FOSMessage Documentation

Release 1.0.x-dev

Titouan Galopin

Aug 23, 2017

Contents

FOSMessage is a PHP 5.4+ framework-agnostic library providing a data structure and common features to set up user-to-user messaging systems.

You can think of it as a model for your messaging features : it will take care of the consistency of the data for you in order to easily create a full-featured messaging system.

Note: This library is currently in development. You can test it in your project (the Composer installation process is very simple), but you should not use it in production for the moment.

This library is based on concepts shared by most modern frameworks (dependency injection, event dispatching, abstract data drivers, etc.) and therefore, it's very easy to set it up in any kind of context.

If you want to set it up in Symfony, *FOSMesageBundle* is being developed in a new version (not ready yet).

Key features

- Conversation-based messaging
- Multiple conversations participants support
- Very easy to implement (at least in most of the cases)
- Framework-agnotic
- Doctrine ORM and Mongo ODM support
- Not linked to user system implementation
- Optionnal tagging system to organize conversations
- Event system to let developer execute actions on key steps
- Implemented in framework-specific bundle / module
- PHP7 and HHVM support

CHAPTER 1

Documentation

Getting started

Requirements

FOSMessage only supports Doctrine ORM for the moment but it will support Doctrine ODM in the future. Therefore, for now, you need Doctrine ORM:

```
composer require doctrine/orm
```

Installation

This bundle is available on Packagist. You can install it using Composer:

```
composer require friendsofsymfony/message:1.0.x-dev
```

Important: You should **not** use development versions in Composer: we are using it here only because the library is currently in development. When the library will be released, change that version to follow semantic versionning.

Configuration (wihout framework)

Step 1: Set up your User model

Note: For the moment, only Doctrine ORM is supported. Doctrine ODM will be available soon.

FOSMessage provides a flexible set of tools organized around three main entites: conversations, messages and persons.

The library provides default entities for conversations and messages and they will be enough for the beginning (see *Customize the default entities* to learn more).

However, you need to configure the library to tell it what your User model is. FOSMessage requires that your user class implement `PersonInterface`. This library does not have any direct dependencies to any particular user system, except that it must implement the above interface.

Your user class may look something like the following:

```php
<?php

use Doctrine\ORM\Mapping as ORM;
use FOS\Message\Model\PersonInterface;

/**
 * @ORM\Entity
 */
class User implements PersonInterface
{
    public function getId()
    {
        return $this->id;
    }

    // Your code ...
}
```

Step 2: Configure the Doctrine entity manager

You need to configure Doctrine for two things:

- use your User model as the entity for FOSMessage ;
- use the default entities provided by Doctrine ;

If you are not using a framework, you need to configure Doctrine manually in order to get a usable EntityManager for FOSMessage.

Here is an example of configuration to help you do so:

```php
<?php

$config = \Doctrine\ORM\Tools\Setup::createConfiguration(true);

/*
 * Tell Doctrine to use both your entities and the default entities from FOSMessage
 */
$config->setMetadataDriverImpl($config->newDefaultAnnotationDriver([
    __DIR__ . '/vendor/friendsofsymfony/message/src/Driver/Doctrine/ORM/Entity',
    __DIR__ . '/src',
], false));

/*
 * If you want to use a debug logger
 */
if ($logger) {
    $config->setSQLLogger($logger);
}
```

```
/*
 * Your database parameters
 */
$dbParams = [
    'driver'   => 'pdo_mysql',
    'host'     => '127.0.0.1',
    'user'     => 'root',
    'password' => 'root',
    'dbname'   => 'fos_message',
];

/*
 * Use the Doctrine event manager to use your User model instead of the FOSMessage␣
↪interface
 * in FOSMessage driver
 */
$rtel = new \Doctrine\ORM\Tools\ResolveTargetEntityListener();
$rtel->addResolveTargetEntity('FOS\\Message\\Model\\PersonInterface', 'Entity\\User',␣
↪[]);

$evm  = new \Doctrine\Common\EventManager();
$evm->addEventListener(Doctrine\ORM\Events::loadClassMetadata, $rtel);

/*
 * Finally, create the Doctrine EntityManager
 */
$entityManager = \Doctrine\ORM\EntityManager::create($dbParams, $config, $evm);
```

Configuration (using Symfony)

While the FOSMessage bundle is not ready, you can still configure Symfony and Doctrine to use the library in your project.

Step 1: Set up your User model

Note: For the moment, only Doctrine ORM is supported. Doctrine ODM will be available soon.

FOSMessage provides a flexible set of tools organized around three main entites: conversations, messages and persons.

The library provides default entities for conversations and messages and they will be enough for the beginning (see *Customize the default entities* to learn more).

However, you need to configure the library to tell it what your User model is. FOSMessage requires that your user class implement `PersonInterface`. This library does not have any direct dependencies to any particular user system, except that it must implement the above interface.

Your user class may look something like the following:

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use FOS\Message\Model\PersonInterface;
```

```
/**
 * @ORM\Entity
 */
class User implements PersonInterface
{
    public function getId()
    {
        return $this->id;
    }

    // Your code ...
}
```

Step 2: Configure the Doctrine entity manager

You need to configure Doctrine for two things:

- use your User model as the entity for FOSMessage ;
- use the default entities provided by Doctrine ;

When you are using Symfony, you can configure the Doctrine entity manager through the DoctrineBundle configuration:

```
# app/config/config.yml

doctrine:
    # ...

    orm:
        auto_generate_proxy_classes: "%kernel.debug%"
        naming_strategy: doctrine.orm.naming_strategy.underscore
        auto_mapping: true

        # The mappings to import the FOSMessage entities
        mappings:
            fos_message:
                type: annotation
                dir: %kernel.root_dir%/../vendor/friendsofsymfony/message/src/Driver/
↪Doctrine/ORM/Entity
                prefix: FOS\Message\Driver\Doctrine\ORM\Entity

        # User your user entity instead of the PersonInterface
        resolve_target_entities:
            FOS\Message\Model\PersonInterface: AppBundle\Entity\User
```

You also need to register a few services:

```
# app/config/services.yml

services:
    fos_message.driver:
        class: FOS\Message\Driver\Doctrine\ORM\DoctrineORMDriver
        arguments: [ "@doctrine.orm.entity_manager" ]

    fos_message.repository:
        class: FOS\Message\Repository
        arguments: [ "@fos_message.driver" ]

    fos_message.event_dispatcher:
        class: FOS\Message\EventDispatcher\SymfonyBridgeEventDispatcher
        arguments: [ "@event_dispatcher" ]

    fos_message.tagger:
        class: FOS\Message\Tagger
        arguments:
            - "@fos_message.driver"
            - "@fos_message.repository"

    fos_message.sender:
        class: FOS\Message\Sender
        arguments:
            - "@fos_message.driver"
            - "@fos_message.event_dispatcher"
```

And then you will be able to use the components as following:

```php
<?php

namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class DefaultController extends Controller
{
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction()
    {
        $repository = $this->get('fos_message.repository');
        $sender = $this->get('fos_message.sender');

        return $this->render('default/index.html.twig');
    }
}
```

Now that you have a configured entity manager, you are ready to start using the library!

Usage

Once you have configured Doctrine and your model, you are ready to use FOSMessage.

FOSMessage is organized around three components : the Repository that fetch conversations and messages, the Sender that start conversations and send replies and the Tagger that let you (the developer) tag conversations to retreive them in the future.

These three components are usually set up automatically in the context of a framework (by the dependency injection). If you are not using a framework, you have to set up these components yourself.

For the moment, as only Doctrine ORM is available in FOSMessage, you have to use the Doctrine ORM driver. In the future, other options will be available.

Choose your driver

The driver is the object linking the library to your persistance layer (Doctrine ORM, Propel, etc.). Thus according to what persistance layer you are using, you have to choose a different driver for FOSMessage.

For Doctrine ORM, you can create the driver as following using the entity manager configured in the **Getting started** chapter:

```php
<?php
$driver = new \FOS\Message\Driver\Doctrine\ORM\DoctrineORMDriver($entityManager);
```

Use the components

The Repository

The Repository is the basis of you messaging system. It let you fetch conversations and messages.

You can create it like this:

```php
<?php
$repository = new \FOS\Message\Repository($driver);
```

It provides 4 methods:

List the converations of a given person

Usually in a messaging system there is an “inbox”: a list of conversations in which the current user is participating.

To retrieve this list, the repository provides the method `getPersonConversations(PersonInterface $person, $tag = null)`.

You can use it either without `$tag` object (we will talk about tags a bit later) to fetch all the conversations of the given user.

Conversations will be sorted descending by date.

For instance, in a controller it could look like this:

```php
<?php

class MessagingController
{
    public function inboxAction()
    {
        // ...
        $repository = new \FOS\Message\Repository($driver);
        $conversations = $repository->getPersonConversations($this->getUser());

        return $this->render('inbox.html.twig', [ 'conversations' => $conversations␣
↪]);
    }
}
```

Find a conversation by its identifier

The method `getConversation($id)` is quite easy to understand: it returns a single conversation by its identifier (or null if none is found).

Note: Note that the security is not handled by the library: you should check if your user is allowed to access the conversation.

For instance, in a controller it could look like this:

```php
<?php

class MessagingController
{
    public function conversationAction($id)
    {
        // ...
        $repository = new \FOS\Message\Repository($driver);
        $conversation = $repository->getConversation($id);

        // Check access
        if (! $conversation->isPersonInConversation($this->getUser())) {
            throw new AccessDeniedHttpException();
        }

        return $this->render('conversation.html.twig', [ 'conversation' =>
↪$conversation ]);
    }
}
```

List the messages of a given conversation

One you have a conversation, you will probably want to display its messages. To do so, you have to use the method `getMessages(ConversationInterface $conversation, $offset = 0, $limit = 20, $sortDirection = 'ASC')`.

This method has 4 arguments:

- the conversation `$conversation` of the messages ;
- the offset in the result set (for pagination) ;
- the limit of messages to get (for pagination) ;
- the sort direction to use (messages will be sorted by date) ;

For instance, in a controller it could look like this:

```php
<?php

class MessagingController
{
    public function conversationAction($id)
    {
        // ...
        $repository = new \FOS\Message\Repository($driver);
        $conversation = $repository->getConversation($id);
        // Check access
        if (! $conversation->isPersonInConversation($this->getUser())) {
            throw new AccessDeniedHttpException();
        }

        $messages = $repository->getMessages($conversation);

        return $this->render('conversation.html.twig', [
            'conversation' => $conversation,
            'messages' => $messages,
        ]);
    }
}
```

Find the link between a person and a conversation

Sometimes you can need to retrieve the link between a user and a conversation (for instance if you customized the entities and stored data in this link).

To do so, the repository provides the method `getConversationPerson(ConversationInterface $conversation, PersonInterface $person)` that will return you an instance of `FOS\Message\ModelConversationPersonInterface`.

The Sender

The Sender let you start conversations and reply to them.

You can create it like this:

```php
<?php
$sender = new \FOS\Message\Sender($driver);
```

It provides 2 methods:

Start a conversation

The method `startConversation(PersonInterface $senderPerson, $recipient, $body, $subject = null)` will start a conversation with a sender and a single or multiple recipient(s). A first message will be posted in this conversation with a given body.

The method has 4 arguments:

- `$senderPerson`: the user who started the conversation ;
- `$recipient`: a single `PersonInterface` object or an array of `PersonInterface` ;
- `$body`: the content of the first message of the conversation ;
- `$subject`: in FOSMessage, subject is not required but you can provide one here ;

This method return the created conversation object (instance of `ConversationInterface`).

For instance, in a controller it could look like this:

```
<?php

class MessagingController
{
    public function startAction(Request $request)
    {
        // ...
        $sender = new \FOS\Message\Sender($driver);

        if ($request->getMethod() == 'POST') {
            $data = ...; // Find the form data for instance ...

            $conversation = $sender->startConversation($this->getUser(), $data[
↪'recipient'], $data['body']);

            return $this->redirect('conversation_view', [ 'id' => $conversation->
↪getId() ]);
        }

        return $this->render('form_start.html.twig');
    }
}
```

Reply to a conversation

Once a user has started a conversation, other members could reply. The method `sendMessage(ConversationInterface $conversation, PersonInterface $senderPerson, $body)` does exactly that by replying to a given conversation, as a given sender with a given body.

The method has 3 arguments:

- `$conversation`: the conversation in which the user want to post a reply ;
- `$senderPerson`: the user who wrote the message ;
- `$body`: the content of the reply ;

This method return the created message object (instance of `MessageInterface`).

For instance, in a controller it could look like this:

```
<?php

class MessagingController
{
    public function replyAction($id)
    {
        // ...
        $repository = new \FOS\Message\Repository($driver);
        $conversation = $repository->getConversation($id);

        // Check access
        if (! $conversation->isPersonInConversation($this->getUser())) {
            throw new AccessDeniedHttpException();
        }

        $sender = new \FOS\Message\Sender($driver);
        if ($request->getMethod() == 'POST') {
            $data = ...; // Find the form data for instance ...

            $message = $sender->sendMessage($conversation, $this->getUser(), $data[
↪'body']);

            return $this->redirect('conversation_view', [ 'id' => $conversation->
↪getId() ]);
        }

        return $this->render('form_reply.html.twig', [ 'conversation' =>
↪$conversation ]);
    }
}
```