/img/favicon.ico'}
    <link rel="shortcut icon" type="image/x-icon" href="{$asset_url}">
{/images}
```

This block returns only one parameter, $asset_url, which is the asset URL in the web directory, e.g. under the web/assets path.

file
This is the path to the file (jokers like '*' are NOT allowed), relative to the template base path.

source
When the asset is in a module directory, you need to use this parameter to specify that the source of the asset has to be searched within the module's path instead of the main template path.

template
You may want to use an asset located in another template of the same type (for example, another front office template). To do so, specify the name of this template in the template parameter

{javascripts}

This directive processes your javascript files.

Example :
```
{javascripts file='assets/js/script.js'}
    <script type="text/javascript" src="{$asset_url}"></script>
{/javascripts}
```

file
This is the path to the file (or files, as jokers like '*' are allowed), relative to the template base path.

source
When the asset is in a module directory, you need to use this parameter to specify that the source of the asset has to be searched within the module's path instead of the main template path.

template
You may want to use an asset located in another template of the same type (for example, another front office template). To do so, specify the name of this template in the template parameter

Loop system

Loops are the most convenient feature in Thelia for frontend developers. Already there in Thelia's first version, they have to be improved for Thelia v2. Loops allow to gather data from your shop and display them in your front view. In Thelia v2, loops are a Smarty v3 plugin.

SYNTAX

```
{ifloop rel="my_associated_content_loop"}
  Associated contents for this product :
  <ul>
    {loop type="associated_content" name="my_associated_content_loop"
    product="12"}
      <li>
        <a href="{$URL}">{$TITLE}</a>
      </li>
    {/loop}
  </ul>
{/ifloop}
{elseloop rel="my_associated_content_loop"}
  No associated content for this product
{/elseloop}
```

{loop} {/loop}

The loop function have at least two mandatory parameters :

name
A unique name used to identify the loop in other functions (ifloop and elseloop)

type
The type of a loop is the type of data you want to retrieve.
For the complete type list, see Thelia documentation at http://doc.thelia.net

Each loop type defines its own parameters, you can search this parameter in Thelia documentation.

{ifloop}/{elseloop}

{ifloop} and {elseloop} are conditional loops. They allow to define a different behaviour depending on if the a classic loop displays something or not.
A conditional loop is therefore linked to a classic loop using the rel attribute which must match a classic loop named attribute.

Resources

Documentation : http://doc.thelia.net
Github repo : https://github.com/thelia/thelia
Website : http://thelia.net
Contact : dev@thelia.net
Authors : Manuel Raynaud, Julien Chanséaume, Benjamin Perche, Franck Allimant
Acknowledgements : Damien Souza, Stéphanie Pinet, Marion Laurent

thelia
NEXT GENERATION E-COMMERCE

thelia
e-commerce solution

Introduction

Thelia is an open source tool for creating e-business websites and managing online content. Created in 2005, the new version of Thelia aims to be the next generation E-commerce system. It is based on Symfony 2 components and meets the following objectives : performance and scalability.

Installation

Requirements

Thelia needs at least php 5.4 and works with php 5.5 and 5.6 for now. For the database, Thelia requires at least mysql 5.5.

PHP extensions

- intl
- mcrypt
- mysql and pdo_mysql
- curl
- gd

Download Thelia

You can download Thelia in two different ways :

FROM THE THELIA WEBSITE

Go to the thelia website (http://thelia.net) and download it.

USING COMPOSER

$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar create-project thelia/thelia your-path 2.0.3

Install Thelia

First of all, create a vhost dedicated to Thelia and put the documentRoot in the web directory. Here again you can install Thelia in two different ways.

USING INSTALL WIZARD

With your favorite browser, navigate to the install directory :
http://yourdomain.tld/[subdomain_if_needed]/install
For example, I have thelia downloaded at http://thelia.net and my vhost is correctly configured, I have to go to this address :
http://thelia.net/install

USING CLI TOOLS

$ php Thelia thelia:install
and follow the instructions
After installing Thelia, remove the web/install directory
After the installation you have an architecture like this :

```
www/ <- your web root directory
    thelia/ <- your thelia directory
        bin
        cache
        core
        setup
        local
                config
                modules
                session
        log
        templates
        web <- the only directory accessible by your web server
```

CLI Tools

Thelia has a command line tool that can help you automate repetitive tasks. Obviously you can develop your own command.

Usage

$ cd to/thelia/repository
$ php Thelia

If you use the command line without any argument, it will display all command and options available.

admin:create
Create a new administrator user
$ php Thelia admin:create

admin:updatePassword
Change administrator password
$ php Thelia admin:updatePassword adminlogin [--password="..."]

cache:clear
Invalidate cache
$ php Thelia cache:clear [--env="..."] [--without-assets] [--with-images]

image-cache:clear
Empty part or whole web space image cache
$ php Thelia :image-cache:clear [subdir]

module:activate
Activate a module
$ php Thelia module:activate module-name

module:deactivate
Deactivate a module
$ php Thelia module:deactivate module-name

module:generate
Generate all needed files for creating a new Module
$ php Thelia module:generate module-name

module:generate:model
Generate model for a specific module
$ php Thelia module:generate:model module-name [--generate-sql]

module:generate:sql
Generate the sql from schema.xml file for a specific module
$ php Thelia module:generate:sql module-name

module:refresh
Refresh module list
$ php Thelia module:refresh

thelia:dev:reloadDB
Erase current database and create new one. All your data will be lost
$ php Thelia thelia:dev:reloadDB

thelia:generate-resources
Outputs admin resources
$ php Thelia thelia:generate-resources [--output[="..."]]

thelia:install
Install Thelia
$ php Thelia thelia:install

thelia:update
Update Thelia database. Before doing that you have to update Thelia files
$ php Thelia thelia:update

Modules

Modules are the best way to extend Thelia functionalities. Payment and delivery methods are all modules.

The structure of a module is exactly the same as Thelia's core. A module can interact with the container in order to add its own services, to create new compilers, etc.

Structure

```
\MyModule
    \Config
        config.xml   <- mandatory
        module.xml   <- mandatory
        routing.xml
        schema.xml
    MyModule.php <- mandatory
    \Loop
        Product.php
        MyLoop.php
    ...
```

CONFIG.XML CONTENT

```
<?xml version="1.0" encoding="UTF-8" ?>
<config xmlns="http://thelia.net/schema/dic/config"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://thelia.net/schema/dic/config http://thelia.net/
        schema/dic/config/thelia-1.0.xsd">

    <loops>
        <loop name="MySuperLoop" class="MyModule\Loop\MySuperLoop" />
    </loops>

    <forms>
        <form name="MyFormName" class="MyModule\Form\MySuperForm" />
    </forms>

    <commands>
        <command class="MyModule\Command\MySuperCommand" />
    </commands>

    <services>
        <service id="Mymodule.service.id" class="MyModule\
        MySuperService"/>
    </services>

    <hooks>
        <hook id="mymodule.hook" class="MyModule\Hook\MySuperHook"
        scope="request">
            <tag name="hook.event_listener" event="main.body.bottom"
            type="front|back|pdf|email" method="onMainBodyBottom" />
        </hook>
    </hooks>

    <exports> </exports>
    <imports> </imports>

</config>
```

loop
Declare a loop. Name and class properties are mandatory. The name is a unique key and the class is the full namespace for the loop class.

form
Declare a form. Name and class properties are mandatory. The name is a unique key and the class is the full namespace for the form class.

command
Declare a command. Name property is mandatory. The class is the full namespace for the command class.

service
Services are the exact same notion as for Symfony services.
See the dedicated chapter below.

hook
Hooks are the entry points thanks to which modules will insert their own code. To configure hooks, you must declare them in the config.xml file.

Example :

```
<hook id="mymodule.hook" class="MyModule\Hook\MySuperHook"
scope="request">
    <tag name="hook.event_listener" event="main.body.bottom" type="front"
    method="onMainBodyBottom" />
</hook>
```

On the hook node, id and class are mandatory.
The id is a unique identifier and the class is the full path to the class.

On the tag node, name and event are mandatory.
The others are not mandatory, here are more details :

name="hook.event_listener" : this never changes.

event : represents the hook code to which it wants to respond.
type : indicates the context of the hook : frontOffice (default), backOffice, pdf or email.
method : indicates the name of the method to call. By default, it will be based on the name of the hook . eg : for product.additional hook, the method will be called onProductAdditional (CamelCase prefixed by on).
active : allows you to activate the hook (set to 1 - default) or not (set to 0) once the module is installed

Import

```
<import id="your.import.id" class="Your\ImportHandler"
category_id="the.category_id">
    <descriptive locale="en_US">
        <title>Your import title </title>
        <!-- you may add an optional description -->
        <description> ... </description>
    </descriptive>
    <descriptive locale="fr_FR">
        <!-- Here's for another locale -->
    </descriptive>
</import>
```

On the import node, id, class and category_id properties are mandatory.
The id is a unique identifier and the class is the full path to the class.

category_id possible values are :

thelia.import.customer : Imports the customers' data
thelia.import.products : Imports the products' data
thelia.import.content : Imports the contents' data
thelia.import.order : Imports the orders' data
thelia.import.modules : module related imports

Export

```
<export id="your.export.id" class="Your\ExportHandler" category_id="the.
category_id">
    <descriptive locale="en_US">
        <title>Your export title </title>
        <!-- you may add an optional description -->
        <description> ... </description>
    </descriptive>
    <descriptive locale="fr_FR">
        <!-- Here's for another locale -->
    </descriptive>
</export>
```

On the export node, **id, class** and **category_id** properties are mandatory.
The **id** is a unique identifier and the class is the full path to the class.

category_id possible values are :

thelia.export.customer : Exports the customers' data
thelia.export.products : Exports the products' data
thelia.export.content : Exports the contents' data
thelia.export.order : Exports the orders' data
thelia.export.modules : module related exports

You can also create a custom category if you want.
For this you have to put something like below :

```
<export_categories>

    <export_category id="your.category.id">
        <title locale="en_US">A title</title>
        <title locale="fr_FR">Un titre</title>
    </export_category>

    <export_category id="your.other.category.id">
        <!-- here's another import category -->
    </export_category>

</export_categories>
```

MODULE.XML CONTENT

Module.xml file is a description of your module. It includes the author's name, his contact details, module version and the version of Thelia it is compatible with.

```
<?xml version="1.0" encoding="UTF-8"?>
<module>
    <fullnamespace>Atos\Atos</fullnamespace>
    <descriptive locale="en_US">
        <title>Atos-sips payment module</title>
    </descriptive>
    <descriptive locale="fr_FR">
        <title>module de paiement Atos-sips</title>
    </descriptive>
    <version>0.9</version>
    <author>
        <name>Manuel Raynaud</name>
        <email>manu@thelia.net</email>
    </author>
    <type>payment</type>
    <thelia>2.0.0</thelia>
    <stability>beta</stability>
</module>
```

fullnamespace
The full namespace of the module's main class.

descriptive
This block can be repeated for as many locale as you want. It includes a title, subtitle, description and postscriptum. Only the title is mandatory.

version
Module version

author
Author information. It includes a name, a company, an email and a website tag. Only the name is mandatory.

thelia
Which version of Thelia your module is compatible with.

type
The type of your module. It can be :

payment : your module is a payment gateway.
delivery : your module is a delivery platform.
classic : all other types of modules.

stability
Your module stability. Can be one of the value below :

alpha, beta, rc, prod, other