# Superdesk Web Publisher Documentation

***Release 0.5.0***

**Sourcefabric z.ú.**

**Apr 16, 2019**

# Contents

*The next-generation publishing platform for newsrooms*

Superdesk Publisher is a lightweight open source renderer for news articles and other content delivered via an API feed. The code is released under the GNU Affero General Public Licence, version 3.

Publisher is designed to work with the Superdesk newsroom management system from Sourcefabric, but it can also be adapted to work with any compatible API. Publisher is a lightweight PHP 7 renderer for HTTP-pushed content in HTML/CSS/JavaScript templates, and it runs on a standard web server or in a Docker container. A PostgreSQL database is also required.

The presentation of articles is taken care of by a flexible, device-responsive themes system, which can be customised to suit your publications.

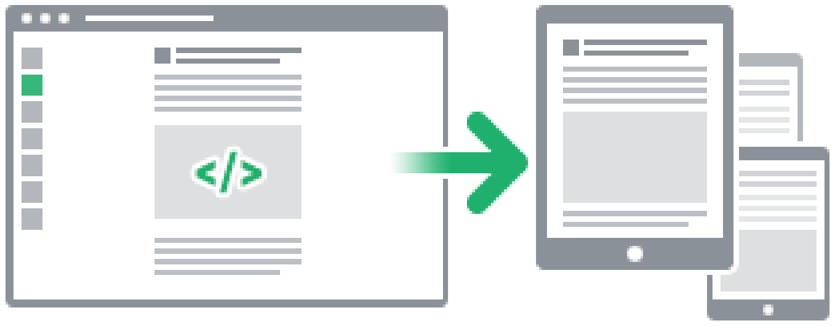
This documentation includes text and code examples from the Symfony and Sylius projects, released under the Creative Commons BY-SA 3.0 licence. Pull requests to improve the documentation are welcome.

# CHAPTER 1

# Manual

**Superdesk Publisher: 360° digital output on an industrial scale**

Publisher is the latest tool made for the Superdesk newsroom suite. It publishes multimedia content across multiple outputs, from websites to apps to outdoor displays to social media, and allows simultaneous, centralised monitoring and management of all of your assets.



Publisher is designed to complement Superdesk, which already powers content creation, production, distribution and curation at media businesses such as global news agencies and print/online newspapers. Superdesk is a structured-data polyglot and converses in formats as diverse as the legacy ANPA 1312 all the way to media-rich NewsML G2. But Publisher can also be extended to handle content created in third-party systems, even in exotic, custom formats.

We encourage any media outfit dealing with multiple outputs, platforms and channels to investigate the power of Publisher. Whether you author and produce content in Superdesk or not, Publisher gives you the real-time multi-tenancy overview you need to ensure that it performs optimally, everywhere.

## 1.1 Introduction

### 1.1.1 Why choose Publisher?

If your organisation already creates and produces content in Superdesk, Publisher is built to work with it natively. If not, but your back-end system or systems are still fit for purpose and your need is to manage a portfolio of digital assets (from multiple websites to apps to social feeds), Superdesk Publisher can be integrated with your legacy tools until their deprecation and replacement.

### 1.1.2 What is Publisher's focus?

- Efficient content delivery to multiple digital channels.
- Rapid development of new website and digital layouts, independently from back-end systems to avoid disrupting editorial workflows.
- All the latest embeds and custom widgets for your web pages.
- Full commercial support from the upstream development and implementation teams.

### 1.1.3 How is Publisher structured?

This graphic shows how the Publisher is configured.

manual/publisher-architecture.png

- Data: Symfony uses Postgres and Doctrine.
- Basic frameworks: Publisher is based on Symfony and the Symfony CMF.
- Publisher: The rest is Publisher

### 1.1.4 Notable features

- Multitenancy
- Built-in support for reverse proxy caching (Varnish, Nginx, internal system as a fallback)
- Live-site management
- Widget support (managed by editors)
- WebSocket communication handling, mainly to refresh views

### 1.1.5 Website architecture

A basic theme must have the following structure:

```
ExampleTheme/                <=== Theme starts here
    views/                   <=== Views directory
        article.html.twig
        base.html.twig
        category.html.twig
        home.html.twig
    translations/            <=== Translations directory
        messages.en.xlf
        messages.de.xlf
```

(continues on next page)

(continued from previous page)

```
    public/                  <=== Assets directory
        css/
        js/
        images/
    theme.json               <=== Theme configuration
```

More on *themes* in chapter *Themes*

## 1.2 Getting started

The Superdesk Publisher themes system is built on fast, flexible and easy-to-use Twig templates.

### 1.2.1 Setting up a new project

Demo Superdesk Publisher themes use Gulp workflow automation (http://gulpjs.com/).

To set up a working environment for theme development based on an existing demo theme, you can follow these steps:

- Fork and clone, or just download the theme from GitHub (https://github.com/SuperdeskWebPublisher/theme-dailyNews, https://github.com/SuperdeskWebPublisher/theme-magazine), or use the default Publisher theme (which is part of the main repo, https://github.com/superdesk/web-publisher/tree/master/src/SWP/Bundle/FixturesBundle/Resources/themes/DefaultTheme)
- Make sure Gulp is working on your system (for how to get it up and running, see here: https://github.com/gulpjs/gulp/blob/master/docs/getting-started.md)
- The Gulp file is already there in the theme, with all necessary methods already implemented. For development purposes, just fire the task "watch" and it will automatically:
  - compile and add all css/scss/sass changes from *public/css/* to *public/dist/style.css*
  - add all js changes from *public/js/* to *public/dist/all.js* file
- For applying changes for production, there is the task 'build' which will also minify css and js and add specific version to these files (to prevent browser caching issues). You can also manually run tasks sass, js, cssmin, jsmin, version, as well as sw (service worker steps that ensure proper pre-caching on the browser side).

To start from scrach, you should know the theme structure:

```
ExampleTheme/                <=== Theme starts here
    views/                   <=== Views directory
        home.html.twig
    translations/            <=== Translations directory
        messages.en.xlf
        messages.de.xlf
    public/                  <=== Assets directory
        css/
        js/
        images/
    theme.json               <=== Theme configuration
```

This comes from *SyliusThemeBundle*, "Flexible theming system for Symfony applications".

You may, if you wish, use Gulp to automate your workflow, or you can use some other system. For further information on this topic, see the chapter 'Themes'.

## 1.2.2 How to Install and Configure Superdesk Publisher with Superdesk

This guide describes the functions of Superdesk Publisher and Superdesk, along with the required steps to run both applications concurrently in a production environment on two **different** servers. (However, both applications can also work on a single machine.)

### How to Install Superdesk?

### Prerequisites:

- VPS or dedicated server: min 2GB RAM, 4GB Free space
- Ubuntu 16.04 server version installed

Superdesk can be installed using one command line script which can be found here: https://github.com/takeit/superdesk-install/blob/master/install

Run command:

```
sudo apt-get install curl -y
&& curl -s https://raw.githubusercontent.com/takeit/superdesk-install/master/install␣
↪| sudo bash
```

**Note:** If you see the following message:

```
The virtual environment was not created successfully because ensurepip is not
available. On Debian/Ubuntu systems, you need to install the python3-venv
package using the following command.
apt-get install python3-venv
You may need to use sudo with that command. After installing the python3-venv
package, recreate your virtual environment.
Failing command: ['/opt/superdesk/env/bin/python3', '-Im', 'ensurepip', ' -- upgrade',
↪ ' -- default-pip']
```

Run:

```
export LC_ALL="en_US.UTF-8"
export LC_CTYPE="en_US.UTF-8"
```

and then again execute the command:

```
sudo apt-get install curl -y
&& curl -s https://raw.githubusercontent.com/takeit/superdesk-install/master/install␣
↪| sudo bash
```

The above command will install all the required dependencies needed by Superdesk. Once this is done, the Superdesk will run on your server. You will be able to access it via your browser: `http://<ip_or_domain>`.

The default login credentials will be:

```
Username: admin
Password: admin
```

### How to Install Superdesk Publisher?

#### Prerequisites:

See here to read more about all requirements.

**Note:** In this guide the Superdesk Publisher will be installed on another server.

#### Setting up the server/VPS

##### 1. Install ElasticSearch

ElasticSearch v5.6 will be used. Run the following command to install ES:

```
curl -L -O https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.6.0.
↪deb &&
sudo dpkg -i elasticsearch-5.6.0.deb && sudo apt-get -y update &&
sudo apt-get -y install --no-install-recommends openjdk-8-jre-headless &&
sudo systemctl enable elasticsearch && sudo systemctl restart elasticsearch
```

The ElasticSearch should be running on port `9200`. You can run the command:

```
curl -s "http://localhost:9200"
```

to find out if all works fine.

##### 2. Install PostgreSQL

Run command:

```
sudo apt-get install postgresql postgresql-contrib -y
```

The default PostgreSQL user is `postgres`.

Set the default PostgreSQL user password:

```
sudo -u postgres psql postgres
\password postgres
```

Hit enter, and a prompt to type a new password will show up.

Type `\q` to exit the postgres console, once you type a new password.

##### 3. Install PHP-FPM 7.2

Let's install PHP-FPM 7.2 with all the required extensions:

```
sudo apt install software-properties-common
sudo LC_ALL=C.UTF-8 add-apt-repository ppa:ondrej/php
sudo apt update
sudo apt install -y php7.2-fpm php7.2-pgsql php7.2-gd php7.2-xml \
php7.2-intl php7.2-zip php7.2-mbstring php7.2-curl php7.2-bcmath
```

## 4. Configure PHP-FPM 7.2

Run command:

```
cd /etc/php/7.2/fpm/pool.d/ &&
sudo curl -s -O https://gist.githubusercontent.com/takeit/
↪2ee16ee50878eeab01a7ca11b69dec10/raw/e9eda2801ac3657495374fcb846c2ff101a3e070/www.
↪conf
&& sudo service php7.2-fpm restart
```

## 5. Install Nginx server

Run command:

```
sudo apt-get -y install nginx
```

## 6. Configure Nginx server

Run command:

```
cd /etc/nginx/sites-enabled/
&& sudo curl -s -O https://gist.githubusercontent.com/takeit/
↪9c895b4d59930a9b550a43a0d26c0e0e/raw/bff973443d244929c8deda70f97b4ae862d9158b/
↪default
&& sudo service nginx restart
```

## 7. Install RabbitMQ server

Run command:

```
sudo apt install -y rabbitmq-server
```

## 8. Install Supervisor

Run command:

```
sudo apt-get install -y supervisor
```

Before starting the installation make sure your server meets all the requirements listed above.

## Completing the Superdesk Publisher installation

The Superdesk Publisher repository can be found on GitHub.

From there the source code can be downloaded and the Superdesk Publisher can be installed on your server.

Follow the guide below.

Assumed our server has `192.168.0.102` IP address. You can change it to your own IP or domain name. But in this guide we will use `192.168.0.102` IP for Superdesk Publisher instance. Superdesk instance will run using `192.168.0.101` IP address.

## 1. Install Composer

```
cd ~/
curl -sS https://getcomposer.org/installer | php
sudo mv composer.phar /usr/local/bin/composer
```

## 2. Download the source code

The default directory where the Publisher source code will be downloaded can be `/var/www/publisher` and all console commands need to be executed inside that directory starting from now on.

Run commands in your terminal:

```
cd /var/www/ && sudo git clone https://github.com/superdesk/web-publisher.git␣
↪publisher
```

Install Superdesk Publisher source code dependencies:

```
HTTPDUSER=$(ps axo user,comm | grep -E '[a]pache|[h]ttpd|[_]www|[w]ww-data|[n]ginx' |␣
↪grep -v root | head -1 | cut -d\  -f1)
&& sudo chown -R "$HTTPDUSER":"$HTTPDUSER" publisher/ && cd publisher
&& sudo -u www-data SYMFONY_ENV=prod composer install --no-dev --optimize-autoloader
```

All the source code dependencies will start to install. Once it is done, you will be asked to fill the `parameters.yml` file which needs to be completed before proceeding.

If you don't know what to set, just simply use default values by hitting "enter" and replace the content of `/var/www/publisher/app/config/parameters.yml` file with:

```
# This file is auto-generated during the composer install
parameters:
    env(DATABASE_HOST): 127.0.0.1
    env(DATABASE_PORT): null
    env(DATABASE_NAME): publisher
    env(DATABASE_USER): postgres
    env(DATABASE_PASSWORD): postgres
    env(DATABASE_SERVER_VERSION): 9
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
    env(SYMFONY_SECRET): SuperSecretTokenPleaseChangeIt
    swp_updater.version.class: SWP\Bundle\CoreBundle\Version\Version
    env(SWP_DOMAIN): 192.168.0.102 # server domain/IP where Superdesk Publisher is␣
↪installed
    cache_servers:
        - 192.168.0.102 # server domain/IP where Superdesk Publisher is installed
    doctrine_cache_driver: array
    sentry.dsn: false
    session_memcached_host: localhost
    session_memcached_port: 11211
    session_memcached_prefix: sess
    session_memcached_expire: 3600
    test_env: doctrine
    allow_origin_cors: '*'
    superdesk_servers:
```

(continues on next page)

(continued from previous page)

```
            - 192.168.0.101 # server domain/IP where Superdesk is installed
    env(ELASTICA_HOST): localhost
    env(ELASTICA_PORT): 9200
    env(RABBIT_MQ_HOST): 127.0.0.1
    env(RABBIT_MQ_PORT): 5672
    env(RABBIT_MQ_USER): guest
    env(RABBIT_MQ_PASSWORD): guest
```

And set proper permissions for `cache` and `logs` directories, run:

```
sudo setfacl -dR -m u:"$HTTPDUSER":rwX -m u:$(whoami):rwX app/cache app/logs
&& sudo setfacl -R -m u:"$HTTPDUSER":rwX -m u:$(whoami):rwX app/cache app/logs
```

### 3. Check Requirements

Check if your server meets the requirements by running:

```
php app/check.php
```

If everything is in order, you should see this message: `Your system is ready to run Symfony projects` on your screen.

### 4. Create the Database and Update the Schema

Inside `/var/www/publisher` directory, run the command to create the database:

```
SYMFONY_ENV=prod php app/console doctrine:database:create
```

And populate the database with the schema, run:

```
SYMFONY_ENV=prod php app/console doctrine:migrations:migrate --no-interaction
```

### 5. Create organization

```
SYMFONY_ENV=prod php app/console swp:organization:create Publisher
```

### 6. Create tenant

```
SYMFONY_ENV=prod php app/console swp:tenant:create <organization_code> 192.168.0.102␣
↪Testing
```

Where `<organization_code>` is the organization code generated by the previous command and `192.168.0.102` is your IP/domain name which points to the server where Superdesk Publisher is installed. Replace it with your and appropriate data.

### 7. Install theme

```
sudo -u www-data SYMFONY_ENV=prod php app/console swp:theme:install <tenant_code> src/
↪SWP/Bundle/FixturesBundle/Resources/themes/DefaultTheme/ -f --activate -p
```

`<tenant_code>` is the Tenant's code generated by previous command. Replace it with the proper value.

**Install theme assets:**

```
sudo -u www-data SYMFONY_ENV=prod php app/console sylius:theme:assets:install
```

### 8. Run supervisor

```
sudo -u www-data SYMFONY_ENV=prod php app/console rabbitmq-supervisor:build --env=prod
```

The Superdesk Publisher should be running and be accessible using your remote server IP, `192.168.0.102` in this case.

### 9. Clear the cache

Run command:

```
SYMFONY_ENV=prod php app/console cache:clear --env=prod
```

## How to Configure Superdesk Publisher with Superdesk?

Now that the Superdesk and Superdesk Publisher applications are installed, it is possible to enable Superdesk Publisher Component inside the Superdesk UI.

Superdesk Publisher Component is a JavaScript component that is a separate dependency and can be included in Superdesk in order to manage Superdesk Publisher application.

The source code of this component can be found at GitHub.

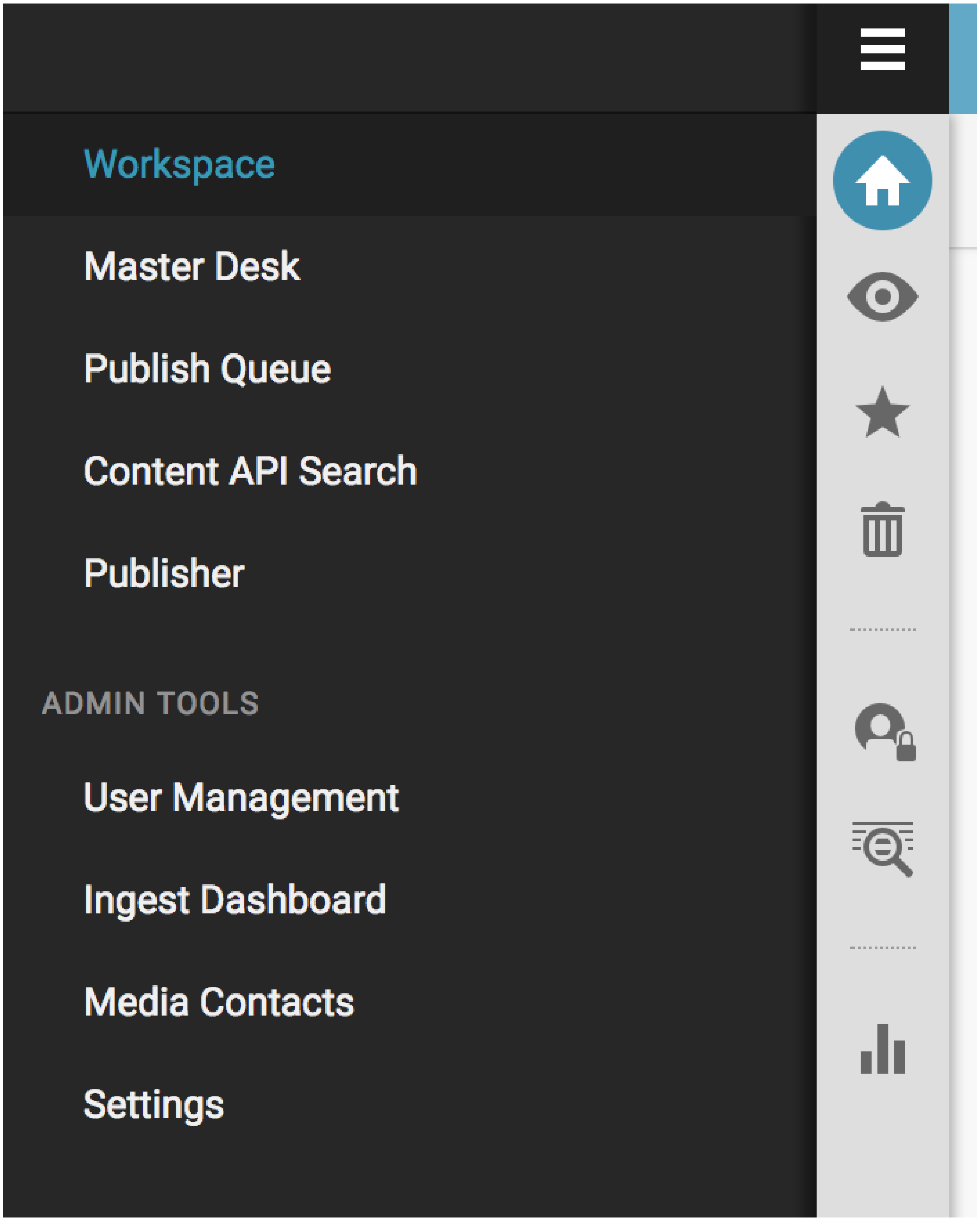
### 1. Update Configuration File

Login to the server where the Superdesk is installed.

Inside `/opt/superdesk/client/dist` directory on your server open the `config.js` and `config.<hash>.js` (e.g. `config.23fr4.js`) files and override the content with the text as below:

```
window.superdeskConfig={
    apps: ['superdesk-publisher'],
    publisher: {
        protocol: "http",
        tenant: '', // subdomain
        domain: '192.168.0.102', // IP address or domain name of your server where␣
↪Superdesk Publisher is installed
        base: 'api/v1'
    },
};
```

That's it! Now, when you log in to Superdesk in the left hamburger menu, you will see the Publisher menu item available:



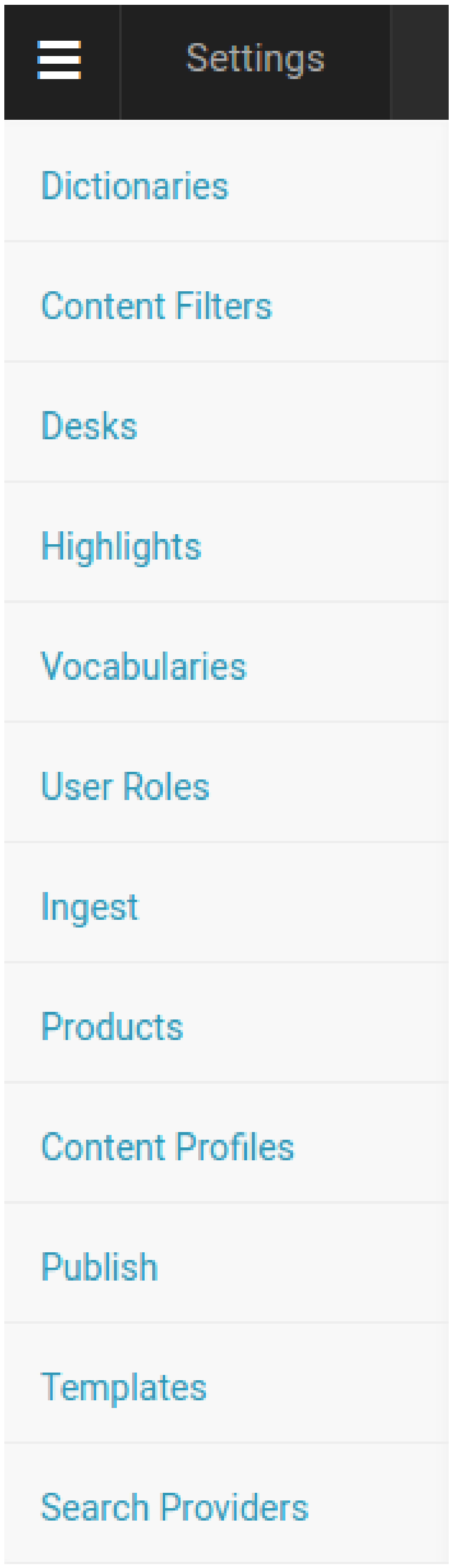
### 2. Configure Subscriber to Publish Content from Superdesk to Superdesk Publisher

You can read more about this in the official Superdesk Publisher documentation.

Thank you for reading to the end of this post! If you liked what you saw, please give us a pat us on the back by starring our project on Github: https://github.com/superdesk/web-publisher.

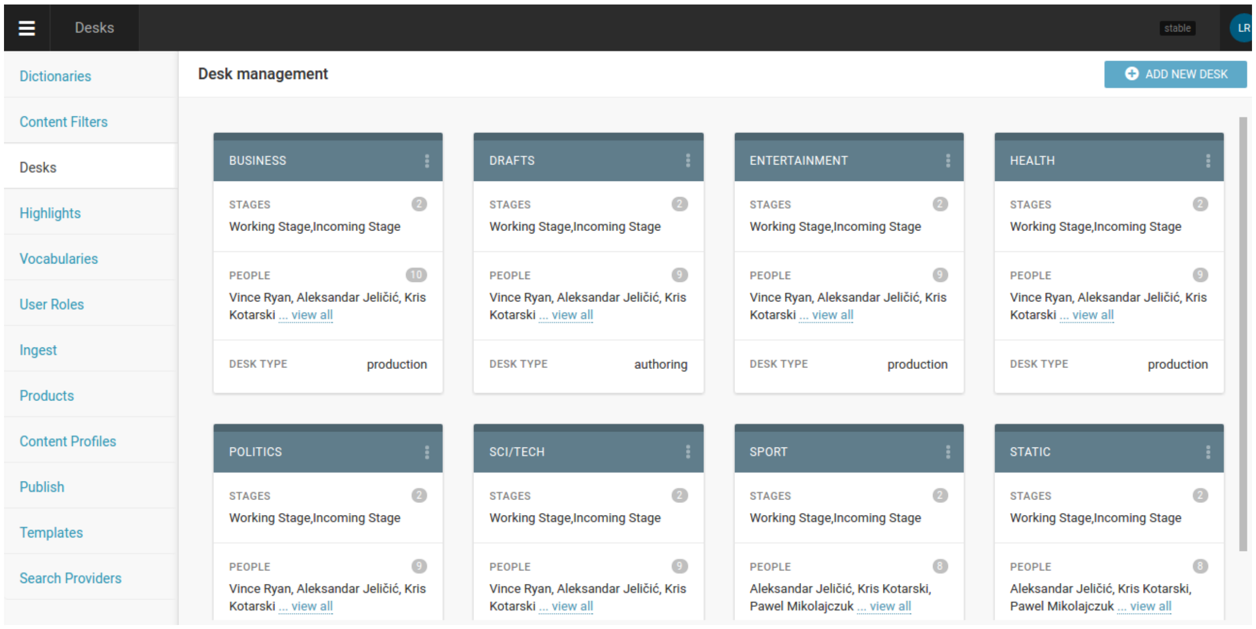
### 1.2.3 Superdesk configuration

As being such flexible and powerful beast, Superdesk needs to be configured correctly in order to deliver right content to right output channel, in appropriate formats and to desired paths based on set-up rules.



We will of course focus only on those settings directly related to successful publishing to Publisher-powered website.

## Desks



BUSINESS
STAGES 2
Working Stage,Incoming Stage
PEOPLE 10
Vince Ryan, Aleksandar Jeličić, Kris Kotarski ... view all
DESK TYPE production

DRAFTS
STAGES 2
Working Stage,Incoming Stage
PEOPLE 9
Vince Ryan, Aleksandar Jeličić, Kris Kotarski ... view all
DESK TYPE authoring

ENTERTAINMENT
STAGES 2
Working Stage,Incoming Stage
PEOPLE 9
Vince Ryan, Aleksandar Jeličić, Kris Kotarski ... view all
DESK TYPE production

HEALTH
STAGES 2
Working Stage,Incoming Stage
PEOPLE 9
Vince Ryan, Aleksandar Jeličić, Kris Kotarski ... view all
DESK TYPE production

POLITICS
STAGES 2
Working Stage,Incoming Stage
PEOPLE 9
Vince Ryan, Aleksandar Jeličić, Kris Kotarski ... view all

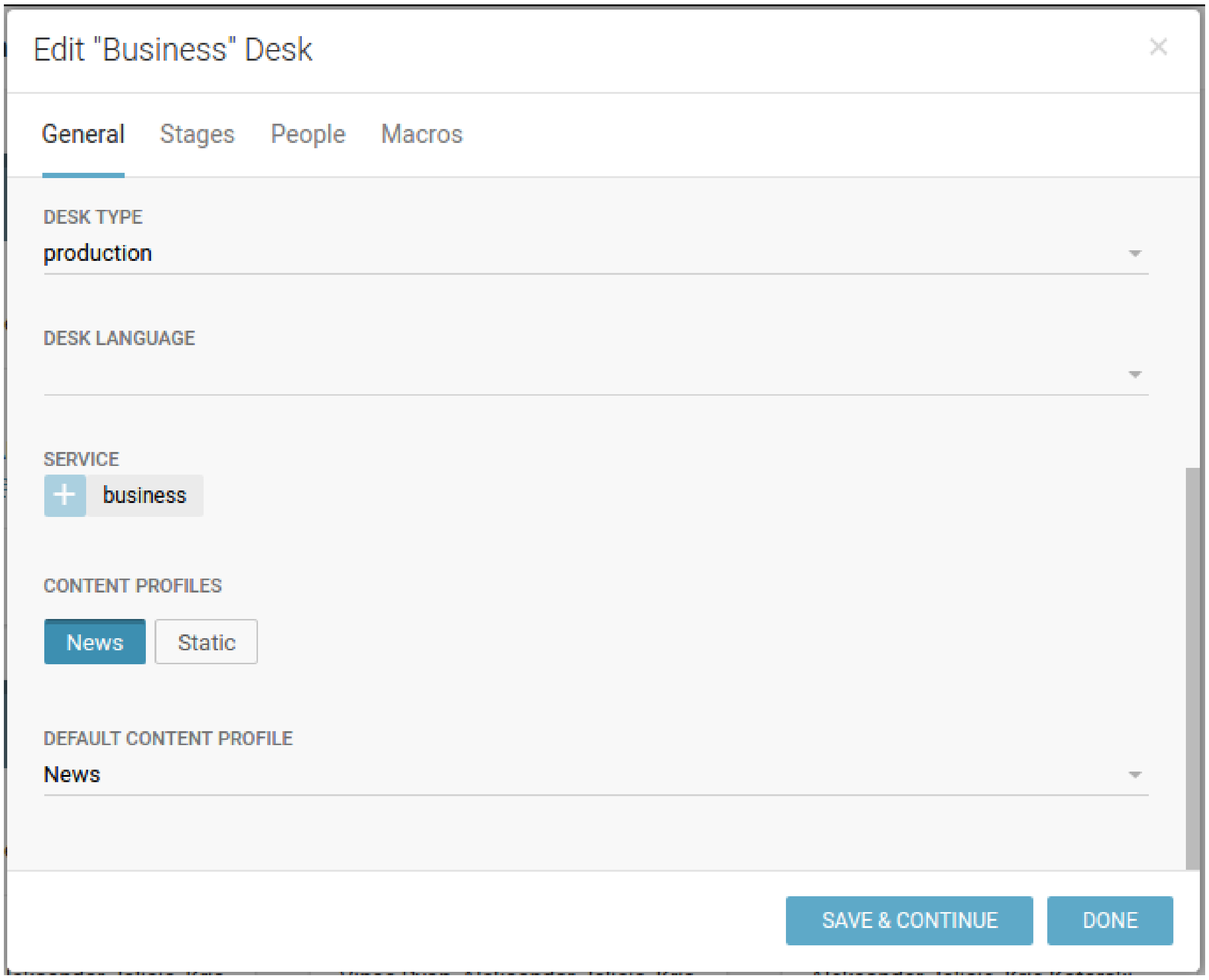
SCI/TECH
STAGES 2
Working Stage,Incoming Stage
PEOPLE 9
Vince Ryan, Aleksandar Jeličić, Kris Kotarski ... view all

SPORT
STAGES 2
Working Stage,Incoming Stage
PEOPLE 8
Aleksandar Jeličić, Kris Kotarski, Pawel Mikolajczuk ... view all

STATIC
STAGES 2
Working Stage,Incoming Stage
PEOPLE 8
Aleksandar Jeličić, Kris Kotarski, Pawel Mikolajczuk ... view all

Concept of Superdesk desks can be used in different ways - they can be created to represent website categories (Politics, Business etc), but also there can be just one or two desks that divide content by its type (news, static pages etc). So based on the logic how desks are used, they can be configured to use specific content type, and to publish content to specific route by default - but don't have to (these decisions, to which route to publish and wich content type to use, can be made separately for every new content item that you create).

Desk settings (accessible upon clicking three-dots icon in desk's right top corner) can be used for this kind of semi-automation:


Edit "Business" Desk

DESK TYPE: production

DESK LANGUAGE:

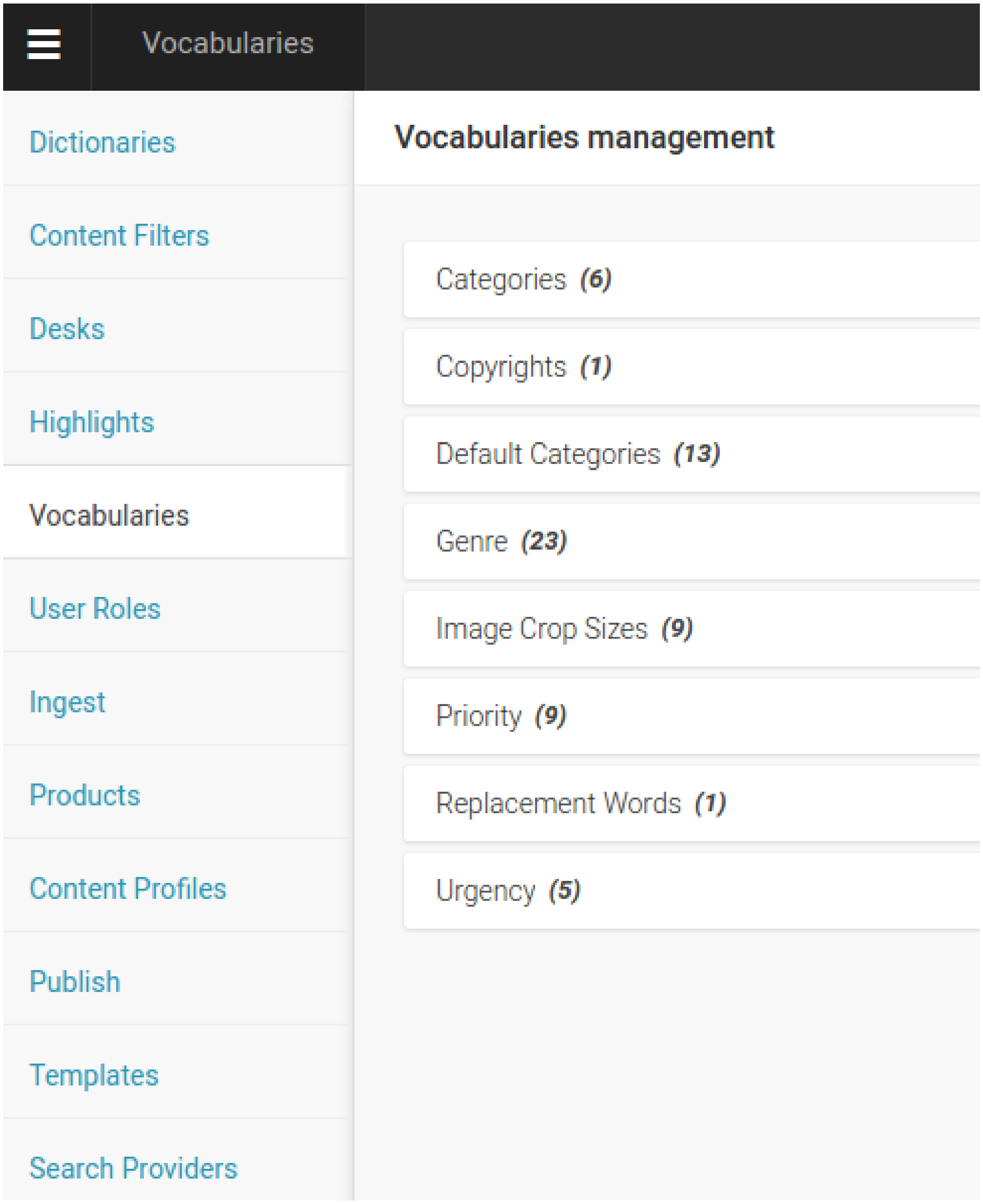
SERVICE: business

CONTENT PROFILES: News, Static

DEFAULT CONTENT PROFILE: News

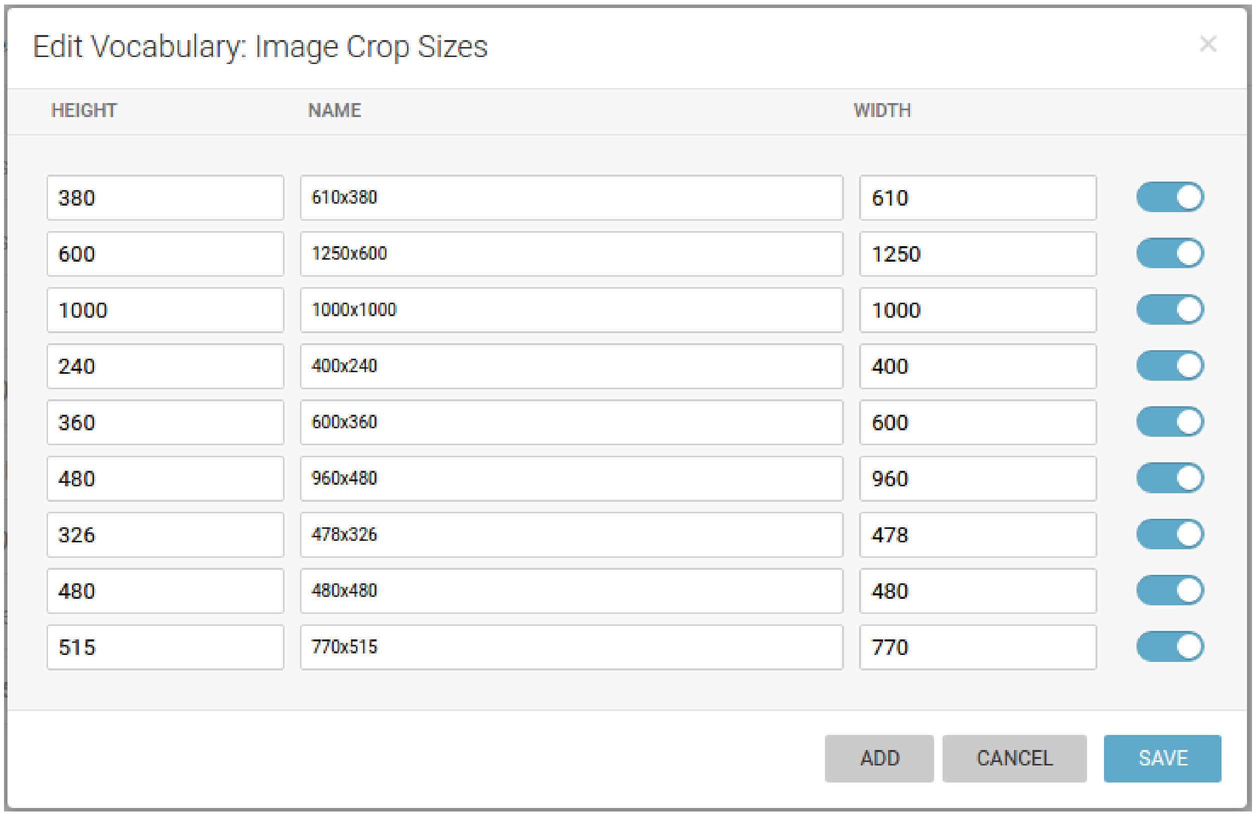
*Service* is directive that tells system to which route to publish by default (and these are created inside Vocabularies/Categories). *Content profiles* is list of configured types of content, and one can be used to be desk's default.

#### Vocabularies management



**Categories** can be used to define *services* (see above) which are assigned to Desks.

**Image Crop Sizes** are definitions of image sizes that are used on your output channels. For example, there can be three websites configured with *Web Site Manager*, and each of them uses several cropping dimensions - lead story on front, second-level stories on front page, story thumbnails, article page big image, etc.



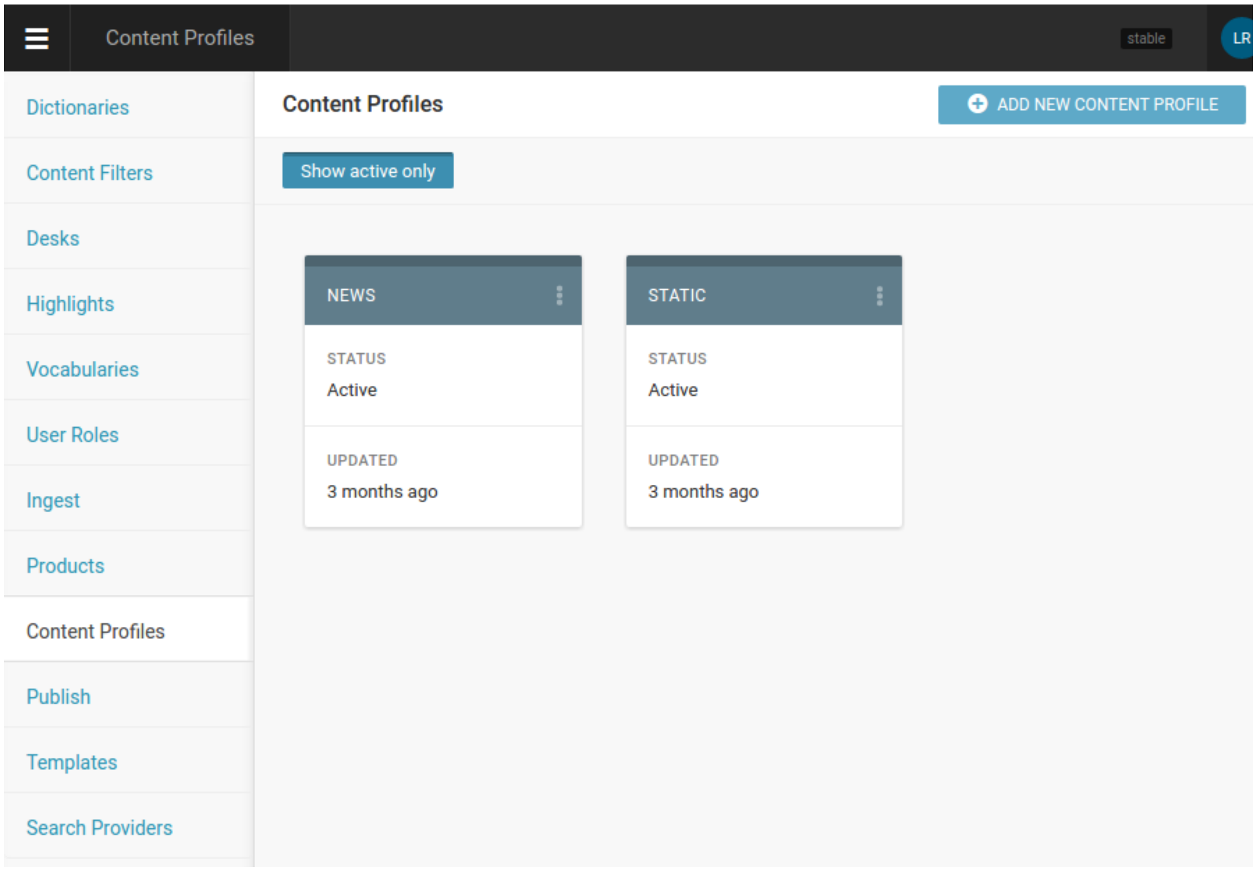
Edit Vocabulary: Image Crop Sizes

| HEIGHT | NAME | WIDTH |
|---|---|---|
| 380 | 610x380 | 610 |
| 600 | 1250x600 | 1250 |
| 1000 | 1000x1000 | 1000 |
| 240 | 400x240 | 400 |
| 360 | 600x360 | 600 |
| 480 | 960x480 | 960 |
| 326 | 478x326 | 478 |
| 480 | 480x480 | 480 |
| 515 | 770x515 | 770 |

And this is exactly what we can configure in *Image crop sizes* settings; original images that are uploaded for the story will be cropped to all these different recangular or square crops and delivered to output channels.

This process is automatic, but can be overviewed by uploaders and editors to be sure that best *point of interest* and *area of interest* are chosen for every single crop size.

### Content profiles



*Content profiles* are the types of content that are being created in Superdesk. They can be dead simple or pretty complex, depending on your needs and nature of the content being created. On our screenshot example above, you can see common situation when output channel receives 'news' articles and 'static' pages.

Schema configuration possibilities for content profile are numerous; this is the list of them all:

- ANPA Category
- Company codes
- Genre
- Byline
- Slug
- Headline
- Priority
- Footer
- SMS
- Abstract
- Feature Media
- Media Description
- Dateline

- Sign Off
- Body footer
- Body HTML
- Editorial Note
- Urgency
- Subject
- Place
- Take Key

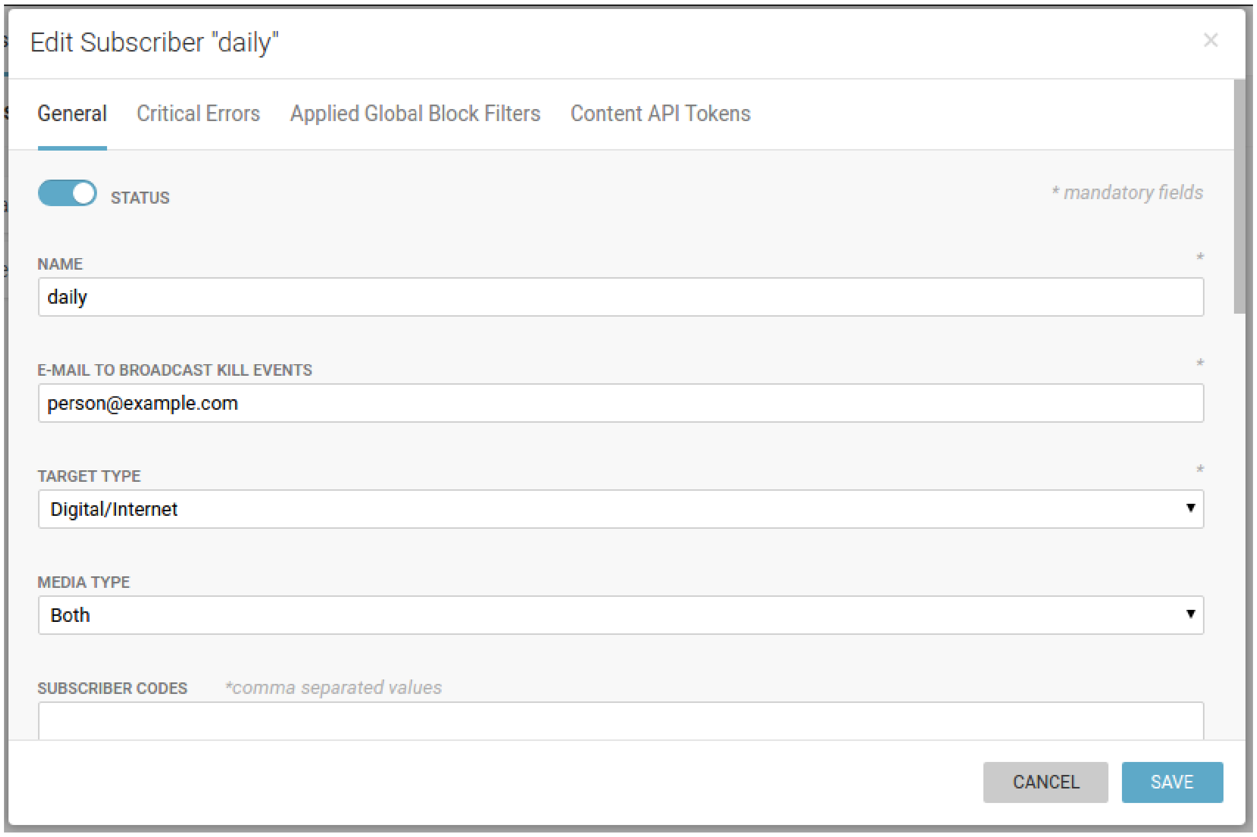
Every of these fields can be fine-tuned to suite your needs - from *length*, *formatting* options and *order*, to *setting its mandatoryness* on or off and *clean paste html* for example. We encourage you to explore these possibilities on your own!

Keep in mind that Superdesk can also deliver **content packages** to outside world. And package can be combination of more content items, every one of different profile.

## Publish

This is where we tell superdesk about our output channels, or **subscribers** (because Superdesk is used not only to send content to Publisher - it is built to serve *all* needs of even biggest news agancies and media companies, *subscriber* can be another news producing company, or anything really, so package they receive may be far more complex and different from only web site publisher needs).

So we can configure only one *subscriber* with one website, or more subscribers - superdesk can feed them all with content that fits their requirements.


Edit Subscriber "daily"

STATUS

* mandatory fields

NAME *
daily

E-MAIL TO BROADCAST KILL EVENTS *
person@example.com

TARGET TYPE *
Digital/Internet

MEDIA TYPE
Both

SUBSCRIBER CODES *comma separated values*

Explanation of options (Publisher-desired values are marked with *):

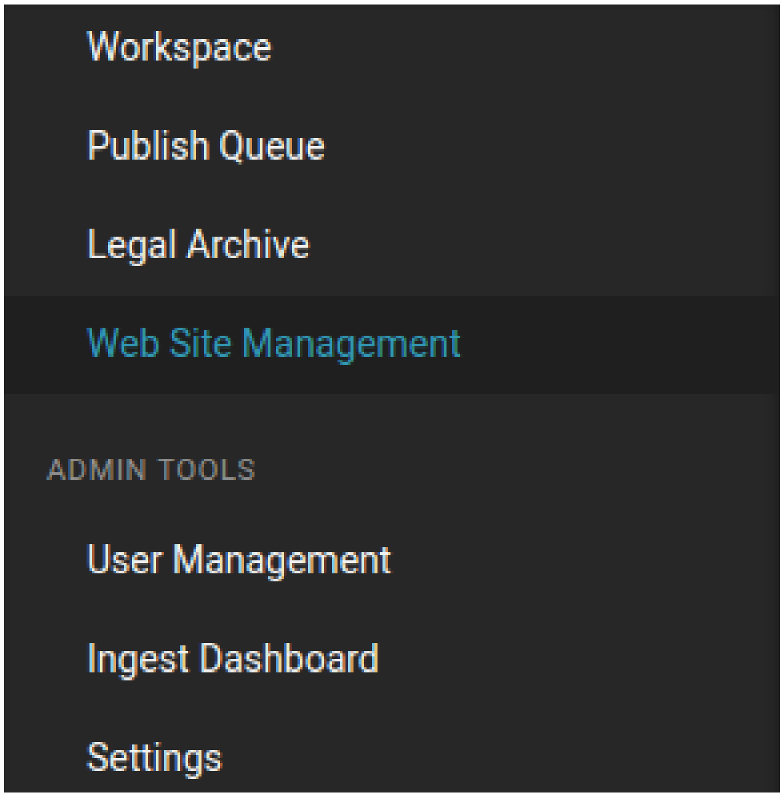
- Status: active or not
- Name
- E-mail to broadcast kill events: kill events are articles that are *killed* in Superdesk and need to be unpublished from output channel. This is automatic process
- Target type: choose between Digital/Internet*, Wire/Paper, or All
- Media type: choose between Media, Non-media or Both*
- Targetable by users: on* or off, option to manually chose if single content item is published to that subscriber
- Products: need to be pre-defined in *Products* section (just create anythnig, name it ie. 'demo' - for Publisher purposes it's not relevant and filling the *name* field is enoguh), but is required by Superdesk to have it)
- **Destinations**
  - Name
  - Format: choose between NINJS*, NewsML G2 or E-mail
  - Delivery type: FTP, E-mail, ODBC, File, Pull or HTTP Push*
  - Package individual items: switch on*/off
  - Resource URL: API endpoint to POST content, https://daily.example.com/app_dev.php/api/v1/content/push
  - Assets URL: https://daily.example.com/app_dev.php/api/v1/assets/push

Destinations can be multiple - in that case same content is sent to more destinations. In real life, it means that one publisher, for example, can have two different websites that serve same content.

Other tabs in Publish dialogue are not relevant for publishing content with Publisher - even the last tab in *Edit Subscriber* dialogue, **Content API Tokens**, as Publisher is getting content from superdesk by **http push** and is not using Superdesk Content API.

### 1.2.4 Configuring Publisher

Publisher integrates into Superdesk simply by adding new option *Web Site Management* to its main left-hand sidebar navigation.



This option, when chosen, opens Publisher configuration which allows setting one or more websites. Setting a website actually means defining routes, creating navigation menus (whose menu items are linked to these routes), and creating content lists.

Detailed explanation of website management steps can be found in chapter *Admin interface*

*Navigation* and *Content Lists* are also managable through *LiveSite Editing*

## 1.2.5 Containers and widgets

Containers are intended to give editors more control over templates. Properly implemented, they can transform a theme.

### What is a Container?

A template file can optionally have one or more containers, *block* elements that can be overriden in specific places. For example, article sidebar content, footer, or front page content blocks. This same container can be placed in many different templates, or even many times in the same template.

Every container can have default parameters and content, and can be hidden when not needed. The container twig tag keeps HTML syntax always up to date with JavaScript live management expectations. Container default values can be overridden by widgets.

### What is a Widget?

Widgets can be attached to a container in any order. Many types of widget can represent different features, for example:

- Newsletter signup form

- Social media components, like a page widget or comments widget
- Simple HTML widget with your own custom HTML rendered by widget
- Airtime player widget

### How to create a new type of widget?

To create a new type of widget, you create a new class in `/src/SWP/Component/TemplatesSystem/Gimme/Widget` which extends the `AbstractWidgetHandler.php` in that same directory.

As well as having to implement the render function, you can define what parameters you're expecting in the widget model by adding a static variable to your class called `$expectedParameters`.

For example:

```
protected static $expectedParameters = array(
    'parameter_name' => [
        'type' => 'string',            // or bool, int, float
        'default' => 'default_value'   // if no default is provided, the parameter␣
↪must be set
    ]
);
```

### Do widgets and containers work with caches?

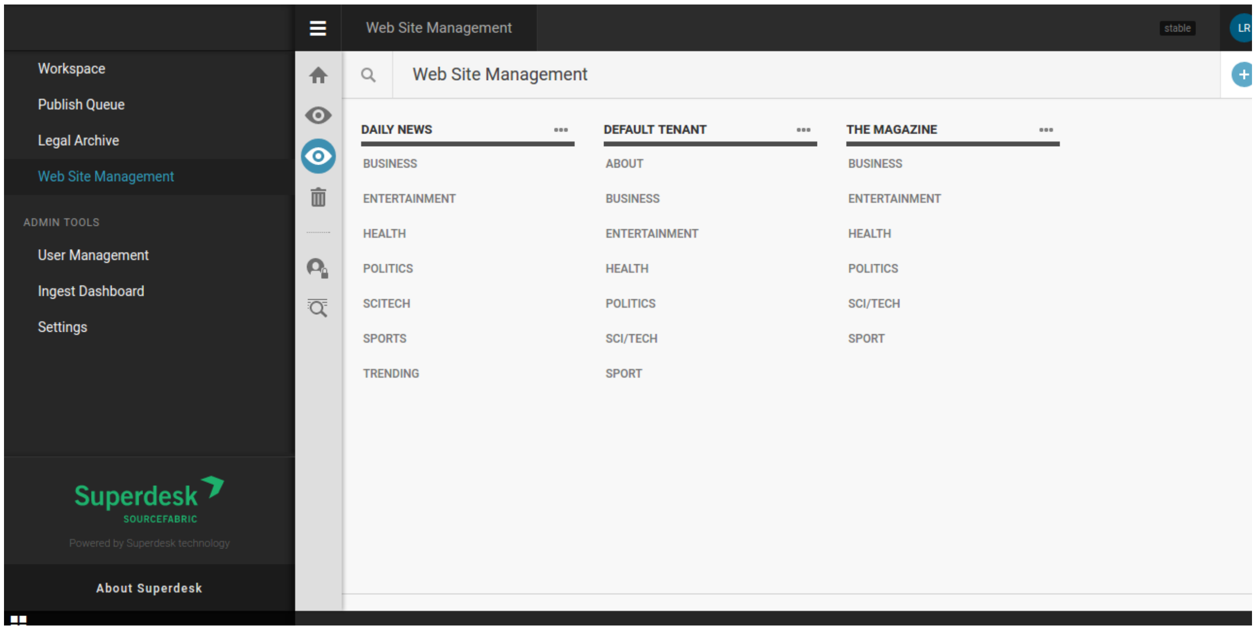
Yes, they are designed to work well with all caching systems used by Publisher.

## 1.2.6 Managing of the websites and routes

When Publisher is installed, it is integrated in Superdesk and it expects output channels (or, in other words - websites) to be set - it is already mentioned in *Configuring Publisher*.

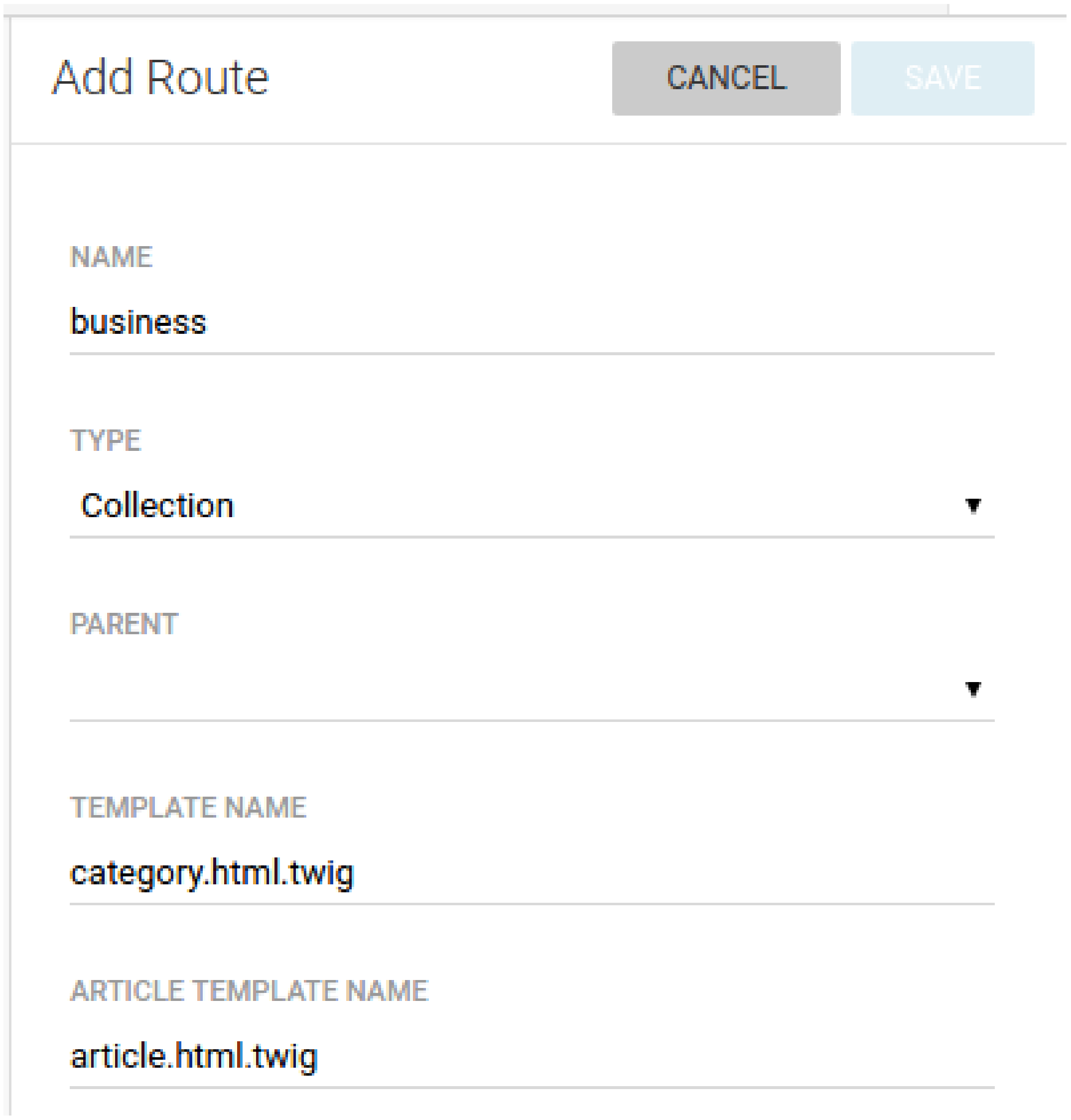
Main concepts are:

- there can be more than one websites that you can configure and feed from Superdesk
- each website is configured firstly by its **routes**; routes can be of type *collection* and *content*.



| DAILY NEWS | DEFAULT TENANT | THE MAGAZINE |
|---|---|---|
| BUSINESS | ABOUT | BUSINESS |
| ENTERTAINMENT | BUSINESS | ENTERTAINMENT |
| HEALTH | ENTERTAINMENT | HEALTH |
| POLITICS | HEALTH | POLITICS |
| SCITECH | POLITICS | SCI/TECH |
| SPORTS | SCI/TECH | SPORT |
| TRENDING | SPORT | |

Route of type *collection* is expected to get articles attached to it - think of it as some kind of category page (Business, or Politics, or simply News). When configuring such route, you need to also specify article template name - the one that will be used to show articles attached to that route.



NAME
business

TYPE
Collection

PARENT

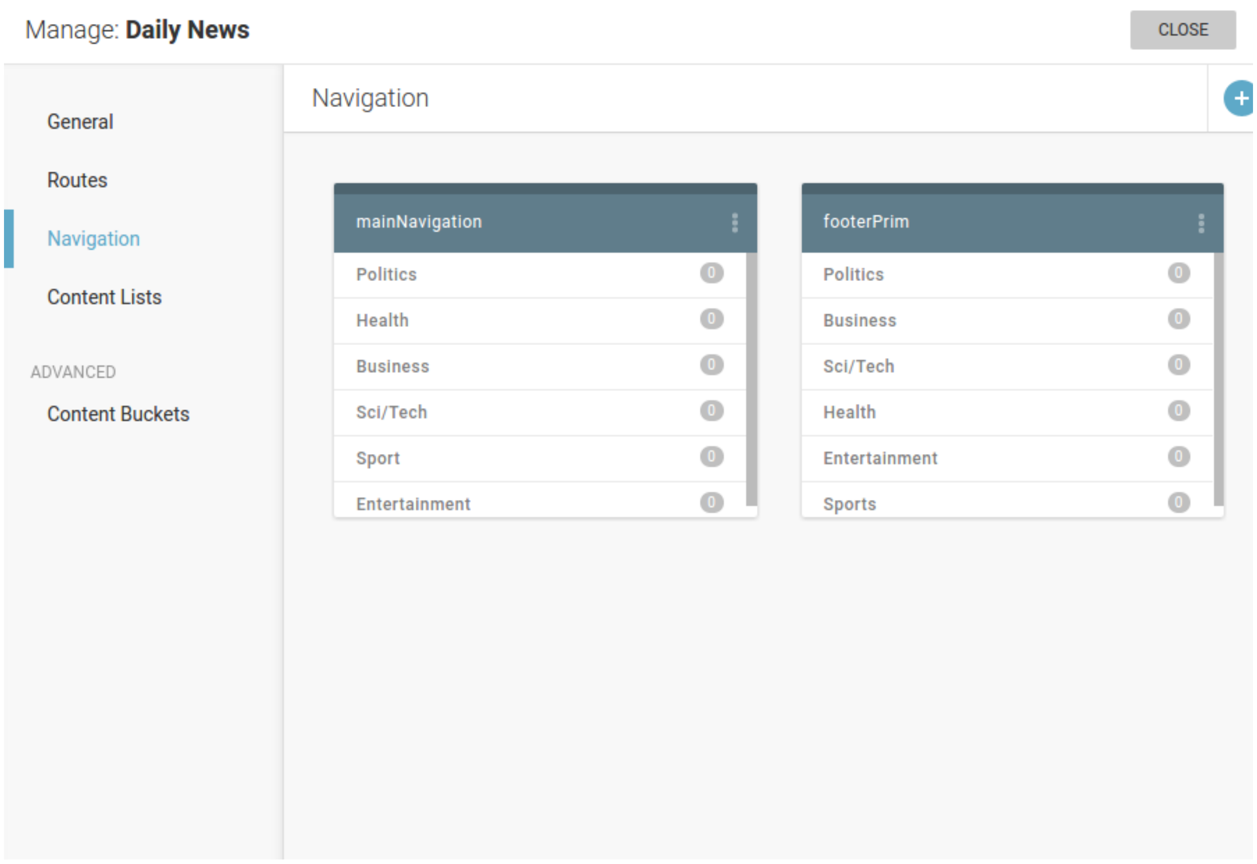
TEMPLATE NAME
category.html.twig

ARTICLE TEMPLATE NAME
article.html.twig

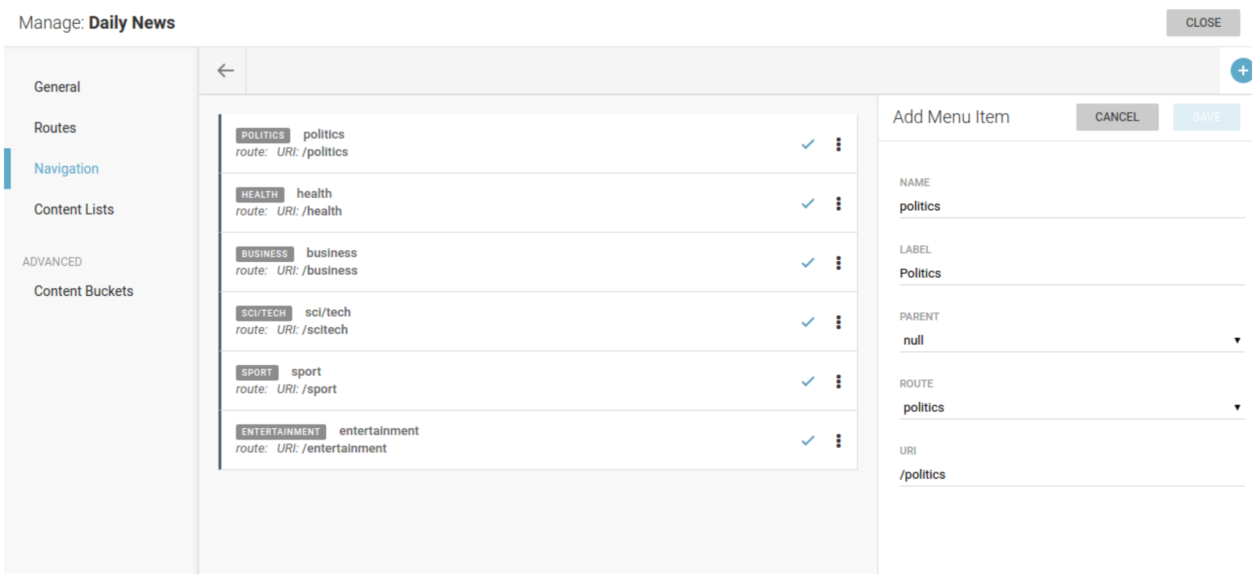
Route of type *content* is the end of the road - it holds the content! No articles are attached to it! So it can be either some special template (contact form, confirmation page, or simply a route that doesn't directly hold attached articles, like 'home' route for example).

### 1.2.7 Navigation management

**Navigations** are menus that you can use on your websites. Advantage of creating them here (and not building navigation menu only in template) is that it can later be managed (meaning menu items added, removed and reordered) even by website editors, and not template developers. And when we say managed, we mean managed either in superdesk Publisher interface, on directly on website frontend, using *LiveSite Editing*.



Only after configuring *Routes*, we can proceed to configure *Navigation*. That's because navigation is built of **menu items**, and menu items are partly defined by routes.



NAME: politics
LABEL: Politics
PARENT: null
ROUTE: politics
URI: /politics

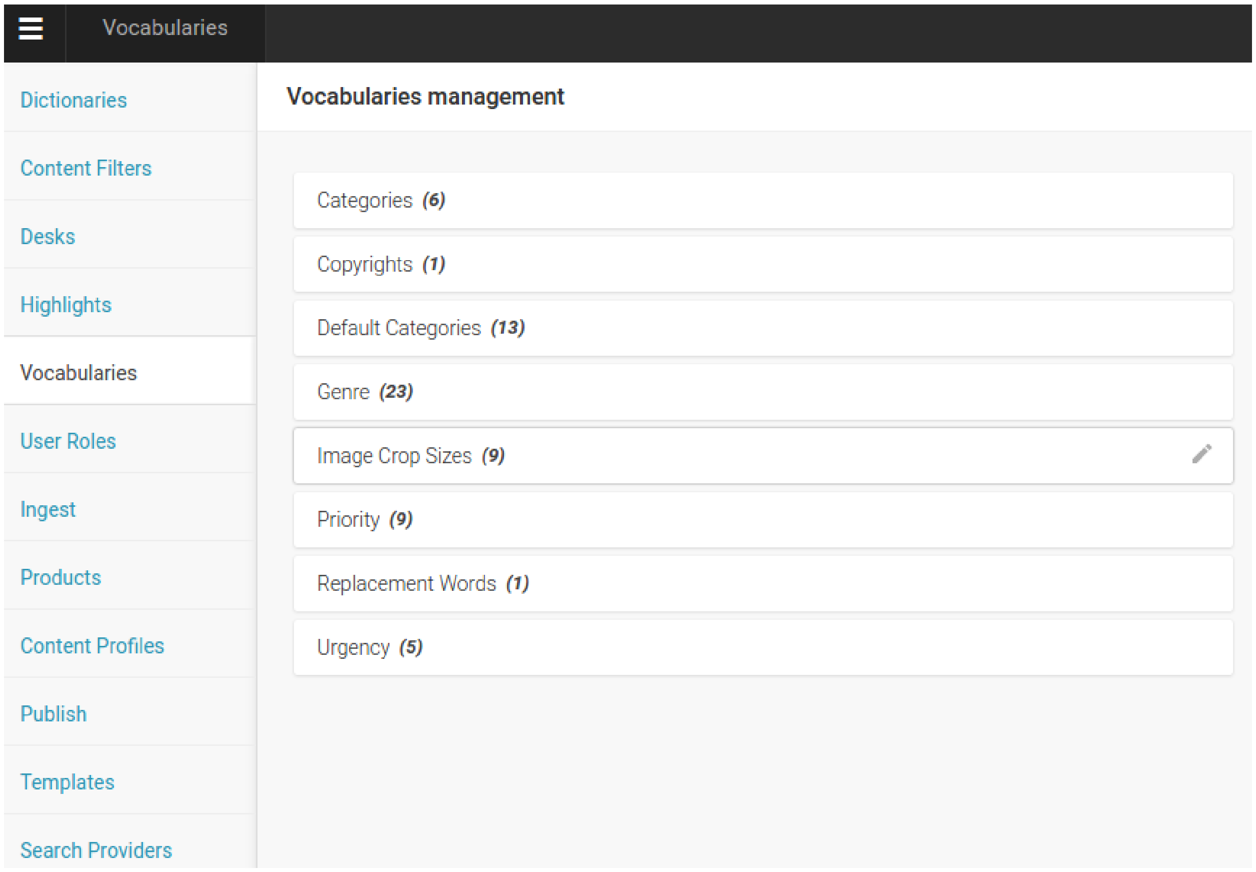
All elements of **menu item** definition are

- Name
- Label - value that is shown on frontend for that menu item in navigation menu

- Parent - useful when building nested, drop-down menus
- Route - one of previously defined routes
- Uri - automatically filled in when route is selected

### 1.2.8 Configure image formats

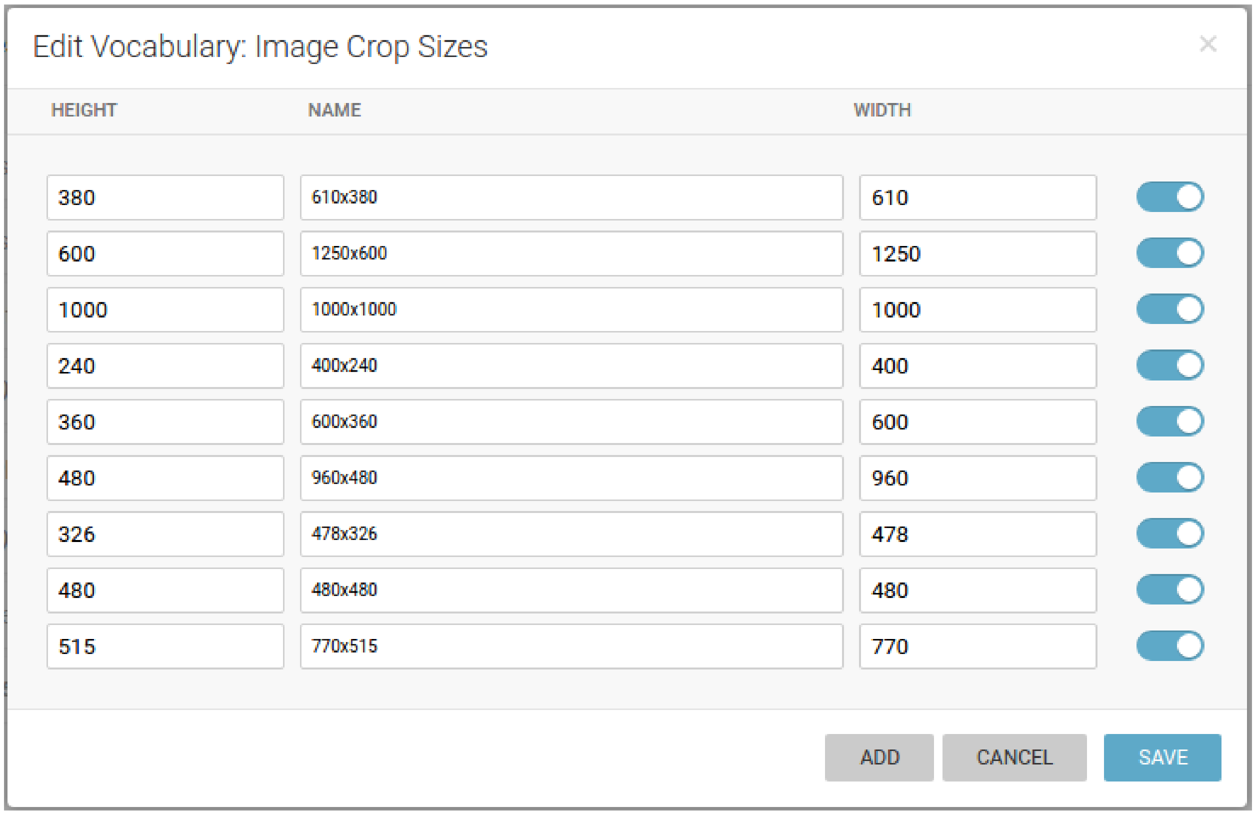
In Superdesk *Settings* -> *Vocabularies* -> *Image Crop sizes* it is possible to define all needed image sizes.



If you are configuring one website, depending on the layout design it can require one image size for lead story on front, another one for second-level stories, thumbnail image for small teasers, big article page image, and finally image size optimized for Facebook sharing for example.

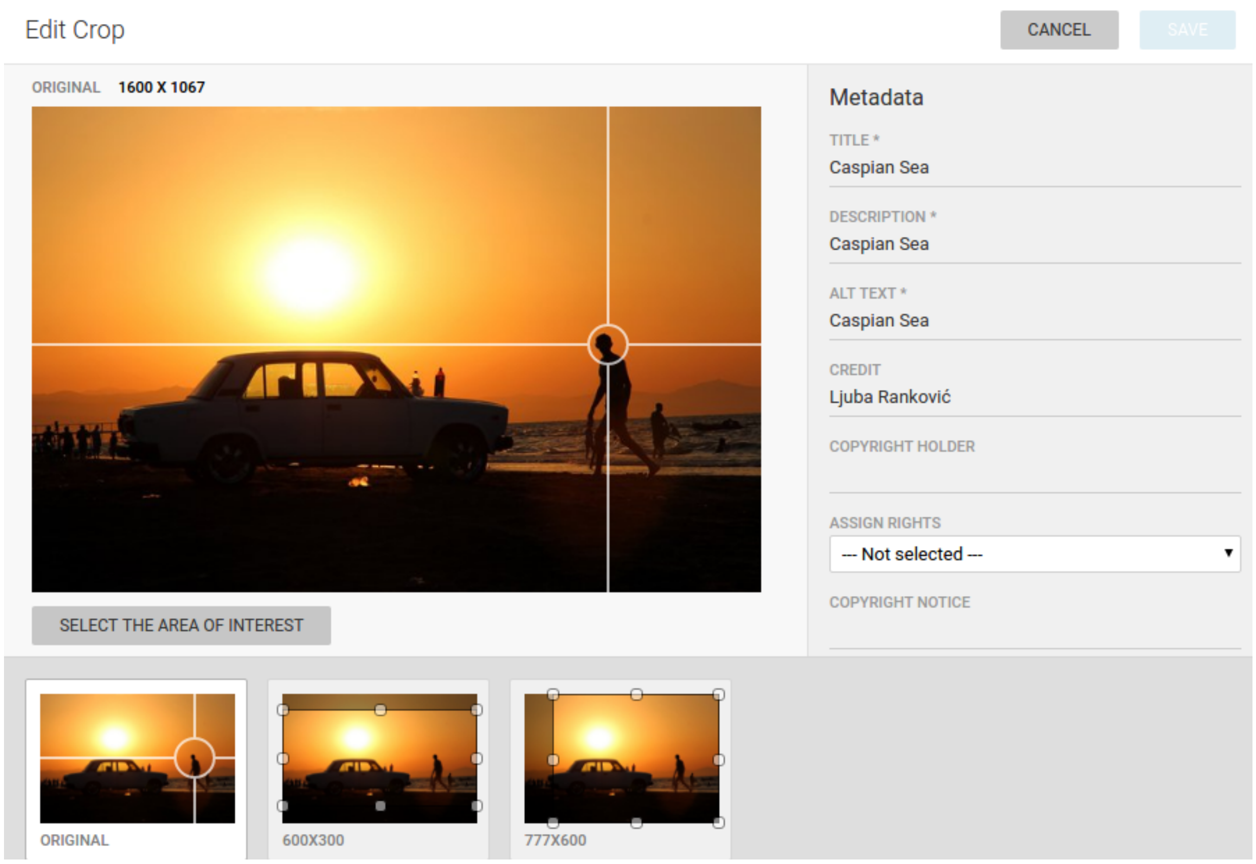
So all these sizes should be crops of originally uploaded image that journalist or editor are adding to the story as feature image (one that represents the story in teasers).

If you have more websites powered by one Superdesk instance, then all of the needed crops are defined here.


Edit Vocabulary: Image Crop Sizes

| HEIGHT | NAME | WIDTH |
|---|---|---|
| 380 | 610x380 | 610 |
| 600 | 1250x600 | 1250 |
| 1000 | 1000x1000 | 1000 |
| 240 | 400x240 | 400 |
| 360 | 600x360 | 600 |
| 480 | 960x480 | 960 |
| 326 | 478x326 | 478 |
| 480 | 480x480 | 480 |
| 515 | 770x515 | 770 |

Even though original image gets cropped automatically, this process can be overviewed and best *point of interest* and *area of interest* are customizable.



TITLE *
Caspian Sea

DESCRIPTION *
Caspian Sea

ALT TEXT *
Caspian Sea

CREDIT
Ljuba Ranković

COPYRIGHT HOLDER

ASSIGN RIGHTS
--- Not selected ---

COPYRIGHT NOTICE

## 1.3 Templates System

The Superdesk Publisher templates system has its own git repository, at: https://github.com/SuperdeskWebPublisher/templates-system

### 1.3.1 Rendering pages with Twig

Superdesk Publisher uses Twig templating engine to render website HTML. **Twig** is modern, flexible, extensible and secure templating system, and has great documentation, as well as active support community at Stack Overflow.

This is how Twig code looks like:

```
{% for user in users %}
* {{ user.name }}
{% else %}
   No users have been found.
{% endfor %}
```

If you are creating completely new theme for your Publisher project, or going to modify some of the existing demo themes, you can follow *this handy guide*.

Generally, if starting from scratch, we advize you to develop your HTML/CSS/JS first with some dummy content, and once it's ready, you can proceed with translating this markup into twig templates.

We have developed three demo themes which can serve as a refference for quick start (more about it *here*)

- *Superdesk Publisher demo theme*, located at */src/SWP/Bundle/FixturesBundle/Resources/themes/DefaultTheme* inside your Publisher instance (this theme is distributed as part of Publisher package)
- *The Modern Times theme*, whose Git repo is here: https://github.com/SuperdeskWebPublisher/theme-dailyNews
- *Magazine theme*, whose Git repo is here: https://github.com/SuperdeskWebPublisher/theme-magazine

### 1.3.2 Creating a page templates

Page template can render content in a straight-forward way: take article delivered to Publisher from Superdesk (or from any other source) and put its elements in markup of your choice (article title in *<h1>*, wrap main body in block-level element, etc).

But page can also be built using *containers* and *widgets* - in that case, editors can change article elements using *LiveSite Editing tool*.

More on that in our Cookbook Entry *Complex static pages with Containers and Widgets*

### 1.3.3 Creating custom templates

#### Setting error pages from theme

Publisher provides simple default templates for error pages. You can find them in `app/Resources/TwigBundle/views/Exception/` directory.

To override these templates from theme you need to create `TwigBundle/views/Exception/` directory in your theme, and put there new error pages files.

Example Structure:

```
ThemeName/
└─ TwigBundle/
   └─ views/
      └─ Exception/
         ├─ error404.html.twig
         ├─ error403.html.twig
         ├─ error500.html.twig
         ├─ error.html.twig      # All other HTML errors
```

#### Testing error pages during theme development

You can use URLs like

```
http://wepublisher.dev/app_dev.php/_error/404
http://wepublisher.dev/app_dev.php/_error/403
http://wepublisher.dev/app_dev.php/_error/500
http://wepublisher.dev/app_dev.php/_error/501 # error.html.twig will be loaded
```

to preview the error page for a given status code as HTML.

### 1.3.4 Properties

### 1.3.5 Templates features

### Custom Twig tags

Gimme allows you to fetch the Meta object you need in any place of your template. It supports single Meta objects (with `gimme` ) and collections of Meta objects (with `gimmelist`).

#### container

The `container` tag has one required parameter and one optional parameter:

- (required) container unique name, for example: *frontpage_sidebar*
- (optional) keyword `with` and default parameters for containers (used to create the container on theme installation).

Here is an example of a container tag:

```
{% container 'frontpage_sidebar' with {
    'width': 400,
    'height': 500,
    'styles': 'border: solid 1px red',
    'cssClass': 'css_class_name',
    'data': {'custom-key': value}
} %}
{% endcontainer %}
```

This container tag will render the HTML code:

```
<div id="frontpage_sidebar" class="swp_container css_class_name" style="width: 400px;␣
↪height: 500px; border: solid 1px red;" data-custom-key="value"></div>
```

The available container parameters are:

- [integer] width - container width
- [integer] height - container height
- [string] styles - container inline styles
- [string] cssClass - container class string
- [string] data - JSON object string with html-data properties (keys and values)

#### gimme

The tag `gimme` has one required parameter and one optional parameter:

- (required) Meta object type (and name of variable available inside block), for example: *article*
- (optional) Keword `with` and parameters for Meta Loader, for example: `{ param:  "value" }`

```
{% gimme article %}
    {# article Meta will be available under "article" variable inside block #}
    {{ article.title }}
{% endgimme %}
```

Meta Loaders sometimes require special parameters - like the article number, language of the article, user id, etc..

```
{% gimme article with { articleNumber: 1 } %}
    {# Meta Loader will use provided parameters to load article Meta #}
    {{ article.title }}
{% endgimme %}
```

### gimmelist

The `gimmelist` tag has two required parameters and two optional parameters:

- (required) Name of variable available inside block: `article`
- (required) Keyword `from` and type of requested Metas in collection: `from articles` with filters passed to Meta Loader as extra parameters (`start`, `limit`, `order`)
- (optional) Keyword `with` and parameters for Meta Loader, for example: `with {foo: 'bar', param1: 'value1'}`
- (optional) Keyword `without` and parameters for Meta Loader, for example: `without {source: 'AAP'}`
- (optional) Keyword `if` and expression used for results filtering
- (optional) Keyword `ignoreContext` and optional array of selected meta to be ignored

Example of the required parameters:

```
{% gimmelist article from articles %}
    {{ article.title }}
{% endgimmelist %}
```

Example with ignoring selected context parameters:

```
{% gimmelist article from articles ignoreContext ['route', 'article'] %}
...
```

Example with ignoring whole context

```
{% gimmelist article from articles ignoreContext [] %}
...
```

Or even without empty array

```
{% gimmelist article from articles ignoreContext %}
...
```

Example with filtering articles by metadata:

```
{% gimmelist article from articles with {metadata: {byline: "Karen Ruhiger", located:
↪"Sydney"}} %}
    {{ article.title }}
{% endgimmelist %}
```

The above example will list all articles by metadata which contain `byline` equals to `Karen Ruhiger` **AND** `located` equals to `Sydney`.

To list articles by authors you can also do:

```
{% gimmelist article from articles with {author: ["Karen Ruhiger", "Doe"]} %}
    {{ article.title }}
    Author(s): {% for author in article.authors %}<img src="{{ url(author.avatar) }}"␣
↪/>{{ author.name }} ({{ author.role }}) {{ author.biography }} - {{ author.jobTitle.
↪name }},{% endfor %}
{% endgimmelist %}
```

It will then list all articles written by `Karen Ruhiger` **AND** `Doe`.

To list articles from the `Forbes` source but without an `AAP` source you can also do:

```
{% gimmelist article from articles with {source: ["Forbes"]} without {source: ["AAP"]}
↪ %}
    {% for source in article.sources %} {{ source.name }} {% endfor %}
{% endgimmelist %}
```

It will then list all articles **with** source `Forbes` and **without** `AAP`.

Listing article's custom fields:

```
{% gimmelist article from articles %}
    {{ article.title }}
    {{ article.extra['my-custom-field'] }}
{% endgimmelist %}
```

Example with usage of all parameters:

```
{% gimmelist article from articles|start(0)|limit(10)|order('id', 'desc')
    with {foo: 'bar', param1: 'value1'}
    contextIgnore ['route', 'article']
    if article.title == "New Article 1"
%}
    {{ article.title }}
{% endgimmelist %}
```

### How to work with `gimmelist` pagination?

`gimmelist` is based on Twig `for` tag, like in Twig there is loop variable available. In addition to default loop properties there is also `totalLength`. It's filled by loader with number of total elements in storage which are matching criteria. Thanks to this addition we can build real pagination.

`TemplateEngine` Bundle provides simple default pagination template file: `pagination.html.twig`.

**Note:** You can override that template with `SWPTemplatesSystemBundle/views/pagination.html.twig` file in Your theme. Or You can use own file used for pagination rendering.

Here is commented example of pagination:

```
{# Setup list and pagination parameters #}
{% set itemsPerPage, currentPage = 1, app.request.get('page', 1) %}
{% set start = (currentPage / itemsPerPage) - 1 %}

{# List all articles from route '/news' and limit them to `itemsPerPage` value␣
↪starting from `start` value #}
{% gimmelist article from articles|start(start)|limit(itemsPerPage) with {'route': '/
↪news'} %}
```

(continues on next page)

(continued from previous page)

```

    <li><a href="{{ url(article) }}">{{ article.title }} </a></li>

    {# Render pagination only at end of list #}
    {% if loop.last  %}
        {#
            Use provided by default pagination template

            Parameters:
            * currentFilters (array) : associative array that contains the current␣
↪route-arguments
            * currentPage (int) : the current page you are in
            * paginationPath (Meta|string) : the route name (or supported by router␣
↪Meta object) to use for links
            * lastPage (int) : represents the total number of existing pages
            * showAlwaysFirstAndLast (bool) : Always show first and last link (just␣
↪disabled)
        #}
        {% include '@SWPTemplatesSystem/pagination.html.twig' with {
            currentFilters: {}|merge(app.request.query.all()),
            currentPage: currentPage,
            paginationPath: gimme.route,
            lastPage: (loop.totalLength/itemsPerPage)|round(1, 'ceil'),
            showAlwaysFirstAndLast: true
        } only %}
    {% endif %}
{% endgimmelist %}
```

For referrence, see original *pagination.html.twig* template (if you want to customize it and use instead of default one):

```
{#
  Source: http://dev.dbl-a.com/symfony-2-0/symfony2-and-twig-pagination/
  Updated by: Simon Schick <simonsimcity@gmail.com>

  Parameters:
    * currentFilters (array) : associative array that contains the current route-
↪arguments
    * currentPage (int) : the current page you are in
    * paginationPath (string) : the route name to use for links
    * showAlwaysFirstAndLast (bool) : Always show first and last link (just disabled)
    * lastPage (int) : represents the total number of existing pages
#}
{% spaceless %}
    {% if lastPage > 1 %}

        {# the number of first and last pages to be displayed #}
        {% set extremePagesLimit = 3 %}

        {# the number of pages that are displayed around the active page #}
        {% set nearbyPagesLimit = 2 %}

        <nav class="pagination">
            <div class="numbers">
                <ul>
                    {% if currentPage > 1 %}
                        <li><a href="{{ path(paginationPath, currentFilters|merge(
↪{page: currentPage-1})) }}">Previous</a></li>
```

(continues on next page)

(continued from previous page)

```

                        {% for i in range(1, extremePagesLimit) if ( i < currentPage -
↪ nearbyPagesLimit ) %}
                            <li><a href="{{ path(paginationPath, currentFilters|merge(
↪{page: i})) }}">{{ i }}</a></li>
                        {% endfor %}

                        {% if extremePagesLimit + 1 < currentPage - nearbyPagesLimit
↪%}
                            <span class="sep-dots">...</span>
                        {% endif %}

                        {% for i in range(currentPage-nearbyPagesLimit, currentPage-
↪1) if ( i > 0 ) %}
                            <li><a href="{{ path(paginationPath, currentFilters|merge(
↪{page: i})) }}">{{ i }}</a></li>
                        {% endfor %}
                    {% elseif showAlwaysFirstAndLast %}
                        <span class="disabled">Previous</span>
                    {% endif %}

                    <li class="current"><a href="{{ path(paginationPath,␣
↪currentFilters|merge({ page: currentPage })) }}">{{ currentPage }}</a></li>

                    {% if currentPage < lastPage %}
                        {% for i in range(currentPage+1, currentPage +␣
↪nearbyPagesLimit) if ( i <= lastPage ) %}
                            <li><a href="{{ path(paginationPath, currentFilters|merge(
↪{page: i})) }}">{{ i }}</a></li>
                        {% endfor %}

                        {% if  (lastPage - extremePagesLimit) > (currentPage +␣
↪nearbyPagesLimit) %}
                            <li><span class="sep-dots">...</span></li>
                        {% endif %}

                        {% for i in range(lastPage - extremePagesLimit+1, lastPage)␣
↪if ( i > currentPage + nearbyPagesLimit ) %}
                            <li><a href="{{ path(paginationPath, currentFilters|merge(
↪{page: i})) }}">{{ i }}</a></li>
                        {% endfor %}

                        <li><a href="{{ path(paginationPath, currentFilters|merge(
↪{page: currentPage+1})) }}">Next</a></li>
                    {% elseif showAlwaysFirstAndLast %}
                        <li><span class="disabled">Next</span></li>
                    {% endif %}
                </ul>
            </div>
        </nav>
    {% endif %}
{% endspaceless %}
```

### How to work with Meta objects

On the template level, every variable in Context and fetched by `gimme` and `gimmelist` is a representation of Meta objects.

**dump**

```
{{ dump(article) }}
```

**print**

```
{{ article }}
```

If the meta configuration has the `to_string` property then the value of this property will be printed, otherwise it will be represented as JSON.

**access property**

```
{{ article.title }}
{{ article['title']}}
```

**generate url**

```
{{ url(article) }}    // absolute url
{{ path(article) }}   // relative path
```

Here's an example using gimmelist:

```
{% gimmelist article from articles %}
    <li><a href="{{ url(article) }}">{{ article.title }} </a></li>
{% endgimmelist %}
```

### Stringy twig extensions

We have extended the twig syntax, adding a number of functions for working with strings from a php library. A list of the functions together with a description of each, and of how they are to be invoked in PHP can be found here: https://github.com/danielstjules/Stringy#instance-methods

To call one of these functions in twig, if it returns a boolean, it is available as a twig function. So, for example, the function contains() can be called like this in twig:

```
{% set string_var = 'contains' %}
{% if contains(string_var, 'tain') %}string_var{% endif %} // will render contains
```

Any php function which returns a string is available in twig as a filter. So, for example, the function between() can be called like this in twig:

```
{% set string_var = 'Beginning' %}
{{ string_var|between('Be', 'ning') }} // will render gin
```

And the function camelize(), which doesn't require any parameters, can simply be called like this:

```
{% set string_var = 'Beginning' %}
{{ string_var|camelize }} // will render bEGINNING
```

### Redirects

We provide two functions which can be used for redirects:

#### redirect

```
{# redirect(route, code, parameters) #}
{{ redirect('homepage', 301, [] }}
```

- route param can be string with ruute name or route meta object (article meta object will work also).
- code is a redirection HTTP code (301 by default)
- parameters - if route requires any it can be provided here

#### notFound

```
{{ notFound('Error message visible for reader' }}
```

`notFound` function will redirect user to 404 error page with provided message (it's usefull when in your custom route, loader can't find requested data).

## 1.3.6 Handling Articles

### Listing Articles

Publisher have concept of Meta Loaders - one of built in loaders covers articles.

The `articles` loader parameters:

- (optional) key `route` - id or name or array of id's used for loading meta (if omitted then current route is used).

```
{% gimmelist article from articles %} <!-- It will use route from context -->
    <img src="{{ url(article) }}" />
{% endgimmelist %}
```

```
{% gimmelist article from articles with {'route': 1} %} <!-- route id -->
    <img src="{{ url(article) }}" />
{% endgimmelist %}
```

```
{% gimmelist article from articles with {'route': '/news'} %} <!-- route staticPrefix␣
↪-->
    <img src="{{ url(article) }}" />
{% endgimmelist %}
```

```
{% gimmelist article from articles with {'route': ['/news', '/sport/*']} %} <!--␣
↪route staticPrefix -->
    <img src="{{ url(article) }}" />
{% endgimmelist %}
```

**Note:** `'/sport/*'` syntax will load articles from main route (`'/sport'`) and all of it 1st level children (eg. `'/sport/football'`).

```
{% gimmelist article from articles with {'route': [1, 2, 3]} %} <!-- array with␣
↪routes id -->
    <img src="{{ url(article) }}" />
{% endgimmelist %}
```

- (optional) key `metadata` - It matches article’s metadata, and you can use all metadata fields that are defined for the article, i.e.: language, located etc.

```
{% gimmelist article from articles with {'metadata':{'language':'en'}} %}
    <img src="{{ url(article) }}" />
{% endgimmelist %}
```

- (optional) key `keywords` - It matches article’s keywords,

```
{% gimmelist article from articles with {'keywords':['keyword1', 'keyword2']} %}
    <img src="{{ url(article) }}" />
{% endgimmelist %}
```

- Filtering out selected articles (useful when you want to exclude articles listed already in content list)

```
{% gimmelist article from articles without {article:[1,2]} %} <!-- pass articles ids␣
↪(collected before) -->
    <img src="{{ url(article) }}" />
{% endgimmelist %}
```

```
{% gimmelist article from articles without {article:[gimme.article]} %} <!-- pass␣
↪articles meta objects -->
    <img src="{{ url(article) }}" />
{% endgimmelist %}
```

- Ordering by article comments count (set by external system)

```
{% gimmelist article from articles|order('commentsCount', 'desc') %}
    <img src="{{ url(article) }}" />
{% endgimmelist %}
```

## 1.3.7 Handling Related Articles

### Listing a collection of Related Articles

Usage:

```
<ul>
{% gimmelist relatedArticle from relatedArticles with { article: gimme.article }} %}
    <li>{{ relatedArticle.article.title }}</li> <!-- Related article's title -->
    <a href="{{ url(relatedArticle.article) }}">Link</a> <!-- Related article's URL --
↪>
    <li>{{ relatedArticle.createdAt|date('Y-m-d') }}</li> <!-- Related article's␣
↪creation date -->
    <li>{{ relatedArticle.updatedAt|date('Y-m-d') }}</li> <!-- Related article's␣
↪update date -->
{% endgimmelist %}
</ul>
```

The above twig code, will render the list of related articles by given article in `with` parameters.

The `{{ relatedArticle.article }}` object is an *article object*.

Specifying `article` parameter is optional. By default the article from context will be loaded.

## 1.3.8 Handling Article Media

### Listing Article Media

Publisher have concept of Meta Loaders - one of built in loaders covers article media.

### Article Media

The `articleMedia` loader have one optional parameter:

- (optional) key `article` - article Meta instance used for loading meta (if omitted then one available in context is used).

Simple usage:

```
{% gimmelist media from articleMedia %} <!-- It will use article from context -->
    <img src="{{ url(media) }}" />
{% endgimmelist %}
```

With optional parameter:

```
{% gimmelist media from articleMedia with {'article': gimme.article} %}
    <img src="{{ url(media) }}" />
{% endgimmelist %}
```

**Note:** Media Meta is handled by default by `url` and `uri` functions. It will return url for original image or file.

### Image Renditions

If provided article media is an Image then it can have custom renditions. You can loop through renditions and display them.

Usage:

```
{% gimmelist media from articleMedia with {'article': gimme.article} %}
    {% if media.renditions is iterable %}
        {% for rendition in media.renditions %}
            <img src="{{ url(rendition) }}" style="width: {{ rendition.width }}px;␣
↪height: {{ rendition.height }}px;" />
        {% endfor %}
    {% endif %}
{% endgimmelist %}
```

Get selected rendition only:

```
{% gimmelist media from articleMedia with {'article': gimme.article} %}
    {% gimme rendition with { 'name': '16-9', 'fallback': 'original' } %}
        <img src="{{ url(rendition) }}" style="width: {{ rendition.width }}px;␣
↪height: {{ rendition.height }}px;" />
```

(continues on next page)

(continued from previous page)

```
    {% endgimme %}
{% endgimmelist %}
```

**Note:** 'original' is default feedback value for single rendition loader.

### Feature Media

If Item comes with `featuremedia` association then Article will have this media set as `featureMedia`.

Usage:

```
{% if gimme.article.featureMedia.renditions is iterable %}
    {% for rendition in gimme.article.featureMedia.renditions %}
        <img src="{{ url(rendition) }}" style="width: {{ rendition.width }}px;␣
↪height: {{ rendition.height }}px;" />
    {% endfor %}
{% endif %}
```

Or get selected rendition:

```
{% gimme rendition with { 'media': gimme.article.featureMedia, 'name': '16-9',
↪'fallback': 'original' } %}
    <img src="{{ url(rendition) }}" style="width: {{ rendition.width }}px; height: {{␣
↪rendition.height }}px;" />
{% endgimme %}
```

## 1.3.9 Handling Article Authors

### Listing a Single Article's Author

Usage:

```
<ul>
{% gimme author with { id: 1 } %}
    <li>{{ author.name }}</li> <!-- Author's name -->
    <li>{{ author.role }}</li> <!-- Author's name -->
    <li>{{ author.biography }}</li> <!-- Author's biography -->
    <li>{{ author.jobTitle.name }}</li> <!-- Author's job title name -->
    <li>{{ author.jobTitle.qcode }}</li> <!-- Author's job title code -->
    {% if author.avatar %}<li> <img src="{{ url(author.avatar) }}"><li>{% endif %} <!-
↪- Author's avatar url. Check first if it's not null - author can be without avatar.␣
↪-->
    <li>{{ author.facebook }}</li> <!-- Author's Facebook -->
    <li>{{ author.instagram }}</li> <!-- Author's Instagram -->
    <li>{{ author.twitter }}</li> <!-- Author's Twitter -->
{% endgimme %}
</ul>
```

Parameters:

```
{% gimme author with { id: 1 } %} {{ author.name }} {% endgimmelist %} - select␣
↪author by it's id.
```

```
{% gimme author with { name: "Tom" } %} {{ author.name }} {% endgimmelist %} - select
↪author by it's name.
```

### Listing a collection of Article's Authors

Usage:

```
<ul>
{% gimmelist author from authors with { role: ["writer"] } without {role: ["subeditor
↪"]} %}
    <li>{{ author.name }}</li> <!-- Author's name -->
    <li>{{ author.role }}</li> <!-- Author's role -->
    <li>{{ author.biography }}</li> <!-- Author's biography -->
    <li>{{ author.jobTitle.name }}</li> <!-- Author's job title name -->
    <li>{{ author.jobTitle.qcode }}</li> <!-- Author's job title code -->
    {% if author.avatar %}<li> <img src="{{ url(author.avatar) }}"><li>{% endif %} <!-
↪- Author's avatar url. Check first if it's not null - author can be without avatar.
↪-->
    <li>{{ author.facebook }}</li> <!-- Author's Facebook -->
    <li>{{ author.instagram }}</li> <!-- Author's Instagram -->
    <li>{{ author.twitter }}</li> <!-- Author's Twitter -->
{% endgimmelist %}
</ul>
```

The above twig code, will render the list of articles where author's role is `writer` and is not `subeditor`.

Filter authors by author's name:

```
{% gimmelist author from authors with { name: ["Tom"] } %}
    {{ author.name }}
{% endgimmelist %}
```

Filter authors by author's slug (automatically created from name. Example: John Doe -> john-doe):

```
{% gimmelist author from authors with { slug: ["john-doe"] } %}
    {{ author.name }}
{% endgimmelist %}
```

Filter authors by author's name and role:

```
{% gimmelist author from authors with { role: ["Writer"], name: ["Tom"] } %}
    {{ author.name }}
{% endgimmelist %}
```

Filter authors by job title:

```
{% gimmelist author from authors with {jobtitle: {name: "quality check"}} %}
    {{ author.name }}
{% endgimmelist %}

{% gimmelist author from authors with {jobtitle: {qcode: "123"}} %}
    {{ author.name }}
{% endgimmelist %}
```

## 1.3.10 Handling Article Slideshows

### Listing a Single Article's Slideshow

Usage:

```
{% gimme slideshow with { name: "slideshow1" } %}
    {{ slideshow.code }} <!-- Slideshow's code -->
    {{ slideshow.createdAt|date('Y-m-d hh:mm') }} <!-- Slideshow's created at datetime -->
    {{ slideshow.updatedAt|date('Y-m-d hh:mm') }} <!-- Slideshow's updated at datetime-->
{% endgimme %}
```

or

```
{% gimme slideshow with { name: "slideshow1", article: gimme.article } %}
    {{ slideshow.code }} <!-- Slideshow's code -->
    {{ slideshow.createdAt|date('Y-m-d hh:mm') }} <!-- Slideshow's created at datetime -->
    {{ slideshow.updatedAt|date('Y-m-d hh:mm') }} <!-- Slideshow's updated at datetime-->
{% endgimme %}
```

Parameters:

```
{% gimme slideshow with { name: "slideshow1", article: gimme.article } %} {{ slideshow.code }} {% endgimme %} - select slideshow by it's code/name and current article.
```

If the `article` parameter is not provided, the slideshow will be loaded for the current article that is set in the context.

### Listing a collection of Article's Slideshows

Usage:

```
{% gimmelist slideshow from slideshows with { article: gimme.article } %}
    {{ slideshow.code }} <!-- Slideshow's code -->
    {{ slideshow.createdAt|date('Y-m-d hh:mm') }} <!-- Slideshow's created at datetime -->
    {{ slideshow.updatedAt|date('Y-m-d hh:mm') }} <!-- Slideshow's updated at datetime-->
{% endgimmelist %}
```

The above twig code will render the list of articles slideshows for the current article set in context.

### Listing all Article's Slideshows Items

Usage:

```
{% gimmelist slideshowItem from slideshowItems with { article: gimme.article } %}
    {% gimme rendition with {'media': slideshowItem.articleMedia, 'name': '770x515', 'fallback': 'original' } %}
        <img src="{{ url(rendition) }}" />
    {% endgimme %}
{% endgimmelist %}
```

The above twig code will render the list of articles slideshows for the current article set in context.

Or if there are audio, video, image files in slideshow:

```
{% gimmelist slideshow from slideshows with { article: gimme.article } %}
    <h2>{{ slideshow.code }}</h2>
    {% gimmelist slideshowItem from slideshowItems with { article: gimme.article, slideshow: slideshow } %}

        {% if slideshowItem.articleMedia.mimetype starts with 'image' %}
            {% gimme rendition with {'media': slideshowItem.articleMedia, 'name': '770x515', 'fallback': 'original' } %}
                <img src="{{ url(rendition) }}" />
            {% endgimme %}
        {% elseif slideshowItem.articleMedia.mimetype starts with 'audio' %}
            <audio src="{{ url(slideshowItem.articleMedia) }}" controls>
                <a href="{{ url(slideshowItem.articleMedia) }}">Download song</a>
            </audio>
        {% elseif slideshowItem.articleMedia.mimetype starts with 'video' %}
            <video src="{{ url(slideshowItem.articleMedia) }}" controls>
                <a href="{{ url(slideshowItem.articleMedia) }}">Download video</a>
            </video>
        {% endif %}

    {% endgimmelist %}
{% endgimmelist %}
```

### Listing all Article's Slideshows and its Items

Usage:

```
{% gimmelist slideshow from slideshows with { article: gimme.article } %}
    {{ slideshow.code }} <!-- Slideshow's code -->
    <!-- Slideshow items -->
    {% gimmelist slideshowItem from slideshowItems with { article: gimme.article, slideshow: slideshow } %}
        {% gimme rendition with {'media': slideshowItem.articleMedia, 'name': '770x515', 'fallback': 'original' } %}
            <img src="{{ url(rendition) }}" />
        {% endgimme %}
    {% endgimmelist %}
    {{ slideshow.createdAt|date('Y-m-d hh:mm') }} <!-- Slideshow's created at datetime -->
    {{ slideshow.updatedAt|date('Y-m-d hh:mm') }} <!-- Slideshow's updated at datetime-->
{% endgimmelist %}
```

The `article` parameter in `gimmelist` is optional. If not provided, it will load slideshows for current article.

### Listing a Single Slideshow and its Items by Name

Usage:

```
{% gimmelist slideshow from slideshows with { article: gimme.article, name: "slideshow1" } %}
    {{ slideshow.code }} <!-- Slideshow's code -->
```

(continues on next page)

(continued from previous page)

```
    <!-- Slideshow items -->
    {% gimmelist slideshowItem from slideshowItems with { article: gimme.article,␣
↪slideshow: slideshow } %}
        {% gimme rendition with {'media': slideshowItem.articleMedia, 'name': '770x515
↪', 'fallback': 'original' } %}
            <img src="{{ url(rendition) }}" />
        {% endgimme %}
    {% endgimmelist %}
    {{ slideshow.createdAt|date('Y-m-d hh:mm') }} <!-- Slideshow's created at␣
↪datetime -->
    {{ slideshow.updatedAt|date('Y-m-d hh:mm') }} <!-- Slideshow's updated at␣
↪datetime-->
{% endgimmelist %}
```

The `article` parameter in `gimmelist` is optional. If not provided, it will load slideshows for current article.

## 1.3.11 Handling Routes

### Listing a Single Route

Usage:

```
<ul>
{% gimme route with { parent: 5, slug: 'test-route', name: 'Test Route'} %}
    <li>{{ route.name }}</li> <!-- Route's name -->
    <li>{{ route.slug }}</li> <!-- Route's slug -->
    <li>{{ url(route) }}</li> <!-- Route's url -->
{% endgimme %}
</ul>
```

- `parent` - an id of parent route
- `slug` - route's slug
- `name` - route's name

### Listing a Routes Collection

Usage:

```
<ul>
{% gimmelist route from routes %}
    <li>{{ route.name }}
{% endgimme %}
</ul>
```

```
<ul>
{% gimmelist route from routes with {parent: 5} %} <!-- possible values for parent:␣
↪(int) 5, (string) 'Test Route', (meta) gimme.route -->
    <li>{{ route.name }}
{% endgimmelist %}
</ul>
```

### 1.3.12 Handling Content List Items

#### Listing Content List Items

**Note:** Content List can store many different content types (articles, events, packages).

##### Content List

Usage:

```
<ul>
{% gimme contentList with { contentListName: "List1" } %}
    <li>{{ contentList.name }}</li> <!-- Content list name -->
    <li>{{ contentList.description }}</li> <!-- Content list description -->
    <li>{{ contentList.type }}</li> <!-- Content list type) -->
{% endgimme %}
</ul>
```

Parameters:

```
{% gimme contentList with { contentListId: 1 } %} - select list by it's id.
```

```
{% gimme contentList with { contentListName: "List Name" } %} - select list by it's␣
↪name.
```

##### Content List Items

Usage:

```
<ul>
{% gimmelist item from contentListItems with { contentListName: "List1" } %}
    <li>{{ item.content.title }}</li> <!-- Article title -->
    <li>{{ item.position }}</li> <!-- Item position in list -->
    <li>{{ item.sticky ? "pinned" : "not pinned" }}</li> <!-- Checks if item is␣
↪sticky (positioned on top of list) -->
{% endgimmelist %}
</ul>
```

or

```
{% gimme contentList with { contentListName: "List1"} %}
    {% cache 'top-articles' {gen: contentList} %}
    <ul>
    {% gimmelist item from contentListItems with { contentList: contentList } %}
        <li>{{ item.content.title }}</li> <!-- Article title -->
        <li>{{ item.position }}</li> <!-- Item position in list -->
        <li>{{ item.sticky ? "pinned" : "not pinned" }}</li> <!-- Checks if item is␣
↪sticky (positioned on top of list) -->
    {% endgimmelist %}
    </ul>
    {% endcache %}
{% endgimme %}
```

**Note:** Passing previously fetched contentList (for cache key generation needs) is good for performance.

Parameters:

```
{% gimmelist item from contentListItems with { contentListId: 1 } %} - select list by␣
↪it's Id.
```

```
{% gimmelist item from contentListItems with { contentListId: 1, sticky: true } %} -␣
↪filter by sticky value.
```

```
{% gimmelist item from contentListItems with { contentListId: 1, sticky: true }␣
↪without {content: [1]} %} - exclude article with id 1 (in case of articles as␣
↪items).
```

### 1.3.13 Keywords

#### List a Single Keyword

Usage:

```
<ul>
{% gimme keyword with { slug: 'big-city' } %}
    <li>{{ keyword.name }}</li> <!-- Keyword's name -->
    <li>{{ keyword.slug }}</li> <!-- Keyword's slug -->
{% endgimme %}
</ul>
```

### 1.3.14 Template Caching

For now we support just template block caching with the `cache` block.

The `Cache` block is simple, and accepts only two parameters: cache key and strategy object (with strategy key and value).

**Note:** Cache blocks can be nested:

```
{% cache 'v1' {time: 900} %}
    {% for item in items %}
        {% cache 'v1' {gen: item} %}
            {# ... #}
        {% endcache %}
    {% endfor %}
{% endcache %}
```

The annotation can also be an expression:

```
{% set version = 42 %}
{% cache 'hello_v' ~ version {time: 300} %}
    Hello {{ name }}!
{% endcache %}
```

There is no need to invalidate keys - the system will clear unused cache entries automatically.

### Strategies

There are two available cache strategies: `lifetime` and `generational`.

With `lifetime` as a strategy key you need to provide `time` with a value in seconds.

```
{# delegate to lifetime strategy #}
{% cache 'v1/summary' {time: 300} %}
    {# heavy lifting template stuff here, include/render other partials etc #}
{% endcache %}
```

With `generational` as a strategy key you need to provide `gen` with object or array as the value.

```
{# delegate to generational strategy #}
{% cache 'v1/summary' {gen: gimme.article} %}
    {# heavy lifting template stuff here, include/render other partials etc #}
{% endcache %}
```

**Note:** You can pass Meta object to `generational` strategy and it will be used for key generation. If Meta value have `created_at` or `updated_at` then those properties will be used, otherwise key will be generated only from object `id`.

### Content list blocks caching

It's important to always use `generational` strategy for content lists (and it items) caching. Publisher will update cache key generated with it every time when items on list are added/removed/reordered or when list criteria are updated or even when article used by list will be unpublished or updated.

```
{% cache 'frontPageManualList' {gen: contentList} %}
    {# get and render list items here #}
{% endcache %}
```

## 1.3.15 How to implement Search using ElasticSearch?

To make use of features (e.g. full-text search) provided by ElasticSearch and it's extension/plugin /bundles/SWPElasticSearchBundle/index created specifically for Superdesk Publisher, you need to do the following steps.

### Create a new Route

You can create a new route using admin interface as described in *Create new route with extension (example: feed/sitemap.rss)* section. In this example the created route under which we will place search will be named: `search`.

### Create a template file

Example search template which loads search and its results when filtered by criteria:

```
# ../view/search.html.twig
<form name="filter" method="get">
    <input type="search" id="filter_search" name="q">
</form>

{% set itemsPerPage, currentPage = 8, app.request.get('page', 1) %}
{% set start = ((currentPage - 1) * itemsPerPage) %}

{% gimmelist article from searchResults|limit(app.request.get('limit',␣
↪10))|order(app.request.get('field', 'publishedAt'), app.request.get(
↪'direction', 'desc')) with {
    term: app.request.get('q', ''),
    page: app.request.get('page', 1),
    routes: app.request.get('route', []),
    term: app.request.get('q', ''),
    publishedBefore: app.request.get('publishedBefore'),
    publishedAfter: app.request.get('publishedAfter'),
    publishedAt: app.request.get('publishedAt'),
    sources: app.request.get('source', []),
    authors: app.request.get('author', []),
    statuses: app.request.get('status', []),
    metadata: app.request.get('metadata', []),
} %}
    <h4>{{ article.title }}</h4>
    <p>{{ article.lead }}</p>

{% if loop.last  %}
    {% include '_tpl/pagination.html.twig' with {
    currentFilters: {}|merge(app.request.query.all()),
    currentPage: currentPage,
    paginationPath: gimme.route,
    lastPage: (loop.totalLength/itemsPerPage)|round(0, 'ceil')
    } only %}

    Showing {{ searchResults|length }} out of {{ loop.totalLength }}␣
↪articles.
{% endif %}
{% endgimmelist %}

<a href="search?route[]=50">Business</a>
<a href="search?route[]=49">Politics</a>
```

Alternatively, to built-in `order` function, you can also use `sort: app.request.get('sort', [])`, parameter to sort by different fields and directions which needs to be passed directly to the `with` statement.

### Available search criteria:

Based on the above template.

| Criteria name | Description | Example | Format |
|---|---|---|---|
| sort | Sorting | sort[publishedAt]=desc | sort[<field>]=<direction> |
| page | Pagination | page=1 | page=<page_number> |
| limit | Items per page | limit=10 | limit=<limit> |
| routes | An array of routes ids | route[]=10&route[]=12 | route[]=<routeId>&route[]=<routeId> |
| q | Search query | q=Lorem ipsum | q=<search_term> |
| publishedBefore | Published before date time | publishedBefore=1996-10-15T00:00:00 | publishedBefore=<datetime> |
| publishedAfter | Published before date time | publishedBefore=1996-10-15T00:00:00 | publishedAfter=<datetime> |
| sources | Sources of articles | source[]=APP&source[]=NTB | source[]=<source>&source[]=<source> |
| authors | An array of authors | author[]=Joe&author[]=Doe | author[]=<auth1>&author[]=<auth2> |
| statuses | An array of statues | status[]=new&status[]=published | status[]=new&status[]=published |
| metadata | An array metadata | metadata[located]=Sydney | metadata[<field>]=<value> |

### 1.3.16 Tips

### 1.3.17 Templates inheritance

Default template name for route and articles in Publisher is `article.html.twig`.

**Inheritance overview:**

```
> article.html.twig
    > Route custom template
        > Article custom template
```

If `route` is collection type then it can have declared two default templates:

- `default_template` used for rendering Route content (eg. /sport).
- `default_articles_template` used for rendering content attached to this route (eg. /sport/usain-bolt-fastest-man-in-theo-world).

**Note:** When route `default_template` property is set but not `default_articles_template`, then Web Publisher will load all articles attached to this route with template chosen in `default_template` (not with `article.html.twig`).

If `content` have assigned custom template then it will always override other already set templates.

### 1.3.18 How to change the Route/Article template name?

You can change default template name values for article and route with API calls (providing it on resource create or resource update calls).

Example resource update calls:

#### article:

```
curl -X "PATCH" -d "article[template_name]=new_template_name.html.twig" -H "Content-
↪type:\ application/x-www-form-urlencoded" /api/v1/content/articles/features
```

#### route:

```
curl -X "PATCH" -d "route[template_name]=custom_route_template.html.twig" -H "Content-
↪type:\ application/x-www-form-urlencoded" /api/v1/content/routes/news
```

## 1.4 Themes

### 1.4.1 About themes and multitenancy

Themes provide templates for the customization of the appearance and functionality of your public-facing websites. Themes can also be translated to support multiple written languages and regional dialects.

The Superdesk Publisher themes system is built on top of fast, flexible and easy to use Twig templates.

By default, themes are located under the `app/themes` directory. A basic theme must have the following structure:

```
ExampleTheme/                <=== Theme starts here
    views/                   <=== Views directory
        home.html.twig
    translations/            <=== Translations directory
        messages.en.xlf
        messages.de.xlf
    screenshots/                 <=== Theme screenshots
        front.jpg
    public/                  <=== Assets directory
        css/
        js/
        images/
    theme.json               <=== Theme configuration
```

Publisher does not support the option of Sylius theme structure to have bundle resources nested inside a theme.

- – *

Superdesk Publisher can serve many websites from one instance, and each website *tenant* can have multiple themes. The active theme for the tenant can be selected with a local settings file, or by API.

If only one tenant is used, which is the default, the theme should be placed under the `app/themes/default` directory, (e.g. `app/themes/default/ExampleTheme`).

If there were another tenant configured, for example `client1`, the files for one of this tenant's themes could be placed under the `app/themes/client1/ExampleTheme` directory.

- – *

Superdesk Publisher provides an easy way to create device-specific templates. This means you only need to put the elements in a particular template which are going to be used on the target device.

The supported device types are: `desktop, phone, tablet, plain`.

A theme with device-specific templates could be structured like this:

```
ExampleTheme/                   <=== Theme starts here
    phone                       <=== Views used on phones
        views/                  <=== Views directory
            home.html.twig
    tablet                      <=== Views used on tablets
        views/                  <=== Views directory
            home.html.twig
    views/                      <=== Default templates directory
        home.html.twig
    translations/               <=== Translations directory
        messages.en.xlf
        messages.de.xlf
    screenshots/                <=== Theme screenshots
        front.jpg
    public/                     <=== Assets directory
        css/
        js/
        images/
    theme.json                  <=== Theme configuration
```

**Note:** If a device is not recognized by the Publisher, it will fall back to the `desktop` type. If there is no `desktop` directory with the required template file, the locator will try to load the template from the root level `views` directory.

More details about theme structure and configuration can be found in the Sylius Theme Bundle documentation.

### 1.4.2 Create or install a theme

To install theme assets you need to run `swp:theme:install` command.

```
The swp:theme:install command installs your custom theme for given tenant:

    app/console swp:theme:install <tenant> <theme_dir>

You need specify the directory (theme_dir) argument to install
theme from any directory:

    app/console swp:theme:install <tenant> /dir/to/theme

Once executed, it will create directory app/themes/<tenant>
where <tenant> is the tenant code you typed in the first argument.

To force an action, you need to add an option: --force:

    app/console swp:theme:install <tenant> <theme_dir> --force

To activate this theme in tenant, you need to add and option --activate:
    app/console swp:theme:install <tenant> <theme_dir> --activate

If option --processGeneratedData will be passed theme installator will
generate declared in theme config elements like: routes, articles, menus, widgets,
content lists and containers
```

### 1.4.3 Work with theme assets

To install theme assets you need to run `sylius:theme:assets:install` command.

Theme assets (JavaScript, CSS etc. files) should be placed inside the theme directory. There are few ways of reading theme assets in your Twig templates. The below how-to describes where to place the assets, how to install it and use it.

#### Load assets from the theme's public directory (`app/themes/<theme-name>/public`)

1. Put the `example.css` asset file inside `<theme-name>/public/css/` directory.
2. Install assets by running command: `php app/console sylius:theme:assets:install`.
3. Make use of the asset file in twig templates:

```
<!-- loads test.css file directly /public/css/ in theme directory -->
<link rel="stylesheet" href="{{ asset('theme/css/example.css') }}" />
```

#### Load assets from the public `web` directory

1. Put the `example.css` asset file directly inside `web` directory.
2. Make use of the asset file in twig templates:

```
<!-- loads asset file directly from `web` dir (`web/example.css`) -->
<link rel="stylesheet" href="{{ asset('example.css') }}" />
```

#### Generate simple links for current theme assets

If You need to get link to asset from outside of twig template then you can use this url:

```
/public/{filePath}

ex. <link rel="stylesheet" href="/public/css/example.css" />
```

Where {filePath} is path for your file from public directory inside theme.

#### Load Service Worker files (from domain root level)

If You want to use service worker or manifest file (it must be placed in root level) then you can use this url:

```
/{fileName}.{fileExtension}

ex. <link rel="manifest" href="/manifest.json">
```

Where {fileName} can be only `sw` or `manifest`.

#### Load bundles' assets

1. Install Symfony assets by running command: `php app/console assets:install`.
2. Make use of the asset file in twig templates:

```
<!-- loads bundle's asset file from bundles dir -->
<link rel="stylesheet" href="{{ asset('bundles/framework/css/body.css') }}" />
```

### Override bundles' assets from the theme

There is a possibility to override bundle specific assets. For example, you have `AcmeDemoBundle` registered in your project. Let's assume there is a `body.css` file placed inside this bundle (`Resources/public/css/body.css`). To override `body.css` file from your theme, you need to place your new `body.css` file inside `app/themes/<theme-name>/AcmeDemoBundle/public` directory:

1. Put the `body.css` asset file inside `app/themes/<theme-name>/AcmeDemoBundle/public` directory.
2. Install assets by running command: `php app/console sylius:theme:assets:install`.
3. Make use of the asset file in twig templates:

```
<link rel="stylesheet" href="{{ asset('theme/acmedemo/css/body.css') }}" />
```

**Note:** `theme` prefix in `{{ asset('theme/css/example.css') }}` indicates that the asset refers to current theme.

## 1.4.4 Translations

The Symfony Translation component supports a variety of file formats for translation files, but in accordance with best practices suggested in the Symfony documentation, the XLIFF file format is preferred. JMSTranslationBundle has been added to the project to facilitate the creation and updating of such files.

The use of abstract keys such as `index.welcome.title` is preferred, with an accompanying description `desc` in English to inform a translator what needs to be translated. This description could simply be the English text which is to be displayed, but additional information about context could be provided to help a translator.

Abstract keys are used for two main reasons:

1. Translation messages are mostly written by developers, and changes might be necessitated later. These changes would then result in changes for all supported languages instead of only for the source language, and some translations might be lost in the process.
2. Some words in English are spelled differently in other languages, depending on their meaning, so providing context is important.

Here is an example of the preferred syntax in twig templates:

```
{{ 'index.welcome.title'|trans|desc('Welcome to Default Theme!') }}
```

Translation labels added to Twig and php files can be extracted and added to XLIFF files using a console command `app/console translation:extract`. This command can be used to create or update a XLIFF file in the locale `en` for the `DefaultTheme` of the FixturesBundle:

```
app/console translation:extract en --dir=./src/SWP/Bundle/FixturesBundle/Resources/
↪themes/DefaultTheme/ --output-dir=./src/SWP/Bundle/FixturesBundle/Resources/themes/
↪DefaultTheme/translations
```

This will create or update a XLIFF file in English called `messages.en.xlf`, which can be used with a translation tool.

### 1.4.5 AMP HTML Integration

Google AMP HTML integration comes with Superdesk Publisher out of the box. This integration gives you a lot of features provided by Google. To name a few: fast loading time and accessibility via Google engines etc. There is no need to install any dependencies, all you need to do is to create AMP HTML compatible theme or use the default one provided by us.

Default AMP HTML theme is bundled in our main Demo Theme and can be installed using `php app/console swp:theme:install` command.

You could also copy it to your own main theme and adjust it in a way you wish.

**Note:** See *Setting up a demo theme* section for more details on how to install demo theme.

#### How to create AMP HTML theme?

You can find more info about it in AMP HTML official documentation.

#### Where to upload AMP HTML theme?

Publisher expects to load AMP HTML theme from main theme directory which is `app/themes/<tenant_code>/<theme_name>`. AMP HTML theme should be placed in `app/themes/<tenant_code>/<theme_name>/amp/amp-theme` folder. `index.html.twig` is the starting template for that theme. If that template doesn't exist, theme won't be loaded. Once the theme is placed in a proper directory, it will be automatically loaded.

To test if the theme has been loaded properly you can access your article at e.g.: `https://example.com/news/my-articles?amp`.

#### Linking AMP page and non-AMP page

To add a link to AMP page from article template in the form of `<link>` tags (which is required by AMP HTML integration for discovery and distribution), you can use `amp` Twig filter:

```
{# app/themes/<tenant_code>/<theme_name>/views/article.html.twig #}
<link rel="amphtml" href="{{ url(gimme.article)|amp }}"> {# https://example.com/news/
↪my-articles?amp #}
```

And from AMP page:

```
{# app/themes/<tenant_code>/<theme_name>/amp/amp-theme/index.html.twig #}
<link rel="canonical" href="{{ url(gimme.article) }}"> {# https://example.com/news/my-
↪articles #}
```

### 1.4.6 Theme Logo

Theme's logo can be uploaded by the API. Once it is uploaded, it can be rendered in theme's template.

There is possibility to upload up to three different logos per theme.

### How to upload custom theme's logo?

The main theme's logo can be uploaded by making a `POST` call with the attached image to the API endpoint: `/theme/logo_upload/`.

To upload second and third logo, a `POST``request to ``/theme/logo_upload/<theme_setting_name>` must be done.

Where `<theme_setting_name>` should be replaced by `theme_logo_second` for second logo and `theme_logo_third` for the third logo.

See `/api/doc` route in your Superdesk Publisher installation to find out more about API documentation.

This endpoint accepts `jpg`, `jpeg` and `png` image extensions.

### How to display path of the theme's logo using API?

Once the theme logos are uploaded using `/theme/logo_upload/` API endpoint, it is a time to display them. The logos paths are stored under the `theme_logo` (main logo), `theme_logo_second` and `theme_logo_third` settings name and can be accessed by calling `/themes/settings/` endpoint.

To get the image, just simply grab the `theme_logo` or `theme_logo_second` or `theme_logo_third` setting from the response and pass its value as an argument to the url: `/theme_logo/<theme_logo_setting_here>` (e.g. `/theme_logo/f2/e9/7c543ecad44807b0acab0b61e09a.png`), then an image will be streamed.

If the value of `theme_logo`, `theme_logo_second` or `theme_logo_third` setting is not set, to get the default logo just use the `/public/{fileName}.{fileExtension}` API endpoint which will help you get any image (logo in this case) from the `public` directory of your theme.

### How to display path of the theme's logo in templates?

There is a `themeLogo` twig function which accepts two arguments. These arguments are: a path to the logo from `public` directory in your theme in this example (Read more about it in *Work with theme assets* chapter). If theme's logo is not uploaded, the function will fallback to the path provided in as a first argument to that function.

The second argument of that function is theme setting name, e.g. `theme_logo_second`. If not passed the default logo is loaded from `theme_logo` setting.

Any path can be provided as an argument.

```
{# app/themes/<tenant_code>/<theme_name>/views/index.html.twig #}
<img src="{{ themeLogo(asset('theme/img/logo.png')) }}">
<img src="{{ themeLogo(asset('theme/img/logo.png'), 'theme_logo_second') }}">
```

## 1.4.7 Theme Settings

Settings of a theme are defined in `theme.json` configuration file which should be present in every theme directory.

An example of `theme.json` file with defined settings will look like:

```
{
    "name": "swp/default-theme",
    "title": "Default Theme",
    "description": "Superdesk Publisher default theme",
    "authors": [
        {
```

(continues on next page)

(continued from previous page)

```
                "name": "Sourcefabric z.ú.",
                "email": "contact@sourcefabric.org",
                "homepage": "https://www.sourcefabric.org",
                "role": "Organization"
            }
        ],
        "settings": {
            "primary_font_family": {
                "label": "Primary Font Family",
                "value": "Roboto",
                "type": "string",
                "help": "The primary font",
                "options": [
                    {"value": "Roboto", "label": "Roboto"},
                    {"value": "Lato", "label": "Lato"},
                    {"value": "Oswald", "label": "Oswald"}
                ]
            },
            "secondary_font_family": {
                "value": "Roboto",
                "type": "string",
                "options": [
                    {"value": "Roboto", "label": "Roboto"},
                    {"value": "Lato", "label": "Lato"},
                    {"value": "Oswald", "label": "Oswald"}
                ]
            },
            "body_font_size": {
                "label": "Body Font Size",
                "value": 14,
                "type": "integer",
                "options": [
                    {"value": 14, "label": "14px"},
                    {"value": 16, "label": "16px"},
                    {"value": 18, "label": "18px"}
                ]
            }
        }
}
```

In the `settings` property of the JSON file are defined the default theme's settings.

Each setting can be overridden by the API. See `/settings/` API endpoint for more details in the `/api/doc` route in your Superdesk Publisher instance.

Read more about settings in *Settings* chapter to find out more.

Every setting is a JSON object which can contain the following properties:

- `label` - Setting's label, will be visible in API when defined,
- `value` - Setting's value, will be visible in API when defined,
- `type` - Setting's type, either it's `string`, `integer`, `boolean` or `array`.
- `help` - Settins's helper text.
- `options` - an array of optional values that can be used to implement select box.

Read more about theme's structure in *Themes* chapter.

#### How to display current theme's settings in templates?

```
{# app/themes/<tenant_code>/<theme_name>/views/index.html.twig #}
{{ themeSetting('primary_font_family') }} # will print "Roboto"
```

In development environment, if the theme's setting doesn't exists an exception will be thrown with a proper message that it does not exist. In production environment no exception will be thrown, the page will render normally.

#### How to display current theme's settings using API?

Theme's settings can be accessed by calling an `/theme/settings/` API endpoint using `GET` method.

#### How to update current theme's settings using API?

To update theme's settings using API, a `PATCH` request must be submitted to the `/settings/` endpoint with the JSON payload:

```
{
    "settings": {
        "name": "primary_font_family",
        "value": "custom font"
    }
}
```

#### How to restore current theme's settings using API?

There is a possibility to restore the current theme's settings to the default ones, defined in the `theme.json` file.

This can be done using API and calling a `/settings/revert/{scope}` endpint using `POST` method. The `scope` parameter should be set to `theme` in order to restore settings for current theme.

### 1.4.8 Example themes

#### Superdesk Publisher demo theme

Theme location: */src/SWP/Bundle/FixturesBundle/Resources/themes/DefaultTheme*

**Publisher Mag** ships within Superdesk Publisher Release, thus can be considered the default and most basic Publisher theme. It serves purpose of showing most common features of the software, such as listing articles, showing article elements and full content, working with menu widgets etc.

**Publisher Mag** also features html and content list widgets which enable live-site editing from frontend.

To create richer user experience, 3rd-party services can be incorporated. In **Publisher Mag** theme we showcase it with Disqus article comments.

#### The Modern Times theme

Theme repo: https://github.com/SuperdeskWebPublisher/theme-dailyNews

**The Modern Times** theme is fresh, fully responsive, highly adaptible Superdesk Publisher theme built primarily to serve those media operations with high daily news production. It offers editors flexibility in ways they can present sets of news (category or tag based; manually curated, fully- or semi-automated content lists; and more).

**The Modern Times** also features several customizable menu, html and content list widgets which enable live-site editing from frontend.

In this theme we also showcase how 3rd-party services can be incorporated for reacher user experience (Open weather data integration for any wetaher station in the world, Disqus article comments, Playbuzz voting poll, Google Custom Search Engine).

#### Magazine theme

Theme repo: https://github.com/SuperdeskWebPublisher/theme-magazine

**Magazine** theme is fresh, fully responsive, simple and ultra fast Superdesk Publisher theme built to serve those media operations that are not primarily focused on daily content production, but on fewer stories per day/week that have longer time span. Naturally this applies to traditional weekly, be-weekly or even monthly type of magazines from the print world.

**Magazine** theme features customizable menu, html and content list widgets which enable live-site editing from frontend.

To create richer user experience, 3rd-party services can be incorporated. In **Magazine** theme we showcase it with Disqus article comments.

## 1.5 Editorial tools

### 1.5.1 Content Lists

What playlists are for music, content lists are for articles - customizable sets of content organized and ordered in a way that suits your output channel the best. For example, when you access some Publisher website homepage as visitor, chances are that the teasers for articles you see are curated by website editors using *Content lists*. The order of articles can be easily updated as news arrive, new articles added and some other removed etc.

#### Manual Content Lists

With *manual content lists*, articles need to be dragg'n'dropped into the list manually. This gives complete freedom to editors which stories are put where. List can be (and should be) limited in length, so when new article is added to let's say top of it, last one drops out.

#### Automatic Content Lists

If list doesn't need to be fully customizable, or when it can be programmed with certain rules to collect appropirate articles, then *automatic content list* stepps in.

For example, some block on the webiste shows most recent articles from section Sport, where metadata location is set to 'Europe' and author is not John Smith; in such situation, obviously editors don't need to put in manual labour, but rather use automated set of rules which add articles to the automatic list.

Built in criteria:

- `route` - an array of route ids, e.g. [1,5]
- `author` - an array of authors, e.g. ["Test Persona","Doe"]
- `publishedBefore` - date string, articles published before this date will be added to the list,

e.g. date: "2017-01-20". (date format must be YYYY-MM-DD)

- `publishedAfter` - date string, articles published after that date will be added to the list, format is the same as in the `publishedBefore` case.
- `publishedAt` - date string, when defined articles matching this publish dates will be added to the list when published, format is the same as in case of `publishedBefore` and `publishedAfter`
- `metadata` - metadata field is json string, e.g. `{"metadata":{"language":"en"}}`. It matches article's metadata, and you can use all metadata fields that are defined for the article, i.e.: language, located etc.

All criteria can be combined together which in the result it will add articles to the list (on publish) depending on your needs.

#### Content Buckets

### 1.5.2 Livesite Editor

#### What is Livesite Editor?

With LiveSite editor, Publisher offers website editors possibility to manage and manupulate containers and widgets web page is built of, directly on frontend. Yes, this means that when such person is authenticated, page structure will get interactive and tool pallettes will appear.

#### Editing content directly in frontend

So when website editor with such privileges authenticates, page structure gets interactive in a way that containers that exist on it are marked with descrete border and label. On hover action, also widgets are marked in different colour, and both *containers* and *widgets* configuration button can activate modal dialog with options.

Options when live-editing containers are

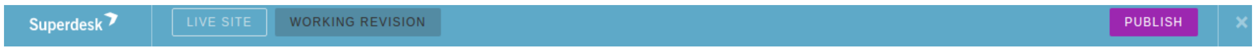
- add new widget to container (possible types of widgets to add are: *Content List Widget*, *HTML Widget*, *Google Adsense Widget*, *Menu Widget*)
- remove widget from container
- re-order widgets inside container

Options when live-editing widgets are

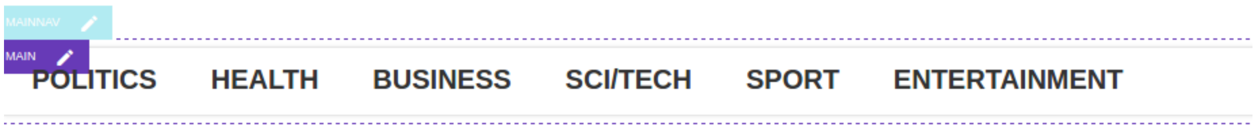
- edit widget name
- depending on widget type, different fileds will be editable. For example: - for Menu Widget, dropdown to chose which menu to link to the widget, and template name of the template that will be used to create menu - for HTML widget, text area with html code

#### Livesite Editor interface

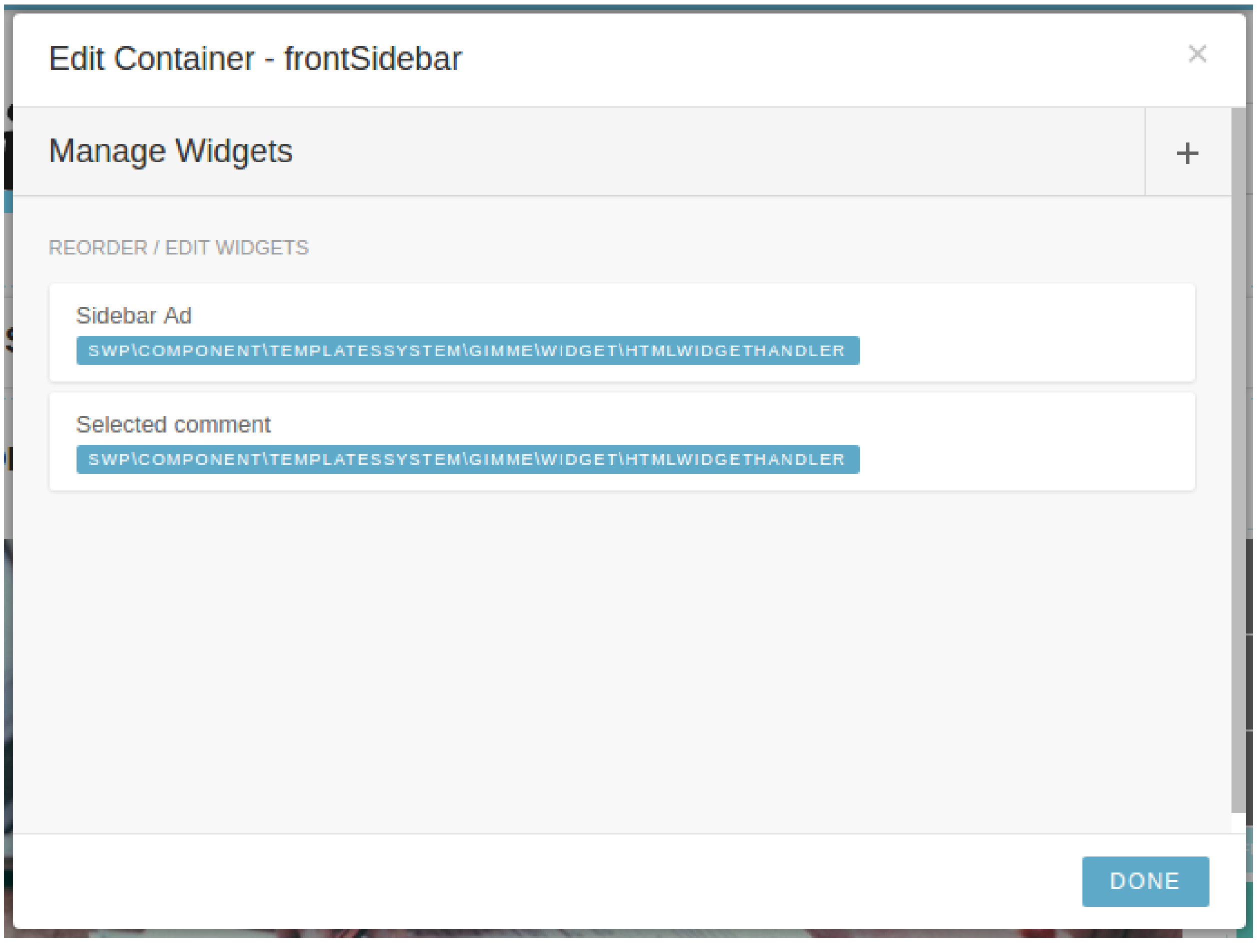
Interface for LiveSite editor is fairly simple; editing is turned on (again, only for authorized users) by pressing the top-left corner button - it reveals top toolbar with options to jump between *live site* and *current revision*, as well as *publish* buttons which deploys applied changes to the live site.



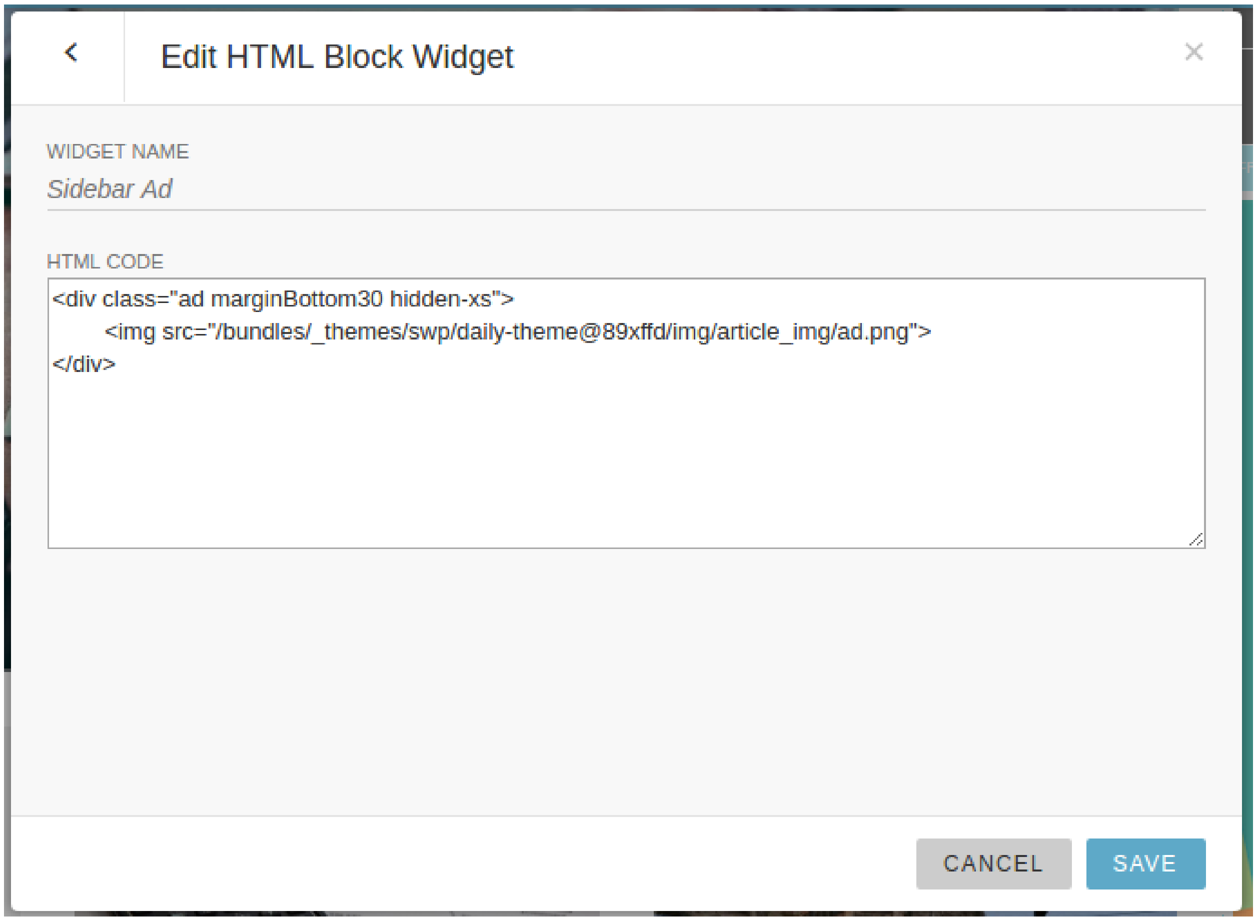
In the same time, as toolbar is activated, containers become marked and interactive on hover, showing widget's borders and label.



Clicking on edit icon next to container label opens options modal


Edit Container - frontSidebar
Manage Widgets
+
REORDER / EDIT WIDGETS
Sidebar Ad
SWP\COMPONENT\TEMPLATESSYSTEM\GIMME\WIDGET\HTMLWIDGETHANDLER
Selected comment
SWP\COMPONENT\TEMPLATESSYSTEM\GIMME\WIDGET\HTMLWIDGETHANDLER
DONE

Clicking on edit icon next to widget label opens options modal



Sidebar Ad

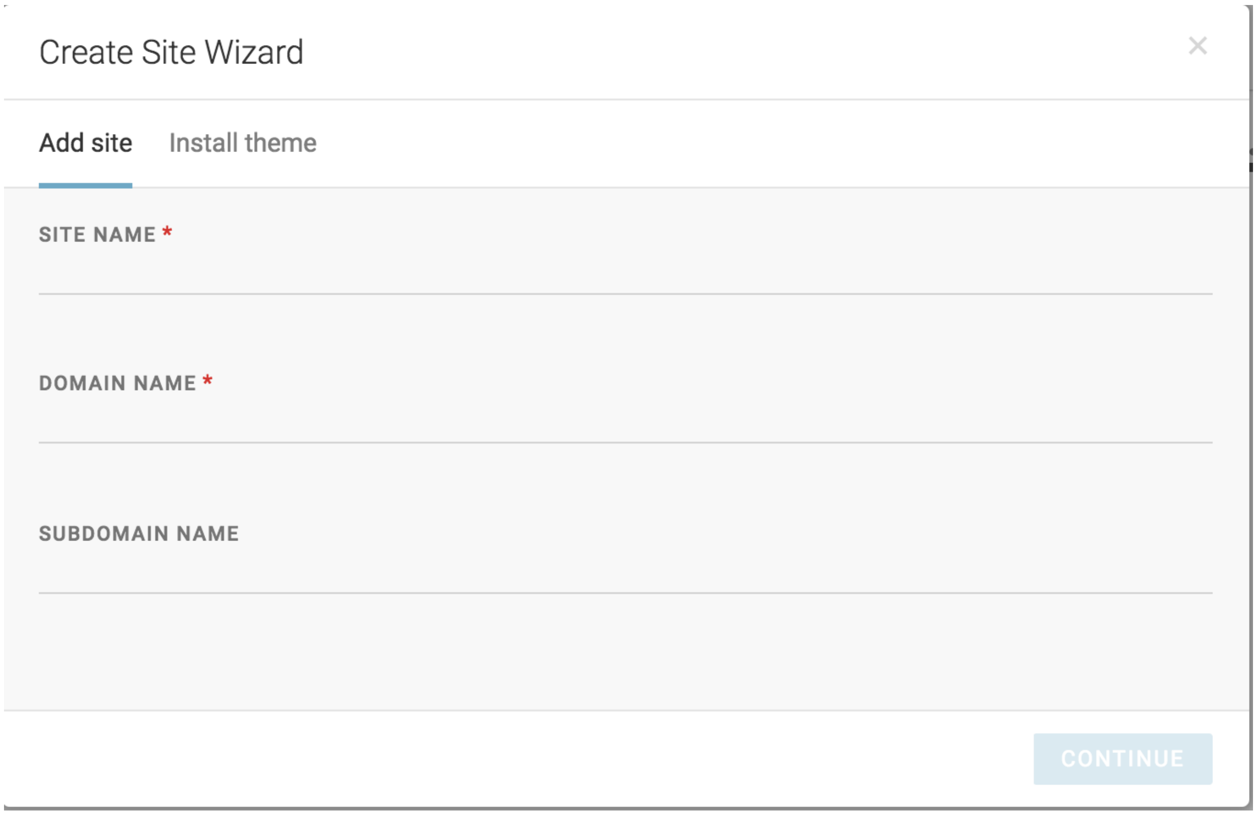
```
<div class="ad marginBottom30 hidden-xs">
      <img src="/bundles/_themes/swp/daily-theme@89xffd/img/article_img/ad.png">
</div>
```

## 1.6 Admin Interface

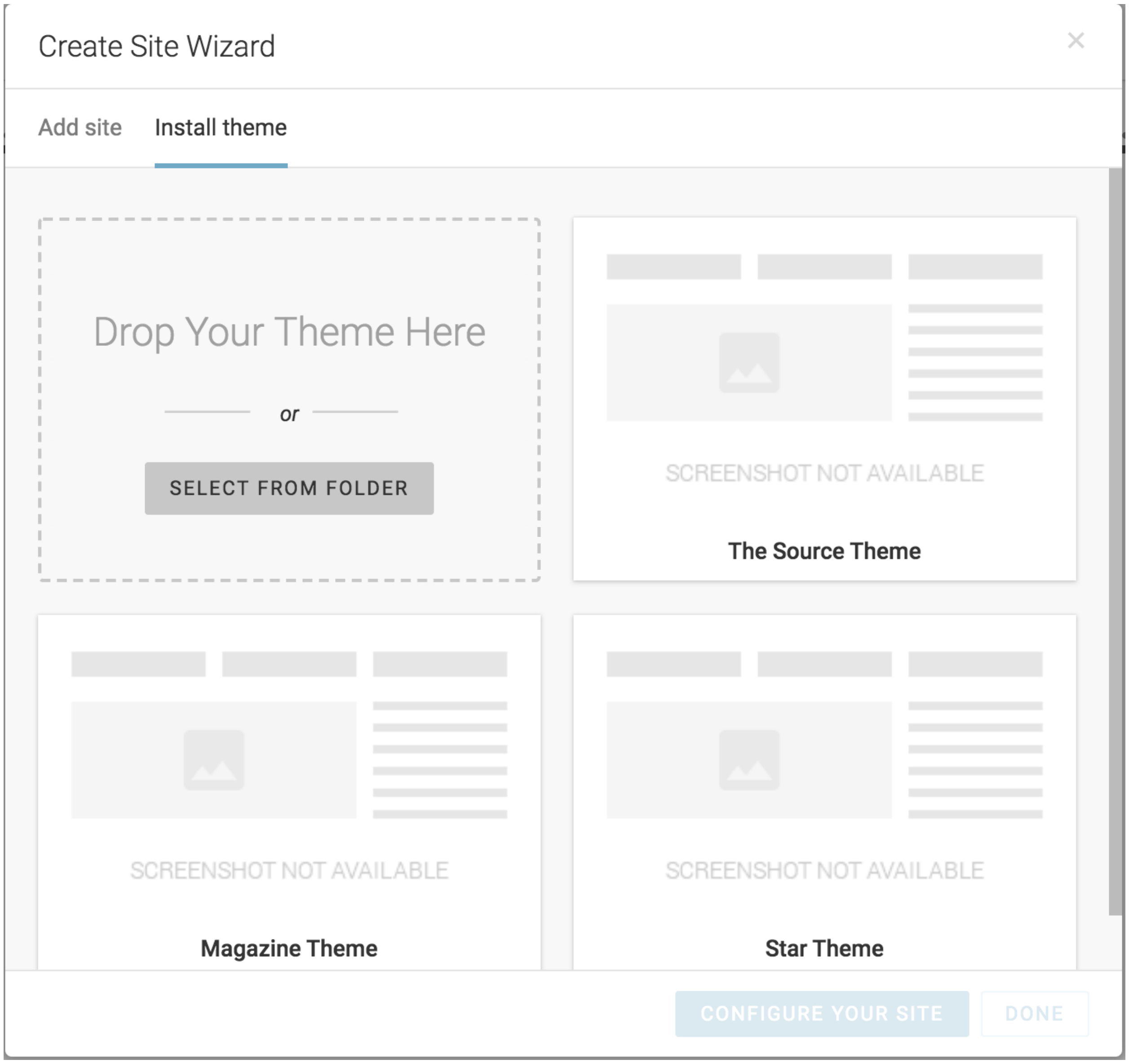
### 1.6.1 Creating new site

Publisher offers possibilites to create and maintain many sites with one instance.

To create new site simply click big plus button in top right corner of **Output Settings** and follow Create Site Wizard steps.



First step is to fill site name and domain that points to publisher instance. Subdomain is optional.

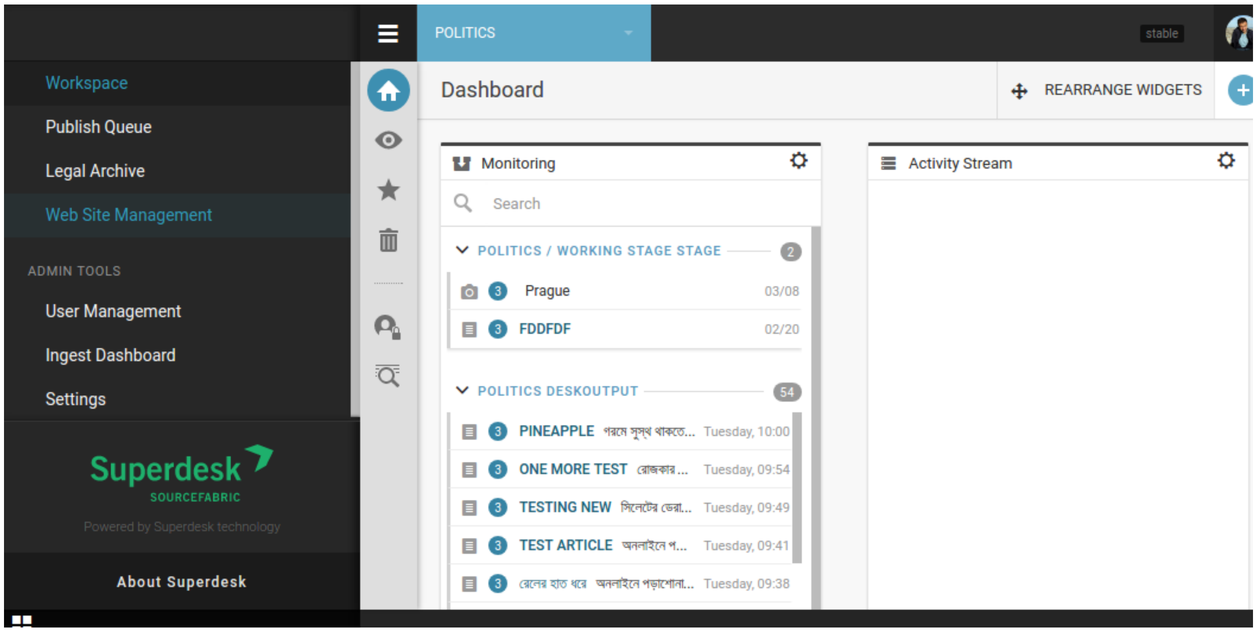


Once site is created wizard will take you to next step where you can upload or use one of already installed themes. Once theme is installed and activated you can proceed to site configuration. **Some themes support predefined data. It means that sample content will be created for you during installation process.**

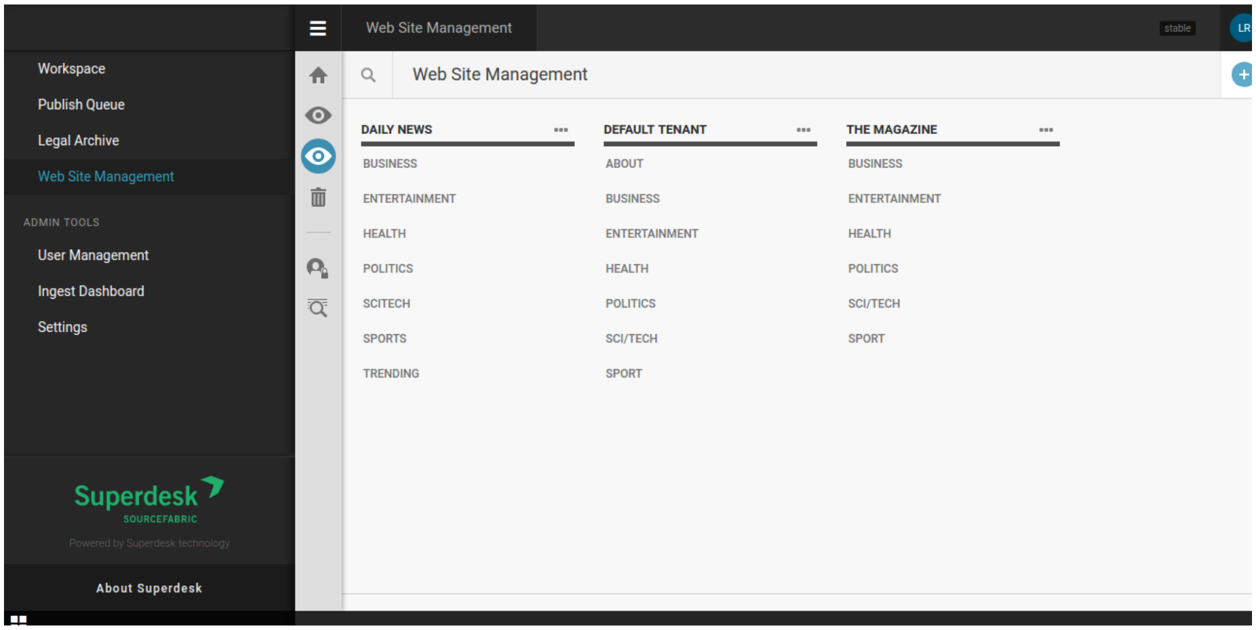
### 1.6.2 Site Management

Publisher is built to work with any News API feed and thus it can be used as frontend solution for, say, Wordpress created content, or custom CMS solution, or just a feed that you subscribe for with some provider.

But if you use Publisher together with Superdesk, then you have **Web Site Management** GUI that integrates seemlesly to the Superdesk environment. From there, you can manage one or many websites (or, more generally, output channels) with as many tenants as you want.

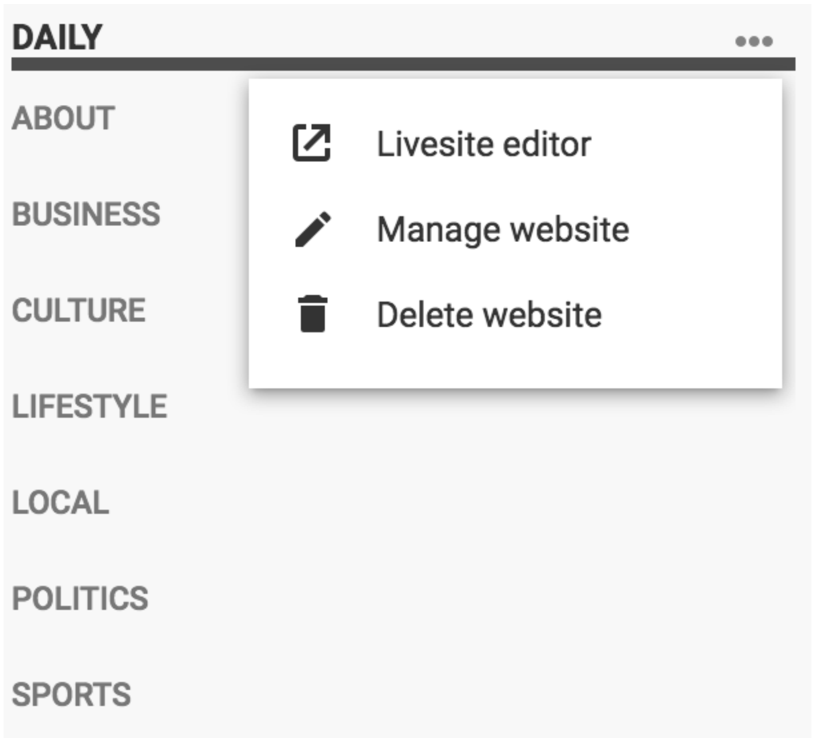


In our example (screenshot 2) you can see three websites configured. As you can see, each site is represented with its name and list of assigned routes.

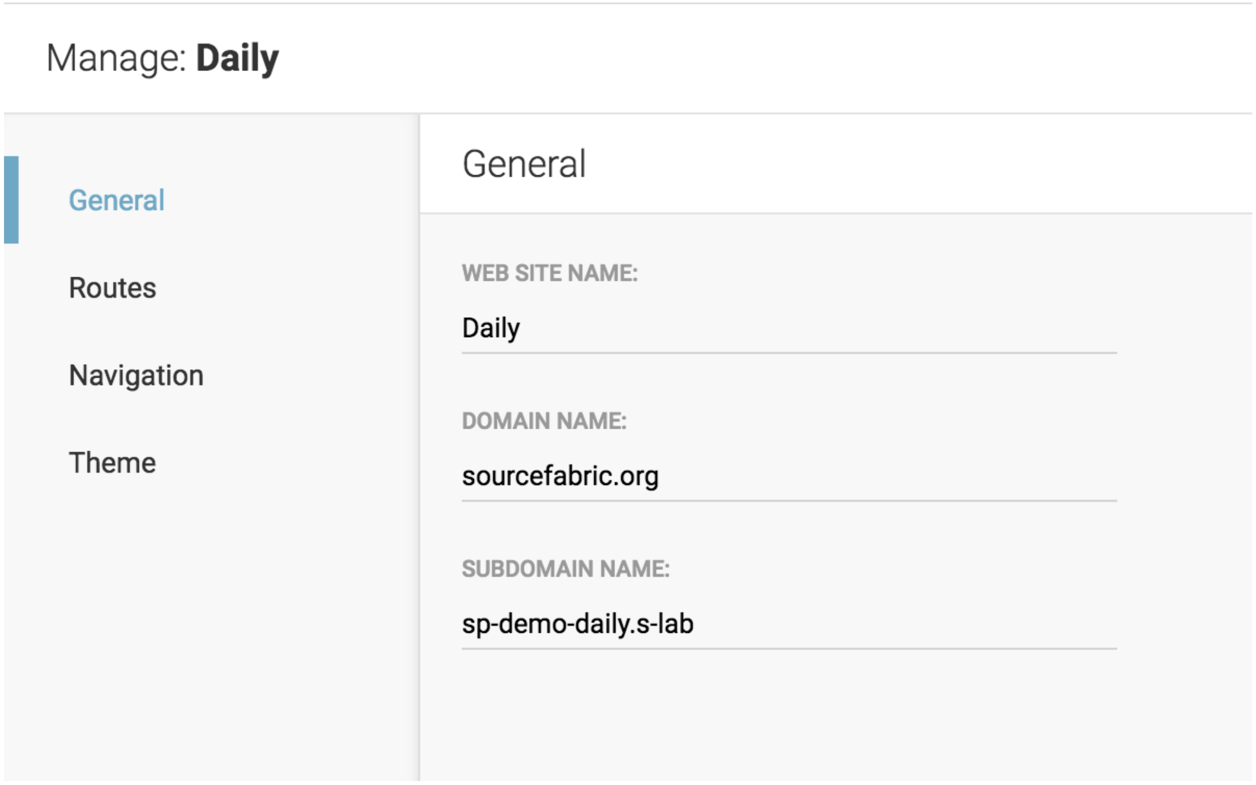


| DAILY NEWS | DEFAULT TENANT | THE MAGAZINE |
|---|---|---|
| BUSINESS | ABOUT | BUSINESS |
| ENTERTAINMENT | BUSINESS | ENTERTAINMENT |
| HEALTH | ENTERTAINMENT | HEALTH |
| POLITICS | HEALTH | POLITICS |
| SCITECH | POLITICS | SCI/TECH |
| SPORTS | SCI/TECH | SPORT |
| TRENDING | SPORT | |

Options to manage or delete websites are available after clicking on the three-dots icon.



Next screenshot shows initial step in managing single website - set its name and (sub)domain.

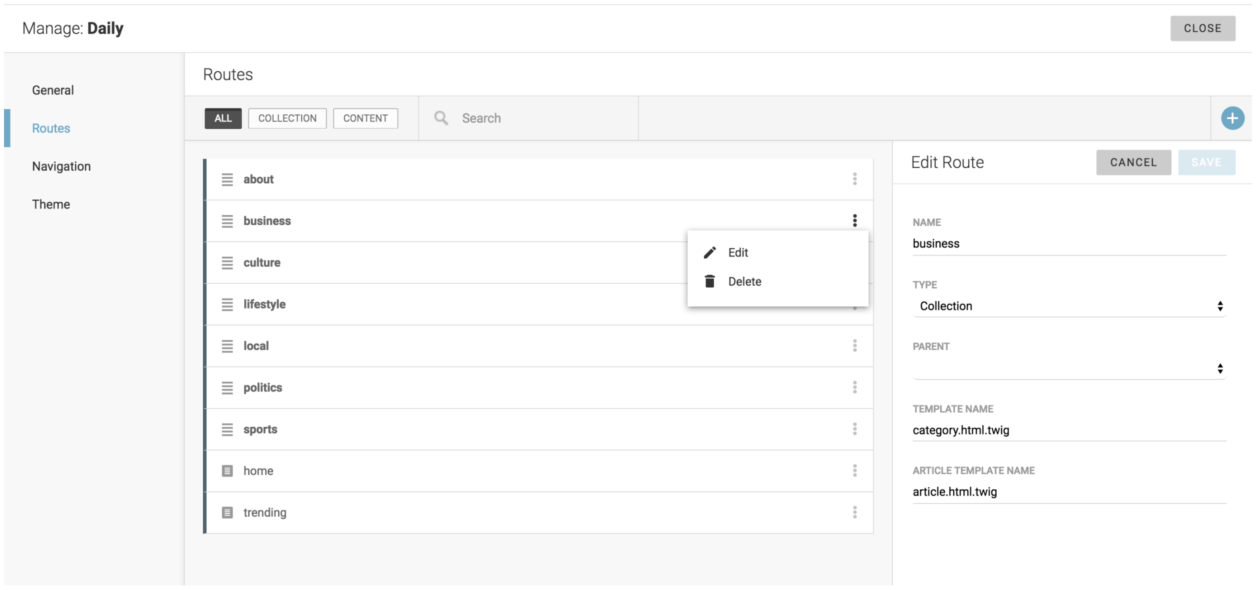


WEB SITE NAME:
Daily

DOMAIN NAME:
sourcefabric.org

SUBDOMAIN NAME:
sp-demo-daily.s-lab

Definition of site routes is the next step. Routes can be of type *collection* or *content* - former is category-like route to which articles are later attached, while latter is an end - with specific content assigned.



NAME
business

TYPE
Collection

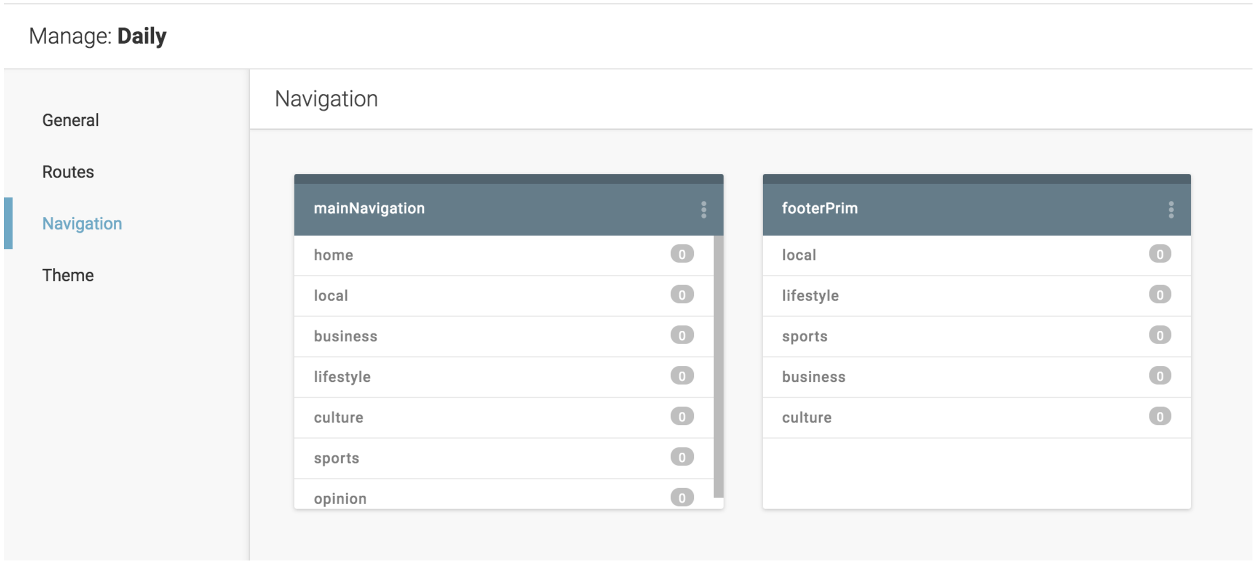
PARENT

TEMPLATE NAME
category.html.twig

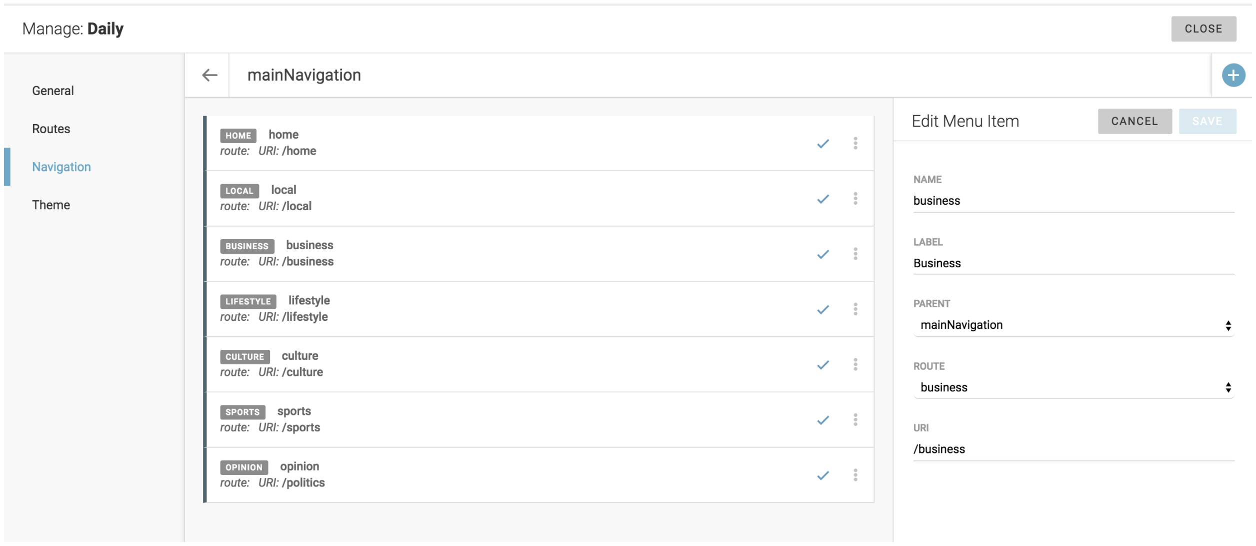
ARTICLE TEMPLATE NAME
article.html.twig

As you can see, route definition consists of name, type (*collection* or *content*), eventual parent route, template used to show this route, and article template to open articles attached to this route (if route is of type *collection*).

Third step in managing a website is to define its navigation. In other words, this option can be used to make navigation menus for header, footer, sidebar or wherever. If created this way, menu can be later also managed through LiveSite editing (of course, menus can also be defined in templates but then they are not dynamic in a way that can be managed y website editors).



Navigation menu consists on menu items - for example Home, Politics, Business etc. Each of these *menu items* is defined by name, label, parent, route (pre-defined in previous step) and uri.



| | |
|---|---|
| HOME home | route: URI: /home |
| LOCAL local | route: URI: /local |
| BUSINESS business | route: URI: /business |
| LIFESTYLE lifestyle | route: URI: /lifestyle |
| CULTURE culture | route: URI: /culture |
| SPORTS sports | route: URI: /sports |
| OPINION opinion | route: URI: /politics |

NAME
business

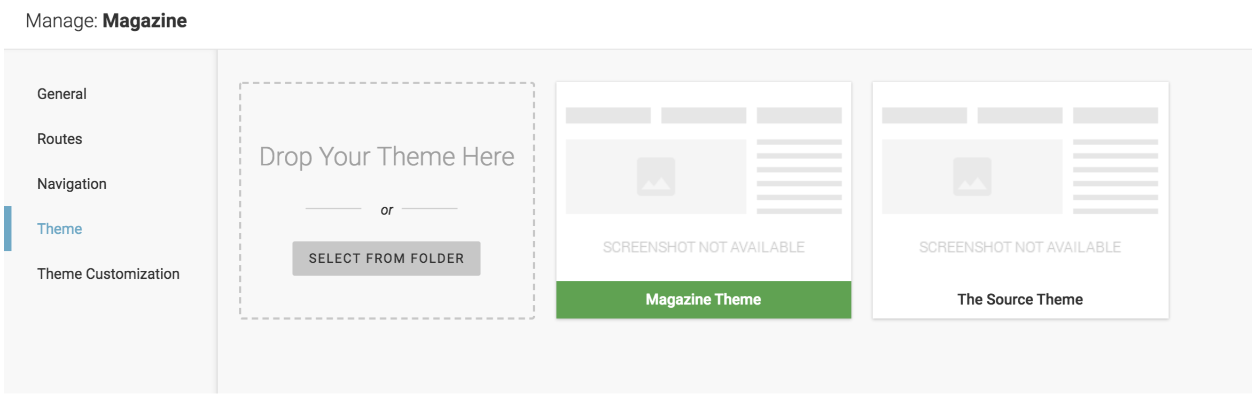
LABEL
Business

PARENT
mainNavigation

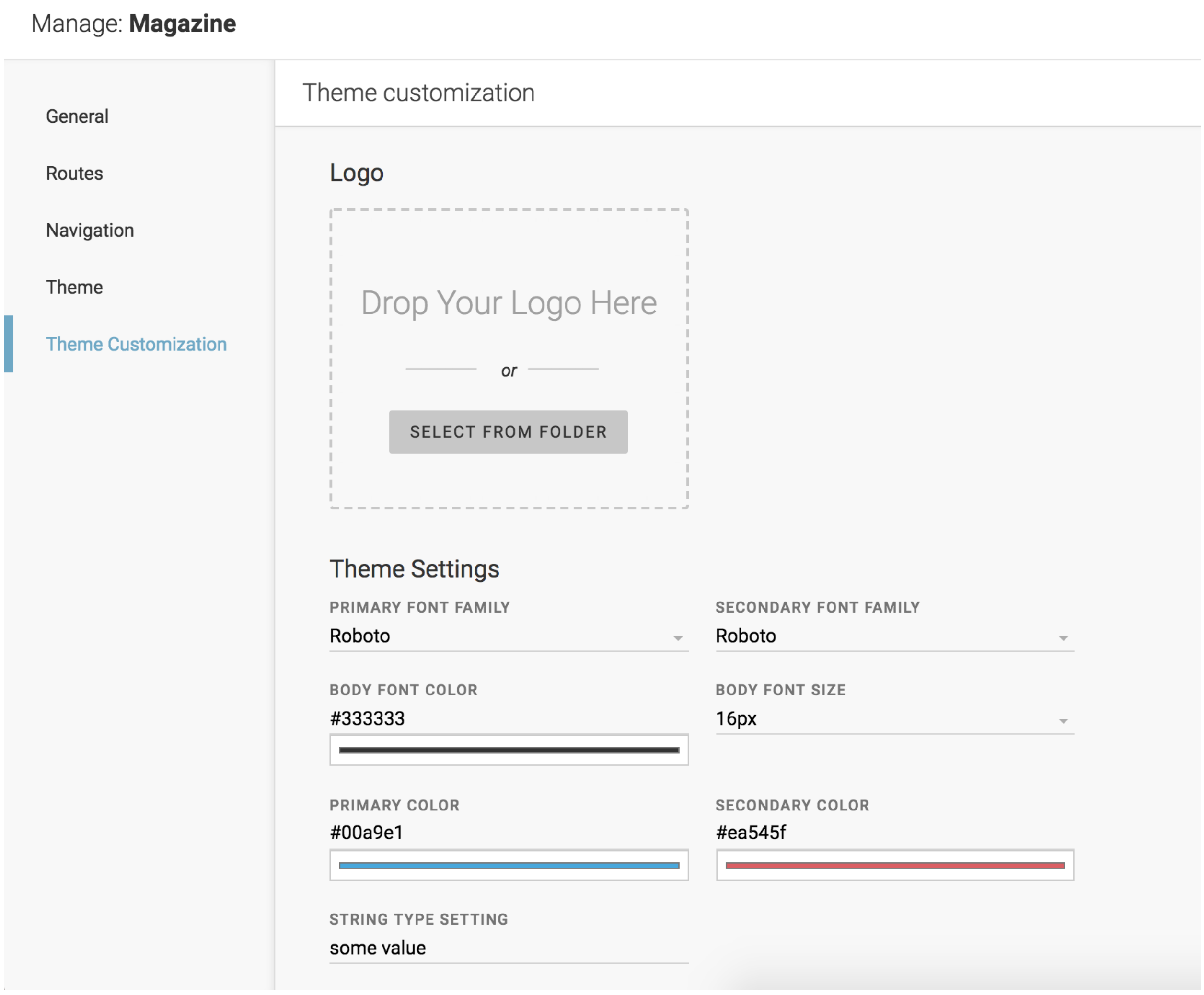
ROUTE
business

URI
/business

Fourth step in site management offers possibilites to choose or change theme.



You can either upload your custom theme or choose one of available themes form list. Currently used theme is marked with green background.

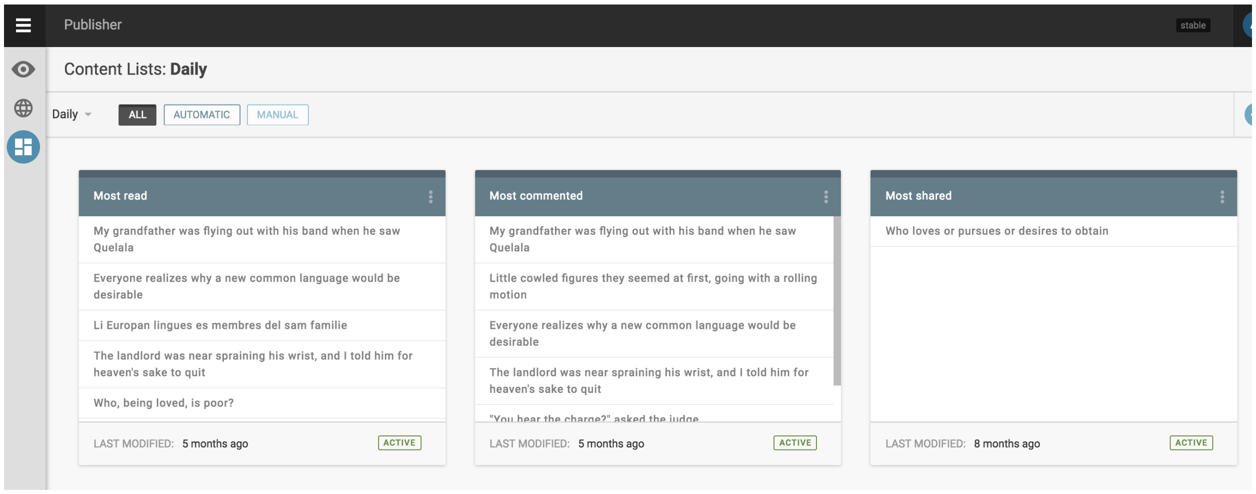


PRIMARY FONT FAMILY: Roboto
SECONDARY FONT FAMILY: Roboto
BODY FONT COLOR: #333333
BODY FONT SIZE: 16px
PRIMARY COLOR: #00a9e1
SECONDARY COLOR: #ea545f
STRING TYPE SETTING: some value

If selected theme supports Theme Settings you can also customise your theme with GUI.

## 1.6.3 Content lists

Publisher offers possibilites to create manual or automatic content lists.



Most read

My grandfather was flying out with his band when he saw Quelala

Everyone realizes why a new common language would be desirable

Li Europan lingues es membres del sam familie

The landlord was near spraining his wrist, and I told him for heaven's sake to quit

Who, being loved, is poor?

LAST MODIFIED: 5 months ago ACTIVE

Most commented

My grandfather was flying out with his band when he saw Quelala

Little cowled figures they seemed at first, going with a rolling motion

Everyone realizes why a new common language would be desirable

The landlord was near spraining his wrist, and I told him for heaven's sake to quit

"You hear the charge?" asked the judge

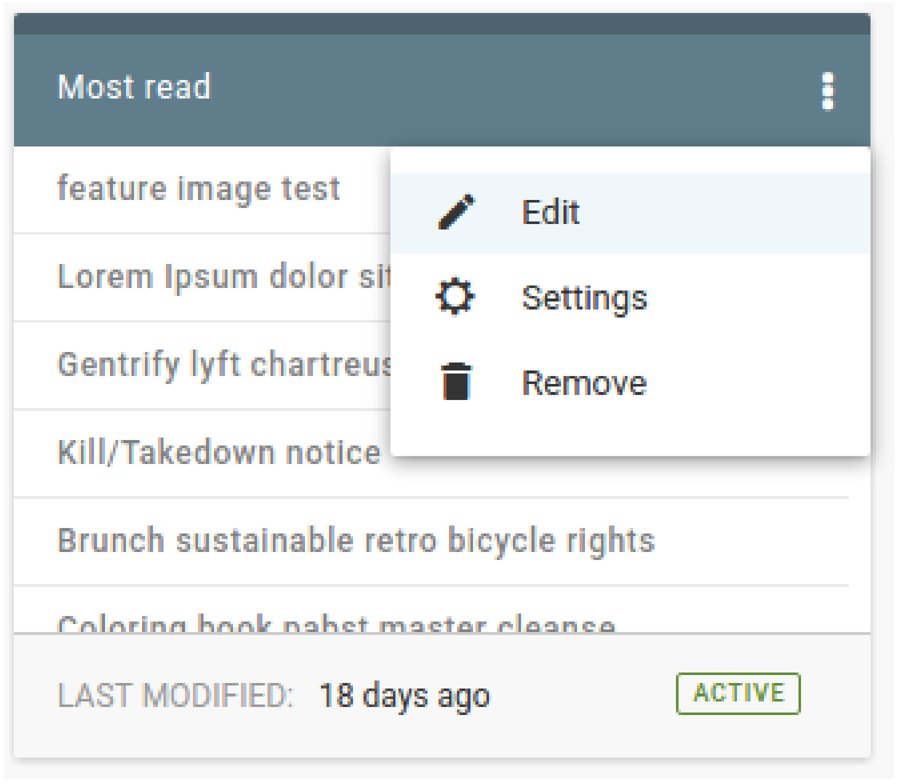
LAST MODIFIED: 5 months ago ACTIVE

Most shared

Who loves or pursues or desires to obtain

LAST MODIFIED: 8 months ago ACTIVE

With automatic content list, it is necessary to set up rules which will be used to fill the list with articles. It can be single rule or combination of several rules. To open the dialog with such options, there is the dialog that opens on three-dot button (another option in this dialogue is *settings*, which enables setting list limit and cache lifetime)



feature image test

Lorem Ipsum dolor sit

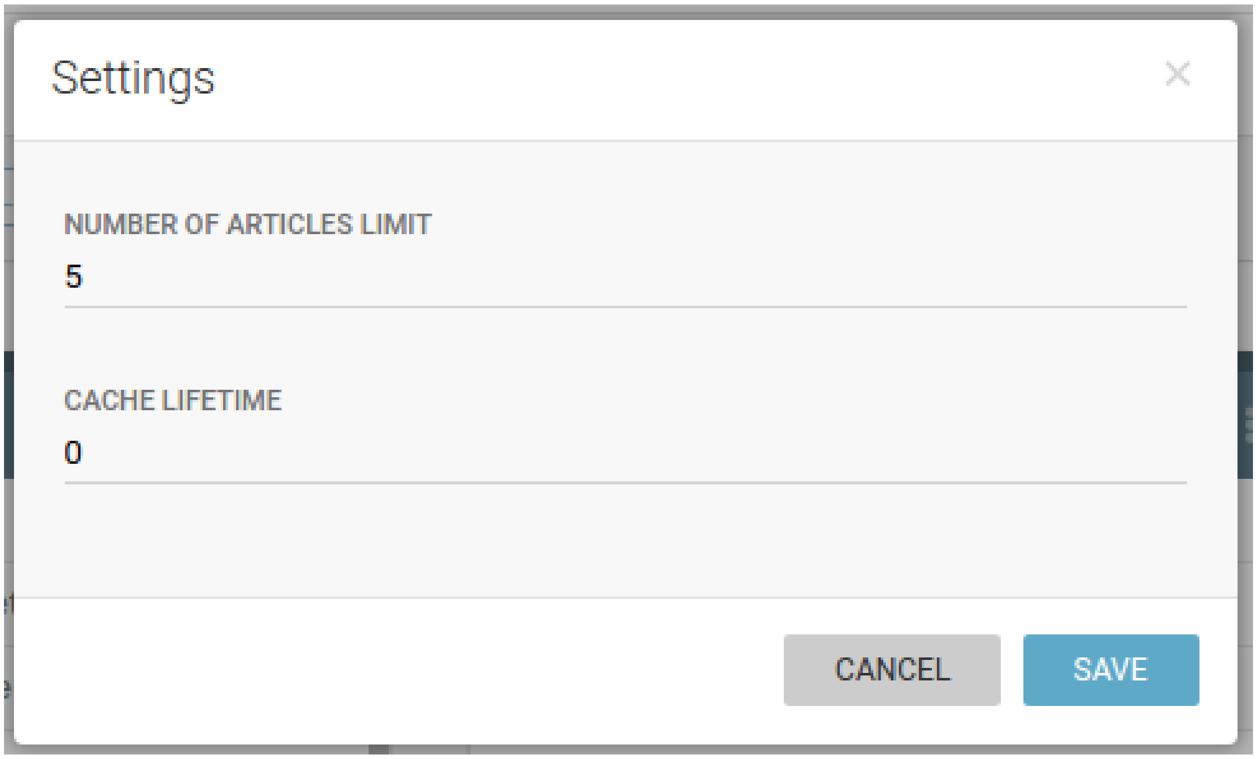
Gentrify lyft chartreus

Kill/Takedown notice

Brunch sustainable retro bicycle rights

Coloring book pabst master cleanse

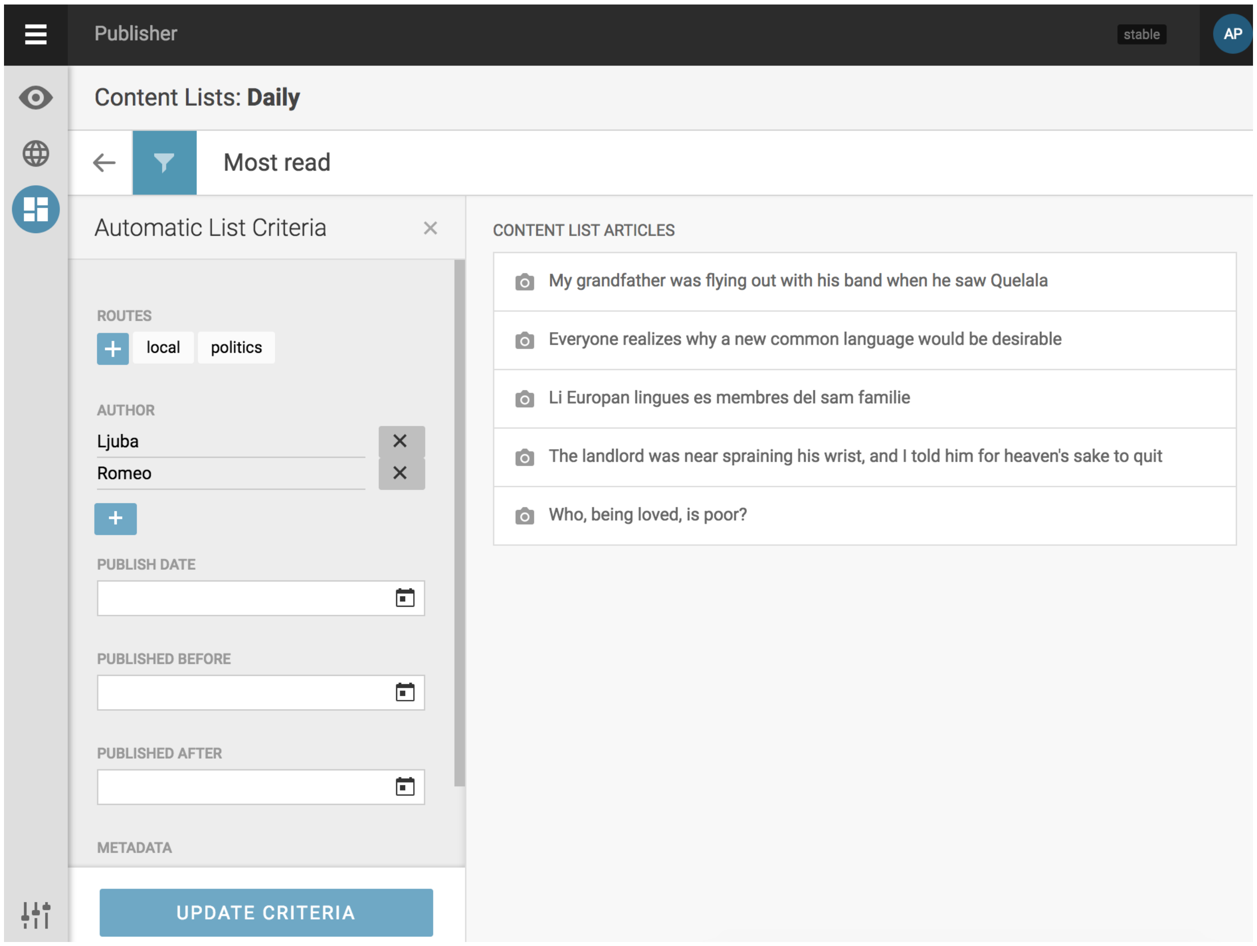
LAST MODIFIED: 18 days ago ACTIVE


Settings

NUMBER OF ARTICLES LIMIT
5

CACHE LIFETIME
0

As you can see, setting automated list criteria is straight-forward process where you can define as many rules as you need.



My grandfather was flying out with his band when he saw Quelala

Everyone realizes why a new common language would be desirable

Li Europan lingues es membres del sam familie

The landlord was near spraining his wrist, and I told him for heaven's sake to quit

Who, being loved, is poor?

# CHAPTER 2

# Cookbooks

## 2.1 Templates Cookbooks

### 2.1.1 Complex static pages with Containers and Widgets

Web site almost always need a way to work with static pages (e.g. About us, Contact, Impressum). For simple 'static' pages such as these, it is usually enough to create page-like articles in CMS back-end - title and one big content field editable in html editor.

But how to approach complex static pages that have several different elements such as boxes, info with fancy changeable backgrounds, custom size graphic elements etc?

In Publisher for such complex, presentation-like static pages we use combination of containers and widgets linked to them - in *LiveSite Editing* mode, editors can then change content of widgets, their order inside parent container, or can turn widgets on or off. And because it's content editing on front-end, visualization of the change is not an obstacle, editors see in real time how their change will affect web page.

First step is defining containers on your page - try to identify zone with specific characteristics and then turn them into containers. For example, page header can be one container, main content bar another, sidebar third, footer fourth container.

It is done in template, with command like this

```
{% container 'frontpage_sidebar' with {
    'styles': 'border: solid 1px red',
    'cssClass': 'css_class_name',
    'data': {'custom-key': value}
} %}
...
{% endcontainer %}
```

Of course, parameters are not mandatory, so your container definition can be as simple as this:

```
{% container 'frontpage_sidebar'%}
...
{% endcontainer %}
```

This will generate block-level element on web page (div) with special class - it is used in live editing to define dynamic canvas where editors can work with widgets.

Containers can have none, one or more widgets - for example, think of frontpage sidebar; if it is defined as container in template, then editors can create several widgets in that container. From time to time they can turn on widget that highlights for example some recent interview; or can put some important announcement on top of the sidebar; etc - and all that without any changes in templates, only by playing wiith *LiveSite Editing* mode.

Default widget that we have in mind here is html widget; it's content is markup that can be prepared externally (even in some html editor), and then just pasted as widget content. so when editors need to change something inside the widget, they will actually apply change in html.

## 2.1.2 Users registration and login

### Registration

User can be registered in Publisher with REST API `/{version}/users/register/` POST request.

```
curl -X POST 'http://webpublisher.dev/api/v1/users/' -H 'Origin: http://webpublisher.
↪dev' -H 'Content-Type: application/x-www-form-urlencoded' -H 'Accept: */*' -H
↪'Connection: keep-alive' -H 'DNT: 1' --data '_format=json&user_registration%5Bemail
↪%5D=pawel.mikolajczuk%40sourcefabric.org&user_registration%5Busername%5D=pawel.
↪mikolajczuk&user_registration%5BplainPassword%5D%5Bfirst%5D=superStronP%40SSword&
↪user_registration%5BplainPassword%5D%5Bsecond%5D=superStronP%40SSword' --compressed
```

After that user will get email message with link for account confirmation. Link will redirect him to `/register/confirmed` page.

#### Customize sender email address:

By default email will be sent from `'contact@{tenant domain}` - example: `contact@example.com`. You can override it by customizing `registration_from_email.confirmation` setting.

#### Customize confirmation email template:

Default template used for confirmation is `@FOSUser/Registration/email.txt.twig` You can override it by customizing `registration_from_email.confirmation` setting.

### Customize account confirmation page template:

After clicking on conformation link (from email) user will be redirected to `/register/confirmed` url. To render this page publisher by default use `'@FOSUser/Registration/confirmed.html.twig`.

You can override it in Your theme (with creating `FOSUser/Registration/confirmed.html.twig` file in your theme.

**Note:** Read more about settings in *SettingsBundle*.

## Login

Publisher don't provide single page for login action, instead that we made sure that login can be placed in any template (or even widget) of You choice. Only hardcoded url's for security are `/security/login_check` (used for user authentication - you need to send your login form data there) and `/security/logout` (used for logging out user).

### Example login form:

```
{% if app.session.has('_security.last_error') %}
    {# show error message after unsuccessful login attempt #}
    {% set error = app.session.get('_security.last_error') %}
    <div>{{ error.messageKey|trans(error.messageData, 'security') }}</div>
{% endif %}
<form action="{{ path('security_login_check') }}" method="POST">
    <input type="text" name="_username" value="{{ app.session.get('_security.last_
↪username') }}" />
    <input type="password" name="_password" value="" />
    <input type="hidden" name="_login_success_path" value="{{ url('homepage') }}">
    <input type="hidden" name="_login_failure_path" value="{{ url('homepage') }}">
    <input type="submit" value="Login" />
</form>
```

Parameters explanation:

- `_username` - User username parameter (You can use `{{ app.session.get('_security.last_username') }}` as a default value - it will be filled with previously entered username when login will fail and user will be redirected to login form again).
- `_password` - User password
- `_login_success_path` - url used for redirection after successful login
- `_login_failure_path` - url used for redirection after unsuccessful login (use `{{ url(gimme.page) }}` to generate url for current page)

### Check if users is logged in:

```
{% if app.user %}
    Hey {{ app.user.username }}. <a href="{{ url('security_logout') }}">Logout</a>.
{% else %}
    {# Show login form #}
{% endif %}
```

### Password Reset

By default, there are a few routes exposed for password reset functionality (provided by FOSUserBundle).

When going to route `/resetting/request` (route name is `fos_user_resetting_request`) the default password reset form will be rendered.

You can override this template in your theme by creating `FOSUserBundle/views/Resetting/request.html.twig` file in your theme.

All templates related to password reset functionality which can be overridden can be found here.

## 2.1.3 Create new route with extension (example: `feed/sitemap.rss`)

Very common case for newspaper website is providing some custom rss feeds. To accomplish that with Publisher you need to create new route (ex: `feed/sitemap`) and attach selected template to it.

After hitting this route you will see output generated by template but you will find out that Content Type of this route response is set to `text/html` - and it can create issues for some parsers.

To fix that problem you just need to rename your route to one with extension at end. For example `feed/sitemap.rss`. Publisher will automatically recognize route extension and will set proper Content-Type header - in this case: `application/rss+xml`.

This feature can be used for generating custom xml files, or even JavaScript files (with extension `.js`).

## 2.1.4 Create new custom route

Our use case will be article authors pages. As we don’t want to have special route for every author - we need to create one witch will fit all our needs.

We need url like that: `publisher.dev/authors/{authorSlug}`. Author slug is a parameter passed to template and will be used by us for loading desired author. Example filled route: `publisher.dev/authors/john-doe`.

We can create route like that with API:

`POST /api/v1/content/routes/`

```json
{
    "route": {
        "name": "Authors",
        "slug": "authors",
        "type": "custom",
        "templateName": "author.html.twig",
        "variablePattern": "/{authorSlug}",
        "requirements": [
            {
                "key": "authorSlug",
                "value": "[a-zA-Z\\-_]+"
            }
        ]
    }
}
```

Important parts:

- type `custom`: says to publisher that we want to set `variablePattern` and `requirements` manually.

- `variablePattern` - string with set parameters placeholders added to end of route.
- `requirements` = array of objects with regular expression for provided parameters.

Now in template `author.html.twig` we can load author by slug provided in url.

```
{% gimme author with { slug: app.request.attributes.get('authorSlug') } %}
    Author name: {{ author.name }}
{% endgimme %}
```

And done - you have custom route with option to pass parameters in url and use them later in template.

### 2.1.5 Most popular (read) articles list

Article meta have property `articleStatistics` and inside it You can find `pageViewsNumber`. To simplify syntax for ordering we created `pageViews` alias.

Here is example how to order articles by their page views number:

```
{% gimmelist article from articles|order('pageViews', 'desc') %} <!-- ordering by␣
↪page views -->
    <a href="{{ url(article) }}">{{ article.title }}</a>
{% endgimmelist %}
```

There is also option to get most popular (ordered by page views) articles from date range.

For example this is how You can list yesterday most popular articles:

```
{% gimmelist article from articles|order('pageViews', 'desc')|dateRange('now', '-1 day
↪') %}
    <a href="{{ url(article) }}">{{ article.title }}</a>
{% endgimmelist %}
```

Filter `dateRange` takes two parameters compatible with PHP strtotime syntax (http://php.net/manual/en/function.strtotime.php):

- start date (<= equal or in past ) - time will be reset to 23:59:59
- end date (>= equal or in feature) - time will be reset to 00:00:00

So `|dateRange('now', '-1 day')` will filter all page views from whole day today and whole yesterday (from midnight)

To activate article page views counting you need to call short twig function.

```
{{ countPageView(gimme.article) }}
```

It will print `<script>` tag with some generated by Publisher javascript code inside.

### 2.1.6 Elements generated on theme installation

To provide better experience for end users after theme installation Publisher can create some default content used by theme. Thanks to this feature theme developer can be sure that all of his article or category templates will be visible without complicated configuration by end user.

Generated elements can be declared in theme.json config file under `generatedData` key. Example:

```
{
 "name": "my/custom-theme",
 "generatedData": { ... }
}
```

Theme generators supports now those elements: `routes`, `menus`, `containers`, `widgets`, and `contentLists`. All elements have this same properties as are supported by API requests, plus few extra like:

- in routes: `numberOfArticles` - number of fake articles generated and attached to route
- in menus: `children` - array of child menus attached to parent one
- in widgets: `containers` - array of containers names where widgets should be automatically attached

## Example routes block

```
"generatedData": {
    "routes": [
        {
            "name": "Politics",                             # required
            "slug": "politics",                             # optional
            "type": "collection",                           # required
            "templateName": "category.html.twig",           # optional
            "articlesTemplateName": "article.html.twig",    # optional
            "numberOfArticles": 1                           # optional (number of␣
↪articles generated and attached to route)
        },
    ...
```

## Example menus block

```
"generatedData": {
    "menus": [
        {
            "name": "mainNavigation",               # required
            "label": "Main Navigation",             # optional
            "children": [                           # optional (array of child menus␣
↪attached to parent one)
                {
                    "name": "home",                 # required
                    "label": "Home",                # optional
                    "uri": "/"
                }
            ]
        },
        {
            "name": "footerPrim",                   # required
            "label": "Footer Navigation",           # optional
            "children": [                           # optional (array of child menus␣
↪attached to parent one)
                {
                    "name": "politics",             # required
                    "label": "Politics",            # optional
                    "route": "Politics"             # optional (route name - can be␣
↪one defined in this config)
```

(continues on next page)

(continued from previous page)

```
                },
    ...
```

### Example containers block

```
"generatedData": {
    "containers": [
        {
            "name": "mainNav",                      # required
            "styles": "border: 1px ",               # optional
            "visible": true,                        # optional (default true)
            "cssClass": "col-md-12",                # optional
            "data": {                               # optional
                "some_extra_data": true
            }
        },
        {
            "name": "frontArticles"                 # required
        }
    ...
```

### Example widgets block

```
"generatedData": {
    "widgets": [
        {
            "name": "NavigationMain",               # required
            "type": "SWP\\Bundle\\TemplatesSystemBundle\\Widget\\MenuWidgetHandler",
↪# required
            "visible": true,                        # required
            "parameters": {                         # optional
                "menu_name": "mainNavigation",
                "template_name": "menu1.html.twig"
            },
            "containers": ["mainNav"]               # optional
        },
    ...
```

### Example contentLists block

```
"generatedData": {
    "contentLists": [
        {
            "name": "Example automatic list",                   # required
            "type": "automatic",                                # required
            "description": "New list",                          # required
            "limit": 5,                                         # optional
            "cacheLifeTime": 30,                                # optional
            "filters": "{\"metadata\":{\"located\":\"Porto\"}}"     # optional
        }
    ...
```

## 2.2 Technical Cookbooks

### 2.2.1 Installing Composer

**Note:** This section is based on Symfony2 documentation.

Composer is the package manager used by modern PHP applications. Use Composer to manage dependencies in your Symfony applications and to install Symfony Components in your PHP projects.

It's recommended to install Composer globally in your system as explained in the following sections.

#### Install Composer on Linux and Mac OS X

To install Composer on Linux or Mac OS X, execute the following two commands:

```
curl -sS https://getcomposer.org/installer | php
sudo mv composer.phar /usr/local/bin/composer
```

**Note:** If you don't have `curl` installed, you can also just download the `installer` file manually at https://getcomposer.org/installer and then run:

```
php installer
sudo mv composer.phar /usr/local/bin/composer
```

#### Install Composer on Windows

Download the installer from getcomposer.org/download, execute it and follow the instructions.

#### Learn more

Read the Composer documentation to learn more about its usage and features.

### 2.2.2 Updating database schema after customizing model's mapping

If you modified one of the model's mapping file or you added completely new model with mapping, you will need to update the database schema.

There are two ways to do it.

- via direct database schema update:

```
$ php app/console doctrine:schema:update --force
```

- via migrations:

We recommend to update the schema using migrations so you can easily rollback and/or deploy new changes to the database without any issues.

```
$ php app/console doctrine:migrations:diff
$ php app/console doctrine:migrations:migrate
```

**Tip:** Read more about the database modifications and migrations in the Symfony documentation here.

### 2.2.3 Meta Loaders

Meta Loaders are services injected into the `SWP\TemplatesSystemBundle\Gimme\Loader\ChainLoader` class and are used for loading specific types of Meta objects.

Every Meta Loader must implement the `SWP\TemplatesSystemBundle\Gimme\Loader\LoaderInterface` interface.

Required methods:

- load
- isSupported

**Example Meta Loader**

```
<?php

namespace SWP\Component\TemplatesSystem\Gimme\Loader;

use SWP\Component\TemplatesSystem\Gimme\Factory\MetaFactory;
use SWP\Component\TemplatesSystem\Gimme\Meta\Meta;
use Symfony\Component\Yaml\Parser;

class ArticleLoader implements LoaderInterface
{
    /**
     * @var string
     */
    protected $rootDir;

    /**
     * @var MetaFactory
     */
    protected $metaFactory;

    /**
     * @param string $rootDir path to application root directory
     */
    public function __construct($rootDir, MetaFactory $metaFactory)
    {
        $this->rootDir = $rootDir;
        $this->metaFactory = $metaFactory;
    }

    /**
     * Load meta object by provided type and parameters.
     *
     * @MetaLoaderDoc(
     *     description="Article Meta Loader provide simple way to test Loader, it␣
↪will be removed when real loaders will be merged.",
```

(continues on next page)

(continued from previous page)

```
     *     parameters={}
     * )
     *
     * @param string $type               object type
     * @param array  $parameters         parameters needed to load required object type
     * @param array  $withoutParameters parameters used to exclude items from result
     * @param int    $responseType       response type: single meta␣
↪(LoaderInterface::SINGLE) or collection of metas (LoaderInterface::COLLECTION)
     *
     * @return Meta|array false if meta cannot be loaded, a Meta instance otherwise
     */
    public function load($type, array $parameters = null, $withoutParameters = [],
↪$responseType = LoaderInterface::SINGLE)
    {
        if (!is_readable($this->rootDir.'/Resources/meta/article.yml')) {
            throw new \InvalidArgumentException('Configuration file is not readable␣
↪for parser');
        }
        $yaml = new Parser();
        $configuration = (array) $yaml->parse(file_get_contents($this->rootDir.'/
↪Resources/meta/article.yml'));

        if ($responseType === LoaderInterface::SINGLE) {
            return $this->metaFactory->create([
                'title' => 'New article',
                'keywords' => 'lorem, ipsum, dolor, sit, amet',
                'don\'t expose it' => 'this should be not exposed',
            ], $configuration);
        } elseif ($responseType === LoaderInterface::COLLECTION) {
            return [
                $this->metaFactory->create([
                    'title' => 'New article 1',
                    'keywords' => 'lorem, ipsum, dolor, sit, amet',
                    'don\'t expose it' => 'this should be not exposed',
                ], $configuration),
                $this->metaFactory->create([
                    'title' => 'New article 2',
                    'keywords' => 'lorem, ipsum, dolor, sit, amet',
                    'don\'t expose it' => 'this should be not exposed',
                ], $configuration),
            ];
        }
    }

    /**
     * Checks if Loader supports provided type.
     *
     * @param string $type
     *
     * @return bool
     */
    public function isSupported($type)
    {
        return in_array($type, ['articles', 'article']);
    }
}
```

### 2.2.4 Widgets

A container may have n widgets, each of which represents a fragment of the part of the page which is represented by the container. See /templates_system/containers_and_widgets for more details.

#### How to create a new type of widget?

Add a new constant to the /src/SWP/Bundle/TemplatesSystemBundle/Model/WidgetModel class, and a reference to a class which extends `AbstractWidgetHandler.php` in `/src/SWP/Component/TemplatesSystem/Gimme/Widget`

For example:

```
class WidgetModel implements WidgetModelInterface, TenantAwareInterface,␣
↪TimestampableInterface
{
    const TYPE_NEW = 1;
    ...

    protected $types = [
        self::TYPE_NEW =>
↪'\\SWP\\Component\\TemplatesSystem\\Gimme\\Widget\\NewWidgetHandler',
        ...
    ];

    ...
```

You must then create that class which you have referenced.

As well as having to implement the render function in this class, you can define what, if any parameters should be set in an instance of a widget model which references this class by adding a static variable called `$expectedParameters`.

For example:

```
class NewWidgetHandler extends AbstractWidgetHandler
{
  protected static $expectedParameters = array(
      'parameter_name' => [
          'type' => 'string',             // or bool, int, float
          'default' => 'default_value'    // if no default is provided, the parameter␣
↪must be set
      ]
  );

  ...
```

If there is no default value, the default value, by default, will be null.

Typically, the render function will use the template engine to render a template which requires the given parameters.

For example:

```
class NewWidgetHandler extends AbstractWidgetHandler
{
  ...

  /**
```

(continues on next page)

(continued from previous page)

```
     * Render widget content.
     *
     * @return string
     */
    public function render()
    {
        return $this->renderTemplate('template_name.html.twig');
    }

    ...
```

The referenced twig file should be located in the theme's views/widgets directory. The implementation of the renderTemplate method in the base class prepends 'widgets/' to the given string, and passes this argument, along with the expectedParameters and their values in an associative array, to the template engine's render method.

### Available widgets

By default there are three types of widgets bundled into the project:

- ContentListWidget
- HtmlWidget (default one)
- GoogleAdsenseWidget
- MenuWidget
- LiveblogWidget
- TemplateWidget

### ContentListWidget

This is the widget responsible for displaying Content Lists' items. For more details about Content Lists and Content List Items see *Content Lists* and *ContentListBundle* sections.

**Default parameters:**

| Parameter | Description | Required? | Default value | Type |
|---|---|---|---|---|
| list_id | List's id | yes | N/A | int |
| template_name | Template name to render | yes | list.html.twig | string |

### LiveblogWidget

This is the widget responsible for displaying Superdesk LiveBlog embeds.

**Default parameters:**

| Parameter | Description | Required? | Default value | Type |
|---|---|---|---|---|
| uri | Liveblog embed (fragment) uri | yes | N/A | string |
| template_name | Template name to render | yes | liveblog.html.twig | string |

Default template file name: `liveblog.html.twig` (default version provided by Publisher can be overriden by theme).

#### TemplateWidget

This widget is used to trigger some smaller template functionality, for example list of most popular articles in last week, or featured author profile. It is by default looking for template in folder *views/widgets*

**Default parameters:**

| Parameter | Description | Required? | Default value | Type |
|---|---|---|---|---|
| template_name | Template name to render | yes | N/A | string |

## 2.2.5 Content Lists

### Automatic Content Lists

#### Pinned articles

This feature gives you a way of exposing more important articles on top of the list.

In automatic content lists you have the possibility to pin/unpin articles from the list. If you decide to pin one of the article from the list, it will always show up on top of the list, no matter how many new articles will be added to that list.

You can pin as many articles as you want.

Pinned article can be unpinned too. Once it's done, the unpinned article will be removed from the top of the list and will remain in the list on the corresponding position.

**By default, articles in Automatic Content List are ordered by "sticky" flag (desc)** and item creation date (desc).

#### How articles are being added to Automatic Content Lists

When a new list is created and relevant criteria/filters are set for that list, articles to be published will automatically be added to the list but only when they meet the list's criteria.

If the list's criteria are not defined, or if the articles do not match the list's criteria, articles will not be added to the list.

Here is an example to demonstrate this:

Given that your list's criteria/filters are set to match "Sports" route so when you publish an article which is assigned to "Sports" route, it will be automatically assigned to the list.

Built in criteria:

- `route` - an array of route ids, e.g. [1,5]
- `author` - an array of authors, e.g. ["Test Persona","Doe"]
- `publishedBefore` - date string, articles published before this date will be added to the list, e.g. date: "2017-01-20". (date format must be YYYY-MM-DD)
- `publishedAfter` - date string, articles published after that date will be added to the list, format is the same as in the `publishedBefore` case.
- `publishedAt` - date string, when defined articles matching this publish dates will be added to the list when published, format is the same as in case of `publishedBefore` and `publishedAfter`
- `metadata` - metadata field is json string, e.g. `{"metadata":{"language":"en"}}`. It matches article's metadata, and you can use all metadata fields that are defined for the article, i.e.: language, located etc.

All criteria can be combined together which in the result it will add articles to the list (on publish) depending on your needs.

### 2.2.6 Article Preview

Article preview is based on user roles. Every article which is not published yet can be previewed by users with special roles assigned to them. This role is named `ROLE_ARTICLE_PREVIEW`.

If a user has `ROLE_ARTICLE_PREVIEW` role assigned, he/she can preview article using url: `domain.com/preview/article/<routeId>/<article-slug>/?auth_token=<token>`.

Where `<routeId>` is route identifier on which you want to preview given article by it's slug (`<article-slug>` parameter).

Important here is to provide token, in order to be authorized to preview an article.

**Tip:** See *API Authentication* section for more details on how to obtain user token.

For example, if you created an article that has a slug `test-article` and this article is assigned to `news` route which id is 5, it will be available for preview under `/preview/article/5/test-article?auth_token=uty56392323==` url but only when the user has `ROLE_ARTICLE_PREVIEW` role assigned. In other cases 403 error will be thrown.

If you are building JavaScript app and you want to preview article, the preview url of an article can be taken and loaded in an iframe for preview.

#### User roles eligible for article preview:

| Role |
|---|
| `ROLE_EDITOR` |
| `ROLE_INTERNAL_API` |
| `ROLE_ADMIN` |
| `ROLE_SUPER_ADMIN` |

### 2.2.7 Rules

Rules are a way to define the business logic inside the system. They define how that business logic should be organized. A good example is a rule defining an article auto-publish. The organization rule can be configured to forward the content received from Superdesk to one of the defined tenants. Then tenant's rule can be configured to automatically publish an article under specific route, so if the content is received in Superdesk Publisher, it can be automatically forwarded to the defined tenants (it won't be published by default) and will be published under the specific route defined by the tenant's rule.

There are two types of rules which are defined in Superdesk Publisher: - organization rules - applied to the level of organization - tenant rules - applied to the level of tenant

#### Auto-publish content based on rules

#### 1. Adding an organization rule to push content to the desired tenants

Let's assume there is a single organization created which contains a single tenant with code `123abc`.

The next step is to create an organization rule by making a `POST` request to the `/api/v1/organization/rules/` API endpoint with the JSON body:

```
{
    "rule":{
      "name":"Test rule",
      "description":"Test rule description",
      "priority":1,
      "expression":"package.getLocated() matches \"/Sydney/\"",
      "configuration":[
        {
          "key":"destinations",
          "value":[
            {
              "tenant":"123abc"
            }
          ]
        }
      ]
    }
  }
```

In the JSON above we define that if the content which comes from Superdesk has a field `located` and it matches the value of `Sydney`, then push the content to the tenant with the code equal to `123abc`.

On the tenant level, a new article will be created based on the pushed content which won't be published by default. Right now, this article can be manually published or route can be assigned to it manually etc.

In order to publish it automatically, read below.

### 2. Adding a tenant rule to autopublish content under specific route

The next step is to create a rules on the level of tenant to automatically publish the article under specific route.

In order to do that, a `POST` request must be made to the `/api/v1/rules/` API endpoint with the JSON body:

```
{
    "rule":{
      "name":"Test tenant rule",
      "description":"Test tenant rule description",
      "priority":1,
      "expression":"article.getMetadataByKey(\"located\") matches \"/Sydney/\"",
      "configuration":[
        {
          "key":"route",
          "value":6
        },
        {
          "key":"published",
          "value":true
        }
      ]
    }
  }
```

The above JSON string defines, if an article's metadata field called `located` equals to `Sydney` then assign route with id `6` to the article and automatically publish it.

### 3. Adding a tenant rule to publish content to Facebook Instant Articles

Article can be also published to Facebook Instant Articles. To do that, create a new tenant rule by making a `POST` request to the `/api/v1/rules/` API endpoint with the JSON body:

```
{
    "rule":{
      "name":"Test tenant rule",
      "description":"Test tenant rule description",
      "priority":1,
      "expression":"article.getMetadataByKey(\"located\") matches \"/Sydney/\"",
      "configuration":[
        {
          "key":"fbia",
          "value":true
        }
      ]
    }
}
```

Note the `fbia` key in the `configuration` property is set to `true`.

If the content will be pushed to the tenant, the content will be also submitted to the Facebook Instant Articles.

Read more about Facebook Instant Articles in *this section*.

### 4. Adding a tenant rule to make an article paywall-secured

Articles can be marked as paywall-secured so an access can be restricted to such articles. To do that, create a new tenant rule by making a `POST` request to the `/api/v1/rules/` API endpoint with the JSON body:

```
{
    "rule":{
      "name":"Make articles paywall-secured",
      "description":"Marks articles as paywall-secured.",
      "priority":1,
      "expression":"article.getMetadataByKey(\"located\") matches \"/Sydney/\"",
      "configuration":[
        {
          "key":"paywallSecured",
          "value":true
        }
      ]
    }
}
```

Note the `paywallSecured` key in the `configuration` property is set to `true`.

If the content will be pushed to the tenant and will match given expression, the “paywall-secured” flag will be set to `true`.

Read more about Paywall in *this section*.

### Evaluation of the rules which match given package/item

Based on the package, there is a possibility to evaluate rules that match given package’s/item’s metadata. If the package/item in NINJS format will be passed to the `/api/v1/organization/rules/evaluate` as a request’s payload, then the (organization and/or tenant) rules that match that package's/item's metadata will be returned.

## 2.2.8 Usage

### What is Output Channel Adapter?

Output Channel Adapter is a service which helps to communicate with the external system, for example, Wordpress. Thanks to the concept of adapters it is possible to exchange data between 3rd party services. It is possible to send the data from Publisher to an external system and also get the data from that system.

The Output Channel Adapters are strictly connected to the concept of Output Channels.

It is possible to choose the type of the external system (e.g. WordPress) if a new tenant is created.

It means that the content which is sent to Superdesk Publisher will not only be stored in the Publisher's storage, but it can also be transmitted to an external system.

In this case, thanks to the output channels, you can send content wherever you want.

Superdesk Publisher acts as a hub where you can control where the content goes.

### Adding New Adapter to work with Output Channels

A new adapter must implement `SWP\Bundle\CoreBundle\Adapter\AdapterInterface` interface.

1. Add a new const (`CUSTOM_TYPE`) to the `SWP\Component\OutputChannel\Model\OutputChannelInterface` interface.
2. Create a custom class:

```php
<?php

// CustomAdapter.php
declare(strict_types=1);

namespace SWP\Bundle\CoreBundle\Adapter;

use GuzzleHttp\ClientInterface;
use SWP\Bundle\CoreBundle\Model\ArticleInterface;
use SWP\Bundle\CoreBundle\Model\OutputChannelInterface;

final class CustomAdapter implements AdapterInterface
{
    /**
     * @var ClientInterface
     */
    private $client;

    /**
     * WordpressAdapter constructor.
     *
     * @param ClientInterface $client
     */
    public function __construct(ClientInterface $client)
    {
        $this->client = $client;
    }
```

(continues on next page)

(continued from previous page)

```

    /**
     * {@inheritdoc}
     */
    public function send(OutputChannelInterface $outputChannel, ArticleInterface
↪$article): void
    {
        $url = $outputChannel->getConfig()['url'];

        $this->client->post($url, [
            'headers' => ['Content-Type' => 'application/json'],
            'body' => $article->getBody(),
            'timeout' => 5,
        ]);
    }

    /**
     * {@inheritdoc}
     */
    public function supports(OutputChannelInterface $outputChannel): bool
    {
        return OutputChannelInterface::TYPE_CUSTOM === $outputChannel->getType();
    }
}
```

3. Create Custom Adapter configuration form type

This form type will define which fields we can specify for the adapter. It can be credentials to connect to the 3rd party service etc.

```
<?php

declare(strict_types=1);

namespace SWP\Bundle\OutputChannelBundle\Form\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Validator\Constraints\NotBlank;
use Symfony\Component\Validator\Constraints\Url;

final class CustomOutputChannelConfigType extends AbstractType
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('url', TextType::class, [
                'constraints' => [
                    new NotBlank(),
                    new Url(),
                ],
            ])
            ->add('key', TextType::class, [
```

(continues on next page)

(continued from previous page)

```
                'constraints' => [
                    new NotBlank(),
                ],
            ])
            ->add('secret', TextType::class, [
                'constraints' => [
                    new NotBlank(),
                ],
            ])
        ;
    }
}
```

4. Include a new type and form type in the `OutputChannelType`

```
<?php

declare(strict_types=1);

namespace SWP\Bundle\OutputChannelBundle\Form\Type;

use SWP\Bundle\CoreBundle\Model\OutputChannel;
use SWP\Component\OutputChannel\Model\OutputChannelInterface;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Form\FormEvent;
use Symfony\Component\Form\FormEvents;
use Symfony\Component\Form\FormInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

final class OutputChannelType extends AbstractType
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('type', ChoiceType::class, [
                'choices' => [
                    'Wordpress' => OutputChannelInterface::TYPE_WORDPRESS,
                    'Custom' => OutputChannelInterface::TYPE_CUSTOM,
                ],
            ])
        ;

        $formModifier = function (FormInterface $form, ?string $type) {
            if (OutputChannelInterface::TYPE_WORDPRESS === $type) {
                $form->add('config', WordpressOutputChannelConfigType::class);
            }

            if (OutputChannelInterface::TYPE_CUSTOM === $type) {
                $form->add('config', CustomOutputChannelConfigType::class);
            }
        };

```

(continues on next page)

(continued from previous page)

```php
        $builder->addEventListener(
            FormEvents::POST_SET_DATA,
            function (FormEvent $event) use ($formModifier) {
                $data = $event->getData();
                if (null !== $event->getData()) {
                    $formModifier($event->getForm(), $data->getType());
                }
            }
        );

        $builder->get('type')->addEventListener(
            FormEvents::POST_SUBMIT,
            function (FormEvent $event) use ($formModifier) {
                $type = $event->getForm()->getData();

                $formModifier($event->getForm()->getParent(), $type);
            }
        );
    }

    /**
     * {@inheritdoc}
     */
    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'csrf_protection' => false,
            'data_class' => OutputChannel::class,
        ]);
    }

    /**
     * {@inheritdoc}
     */
    public function getBlockPrefix(): string
    {
        return 'swp_output_channel';
    }
}
```

5. Register your new adapter

Your new adapter must be registered so it can be detected by the system and used by the Publisher. It can be done by taggin a service with `swp.output_channel_adapter` tag.

```yaml
services:
    # ..
    SWP\Bundle\CoreBundle\Adapter\CustomAdapter:
        public: true
        arguments:
            - '@GuzzleHttp\Client'
        tags:
            - { name: swp.output_channel_adapter, alias: custom_adapter }
```

6. Create a new tenant with output channel which will use the newly created adapter

Now, when you want to create a new tenant, it will be possible to choose your output channel type and define the configuration which will use the newly created `CustomAdapter`.

```
curl -X POST \
  http://example.com/api/v1/tenants/ \
  -H 'Authorization: key' \
  -H 'Cache-Control: no-cache' \
  -H 'Content-Type: application/json' \
  -d '      {
        "tenant": {
          "domainName": "example.com",
          "name": "Custom tenant",
          "subdomain": "custom",
          "outputChannel": {
            "type": "custom",
            "config": {
              "url": "https://api.custom.com",
              "key": "private key",
              "secret": "secret"
            }
          }
        }
      }'
```

### Using Wordpress Adapter

Usage:

```
// example.php
// ..

use SWP\Bundle\CoreBundle\Adapter\WordpressAdapter;
use SWP\Bundle\CoreBundle\Model\Article;
use SWP\Component\OutputChannel\Model\OutputChannel;
// ..

$article = new Article();

$guzzle = new GuzzleHttp\Client();
// ...

$wordpressAdapter = new WordpressAdapter($guzzle);
$outputChannel = new OutputChannel();
$outputChannel->setType('wordpress');
// ...

if ($adapter->supports($outputChannel)) {
    $adapter->send($outputChannel, $article);
    // ...
}
```

### Using Composite Output Channel Adapter

The Composite Output Channel Adapter service loops for each of the registered adapter, checks if adapter supports given output channel and executes appropriate adapter functions.

Usage:

```php
<?php
 // example.php
 // ..

 use SWP\Bundle\CoreBundle\Adapter\CompositeOutputChannelAdapter;
 use SWP\Bundle\CoreBundle\Adapter\WordpressAdapter;
 use SWP\Bundle\CoreBundle\Model\Article;
 use SWP\Component\OutputChannel\Model\OutputChannel;
 // ..

 $article = new Article();
 $guzzle = new GuzzleHttp\Client();
 // ...

 $wordpressAdapter = new WordpressAdapter($guzzle);
 // ...

 $compositeAdapter = new CompositeOutputChannelAdapter();
 $compositeAdapter->addAdapter($wordpressAdapter);
 // ...
 $outputChannel = new OutputChannel();
 $outputChannel->setType('wordpress');
 // ...

 $compisiteAdapter->send($outputChannel, $article);
 // ...
```

### 2.2.9 WebSocket Communication

There is a WebSocket server where the push notifications can be sent to the connected clients. These push notifications are used to refresh the views or in other words, keeps everything synchronized.

In the background, the WebSocket server is using ZeroMQ queue (**'WAMP sub-protocol and PubSub patterns<http://socketo.me/docs/wamp>'_**) and from there it sends everything to clients. There is no communication from client to server, all changes are handled via API. For example, if the new content is pushed to Publisher, it is immediately sent to all the clients, meaning that the new content has been delivered.

#### Authentication

To keep the WebSocket server protected from unauthorized persons there is an authentication mechanism implemented which is based on tokens. Only clients with a valid token can access the WebSocket server. It means if you want to connect to the WebSocket server you have to authenticate via API first. This is the standard token which can be also used to access API. Read more about ../internal_api/authentication. The obtained token must be placed as a query parameter inside the WS url: `ws://127.0.0.1:8080?token=<token>`, where `<token>` is the proper token.

#### How it works?

A client must connect to the WebSocket server and subscribe to the specific topic. In this case it is `package_created`.

If the new content will be sent to the Publisher, we will automatically receive info from the WebSocket server about newly delivered package/content. Based on that info we can refresh or update existing view.

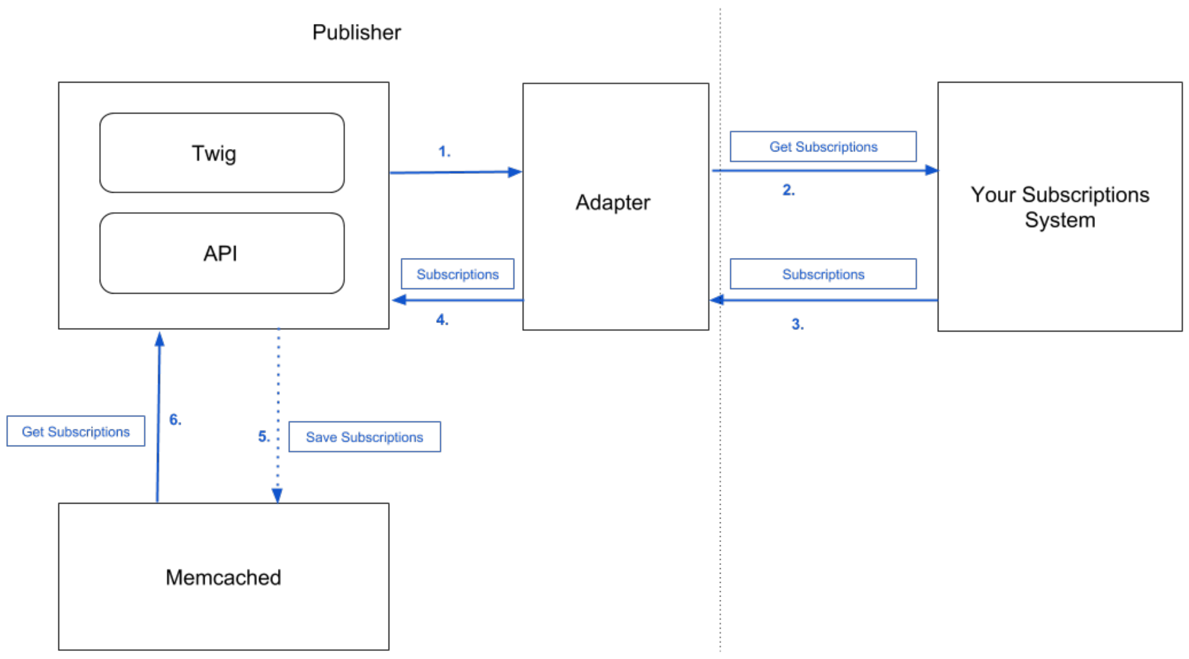
The default WebSocket server port is `8080` and host `127.0.0.1`.

```
<script language="javascript" type="text/javascript" src="https://cdn.rawgit.com/
↪cboden/fcae978cfc016d506639c5241f94e772/raw/
↪e974ce895df527c83b8e010124a034cfcf6c9f4b/autobahn.js"></script>
<script>
    var conn = new ab.Session('ws://127.0.0.1:8080?token=12345',
        function() {
            conn.subscribe('package_created', function(topic, data) {
                // This is where you would add the new article to the DOM (beyond the␣
↪scope of this tutorial)
                console.log('New article published to "' + topic + '" : ' + data.
↪title);
            });
        },
        function() {
            console.warn('WebSocket connection closed');
        },
        {'skipSubprotocolCheck': true}
    );
</script>
```

### 2.2.10 Paywall

Paywall only retrieves the subscriptions data from an external Subscriptions System. In order to buy subscriptions, you have to directly interact with the 3rd party Subscriptions System if you want to create new subscriptions. For example, a simple JS application can be written where users can buy an access to the website.

#### How it works



1. Your app (via API) or web front-end (via Twig) requests the subscriptions data.
2. The Publisher passes the request parameters to the adapter.

3. Adapter calls external Subscriptions System.
4. Once the connection is initialized and the adapter has made a request to Subscriptions System, the Subscriptions System communicates that information and returns a list of subscriptions or a single subscription.
5. Publisher then saves the retrieved subscriptions in the Memcached and caches it for 24 hours, by default. If the request parameters did not change, each request to get subscriptions will be executed to Memcached server.
6. The Publisher gets the subscription(s) from the Memcached.

### How to configure subscriptions cache lifetime

By default, the retrieved subscriptions are cached for 24 hours (86400 seconds). This can be changed by setting `env(SUBSCRIPTIONS_CACHE_LIFETIME)` env var or parameter value to different number of seconds.

### How to render user subscriptions in Twig

To render user subscriptions:

```
{% if app.user %}
    Hey {{ app.user.username }}. <a href="{{ url('security_logout') }}">Logout</a>.
    {% gimmelist subscription from subscriptions with { user: app.user, articleId: 10,
↪ routeId: 20 } %}
        {{ subscription.id }} # subscription's id, this is set in the Paywall Adapter
        {{ subscription.code }} # subscription's code, this is set in the Paywall␣
↪Adapter
        {{ subscription.active }} # is subscription active, this is set in the␣
↪Paywall Adapter
        {{ subscription.type }} # type of the subscription, this is set in the␣
↪Paywall Adapter
        {{ subscription.details.articleId }} # an array with more details about the␣
↪subscription, this is set in the Paywall Adapter
        {{ subscription.updatedAt|date('Y-m-d') }}
        {{ subscription.createdAt|date('Y-m-d') }}
    {% endgimmelist %}
{% endif %}
```

To render a single user subscription by article id:

```
{% if app.user %}
    Hey {{ app.user.username }}. <a href="{{ url('security_logout') }}">Logout</a>.
    {% gimme subscription with { user: app.user, articleId: 10 } %}
        {{ subscription.id }}
        # ...
    {% endgimme %}
{% endif %}
```

To render a single user subscription by article id and name:

```
{% if app.user %}
    Hey {{ app.user.username }}. <a href="{{ url('security_logout') }}">Logout</a>.
    {% gimme subscription with { user: app.user, articleId: 10, name: "premium_content
↪" } %}
        {{ subscription.id }}
        {{ subscription.details.name }}
        # ...
```

(continues on next page)

(continued from previous page)

```
    {% endgimme %}
{% endif %}
```

### How to check if used is "paywall-secured"

Route and the article objects can be marked as "paywall-secured". This can be done via Routes API and Articles API by setting the value of `paywallSecured` property to `true`.

To check if the article or route is "paywall-secured" do:

Articles:

```
{% gimmelist article from articles %}
    {% if article.paywallSecured %}
        # render content of the article
    {% else %}
        # need to buy an access to read this article
    {% endif %}
    # ...
{% endgimmelist %}
```

Routes:

```
{% gimmelist route from routes %}
    {% if route.paywallSecured %}
        # render articles under this route
    {% else %}
        # need to buy an access to read this section
    {% endif %}
    # ...
{% endgimme %}
```

### How to mark articles as paywall-secured using rules

Read more about it in *this section*.

### How to mark articles as paywall-secured by manually publishing packages

You can also directly publish a package and mark articles as "paywall-secured" by making a `POST` request to `/api/v1/packages/<package_id>/publish/` API endpoint with body:

```
{
    "publish":{
      "destinations":[
        {
          "tenant":"123abc",
          "route":6,
          "fbia":false,
          "published":true,
          "paywallSecured":true
        }
      ]
    }
}
```

#### How to override paywall-secured option using publish destination

If there is a rule configured that marks all the articles matching given expression as "paywall-secured", you can use publish destinations to override existing publish workflow for specific packages on specific tenants.

To do this, make a `POST` request to `/api/v1/organization/destinations/` API endpoint with body:

```
{
  "publish_destination":{
    "tenant":"123abc",
    "route":5,
    "fbia":false,
    "published":true,
    "paywallSecured":false,
    "packageGuid": "urn:newsml:sd-master.test.superdesk.org:2022-09-19T09:26:52.
↪402693:f0d01867-e91e-487e-9a50-b638b78fc4bc"
  }
}
```

The following destination will be processed when package will be published. The package will be published to tenant with code `123abc`, route with id `5` and won't be marked as "paywall-secured" even if there is a rule marking it as paywall-secured.

## 2.3 Editors Cookbooks

### 2.3.1 Facebook Instant Articles Integration

Superdesk Publisher have build in integration with Facebook Instant Articles. This cookbook describes all steps needed for proper configuration.

#### Step 1. Register Facebook Page and Application in Publisher

**Note:** As at the moment of creation this documentation there is no UI in Superdesk for this feature, needed actions will be described with CURL direct API calls.

**Note:** Publisher API request require authentication. Read more about this here: *API authentication*

Instant Articles are strongly connected with Facebook Page. To start you need to enable that feature in Your Facebook Page settings. After that call our API to register that page in Publusher.

**Facebook Page** can be registered in Publisher with REST API `/api/{version}/facebook/pages/` POST request.

Required parameters:

- pageId - Unique ID of your Facebook Page
- name - Facebook Page Name

```
curl -X POST 'http://webpublisher.dev/api/v1/facebook/pages/' -H 'Origin: http://
↪webpublisher.dev' -H 'Content-Type: application/x-www-form-urlencoded' -H 'Accept:␣
↪*/*' -H 'Connection: keep-alive' -H 'DNT: 1' -d "facebook_
↪page[pageId]=1234567890987654321&facebook_page[name]=Test Page" --compressed
```

(continues on next page)

(continued from previous page)

Next step is registering Facebook Application (You need to create it first on Facebook Platform). Application is used for retrieving `never expired access token` - it will be used by Publisher in Facebook API calls.

**Facebook Application** can be registered in Publisher with REST API `/api/{version}/facebook/applications/` POST request.

Required parameters:

- appId - Unique ID of your Facebook Application
- appSecret - Generated by Facebook Application secret

```
curl -X POST 'http://webpublisher.dev/api/v1/facebook/applications/' -H 'Origin:␣
↪http://webpublisher.dev' -H 'Content-Type: application/x-www-form-urlencoded' -H
↪'Accept: */*' -H 'Connection: keep-alive' -H 'DNT: 1' -d "facebook_
↪application[appId]=1234567890987654321&facebook_
↪application[appSecret]=superS3cretSecretFromFacebook" --compressed
```

### Step 2. Facebook Page/Application authentication

Assuming that in your database you have Application with id `123456789` and Page with id `987654321` (and both it exists on Facebook platform), You need to call this url `(route: swp_fbia_authorize)`: `/facebook/instantarticles/authorize/123456789/987654321`

In response You will be redirected to Facebook where You will need allow for all required permissions.

After that Facebook will redirect You again to application where (in background - provided by Facebook `code` will be exchanged for access token and that access) you will get JSON response with `pageId` and `accessToken` (never expiring access token).

### Step 3. Create Content List (bucket)

In most of the cases you don't want to push everything to Instant Articles. Publisher allows to define rules for articles selected for Instant Articles publication. This solution is based on Content Lists. Content list allows You to define custom criteria and check them on every published article - if article matches criteria then it's added to that Content List and automatically published to Instant Articles.

Content List can be created in Publisher with REST API `/api/{version}/content/lists/` POST request.

Required parameters:

- name - Content List name
- type - Content List type, in this case it must be "bucket"
- expression - (optional) Expression used for testing published articles eg.: `article.getPriority() > 4`

```
curl -X POST 'http://webpublisher.dev/api/{version}/content/lists/' -H 'Origin: http:/
↪/webpublisher.dev' -H 'Content-Type: application/x-www-form-urlencoded' -H 'Accept:␣
↪*/*' -H 'Connection: keep-alive' -H 'DNT: 1' -d "content_list[name]=Facebook␣
↪Instant Articles&content_list[type]=bucket" --compressed
```

This list don't have `expression` parameter defined so it will catch all published articles.

### Step 3. Create Feed

Feed's are used to connecting Facebook Pages and Content Lists. Thanks to them, you can send selected articles to different Facebook Pages.

Feed can be created in Publisher with REST API `/api/{version}/facebook/instantarticles/feed/` POST request.

Required parameters:

- contentBucket - Content List id
- facebookPage - Facebook Page id (from publisher)
- mode - Instant Article publishing mode: 0 (devlopment) or 1 (production)

```
curl -X POST 'http://webpublisher.dev/api/{version}/content/lists/' -H 'Origin: http:/
↪/webpublisher.dev' -H 'Content-Type: application/x-www-form-urlencoded' -H 'Accept:␣
↪*/*' -H 'Connection: keep-alive' -H 'DNT: 1' -d "facebook_instant_articles_
↪feed[contentBucket]=1&facebook_instant_articles_feed[facebookPage]=1&&facebook_
↪instant_articles_feed[mode]=0" --compressed
```

### Step 4. Create Instant Article template

Instant Article is created from parsed template file. Look and feel of Instant Article can be controlled by templates files in theme. File must be located here: `views\platforms\facebook_instant_article.html.twig`. Publisher autmatically attaches current article meta like in regular page template (remember that there is `gimme.route` set in this case).

Minimal code for Instant Article templates need to look like that:

```
<!doctype html>
<html lang="en" prefix="op: http://media.facebook.com/op#">
    <head>
        <meta charset="utf-8">
        <link rel="canonical" href="{{ url(gimme.article) }}">
    </head>
    <body>
        <header>
            <h1>{{ gimme.article.title }}</h1>
            <time class="op-published" datetime="{{ gimme.article.publishedAt|date("Y-
↪m-d\\Th:i:s\\Z", false) }}">{{ gimme.article.publishedAt|date("Y-m-d h:i ") }}</
↪time>

        </header>
        <p>{{ gimme.article.lead }}</p>
        {{ gimme.article.body|raw }}
    </body>
</html>
```

HTML code of article body (`gimme.article.body`) will be parsed by `Transformer`. Transformer will try to match html elements to Instant Articles tags (for example images). If he will not recognize some elements then they will be removed. You can preview how You template works with currently published articles here: `/facebook/instantarticles/preview/{articleId}`.

**Note:** Preview url is available only in Publisher Development mode. To send that article to FBIA library from preview add *?listId={content bucket Id}* to url.

#### Step 5. Publish Your articles to Facebook Instant Articles

After all previous steps - publishing should happen automatically just after publishing article matching Content List criteria.

### 2.3.2 Adding new menus

Adding new menu and its items is very easy. Once the menu is created the menu widget will be automatically created so you can use it instantly without a need to create it manually.

Newly created menu widget will, by default, use `menu.html.twig` template and will point to the menu you created.

Here is an example to demonstrate this:

Once the new menu with name `Sports` is created, menu widget will be automatically created with a name: `Sports` and template: `menu.html.twig`. New menu widget, by default, will point to newly created menu (with name `Sports`). When this menu widget will be used in template, the menu assigned to that widget will be automatically rendered i.e. the `Sports` menu.

**Note:** Menu widget will be created automatically for you only for the main menus (root menus), not for the menu items.

## 2.4 Implementators Cookbooks

### 2.4.1 Secure content push to Publisher

Content can be pushed to Publisher via HTTP requests from any system. But it's important to store and publish only requests from approved by us sources.

To verify incoming requests we use special header: `x-superdesk-signature`. Value of this header have format like that: `sha1={token}`.

`token` is a result of HMAC (keyed-Hash Message Authentication Code) function. It's created from request content and `secret token` value with sha1 algorithm applied on it.

`secret token` can be defined in Organization (when created or updated). Example command:

```
php app/console swp:organization:update OrganizationName --secretToken secret_token
```

Organization secret token is not visible in any API.

If token is set in organization then Publisher will reject all requests without `x-superdesk-signature` header or with wrong value in it.

### 2.4.2 [Migration] Redirect articles from previous url's to publisher

Almost always after content migration you need to make sure that all previous links to articles works ok with new system. In many publishing systems articles are identified by article/post numbers or some special codes. In Publisher we use unique combination of route and article slug's. Because of that it's impossible to redirect requests only with server redirects.

Publisher provide solution for this case. In short: based on package external data (imported from external system) we localize articles and return redirect responses with link to new article location.

### External data

With special API endpoint (secured with secret code) You can associate pair's of keys and values with imported article (package).

Example of setting external data (from our behat feature):

```
When I add "Content-Type" header equal to "application/json"
And I add "x-publisher-signature" header equal to
↪"sha1=0dcd1953d72dda47f4a4acedfd638a3c58def7bc"
And I send a "PUT" request to "/api/v1/packages/extra/test-news-article" with body:
"""
{
  "articleNumber": "123456",
  "some other key": "some other value"
}
"""

Then the response status code should be 200
```

### Redirect url

In our case server is responsible for composing redirect (supported with regular expressions) url containing article identifier from original url and pushed as external data to package.

For example for article with url like that: `/en/sport/123456/mundial-winner` we need to push indetifier (`123456`) as a external data to `/api/v1/packages/extra/mundial-winner`.

After that server can rediret our url to this one: `/redirecting/extra/articleNumber/123456`. Publisher in response will return redirect response (with code 301) to new article location.

## 2.4.3 Setup Wordpress as a Publisher Output Channel

Output channel allows you to use tenant as a bridge for publishing content in external systems. Thanks to it content pushed to Publisher can be automatically published also in Wordpress or other even internal systems.

We assume that Publisher is installed and running. So now we need Wordpress instance.

1. Download: `wget https://wordpress.org/latest.zip`
2. Unpack: `unzip latest.zip`
3. Download router.php file: `cd wordpress && wget https://gist.githubusercontent.com/ginfuru/1dfd9a054f27d268e9e3f445896150f5/raw/9f5a4c71e9bd6592e113914e64f7c36c31c5a1ad/router.php`
4. Run Wordpress with built in php server: `php -S wordpress.test:8080 router.php`
5. Install Wordpress
6. Install plugin "Application Passwords" (author: George Stephanis). Dont forget to activate it ;)

7. Go to `http://wordpress.test:8080/wp-admin/profile.php` adn create new Application Password - it's on bottom of page. 7. Encode your password with `echo -n "admin:8e7M k22B znze mLVF 3vmc i4Vc" | base64` (run this in terminal) 9. Copy generated password (we will use it later). 10. Create new tenant for wordpress:

```json
{
  "tenant": {
    "domainName": "yourpublisherdomain.com",
    "name": "Wordpress Output Channel",
    "subdomain": "validvhostsubdomain",
    "outputChannel": {
      "type": "wordpress",
      "config": {
        "url": "http://wordpress.test:8080",
        "authorization_key": "Basic YWRtaW46OGU3TSBrMjJCIHpuemUgbUxWRiAzdm1jIGk0VmM="
      }
    }
  }
}
```

In `authorization_key` we type "Basic " and generated before password.

From now all content published to created tenant will be published, unpublished and updated in Wordpress instance.

CHAPTER 3

# Reference

In this chapter you can explore complete reference of things that exist in Publisher, list of functionality, how to use certain functions etc. This reference is meant to be quick reminder and sum up of everythnig that is already described in other chapters.

## 3.1 System Requirements

To run Publisher, meet these requirements:

- PHP >= 7.0
  - iconv needs to be enabled
  - Intl needs to be installed with ICU 4+
  - pdo needs to be enabled
  - JSON needs to be enabled
  - ctype needs to be enabled
  - Your php.ini needs to have the date.timezone setting
  - PHP tokenizer needs to be enabled
  - mbstring functions need to be enabled
  - POSIX needs to be enabled (only on *nix)
  - CURL and php-curl need to be enabled
  - php.ini recommended settings
    - short_open_tag = Off
    - magic_quotes_gpc = Off
    - register_globals = Off

        * session.auto_start = Off

- Postgresql >= 9.4
  - pdo-pgsql
- Memcached
  - memcached (running)
  - php-memcached

## 3.2 Twig Extensions

Publisher provides its own Twig functions and filters in addition to the standard set of Twig functions which you can use in website templates.

### 3.2.1 Functions

- container
- gimme
- gimmelist
- redirects
- Stringy twig extensions
- knp-time-bundle

# CHAPTER 4

# Bundles

## 4.1 Fixtures Bundle

### 4.1.1 Overview

The Fixtures Bundle helps developers to create fixtures or fake data that can be used for development and/or testing purposes.

It relies on the following 3rd party libraries:

DoctrineFixturesBundle (gives possibility to load data fixtures programmatically into the Doctrine ORM or ODM)

fzaninotto/Faker (generates fake data for you)

nelmio/alice (gives you a few essential tools to make it very easy to generate complex data with constraints in a readable and easy to edit way)

It also provides the possibility to set up a ready-to-use demo theme, for development.

### 4.1.2 How to use fixtures

The following chapter describes how to make use of the Fixtures Bundle features.

### 4.1.3 Creating a simple PHP fixture class

Fixtures should be created inside `SWP\FixturesBundle\DataFixtures\<db_driver>` directory and by convention, should be called like: `LoadPagesData`, `LoadArticlesData`, `LoadUsersData` etc.

Replace `db_driver` either with `ORM` for the Doctrine ORM or `PHPCR` for the Doctrine PHPCR ODM.

Example Fixture class:

```php
<?php
namespace SWP\FixturesBundle\DataFixtures\ORM;

use Doctrine\Common\DataFixtures\FixtureInterface;
use Doctrine\Common\Persistence\ObjectManager;
use SWP\Bundle\FixturesBundle\AbstractFixture;

class LoadPagesData extends AbstractFixture implements FixtureInterface
{
    /**
     * {@inheritdoc}
     */
    public function load(ObjectManager $manager)
    {
        $env = $this->getEnvironment();
        $this->loadFixtures(
            '@SWPFixturesBundle/Resources/fixtures/ORM/'.$env.'/page.yml',
            $manager
        );
    }
}
```

Each fixture class extends AbstractFixture class and implements FixtureInterface. This way we can use the `load` method, inside which we can create some objects using PHP and/or make use of nelmio/alice and fzaninotto/Faker by loading fixtures in YAML format, as shown in the example above.

## 4.1.4 Creating a simple Alice fixture (YAML format)

For more details on how to create Alice fixtures, please see the Alice documentation as a reference.

Example Alice fixture:

```yaml
SWP\CoreBundle\Entity\Page:
    page1:
        name: "About Us"
        type: 1
        slug: "about-us"
        templateName: "static.html.twig"
        contentPath: "/swp/content/about-us"
    page2:
        name: "Features"
        type: 1
        slug: "features"
        templateName: "features.html.twig"
        contentPath: "/swp/content/features"
    page3:
        name: "Get Involved" # we can also use faker library formatters like:
↪<paragraph(20)> etc
        type: 1
        slug: "get-involved"
        templateName: "involved.html.twig"
        contentPath: "/swp/content/get-involved"
```

The above configuration states that we want to persist into the database three objects of type `SWP\CoreBundle\Entity\Page`. We can use the faker formatters where, for example, `<paragraph(20)>` is one of the fzaninotto/Faker formatters, which tells Alice to generate 20 paragraphs filled with fake data.

By convention, Alice YAML files should be placed inside `Resources/fixtures/<db_driver>/<environment>`, where <environment> is the current environment name (dev, test).

For instance, having the `Resources/fixtures/ORM/test/page.yml` Alice fixture, we will be able to persist fake data defined in the YAML file into the database (using Doctrine ORM driver), only when the `test` environment is set or defined differently in `SWP\FixturesBundle\DataFixtures\ORM\LoadPagesData.php`.

There is a lot of flexibility on how to define fixtures, so it's up to the developer how to create them.

### 4.1.5 Loading all fixtures

**Note:** Remember to update your database schema before loading fixtures! To do this, run in a console:

```
php app/console doctrine:schema:update --force
```

Once you have your fixtures defined, we can simply load them. To do that you must execute console commands.

To load Doctrine ORM fixtures:

```
php app/console doctrine:fixtures:load --append
```

See `php app/console doctrine:fixtures:load --help` for more details.

After executing the commands above, your database will be filled with the fake data, which can be used by themes.

### 4.1.6 Setting up a demo theme

To make it easier to start with the Web Publisher, we created a simple demo theme. To set this theme as an active one, you need to execute the following command in a console:

```
php app/console swp:theme:install 123abc src/SWP/Bundle/FixturesBundle/Resources/
↪themes/DefaultTheme/ -f -p
```

This command will install default theme for the default tenant which was already created by loading fixtures (see above).

See `php app/console swp:theme:install --help` for more details.

## 4.2 MultiTenancyBundle

This bundle provides the tools to build multi-tenant architecture for your PHP applications.

*The MultiTenancy Component*, which is used by this bundle, provides a generic interfaces to create different implementations of multi-tenancy in PHP applications.

The idea of this bundle is to have the ability to create multiple websites (tenants) within many organizations.

So far, we aim to support two persistence backends:

- Doctrine ORM
- Doctrine PHPCR ODM

**Note:** This documentation describes installation and configuration for Doctrine PHPCR ODM at the moment.

**Features:**

- Allows to create many organizations
- Allows to create many websites (tenants) within a single organization
- Organization can have multiple websites assigned
- Each website have parent organization
- Allows to create default organization and default website (tenant)

E.g. The Vox Media organization can have multiple websites: The Verge. Polygon, Eater etc. Each website has it's own content.

## 4.2.1 Prerequisites

This version of the bundle requires Symfony >= 2.6 and PHP version >=7.0.

## 4.2.2 Installation

### Install the Bundle with Composer

In your project directory, execute the following command to download the latest stable version of the MultiTenancyBundle:

```
composer require swp/multi-tenancy-bundle
```

This command requires you to have Composer installed globally. If it's not installed globally, download the `.phar` file locally as explained in Composer documentation.

### Enable the bundle and its dependencies

**Note:** By default Jackalope Doctrine DBAL is required for PHPCR ODM in this bundle. See Choosing a PHPCR Implementation for alternatives.

Enable the bundle and its dependencies (DoctrinePHPCRBundle, DoctrineBundle) by adding the following lines in the `app/AppKernel.php` file:

```php
// app/AppKernel.php

// ...
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            // ...

            new Doctrine\Bundle\PHPCRBundle\DoctrinePHPCRBundle(),
            new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
            new SWP\MultiTenancyBundle\SWPMultiTenancyBundle(),
        );

        // ...
```

(continues on next page)

(continued from previous page)

```
    }

    // ...
}
```

**Note:** All dependencies will be installed automatically. You will just need to configure the respective bundles.

### Configure the SWPMultiTenancyBundle

Let's enable PHPCR persistence backend.

- *YAML*

```yaml
# app/config/config.yml
swp_multi_tenancy:
    persistence:
        phpcr:
            # if true, PHPCR is enabled in the service container
            enabled: true
```

- *XML*

```xml
<!-- app/config/config.xml -->
<?xml version="1.0" encoding="UTF-8" ?>
<swp_multi_tenancy>
    <persistence>
        <phpcr>
            <!-- if true, PHPCR is enabled in the service container -->
            <enabled>true</enabled>
        </phpcr>
    </persistence>
</swp_multi_tenancy>
```

**Tip:** See *CMF RoutingBundle Integration*.

**Note:** See *Configuration Reference* for more details.

### DoctrinePHPCRBundle Configuration

See how to set PHPCR Session Configuration.

### Add the domain parameter

Add the following parameter to your parameters file, so the current tenant can be resolved and matched against the configured domain.

```yaml
# app/config/parameters.yml
domain: example.com
```

### Update your database schema

**Note:** This step assumes you have already configured and created the database.

Execute the following commands in the console:

```
php app/console doctrine:schema:update --force
php app/console doctrine:phpcr:repository:init
php app/console swp:organization:create --default
php app/console swp:tenant:create --default
php app/console doctrine:phpcr:repository:init
```

That's it, the bundle is configured properly now!

## 4.2.3 Services

### TenantContext

The **TenantContext** service allows you to manage the currently used tenant. Its `getTenant` method gets the current tenant by resolving its subdomain from the request.

For example, if the host name is: `subdomain.example.com` the TenantContext will first resolve the subdomain from the host provided in the parameters file domain, and then it will try to find the object of instance *TenantContextInterface* in the storage. When found, it will return the tenant object.

You can also set the current tenant in the context, so whenever you request the current tenant from the context it will return you the object you set. The `setTenant` method is used to set the tenant. It accepts as the first parameter an object of type *TenantContextInterface*.

### TenantAwareRouter

**Note:** This service requires the CMF Routing Bundle to be installed and configured.

The TenantAwareRouter generates tenant-aware routes. It extends DynamicRouter from the CMF Routing Bundle.

In some cases you may need to generate a statically configured route. Let's say we have a path defined in PHPCR: `/swp/<organization_code>/<tenant_code>/routes/articles/features`. If you want to generate a route for the current tenant in a Twig template, you could use the following code:

```
<a href="{{ path('/routes/articles/features') }}">Features</a>
```

The TenantAwareRouter will resolve the current tenant from the host name and will internally create a route `/swp/<organization_code>/<tenant_code>/routes/articles/features` where `swp` is the root path defined in the bundle configuration, `<tenant_code>` is the current tenant's unique code, and `routes` is the configured `route_basepaths`.

The result will be:

```
<a href="/articles/features">Features</a>
```

You can also generate the route by content path:

```
<a href="{{ path(null, {'content_id': '/content/articles/features'}) }}">Features</a>
```

If the content is stored under the path `/swp/<organization_code>/<tenant_code>/content/articles/features` in the PHPCR tree, the router will search for the route for that content and will return the route associated with it. In this case, the associated route is `/swp/<organization_code>/<tenant_code>/routes/articles/features` so it will generate the same route: `/articles/features` as in the example above.

**Note:** We do not recommend hard-coding the route name in the template because if the route is removed, the page will break.

See *CMF RoutingBundle Integration* on how to enable and make use of this router.

#### PrefixCandidates

**Note:** This service requires the CMF Routing Bundle to be installed and configured.

This service extends Symfony CMF RoutingBundle PrefixCandidates service, to set tenant-aware prefixes. Prefixes are used to generate tenant-aware routes. Prefixes are built from the configured root path, which by default is `/swp` and from `route_basepaths` which you can set in the configuration file.

See the *Full Default Configuration* reference for more details.

### 4.2.4 PHPCR ODM Repository Initializer

#### PHPCRBasePathsInitializer

**Note:** This service requires DoctrinePHPCRBundle to be installed and configured.

The Initializer is the PHPCR equivalent of the ORM schema tools. PHPCRBasePathsInitializer creates base paths in the content repository based on tenants and organizations, configures and registers PHPCR node types. It is disabled by default, but can be enabled in the configuration when using PHPCR ODM persistence backend.

You can execute this initializer, together with the generic one, by running the following command:

```
php app/console doctrine:phpcr:repository:init
```

Running this command will trigger the generic initializer which is provided by the DoctrinePHPCRBundle. The generic initializer will be fired before this one, and will create the root base path in the content repository.

See *CMF RoutingBundle Integration* on how to enable this initializer.

### 4.2.5 Repositories

#### TenantRepository

This repository allows you to fetch a single tenant by its subdomain name and all available tenants from the Doctrine ORM storage. It extends EntityRepository from Doctrine.

This service implements *TenantRepositoryInterface* and it has three methods:

- findBySubdomain($subdomain) - Finds the tenant by subdomain. `$subdomain` is the subdomain of string type.
- findByCode($code) - Finds the tenant by code. `$code` is the unique code of string type.
- findAvailableTenants() - Finds all available tenants. Returns an array of all tenants.

## 4.2.6 SQL Query Filters

### TenantableFilter

This filter adds the where clause to the select queries, to make sure the query will be executed for the current tenant. If the tenant exists in the context and the tenant id is 1, it will add `WHERE tenant_id = 1` to every select query. This way, we always make sure we get the data for the current tenant.

In order to make use of the filter every class needs to implement *TenantAwareInterface* which indicates that it should be associated with the specific tenant.

It extends `Doctrine\ORM\Query\Filter\SQLFilter`.

When PHPCR ODM persistence backend is enabled it will rely on tenant's unique code instead of the tenant id. In this case, if the tenant exists in the context and the tenant code is 123abc, it will add `WHERE tenant_id = 123abc` to every select query.

## 4.2.7 Event Listeners

### TenantableListener

This event listener runs on every kernel request (`kernel.request` event). If the tenant is set in the TenantContext it enables Doctrine ORM Query *TenantableFilter*, otherwise it doesn't do anything. Its responsibility is to ensure that every SQL select query will be tenant-aware (`tenant_id` will be added in the query).

### TenantSubscriber

This subscriber subscribes to every Doctrine ORM `prePersist` event, when persisting the data. It makes sure that the persisted object (which needs to implement *TenantAwareInterface*) will be associated with the current tenant when saving the object.

## 4.2.8 Console Commands

This section describes console commands available in this bundle.

### Create a new organization

To make use of this bundle, you need to first create the default organization. You may also need to create some other, custom organization if needed.

**Note:** This command persists organizations in database depending on your enabled persistence backend. If the PHPCR backend is enabled it will store tenants in PHPCR tree.

To create the default organization, execute the following console command:

```
php app/console swp:organization:create --default
```

To create a custom organization which will be disabled by default, use the command:

```
php app/console swp:organization:create --disabled
```

To create a custom organization, execute the following console command:

```
php app/console swp:organization:create
```

Run `php app/console swp:organization:create --help` to see more details of how to use this command.

### List available organizations

This command list all available organizations.

Usage:

```
php app/console swp:organization:list
```

Run `php app/console swp:organization:list --help` to see more details of how to use this command.

### Create a new tenant

To make use of this bundle, you need to first create the default tenant. You may also need to create some other, custom tenants.

**Note:** This command persists tenants in database depending on your enabled persistence backend. If the PHPCR backend is enabled it will store tenants in PHPCR tree.

To create the default tenant, execute the following console command:

```
php app/console swp:tenant:create --default
```

**Note:** When creating default tenant the command requires you to have the default organization created.

To create a custom tenant which will be disabled by default, use the command:

```
php app/console swp:tenant:create --disabled
```

To create a custom tenant, execute the following console command:

```
php app/console swp:tenant:create
```

You will need to specify organization unique code so tenant can be assigned to the organization.

Run `php app/console swp:tenant:create --help` to see more details of how to use this command.

### 4.2.9 Twig Extension

This bundle provides Twig global variables in your project. See below for more details.

#### Global variables

- `tenant` - provides data about the current website/tenant. This variable is an object of type *TenantInterface*

Usage:

```
{{ tenant.name }}
{{ tenant.subdomain }}
{{ tenant.organization.name }} # get tenant's organization name
{# ... #}
```

### 4.2.10 Configuration Reference

The SWPMultiTenancyBundle can be configured under the `swp_multi_tenancy` key in your configuration file. This section describes the whole bundle's configuration.

#### Full Default Configuration

```
# app/config/config.yml
swp_multi_tenancy:
    use_orm_listeners: false
    persistence:
        phpcr:
            enabled: true
            basepath: "/swp"
            route_basepaths: ["routes"]
            content_basepath: "content"
            menu_baseapth: "menu"
            media_baseapth: "media"
            tenant_aware_router_class: ␣
↪SWP\MultiTenancyBundle\Routing\TenantAwareRouter
            classes:
                tenant:
                    model: SWP\Component\MultiTenancy\Model\Tenant
                    repository: ␣
↪SWP\Bundle\MultiTenancyBundle\Doctrine\PHPCR\TenantRepository
                    factory: SWP\Component\MultiTenancy\Factory\TenantFactory
                    object_manager_name: ~
                organization:
                    model: SWP\Component\MultiTenancy\Model\Organization
                    repository: ␣
↪SWP\Bundle\MultiTenancyBundle\Doctrine\PHPCR\OrganizationRepository
                    factory: SWP\Bundle\MultiTenancyBundle\Factory\OrganizationFactory
                    object_manager_name: ~
        orm:
            enabled: true
            classes:
                tenant:
                    model: SWP\Component\MultiTenancy\Model\Tenant
                    repository: ␣
↪SWP\Bundle\MultiTenancyBundle\Doctrine\ORM\TenantRepository
```

(continues on next page)

(continued from previous page)

```
                        factory: SWP\Component\MultiTenancy\Factory\TenantFactory
                        object_manager_name: ~
                    organization:
                        model: SWP\Component\MultiTenancy\Model\Organization
                        repository:␣
↪SWP\Bundle\MultiTenancyBundle\Doctrine\ORM\OrganizationRepository
                        factory: SWP\Bundle\MultiTenancyBundle\Factory\OrganizationFactory
                        object_manager_name: ~
```

## persistence

### use_orm_listeners

**type**: `Boolean` **default**: `false`

Use this setting to activate the `TenantableListener` and `TenantableSubscriber`. This will enable tenantable SQL extension and will make sure your Doctrine ORM entities are tenant aware. See *Event Listeners* for more details.

## `persistence`

### `phpcr`

```
# app/config/config.yml
swp_multi_tenancy:
    # ..
    persistence:
        phpcr:
            enabled: true
            basepath: "/swp"
            route_basepaths: ["routes"]
            content_basepath: "content"
            menu_baseapth: "menu"
            media_baseapth: "media"
            tenant_aware_router_class:␣
↪SWP\MultiTenancyBundle\Routing\TenantAwareRouter
```

#### `enabled`

**type**: `boolean` **default**: `false`

If `true`, PHPCR is enabled in the service container.

PHPCR can be enabled by multiple ways such as:

```
phpcr: ~ # use default configuration
# or
phpcr: true # straight way
# or
phpcr:
    route_basepaths: ... # or any other option under 'phpcr'
```

### `basepath`

**type**: `string` **default**: `/swp`

The basepath for documents in the PHPCR tree.

### `route_basepaths`

**type**: `array` **default**: `['routes']`

A set of paths where routes should be located in the PHPCR tree.

### `content_basepath`

**type**: `string` **default**: `content`

The basepath for content objects in the PHPCR tree. This information is used to offer the correct subtree to select content documents.

### `media_basepath`

**type**: `string` **default**: `media`

The basepath for media objects in the PHPCR tree. This information is used to offer the correct subtree to select media documents.

### `menu_basepath`

**type**: `string` **default**: `media`

The basepath for menu objects in the PHPCR tree. This information is used to offer the correct subtree to select menu documents.

### `tenant_aware_router_class`

**type**: `string` **default**: `SWP\MultiTenancyBundle\Routing\TenantAwareRouter`

The TenantAwareRouter service's fully qualified class name to use.

### `classes`

```yaml
# app/config/config.yml
swp_multi_tenancy:
    # ..
    persistence:
        phpcr:
            # ..
            classes:
                tenant:
                    model: SWP\Component\MultiTenancy\Model\Tenant
```

(continues on next page)

(continued from previous page)

```
                    repository:␣
↪SWP\Bundle\MultiTenancyBundle\Doctrine\PHPCR\TenantRepository
                    factory: SWP\Component\MultiTenancy\Factory\TenantFactory
                    object_manager_name: ~
                organization:
                    model: SWP\Component\MultiTenancy\Model\Organization
                    repository:␣
↪SWP\Bundle\MultiTenancyBundle\Doctrine\PHPCR\OrganizationRepository
                    factory: SWP\Bundle\MultiTenancyBundle\Factory\OrganizationFactory
                    object_manager_name: ~
```

#### `tenant.model`

**type**: `string` **default**: `SWP\Component\MultiTenancy\Model\Tenant`

The FQCN of the Tenant model class which is of type *TenantInterface*.

#### `tenant.factory`

**type**: `string` **default**: `SWP\Component\MultiTenancy\Factory\TenantFactory`

The FQCN of the Tenant Factory class.

#### `tenant.repository`

**type**: `string` **default**: `SWP\Bundle\MultiTenancyBundle\Doctrine\PHPCR\TenantRepository`

The FQCN of the Tenant Repository class.

#### `tenant.object_manager_name`

**type**: `string` **default**: `null`

The name of the object manager. If set to null it defaults to *default*. If PHPCR ODM persistence backend is enabled it will register `swp.object_manager.tenant` service which is an alias for "doctrine_phpcr.odm.default_document_manager".

#### `organization.model`

**type**: `string` **default**: `SWP\Component\MultiTenancy\Model\Organization`

The FQCN of the Organization model class which is of type *OrganizationInterface*.

#### `organization.factory`

**type**: `string` **default**: `SWP\Bundle\MultiTenancyBundle\Factory\OrganizationFactory`

The FQCN of the Organization Factory class.

### `organization.repository`

**type**: `string` **default**: `SWP\Bundle\MultiTenancyBundle\Doctrine\PHPCR\OrganizationRepository`

The FQCN of the Organization Repository class.

### `organization.object_manager_name`

**type**: `string` **default**: `null`

The name of the object manager. If set to null it defaults to *default*. If PHPCR ODM persistence backend is enabled it will register `swp.object_manager.organization` service which is an alias for `doctrine_phpcr.odm.default_document_manager`.

### `orm`

```yaml
# app/config/config.yml
swp_multi_tenancy:
    # ..
    persistence:
        orm:
            enabled: true
```

### `enabled`

**type**: `boolean` **default**: `false`

If `true`, ORM is enabled in the service container.

ORM can be enabled by multiple ways such as:

```yaml
orm: ~ # use default configuration
# or
orm: true # straight way
# or
orm:
    enabled: true ... # or any other option under 'orm'
```

### `classes`

```yaml
# app/config/config.yml
swp_multi_tenancy:
    # ..
    persistence:
        orm:
            # ..
            classes:
                tenant:
                    model: SWP\Component\MultiTenancy\Model\Tenant
                    repository:␣
↪SWP\Bundle\MultiTenancyBundle\Doctrine\ORM\TenantRepository
```

(continues on next page)

(continued from previous page)

```
                    factory: SWP\Component\MultiTenancy\Factory\TenantFactory
                    object_manager_name: ~
                organization:
                    model: SWP\Component\MultiTenancy\Model\Organization
                    repository:␣
↪SWP\Bundle\MultiTenancyBundle\Doctrine\ORM\OrganizationRepository
                    factory: SWP\Bundle\MultiTenancyBundle\Factory\OrganizationFactory
                    object_manager_name: ~
```

### `tenant.model`

**type**: `string` **default**: `SWP\Component\MultiTenancy\Model\Tenant`

The FQCN of the Tenant model class which is of type *TenantInterface*.

### `tenant.factory`

**type**: `string` **default**: `SWP\Component\MultiTenancy\Factory\TenantFactory`

The FQCN of the Tenant Factory class.

### `tenant.repository`

**type**: `string` **default**: `SWP\Bundle\MultiTenancyBundle\Doctrine\ORM\TenantRepository`

The FQCN of the Tenant Repository class.

### `tenant.object_manager_name`

**type**: `string` **default**: `null`

The name of the object manager. If set to null it defaults to *default*. If Doctrine ORM persistence backend is enabled it will register `swp.object_manager.tenant` service which is an alias for `doctrine.orm.default_entity_manager`.

### `organization.model`

**type**: `string` **default**: `SWP\Component\MultiTenancy\Model\Organization`

The FQCN of the Organization model class which is of type *OrganizationInterface*.

### `organization.factory`

**type**: `string` **default**: `SWP\Bundle\MultiTenancyBundle\Factory\OrganizationFactory`

The FQCN of the Organization Factory class.

#### `organization.repository`

**type**: `string` **default**: `SWP\Bundle\MultiTenancyBundle\Doctrine\ORM\OrganizationRepository`

The FQCN of the Organization Repository class.

#### `organization.object_manager_name`

**type**: `string` **default**: `null`

The name of the object manager. If set to null it defaults to *default*. If Doctrine ORM persistence backend is enabled it will register `swp.object_manager.organization` service which is an alias for `doctrine.orm.default_entity_manager`.

## 4.2.11 Tutorials

### CMF RoutingBundle Integration

The SWPMultiTenancyBundle can be integrated with the CMF RoutingBundle. This section describes how to integrate the CMF Routing Bundle when using PHPCR ODM or ORM as persistence backends.

**Note:** If you don't have CMF RoutingBundle installed, see the documentation on how to install and configure it.

#### Doctrine PHPCR ODM integration

#### Enable PHPCR persistence backend

Make sure the PHPCR persistence backend is enabled in CMF RoutingBundle.

You need to enable the PHPCR as a persistence backend for the SWPMultiTenancyBundle and fully integrate this bundle with the CMF RoutingBundle. Add the following lines to the configuration file:

```yaml
# app/config/config.yml
swp_multi_tenancy:
    persistence:
        phpcr:
            # if true, PHPCR is enabled in the service container
            enabled: true
            # route base paths under which routes will be stored
            route_basepaths: ["routes"]
            # PHPCR content base path under which content will be stored
            content_basepath: "content"
```

Once the `enabled` property is set to true, *PHPCRBasePathsInitializer*, *TenantAwareRouter* and *PrefixCandidates* will be available in the application.

#### Enable TenantAwareRouter

To register the TenantAwareRouter service in the CMF RoutingBundle, add the following lines to your configuration file:

```yaml
cmf_routing:
    chain:
        routers_by_id:
            # other routers
            # TenantAwareRouter with the priority of 150
            swp_multi_tenancy.tenant_aware_router: 150
```

The RoutingBundle example configuration can be found here:

```yaml
cmf_routing:
    chain:
        routers_by_id:
            # default Symfony Router
            router.default: 200
            # TenantAwareRouter
            swp_multi_tenancy.tenant_aware_router: 150
            # CMF Dynamic Router
            cmf_routing.dynamic_router: 100
    dynamic:
        route_collection_limit: 100
        persistence:
            phpcr:
                enabled: true
```

**Note:** Please see the documentation of the CMF RoutingBundle for more details.

### Doctrine ORM integration

Not implemented yet.

### Creating a custom Tenant class

**In this tutorial you will learn how to create custom Tenant class. For example, you want** to store info about tenant's theme name and you want to make use of it in your project.

**Note:** This tutorial covers creating a custom Tenant class for PHPCR ODM.

This new class must implement *TenantInterface* which is provided by *The MultiTenancy Component*, or you can extend the default *Tenant* class, which is also part of the MultiTenancy Component.

Create an interface first which will require to implement theme name behaviour.

```php
<?php

namespasce Acme\AppBundle\Document;

use SWP\Component\MultiTenancy\Model\TenantInterface as BaseTenantInterface;

interface ThemeAwareTenantInterface extends BaseTenantInterface
{
    /**
```

(continues on next page)

(continued from previous page)

```php
     * @return string
     */
    public function getThemeName();

    /**
     * @param string $themeName
     */
    public function setThemeName($themeName);
}
```

Let's create a new Tenant class now:

```php
<?php

namespaspce Acme\AppBundle\Document;

use Acme\AppBundle\Document\ThemeAwareTenantInterface;
use SWP\Component\MultiTenancy\Model\Tenant as BaseTenant;

class Tenant extends BaseTenant implements ThemeAwareTenantInterface
{
    /**
     * @var string
     */
    protected $themeName;

    /**
     * {@inheritdoc}
     */
    public function getThemeName()
    {
        return $this->themeName;
    }

    /**
     * {@inheritdoc}
     */
    public function setThemeName($themeName)
    {
        $this->themeName = $themeName;
    }
}
```

Create a mapping file for your newly created document:

```yaml
# src/Acme/AppBundle/Resources/config/doctrine/Document.Tenant.phpcr.yml

Acme\AppBundle\Document\Tenant:
referenceable: true
fields:
    themeName:
        type: string
        nullable: true
```

Once your class is created, you can now put its FQCN into the MultiTenancy bundle's configuration:

```yaml
# app/config/config.yml
swp_multi_tenancy:
    persistence:
        phpcr:
            enabled: true
            # ..
            classes:
                tenant:
                    model: Acme\AppBundle\Document\Tenant
```

From now on your custom class will be used and you will be able to make use of the `$themeName` property in your app.

**Tip:** See *Configuration Reference* for more configuration details.

That's it, you can now refer to `Acme\AppBundle\Document\Tenant` to manage tenants in the PHPCR tree.

## 4.3 StorageBundle

This bundle provides tools to build a persistence-agnostic storage layer.

*The Storage Component*, which is used by this bundle, provides generic interfaces to create different types of repositories, factories, and drivers which can be used for various storages. So far, this bundle supports:

- Doctrine ORM
- Doctrine PHPCR ODM

By default this bundle uses the Doctrine ORM persistence backend. If you would like to make use of, for example, Doctrine PHPCR, you would need to install and configure DoctrinePHPCRBundle.

### 4.3.1 Prerequisites

This version of the bundle requires Symfony >= 2.8 and PHP version >=5.6.

### 4.3.2 Installation

#### Install the Bundle with Composer

In your project directory execute the following command to download the latest stable version:

```
composer require swp/storage-bundle
```

This command requires you to have Composer installed globally. If it's not installed globally, download the `.phar` file locally as explained in Composer documentation.

#### Enable the bundle and its dependencies

Enable the bundle and its dependency (DoctrineBundle) by adding the following lines in the `app/AppKernel.php` file:

```php
// app/AppKernel.php

// ...
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            // ...

            new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
            new SWP\Bundle\StorageBundle\SWPStorageBundle(),
        );

        // ...
    }

    // ...
}
```

**Note:** All dependencies will be installed automatically. You will just need to configure the respective bundles if needed.

That's it, the bundle is configured properly now!

## 4.3.3 Usage

### How to Provide Model Classes for several Doctrine Implementations

When building a bundle that could be used not only with Doctrine ORM but also the CouchDB ODM, MongoDB ODM or PHPCR ODM, you should still only write one model class. The Doctrine bundles provide a compiler pass to register the mappings for your model classes. This bundle helps you easily register these mappings.

**Note:** See Symfony documentation for more details.

Let's say you created a new bundle called **ContentBundle** and you want to register your model classes mappings. You could do that in a normal way by defining it in the configuration file as follows:

```yaml
# app/config/config.yml

doctrine:
    # ..
    orm:
        entity_managers:
            default:
                # ..
                auto_mapping: false
                mappings:
                    AcmeContentBundle:
                        type: yml
                        prefix: Acme\ContentBundle\Model
                        dir: Resources/config/doctrine
```

If you want to have a flexible way to register your model classes mappings both for Doctrine ORM and PHPCR ODM using a single model class, extend your class with `SWP\Bundle\StorageBundle\DependencyInjection\Bundle\Bundle` from `StorageBundle`.

---

**Note:** SWPStorageBundle is able to load mappings in XML, YAML and annotation formats. By default YAML mappings files are loaded. You can change this by setting the `$mappingFormat` property to: `protected $mappingFormat = BundleInterface::MAPPING_XML;` (see `SWP\Component\Storage\Bundle\BundleInterface`) if you wish to load mapping files in XML format.

---

For example, to extend the AcmeContentBundle, you could use the code:

```php
// src/Acme/ContentBundle/AcmeContentBundle.php
namespace Acme\ContentBundle\AcmeContentBundle;

use SWP\Bundle\StorageBundle\DependencyInjection\Bundle\Bundle;
use SWP\Bundle\StorageBundle\Drivers;
use Symfony\Component\DependencyInjection\ContainerBuilder;

class AcmeContentBundle extends Bundle
{
    /**
     * {@inheritdoc}
     */
    public function getSupportedDrivers()
    {
        return [
            Drivers::DRIVER_DOCTRINE_ORM,
            Drivers::DRIVER_DOCTRINE_PHPCR_ODM,
        ];
    }

    /**
     * {@inheritdoc}
     */
    public function getModelClassNamespace()
    {
        return 'Acme\\ContentBundle\\Model';
    }
}
```

Your bundle can now support multiple drivers. According to the example above, your `Article` model class namespace is specified in the `getModelClassNamespace()` method. Two drivers are configured: PHPCR and ORM. In this case, you can create a model class for PHPCR and ORM and extend the default `Acme\ContentBundle\Model\Article` class. You would then have different implementations for PHPCR and ORM using the same model class:

- `Acme\ContentBundle\ODM\PHPCR\Article` should extend `Acme\ContentBundle\Model\Article`
- `Acme\ContentBundle\ORM\Article` should extend `Acme\ContentBundle\Model\Article`

All that you need to do now is to place the mapping files for each model classes. In this case the `Acme\ContentBundle\ODM\PHPCR\Article` class mapping should be placed inside the `Resources/config/doctrine-phpcr` directory. The mappings for ORM classes should be placed inside the `Resources/config/doctrine-orm` directory.

**Note:** A reference to the directories where mapping files should be placed for model classes is generated automatically, based on the supported driver. In the case of PHPCR it will be `Resources/config/doctrine-phpcr` and in the case of Doctrine ORM it will be `Resources/config/doctrine-orm`. These directories should be created manually if they don't exist already.

The `getSupportedDrivers` defines supported drivers by the `ContentBundle` e.g. PHPCR ODM, MongoDB ODM etc. You should use the `SWP\Bundle\StorageBundle\Drivers` class to specify the supported drivers, as shown in the example above.

The `Drivers` class provides drivers' constants:

```php
namespace SWP\Bundle\StorageBundle;

class Drivers
{
    const DRIVER_DOCTRINE_ORM = 'orm';
    const DRIVER_DOCTRINE_MONGODB_ODM = 'mongodb';
    const DRIVER_DOCTRINE_PHPCR_ODM = 'phpcr';
}
```

### How to automatically register Services required by the configured Storage Driver

This bundle enables you to register required services on the basis of the configured storage driver, to standardize the definitions of registered services.

By default, this bundle registers:

- repository services
- factory services
- object manager services
- parameters

Based on the provided model class it will register the default factory and repository, where you will be able to create a new object based on the provided model class name, adding or removing objects from the repository.

### Set Configuration for your Bundle

Let's start from your bundle configuration, where you will need to specify the default configuration. In this example let's assume you already have a bundle called `ContentBundle` and you want to have working services to manage your resources.

The default `Configuration` class would look something like this:

```php
<?php

namespace Acme\ContentBundle\DependencyInjection;

use Symfony\Component\Config\Definition\Builder\TreeBuilder;
use Symfony\Component\Config\Definition\ConfigurationInterface;

class Configuration implements ConfigurationInterface
{
```

(continues on next page)

(continued from previous page)

```
    /**
     * {@inheritdoc}
     */
    public function getConfigTreeBuilder()
    {
        $treeBuilder = new TreeBuilder();
        $treeBuilder->root('acme_content');

        return $treeBuilder;
    }
}
```

Now, let's add the configuration for the `Acme\ContentBundle\ODM\PHPCR\Article` model class, for the PHPCR driver:

```
<?php

namespace Acme\ContentBundle\DependencyInjection;
// ..

use Acme\ContentBundle\ODM\PHPCR\Article;

class Configuration implements ConfigurationInterface
{
    /**
     * {@inheritdoc}
     */
    public function getConfigTreeBuilder()
    {
        $treeBuilder = new TreeBuilder();
        $treeBuilder->root('acme_content')
            ->children()
                ->arrayNode('persistence')
                    ->addDefaultsIfNotSet()
                    ->children()
                        ->arrayNode('phpcr')
                            ->addDefaultsIfNotSet()
                            ->canBeEnabled()
                            ->children()
                                ->arrayNode('classes')
                                    ->addDefaultsIfNotSet()
                                    ->children()
                                        ->arrayNode('article')
                                            ->addDefaultsIfNotSet()
                                            ->children()
                                                ->scalarNode('model')->
↪cannotBeEmpty()->defaultValue(Article::class)->end()
                                                ->scalarNode('repository')->
↪defaultValue(null)->end()
                                                ->scalarNode('factory')->
↪defaultValue(null)->end()
                                                ->scalarNode('object_manager_name')->
↪defaultValue(null)->end()
                                            ->end()
                                        ->end()
                                    ->end()
```

(continues on next page)

(continued from previous page)

```
                                ->end()
                            ->end()
                        ->end() // phpcr
                    ->end()
                ->end()
            ->end();

        return $treeBuilder;
    }
}
```

**Note:** The `repository`, `factory` and `object_manager_name` nodes are configured to use `null` as the default value. It means that the default factory, repository and object manager services will be registered in the container.

## Register configured classes in your Extension class

Now that you have the configuration defined, it is time to register those classes using the `Extension` class in your bundle. By default, this class is generated inside the `DependencyInjection` folder in every Symfony Bundle.

In this `ContentBundle` example it will be located under the namespace `Acme\ContentBundle\DependencyInjection`. The fully qualified class name will be `Acme\ContentBundle\DependencyInjection\AcmeContentExtension`.

You need to extend this class by the `SWP\Bundle\StorageBundle\DependencyInjection\Extension\Extension` class, which will give you access to register configured classes needed by the storage. The `registerStorage` method will do the whole magic for you. See the code below:

```
<?php

namespace Acme\ContentBundle\DependencyInjection;

// ..
use SWP\Bundle\StorageBundle\Drivers;
use SWP\Bundle\StorageBundle\DependencyInjection\Extension\Extension;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\Config\FileLocator;
use Symfony\Component\DependencyInjection\Loader;

class AcmeContentExtension extends Extension
{
    /**
     * {@inheritdoc}
     */
    public function load(array $configs, ContainerBuilder $container)
    {
        $config = $this->processConfiguration(new Configuration(), $configs);
        $loader = new Loader\YamlFileLoader($container, new FileLocator(__DIR__.'/../
↪Resources/config'));
        $loader->load('services.yml');

        if ($config['persistence']['phpcr']['enabled']) {
            $this->registerStorage(Drivers::DRIVER_DOCTRINE_PHPCR_ODM, $config[
↪'persistence']['phpcr'], $container);
```

(continues on next page)

(continued from previous page)

```
            }
        }
}
```

If the PHPCR persistence backend is enabled, it will register the following services in the container:

| Service ID | Class name |
|---|---|
| swp.factory.article | SWP\Bundle\StorageBundle\Factory\Factory |
| swp.repository.article.class \| Acme\ContentBundle\PHPCR\Article | |
| swp.repository.article | SWP\Bundle\StorageBundle\Doctrine\ODM\PHPCR\DocumentRepository |

together with all parameters:

| Parameter Name | Value |
|---|---|
| swp.factory.article.class | SWP\Bundle\StorageBundle\Factory\Factory |
| swp.model.article.class | Acme\ContentBundle\PHPCR\Article |
| swp.repository.article.class | SWP\Bundle\StorageBundle\Doctrine\ODM\PHPCR\DocumentRepository |

If your configuration supports Doctrine ORM instead of PHPCR, the default service definitions would be:

| Service ID | Class name |
|---|---|
| swp.factory.article | SWP\Bundle\StorageBundle\Factory\Factory |
| swp.object_manager.article | alias for "doctrine.orm.default_entity_manager" |
| swp.repository.article | SWP\Bundle\StorageBundle\Doctrine\ORM\EntityRepository |

And all parameters in the container would look like:

| Parameter Name | Value |
|---|---|
| swp.factory.article.class | SWP\Bundle\StorageBundle\Factory\Factory |
| swp.model.article.class | Acme\ContentBundle\ORM\Article |
| swp.repository.article.class | SWP\Bundle\StorageBundle\Doctrine\ORM\EntityRepository |

You could then access parameters from the container, as visible below:

```php
<?php
//..
$className = $container->getParameter('swp.model.article.class');
var_dump($className); // will return Acme\ContentBundle\PHPCR\Article
```

Now, register all classes in the configuration file:

```yaml
# app/config/config.yml
swp_content:
    persistence:
        phpcr: true
```

The above configuration is equivalent to:

```yaml
# app/config/config.yml
swp_content:
    persistence:
```

(continues on next page)

(continued from previous page)

```yaml
        phpcr:
            enabled: true
            classes:
                article:
                    model: Acme\ContentBundle\ODM\PHPCR\Article
                    factory: ~
                    repository: ~
                    object_manager_name: ~
```

### How to create and use custom repository service for your model

For some use cases you would need to implement your own methods in the repository, like `findOneBySlug()` or `findAllArticles()`. It's very easy!

You need to create your custom implementation for the repository. In this example you will create a custom repository for the `Article` model class and Doctrine PHPCR persistence backend.

Firstly, you need to create your custom repository interface. Let's name it `ArticleRepositoryInterface` and extend it by the `SWP\Component\Storage\Repository\RepositoryInterface` interface:

```php
<?php

namespace Acme\ContentBundle\PHPCR;

use Acme\ContentBundle\Model\ArticleInterface;
use SWP\Component\Storage\Repository\RepositoryInterface;

interface ArticleRepositoryInterface extends RepositoryInterface
{
    /**
     * Find one article by slug.
     *
     * @param string $slug
     *
     * @return ArticleInterface
     */
    public function findOneBySlug($slug);

    /**
     * Find all articles.
     *
     * @return mixed
     */
    public function findAllArticles();
}
```

Secondly, you need to create your custom repository class. Let's name it `ArticleRepository` and implement the `ArticleRepositoryInterface` interface:

```php
<?php

namespace Acme\ContentBundle\PHPCR;

use Acme\ContentBundle\Model\ArticleRepositoryInterface;
use SWP\Bundle\StorageBundle\Doctrine\ODM\PHPCR\DocumentRepository;
```

(continues on next page)

(continued from previous page)

```

class ArticleRepository extends DocumentRepository implements␣
↪ArticleRepositoryInterface
{
    /**
     * {@inheritdoc}
     */
    public function findOneBySlug($slug)
    {
        return $this->findOneBy(['slug' => $slug]);
    }

    /**
     * {@inheritdoc}
     */
    public function findAllArticles()
    {
        return $this->createQueryBuilder('o')->getQuery();
    }
}
```

**Note:** If you want to create a custom repository for the Doctrine ORM persistence backend, you need to extend your custom repository class by the `SWP\Bundle\StorageBundle\Doctrine\ORM\EntityRepository` class.

The last step is to add your custom repository to the configuration file:

```
# app/config/config.yml
swp_content:
    persistence:
        phpcr:
            enabled: true
            classes:
                article:
                    model: Acme\ContentBundle\ODM\PHPCR\Article
                    factory: ~
                    repository: Acme\ContentBundle\PHPCR\ArticleRepository
                    object_manager_name: ~
```

**Note:** Alternatively, you could add it directly in your `Configuration` class.

**Note:** You can change repository class by simply changing your bundle configuration, without needing to change the code.

### How to create and use custom factory service for your model

You may need to have a different way of creating objects than the default way of doing it. Imagine you need to create an `Article` object with the route assigned by default.

**Note:** In this example you will create a custom factory for your `Article` object and Doctrine PHPCR persistence backend.

---

Let's create a custom interface for your factory. Extend your custom class by the `SWP\Component\Storage\Factory\FactoryInterface` class:

```php
<?php

namespace Acme\ContentBundle\Factory;

use SWP\Bundle\ContentBundle\Model\ArticleInterface;
use SWP\Component\Bridge\Model\PackageInterface;
use SWP\Component\Storage\Factory\FactoryInterface;

interface ArticleFactoryInterface extends FactoryInterface
{
    /**
     * Create a new object with route.
     *
     * @param string $route
     *
     * @return ArticleInterface
     */
    public function createWithRoute($route);
}
```

Create the custom Article factory class:

```php
<?php

namespace Acme\ContentBundle\Factory;

use SWP\Component\Storage\Factory\FactoryInterface;

class ArticleFactory implements ArticleFactoryInterface
{
    /**
     * @var FactoryInterface
     */
    private $baseFactory;

    /**
     * ArticleFactory constructor.
     *
     * @param FactoryInterface $baseFactory
     */
    public function __construct(FactoryInterface $baseFactory)
    {
        $this->baseFactory = $baseFactory;
    }

    /**
     * {@inheritdoc}
     */
    public function create()
    {
        return $this->baseFactory->create();
    }
```

(continues on next page)

(continued from previous page)

```

    /**
     * {@inheritdoc}
     */
    public function createWithRoute($route)
    {
        $article = $this->create();
        // ..
        $article->setRoute($route);

        return $article;
    }
}
```

Create a compiler pass to override the default Article factory class with your custom factory on container compilation:

```php
<?php

namespace Acme\ContentBundle\DependencyInjection\Compiler;

use SWP\Component\Storage\Factory\Factory;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\DependencyInjection\Compiler\CompilerPassInterface;
use Symfony\Component\DependencyInjection\Definition;
use Symfony\Component\DependencyInjection\Parameter;

class RegisterArticleFactoryPass implements CompilerPassInterface
{
    /**
     * {@inheritdoc}
     */
    public function process(ContainerBuilder $container)
    {
        if (!$container->hasDefinition('swp.factory.article')) {
            return;
        }

        $baseDefinition = new Definition(
            Factory::class,
            [
                new Parameter('swp.model.article.class'),
            ]
        );

        $articleFactoryDefinition = new Definition(
            $container->getParameter('swp.factory.article.class'),
            [
                $baseDefinition,
            ]
        );

        $container->setDefinition('swp.factory.article', $articleFactoryDefinition);
    }
}
```

Don’t forget to register your new compiler pass in your Bundle class (`AcmeContentBundle`):

```php
<?php

use Acme\ContentBundle\DependencyInjection\Compiler\RegisterArticleFactoryPass;
// ..

/**
 * {@inheritdoc}
 */
public function build(ContainerBuilder $container)
{
    parent::build($container);
    $container->addCompilerPass(new RegisterArticleFactoryPass());
}
```

The last thing required to make use of your new factory service is to add it to the configuration file, under the `factory` node:

```yaml
# app/config/config.yml
swp_content:
    persistence:
        phpcr:
            enabled: true
            classes:
                article:
                    model: Acme\ContentBundle\ODM\PHPCR\Article
                    factory: Acme\ContentBundle\Factory\ArticleFactory
                    repository: ~
                    object_manager_name: ~
```

**Note:** Alternatively, you could add it directly in your `Configuration` class.

You would then be able to use the factory like so:

```php
$article = $this->get('swp.factory.article')->createWithRoute('some-route');
// or create flat object
$article = $this->get('swp.factory.article')->create();
```

**Note:** You can change factory class by simply changing your bundle configuration, without needing to change the code.

### Configuring object manager for your model

As you can see, there is the `object_manager_name` option in the `Configuration` class, which is the default Object Manager (Contract for a Doctrine persistence layer) name.

In the case of Doctrine ORM it's `doctrine.orm.default_entity_manager`, in PHPCR it's `doctrine_phpcr.odm.default_document_manager`.

If you set this option to be, for example, `test` the `doctrine.orm.test_entity_manager` object manager service's id will be used. Of course this new `test` document, in the case of PHPCR, should be first configured in the Doctrine PHPCR Bundle as described in the bundle documentation on multiple document managers. For Doctrine ORM it should be configured as shown in the Doctrine ORM Bundle documentation on multiple entity managers.

The possibility of defining a default Object Manager for a Doctrine persistence layer, and making use of it in the registered repositories and factories in your Bundle, is very useful in case you are using different databases or even different sets of entities.

**Note:** Factories and repositories are defined as a services in Symfony container to have better flexibility of use.

### Resolve target entities

This chapter is strictly related to How to Define Relationships with Abstract Classes and Interfaces so please read it first.

This functionality allows you to define relationships between different entities without making them hard dependencies. All you need to do is to define `interface` node in your bundle's *Configuration* class.

See example below:

```php
<?php

namespace Acme\Bundle\CoreBundle\DependencyInjection;
// ..

use Acme\Component\MultiTenancy\Model\Tenant;
use Acme\Component\MultiTenancy\Model\TenantInterface;
use Acme\Component\MultiTenancy\Model\Organization;
use Acme\Component\MultiTenancy\Model\OrganizationInterface;

class Configuration implements ConfigurationInterface
{
    /**
     * {@inheritdoc}
     */
    public function getConfigTreeBuilder()
    {
        $treeBuilder = new TreeBuilder();
        $treeBuilder->root('acme_core')
            ->children()
                ->arrayNode('persistence')
                    ->addDefaultsIfNotSet()
                    ->children()
                        ->arrayNode('phpcr')
                            ->addDefaultsIfNotSet()
                            ->canBeEnabled()
                            ->children()
                                ->arrayNode('classes')
                                    ->addDefaultsIfNotSet()
                                    ->children()
                                        ->arrayNode('tenant')
                                            ->addDefaultsIfNotSet()
                                            ->children()
                                                ->scalarNode('model')->
↪cannotBeEmpty()->defaultValue(Tenant::class)->end()
                                                ->scalarNode('interface')->
↪cannotBeEmpty()->defaultValue(TenantInterface::class)->end()
                                                ->scalarNode('repository')->
↪defaultValue(null)->end()
```

(continues on next page)

(continued from previous page)

```
                                                ->scalarNode('factory')->defaultValue(null)->end()
                                                ->scalarNode('object_manager_name')->defaultValue(null)->end()
                                            ->end()
                                        ->end()
                                        ->arrayNode('organization')
                                            ->addDefaultsIfNotSet()
                                            ->children()
                                                ->scalarNode('model')->cannotBeEmpty()->defaultValue(Organization::class)->end()
                                                ->scalarNode('interface')->cannotBeEmpty()->defaultValue(OrganizationInterface::class)->end()
                                                ->scalarNode('repository')->defaultValue(null)->end()
                                                ->scalarNode('factory')->defaultValue(null)->end()
                                                ->scalarNode('object_manager_name')->defaultValue(null)->end()
                                            ->end()
                                        ->end()
                                    ->end()
                                ->end()
                            ->end()
                        ->end() // phpcr
                    ->end()
                ->end()
            ->end();

        return $treeBuilder;
    }
}
```

In this case you will be able to specify your interface for your model via config file:

```yaml
# app/config/config.yml
swp_content:
    persistence:
        phpcr:
            enabled: true
            classes:
                tenant:
                    model: Acme\Bundle\CoreBundle\Model\Tenant # extends default Acme\Component\MultiTenancy\Model\Tenant class
                    interface: ~
                    # ..
                organization:
                    model: Acme\Bundle\CoreBundle\Model\Organization # extends default Acme\Component\MultiTenancy\Model\Organization class
                    interface: ~
                    # ..
```

**Now, no matter which model (implementing for example `Acme\Component\MultiTenancy\Model\OrganizationInte`**
you will use in your bundle's configuration above, the interface will be automatically resolved to defined entity and will be used by your mapping file without a need to change any extra code or configuration setup.

The above is equivalent to if the Tenant has a relation to Organization and vice versa.

```yaml
# app/config/config.yml
doctrine:
    # ...
    orm:
        # ...
        resolve_target_entities:
            Acme\Component\MultiTenancy\Model\OrganizationInterface: Acme\Bundle\CoreBundle\Model\Organization
            Acme\Component\MultiTenancy\Model\TenantInterface: Acme\Bundle\CoreBundle\Model\Tenant
```

In this example above every time you will want to change your model inside your bundle's configuration you would also need to care about the Doctrine config as the specified entity will not change automatically to a new one which was defined in bundle's config.

### Inheritance Mapping

By default every entity inside bundle should be mapped as Mapped superclass. This bundle helps you manage and simplify inheritance mapping in case you want to use default mapping or extend it. In this case the following applies:

- If you do not configure your custom class, the default mapped superclasses become entites.
- Otherwise they become mapped superclasses and move the conflicting mappings (these which you cannot normally configure on mapped superclass) to your class mapping. For example, you do not need anymore to map Organization -> Tenants inside your custom class, it is copied transparently from the bundle.
- It also works on all levels, so you can cleanly override the core bundle models! If you configure other class than core one, your entity will be used and the core model will remain mapped superclass.

**Note:** This feature and its description has been ported from Sylius project. See related issue.

## 4.4 BridgeBundle

This bundle provides tools which help you to integrate Superdesk data with Superdesk Web Publisher.

*The Bridge Component*, which is used by this bundle, provides a generic interface to create different type of validators, data transformers, and models, which in turn help to pull and process data from Superdesk.

### 4.4.1 Prerequisites

This version of the bundle requires Symfony >= 2.8 and PHP version >=5.6.

### 4.4.2 Installation

#### Install the Bundle with Composer

In your project directory execute the following command to download the latest stable version:

```
composer require swp/bridge-bundle jms/serializer-bundle swp/jms-serializer-bridge
```

This command requires you to have Composer installed globally. If it's not installed globally, download the .phar file locally as explained in Composer documentation.

### Enable the bundle and its dependencies

Enable the bundle by adding the following lines in the app/AppKernel.php file:

```php
// app/AppKernel.php

// ...
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            // ...
            new JMS\SerializerBundle\JMSSerializerBundle(),
            new SWP\Bundle\BridgeBundle\SWPBridgeBundle()
        );

        // ...
    }

    // ...
}
```

**Note:** All dependencies will be installed automatically. You will just need to configure the respective bundles if needed.

That's it, the bundle is configured properly now!

## 4.4.3 Usage

### Using Validator Chain Service

The Validator Chain service is used to register all validators with a tag validator.http_push_validator.

Usage:

```php
// ...
use Symfony\Component\HttpFoundation\Response;

public function indexAction()
{
    $value = 'some value';
    $result = $this->get('swp_bridge.http_push.validator_chain')->isValid($value);

    return new Response($result);
}
```

### How to Create and Register Custom Validators

Validators are used to process incoming request content and validate it against the specific schema. Read more about it in the Bridge component documentation (in the *Usage* section).

### Creating the Validator Class

A new Validator has to implement the `SWP\Component\Bridge\Validator\ValidatorInterface` and `SWP\Component\Bridge\Validator\ValidatorOptionsInterface` interfaces.

`CustomValidator` class example:

```php
<?php

namespace Acme\DemoBundle\Validator;

use SWP\Component\Bridge\Validator\ValidatorInterface;
use SWP\Component\Bridge\Validator\ValidatorOptionsInterface

final class CustomValidator implement ValidatorInterface, ValidatorOptionsInterface
{
    /**
     * @var string
     */
    private $schema = 'custom schema';

    /**
     * {@inheritdoc}
     */
    public function isValid($data)
    {
        // custom validation here
    }

    /**
     * {@inheritdoc}
     */
    public function getSchema()
    {
        return $this->schema;
    }

    /**
     * {@inheritdoc}
     */
    public function getFormat()
    {
        return 'custom';
    }
}
```

### Configuring the Validator

To register your new validator, simply add a definition to your services file and tag it with a special name: `validator.http_push_validator`:

```yaml
# Resources/config/services.yml
acme_my_custom_validator:
    class: 'Acme\DemoBundle\Validator\CustomValidator'
    tags:
        - { name: validator.http_push_validator, alias: http_push.custom }
```

**Note:** You can use the `SWP\Component\Bridge\Validator\JsonValidator` abstract class if you wish to create custom JSON validator.

### Using Transformer Chain Service

Transformer Chain service is used to register all transformers with a tag `transformer.http_push_transformer`.

Usage:

```php
// ...

public function indexAction()
{
    $value = 'some value';
    $result = $this->get('swp_bridge.http_push.transformer_chain')->transform($value);
    $result = $this->get('swp_bridge.http_push.transformer_chain')->reverseTransform(
↪$value);
}
```

### How to Create and Register Custom Data Transformers

Data transformers are used to transform one value/object into another. Read more about it in the Bridge component documentation (in the *Usage* section).

### Creating the Data Transformer Class

To create a new Data Transformer, your new class should implement the `SWP\Component\Bridge\Transformer\DataTransformerInterface` interface.

`CustomValidator` class example:

```php
<?php

namespace Acme\DemoBundle\Transformer;

use Acme\DemoBundle\Model\Custom;
use SWP\Component\Bridge\Exception\MethodNotSupportedException;
use SWP\Component\Bridge\Exception\TransformationFailedException;
use SWP\Component\Bridge\Validator\ValidatorInterface;
use SWP\Component\Common\Serializer\SerializerInterface;

final class JsonToObjectTransformer implements DataTransformerInterface
{
    /**
```

(continues on next page)

(continued from previous page)

```php
     * @var SerializerInterface
     */
    private $serializer;

    /**
     * @var ValidatorInterface
     */
    private $validatorChain;

    /**
     * JsonToPackageTransformer constructor.
     *
     * @param SerializerInterface $serializer
     * @param ValidatorInterface  $validatorChain
     */
    public function __construct(SerializerInterface $serializer, ValidatorInterface
↪$validatorChain)
    {
        $this->serializer = $serializer;
        $this->validatorChain = $validatorChain;
    }

    /**
     * {@inheritdoc}
     */
    public function transform($json)
    {
        if (!$this->validatorChain->isValid($json)) {
            throw new TransformationFailedException('None of the chained validators␣
↪were able to validate the data!');
        }

        return $this->serializer->deserialize($json, Custom::class, 'json');
    }

    /**
     * {@inheritdoc}
     */
    public function reverseTransform($value)
    {
        throw new MethodNotSupportedException('reverseTransform');
    }
}
```

### Configuring the Data Transformer

To register your new Data Transformer, simply add a definition to your services file and tag it with a special name: `transformer.http_push_transformer`:

```yaml
# Resources/config/services.yml
acme_my_custom_transformer:
    class: 'Acme\DemoBundle\Transformer\CustomTransformer'
    arguments:
        - '@swp.serializer'
        - '@swp_bridge.http_push.validator_chain'
```

(continues on next page)

(continued from previous page)

```
    tags:
        - { name: transformer.http_push_transformer, alias: transformer.json_to_
↪object }
```

### Enabling a separate Monolog channel for Validators

It is possible to enable a separate Monolog channel to which all validators related logs will be forwarded. An example log entry might be logged when the incoming payload can not be validated properly.. You could have then a separate log file for which will be usually saved under the directory `app/logs/` in your application and will be named, for example: `swp_validators_<env>.log`. By default, a separate channel is disabled. You can enable it by adding an extra channel in your Monolog settings (in one of your configuration files):

```
# app/config/config.yml
monolog:
    handlers:
        swp_validators:
            level:    debug
            type:     stream
            path:     '%kernel.logs_dir%/swp_validators_%kernel.environment%.log'
            channels: swp_validators
```

For more details see the Monolog documentation.

## 4.5 ContentBundle

This bundle provides functionality for defining the content of a site managed with Superdesk Web Publisher.

### 4.5.1 Usage

#### Rules to assign routes and/or templates to articles

With the API, it is possible to create and manage rules in order to assign a route and/or a template to content received in a package from a provider based on the metadata in that package.

The rules themselves are to be written in Symfony's expression language, documentation for which can be found here: http://symfony.com/doc/current/components/expression_language/syntax.html

The article document generated from the package can be referenced directly in the rule. So, here is an example of a rule:

```
'article.getMetadataByKey("var_name") matches "/regexExp/"'
# ..
'article.getMetadataByKey("urgency") > 1'
# or
'article.getMetadataByKey("lead") matches "/Text/"'
# etc.
```

A priority can also be assigned to the rule. This is simply an integer. The rules are ordered by their priority (the greater the value, the higher the priority) before searching for the first one which matches.

The route to be assigned is identified by its id, for example:

```
'articles/features'
```

If a template name parameter (*templateName*) is given with the rule, this template will be assigned to the article instead of the one in the route.

Every rule is automatically processed when the content is pushed to `api/v1/content/push` API endpoint. The `SWP\Bundle\ContentBundle\EventListener\ProcessArticleRulesSubscriber` subscriber subscribes to `SWP\Bundle\ContentBundle\ArticleEvents::PRE_CREATE` event and runs the processing when needed.

## Pushing Content to Web Publisher

### Content Push API Endpoint

#### Content Push API Endpoint

| Method | URL |
|---|---|
| POST | /api/v1/content/push |

#### Resource details

| | |
|---|---|
| Response format | JSON |
| Authentication | No |

#### Example Request

```
POST https://<tenant_name>.domain.com/api/v1/content/push
```

#### Example Response

```
{
    "status": "OK"
}
```

Response status code is `201`.

### Which content formats are allowed to be pushed?

The Superdesk Web Publisher can receive content of different formats, by default the IPTC's ninjs format is supported.

- ninjs - standardizes the representation of news in JSON - a lightweight, easy-to-parse, data interchange format.

In Superdesk Web Publisher we use an extension of the standard IPTC's ninjs format (to validate incoming request's content), which is extended by some additional fields:

- *uri* was replaced by *guid*: *uri* should be the resource identifier on the web but since the item was not published yet it can't be determined at this point
- added *priority* field
- added *service* field
- added *slugline* field
- added *keywords* field

- *associations* dictionary may contain entire items like in this ninjs example: http://dev.iptc.org/ninjs-Examples-3

By default, Superdesk Web Publisher is meant to work with our in-house content creation software called Superdesk which uses the above extension of ninjs format to store the data. That's why in Web Publisher the incoming content is being validated in the same format.

In the future we could also support other IPTC formats, like: NewsML-G2, NITF etc. So you would be able to push content to Web Publisher in format that fits you best.

Superdesk ninjs extension schema used by Web Publisher:

```
{
    "$schema": "http://json-schema.org/draft-03/schema#",
    "id" : "http://www.iptc.org/std/ninjs/ninjs-schema_1.1.json#",
    "type" : "object",
    "title" : "IPTC ninjs - News in JSON - version 1.1 (approved, 2014-03-12) /␣
↪document revision of 2014-11-15: geometry_* moved under place",
    "description" : "A news item as JSON object -- copyright 2014 IPTC -␣
↪International Press Telecommunications Council - www.iptc.org - This document is␣
↪published under the Creative Commons Attribution 3.0 license, see  http://
↪creativecommons.org/licenses/by/3.0/  $$comment: as of 2014-03-13 ",
    "additionalProperties" : false,
    "patternProperties" : {
        "^description_[a-zA-Z0-9_]+" : {
            "description" : "A free-form textual description of the content of the␣
↪item. (The string appended to description_ in the property name should reflect the␣
↪format of the text)",
            "type" : "string"
        },
        "^body_[a-zA-Z0-9_]+" : {
            "description" : "The textual content of the news object. (The string␣
↪appended to body_ in the property name should reflect the format of the text)",
            "type" : "string"
        }
    },
    "properties" : {
        "guid" : {
            "description" : "The identifier for this news object",
            "type" : "string",
            "format" : "guid",
            "required" : true
        },
        "type" : {
            "description" : "The generic news type of this news object",
            "type" : "string",
            "enum" : ["text", "audio", "video", "picture", "graphic", "composite"]
        },
        "slugline" : {
            "description" : "The slugline",
            "type" : "string",
            "required" : true
        },
        "mimetype" : {
            "description" : "A MIME type which applies to this news object",
            "type" : "string"
        },
        "representationtype" : {
            "description" : "Indicates how complete this representation of a news␣
↪item is",
```

(continues on next page)

(continued from previous page)

```
            "type" : "string",
            "enum" : ["complete", "incomplete"]
        },
        "profile" : {
            "description" : "An identifier for the kind of content of this news object
↪",
            "type" : "string"
        },
        "version" : {
            "description" : "The version of the news object which is identified by␣
↪the uri property",
            "type" : "string"
        },
        "versioncreated" : {
            "description" : "The date and time when this version of the news object␣
↪was created",
            "type" : "string",
            "format" : "date-time"
        },
        "embargoed" : {
            "description" : "The date and time before which all versions of the news␣
↪object are embargoed. If absent, this object is not embargoed.",
            "type" : "string",
            "format" : "date-time"
        },
        "pubstatus" : {
            "description" : "The publishing status of the news object, its value is␣
↪*usable* by default.",
            "type" : "string",
            "enum" : ["usable", "withheld", "canceled"]
        },
        "urgency" : {
            "description" : "The editorial urgency of the content from 1 to 9. 1␣
↪represents the highest urgency, 9 the lowest.",
            "type" : "number"
        },
        "priority" : {
            "description" : "The editorial priority of the content from 1 to 9. 1␣
↪represents the highest priority, 9 the lowest.",
            "type" : "number"
        },
        "copyrightholder" : {
            "description" : "The person or organisation claiming the intellectual␣
↪property for the content.",
            "type" : "string"
        },
        "copyrightnotice" : {
            "description" : "Any necessary copyright notice for claiming the␣
↪intellectual property for the content.",
            "type" : "string"
        },
        "usageterms" : {
            "description" : "A natural-language statement about the usage terms␣
↪pertaining to the content.",
            "type" : "string"
        },
        "language" : {
```

(continues on next page)

(continued from previous page)

```
            "description" : "The human language used by the content. The value should␣
↪follow IETF BCP47",
            "type" : "string"
        },
        "service" : {
            "description" : "A service e.g. World Photos, UK News etc.",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of a service",
                        "type" : "string"
                    },
                    "code" : {
                        "description": "The code for the service in a scheme (=␣
↪controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    }
                }
            }
        },
        "person" : {
            "description" : "An individual human being",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of a person",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news␣
↪object to the person",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled␣
↪vocabulary) which includes a code for the person",
                        "type" : "string",
                        "format" : "uri"
                    },
                    "code" : {
                        "description": "The code for the person in a scheme (=␣
↪controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    }
                }
            }
        },
        "organisation" : {
            "description" : "An administrative and functional structure which may act␣
↪as as a business, as a political party or not-for-profit party",
            "type" : "array",
```

(continues on next page)

(continued from previous page)

```
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of the organisation",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news␣
↪object to the organisation",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled␣
↪vocabulary) which includes a code for the organisation",
                        "type" : "string",
                        "format" : "uri"
                    },
                    "code" : {
                        "description": "The code for the organisation in a scheme (=␣
↪controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    },
                    "symbols" : {
                        "description" : "Symbols used for a finanical instrument␣
↪linked to the organisation at a specific market place",
                        "type" : "array",
                        "items" : {
                            "type" : "object",
                            "additionalProperties" : false,
                            "properties" : {
                                "ticker" : {
                                    "description" : "Ticker symbol used for the␣
↪financial instrument",
                                    "type": "string"
                                },
                                "exchange" : {
                                    "description" : "Identifier for the marketplace␣
↪which uses the ticker symbols of the ticker property",
                                    "type" : "string"
                                }
                            }
                        }
                    }
                }
            }
        },
        "place" : {
            "description" : "A named location",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "patternProperties" : {
                    "^geometry_[a-zA-Z0-9_]+" : {
                        "description" : "An object holding geo data of this place.␣
↪Could be of any relevant geo data JSON object definition.",
```

(continues on next page)

(continued from previous page)

```
                        "type" : "object"
                    }
                },
                "properties" : {
                    "name" : {
                        "description" : "The name of the place",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news␣
↪object to the place",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled␣
↪vocabulary) which includes a code for the place",
                        "type" : "string",
                        "format" : "uri"
                    },
                    "qcode" : {
                        "description": "The code for the place in a scheme (=␣
↪controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    },
                    "state" : {
                        "description" : "The state for the place",
                        "type" : "string"
                    },
                    "group" : {
                        "description" : "The place group",
                        "type" : "string"
                    },
                    "name" : {
                        "description" : "The place name",
                        "type" : "string"
                    },
                    "country" : {
                        "description" : "The country name",
                        "type" : "string"
                    },
                    "world_region" : {
                        "description" : "The world region",
                        "type" : "string"
                    }
                }
            }
        },
        "subject" : {
            "description" : "A concept with a relationship to the content",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of the subject",
                        "type" : "string"
```

(continues on next page)

(continued from previous page)

```
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news object to the subject",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled vocabulary) which includes a code for the subject",
                        "type" : "string",
                        "format" : "uri"
                    },
                    "code" : {
                        "description": "The code for the subject in a scheme (= controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    }
                }
            }
        },
        "event" : {
            "description" : "Something which happens in a planned or unplanned manner",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of the event",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news object to the event",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled vocabulary) which includes a code for the event",
                        "type" : "string",
                        "format" : "uri"
                    },
                    "code" : {
                        "description": "The code for the event in a scheme (= controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    }
                }
            }
        },
        "object" : {
            "description" : "Something material, excluding persons",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
```

(continues on next page)

(continued from previous page)

```
                "properties" : {
                    "name" : {
                        "description" : "The name of the object",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news␣
↪object to the object",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled␣
↪vocabulary) which includes a code for the object",
                        "type" : "string",
                        "format" : "uri"
                    },
                    "code" : {
                        "description": "The code for the object in a scheme (=␣
↪controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    }
                }
            }
        },
        "byline" : {
            "description" : "The name(s) of the creator(s) of the content",
            "type" : "string"
        },
        "headline" : {
            "description" : "A brief and snappy introduction to the content, designed␣
↪to catch the reader's attention",
            "type" : "string"
        },
        "located" : {
            "description" : "The name of the location from which the content␣
↪originates.",
            "type" : "string"
        },
        "keywords": {
            "description" : "Content keywords",
            "type" : "array"
        },
        "renditions" : {
            "description" : "Wrapper for different renditions of non-textual content␣
↪of the news object",
            "type" : "object",
            "additionalProperties" : false,
            "patternProperties" : {
                "^[a-zA-Z0-9]+" : {
                    "description" : "A specific rendition of a non-textual content of␣
↪the news object.",
                    "type" : "object",
                    "additionalProperties" : false,
                    "properties" : {
                        "href" : {
                            "description" : "The URL for accessing the rendition as a␣
↪resource",
```

(continues on next page)

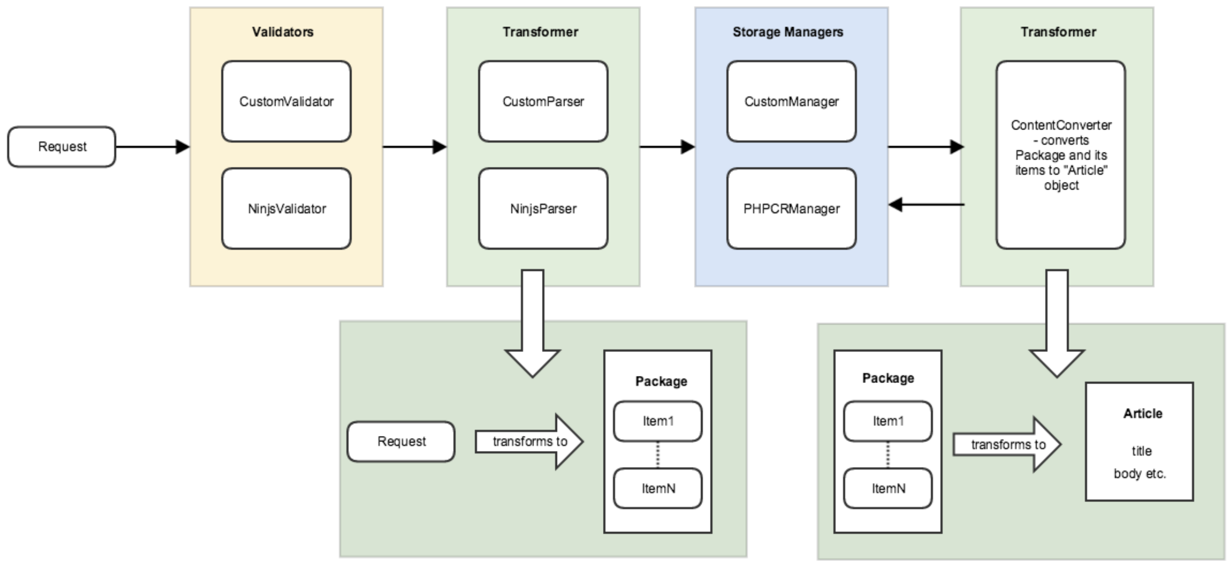
(continued from previous page)

```
                            "type" : "string",
                            "format" : "uri"
                        },
                        "mimetype" : {
                            "description" : "A MIME type which applies to the rendition",
                            "type" : "string"
                        },
                        "title" : {
                            "description" : "A title for the link to the rendition resource",
                            "type" : "string"
                        },
                        "height" : {
                            "description" : "For still and moving images: the height of the display area measured in pixels",
                            "type" : "number"
                        },
                        "width" : {
                            "description" : "For still and moving images: the width of the display area measured in pixels",
                            "type" : "number"
                        },
                        "sizeinbytes" : {
                            "description" : "The size of the rendition resource in bytes",
                            "type" : "number"
                        }
                    }
                }
            }
        },
        "associations" : {
            "description" : "Content of news objects which are associated with this news object.",
            "type" : "object",
            "additionalProperties" : false,
            "patternProperties" : {
                "^[a-zA-Z0-9]+" :  { "$ref": "http://www.iptc.org/std/ninjs/ninjs-schema_1.0.json#" }
            }
        }
    }
}
```

## How Content is Pushed to Web Publisher?

Making a `POST` request to `api/v1/content/push` API endpoint, you can push whatever content you want in request's payload. See *Content Push API Endpoint* section.

The below diagram shows the data flow from the moment of their receipt by the `content/push` API endpoint until it's saved as a resulting `Article` object in the persistence backend.



As you can see the request is being sent first, then the request's content is validated using Web Publisher validators (see validators *Usage* section for more info about validators).

Once the content passes the validation (i.e. submitted content is for example in ninjs format), a respective transformer/parser (see *Using Transformer Chain Service* for more details) transforms (according to submitted content) the incoming data to format which is understandable by Web Publisher, i.e. `Package` and `Item` objects which are reflecting the submitted content.

Once this is done, the converted request's content is being persisted in Web Publisher persistence backend as a representation of `Package` and `Item` objects.

The last step is converting already persisted `Package` and `Item` objects to `Article` object which is used by Web Publisher internally and on which every operation is being made.

### How the article's slug is being generated?

The article's slug is being generated from the `slugline` field, if it is not empty, else the `headline` property's value (according to ninjs IPTC format) is used to populate the article's slug.

**Warning:** If the `slugline` property in incoming data is missing or is empty, the article's slug will be generated from the `headline`, which means if you would want to change the `headline` and submit content again, a new article will be created instead as the article's slug will be generated from the `headline` field.

### What happens when I want to change article's title?

You can change existing article's title in the content that you are sending to Web Publisher. Let's say we have a simple `text` *item* or *package* in ninjs format, as it is defined according to *Which content formats are allowed to be pushed?*.

Once the item/package `headline` is changed and the whole content is pushed to Web Publisher again, the article's title will be updated automatically.

### What happens when I want to change article's slug?

If an article already exists and you want to change the article's slug, the content which you used to create the article for the first time should be re-sent with modified `slugline` property. Once you change the `slugline` property's value and submit it again to Web Publisher, a new article will be created.

### How do I auto-publish an article?

In some cases, you will need to publish an article automatically, without additional action. In this bundle a special logic has been implemented which is responsible for the auto publishing articles, which is based on the rules. You can read more about rules in sections: *Rules to assign routes and/or templates to articles*, RuleBundle - *Usage* section.

All you need to do in order to auto-publish your articles, you need to first add a rule. If the article will match the rule, it will be auto published.

Create a new rule:

```
$ curl 'http://localhost/api/v1/rules/' -H 'Content-Type: application/x-www-form-
↪urlencoded' --data 'rule%5Bpriority%5D=1&rule%5Bexpression%5D=article.
↪getMetadataByKey(%22located%22)+matches+%22%2FSydney%2F%22&rule%5Bconfiguration%5D
↪%5B0%5D%5Bkey%5D=published&rule%5Bconfiguration%5D%5B0%5D%5Bvalue%5D=true' --
↪compressed
```

Submitted rule's expression:

```
article.getMetadataByKey("located") matches "/Sydney/"
```

Submitted rule's configuration:

```
rule[configuration][0][key]: published
rule[configuration][0][value]: true
```

It means that if the above rule's expression matches any article, it will apply the configuration to it - in this case it will publish article.

## 4.6 RuleBundle

This bundle provides a simple business rules engine for Symfony applications.

*The Rule Component*, which is used by this bundle, provides a generic interface to create different type of rule applicators, models etc., which in turn help to create powerful business rules engine.

It means you can create your own rules and apply them to whatever objects you need:

```
# Get the special price if
user.getGroup() in ['good_customers', 'collaborator']

# Promote article to the homepage when
article.commentCount > 100 and article.category not in ["misc"]

# Send an alert when
product.stock < 15
```

### 4.6.1 Prerequisites

This version of the bundle requires Symfony >= 2.8 and PHP version >=5.6.

### 4.6.2 Installation

#### Install the Bundle with Composer

In your project directory execute the following command to download the latest stable version:

```
composer require swp/rule-bundle
```

This command requires you to have Composer installed globally. If it's not installed globally, download the `.phar` file locally as explained in Composer documentation.

#### Enable the bundle and its dependencies

Enable the bundle by adding the following lines in the `app/AppKernel.php` file:

```php
// app/AppKernel.php

// ...
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            new Burgov\Bundle\KeyValueFormBundle\BurgovKeyValueFormBundle(),
            new SWP\Bundle\RuleBundle\SWPStorageBundle()
            // ...
            new SWP\Bundle\RuleBundle\SWPRuleBundle()
        );

        // ...
    }

    // ...
}
```

**Note:** All dependencies will be installed automatically. You will just need to configure the respective bundles if needed.

#### Configure the bundle

- *YAML*

```yaml
# app/config/config.yml
swp_rule:
    persistence:
        orm:
            # if true, ORM is enabled as a persistence backend
            enabled: true
```

**Note:** By default this bundle supports only Doctrine ORM as a persistence backend.

### Update Database Schema

Run the following command:

```
$ php app/console doctrine:schema:update --force
```

That's it, the bundle is configured properly now!

## 4.6.3 Usage

### How to Create and Register Custom Rule Applicator

Rule applicators are used to apply given rule's configuration to an object. Read more about it in the Rule component documentation (in the *Usage* section).

### Creating the Rule Applicator Class

A new Rule Applicator has to implement the `SWP\Component\Rule\Applicator\RuleApplicatorInterface` interface.

`ArticleRuleApplicator` class example:

```php
<?php

namespace Acme\DemoBundle\Applicator;
// ..
use Psr\Log\LoggerInterface;
use SWP\Bundle\ContentBundle\Model\ArticleInterface;
use SWP\Bundle\ContentBundle\Provider\RouteProviderInterface;
use SWP\Component\Rule\Applicator\RuleApplicatorInterface;
use SWP\Component\Rule\Model\RuleSubjectInterface;
use SWP\Component\Rule\Model\RuleInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

final class ArticleRuleApplicator implements RuleApplicatorInterface
{
    /**
     * @var RouteProviderInterface
     */
    private $routeProvider;

    /**
     * @var LoggerInterface
     */
    private $logger;

    /**
     * @var array
     */
    private $supportedKeys = ['route', 'templateName'];
```

(continues on next page)

(continued from previous page)

```php

    /**
     * ArticleRuleApplicator constructor.
     *
     * @param RouteProviderInterface $routeProvider
     * @param LoggerInterface        $logger
     */
    public function __construct(RouteProviderInterface $routeProvider, LoggerInterface $logger)
    {
        $this->routeProvider = $routeProvider;
        $this->logger = $logger;
    }

    /**
     * {@inheritdoc}
     */
    public function apply(RuleInterface $rule, RuleSubjectInterface $subject)
    {
        $configuration = $this->validateRuleConfiguration($rule->getConfiguration());

        if (!$this->isAllowedType($subject) || empty($configuration)) {
            return;
        }

        /* @var ArticleInterface $subject */
        if (isset($configuration[$this->supportedKeys[0]])) {
            $route = $this->routeProvider->getOneById($configuration[$this->supportedKeys[0]]);

            if (null === $route) {
                $this->logger->warning('Route not found! Make sure the rule defines an existing route!');

                return;
            }

            $subject->setRoute($route);
        }

        $subject->setTemplateName($configuration[$this->supportedKeys[1]]);

        $this->logger->info(sprintf(
            'Configuration: "%s" for "%s" rule has been applied!',
            json_encode($configuration),
            $rule->getExpression()
        ));
    }

    /**
     * {@inheritdoc}
     */
    public function isSupported(RuleSubjectInterface $subject)
    {
        return $subject instanceof ArticleInterface && 'article' === $subject->getSubjectType();
    }
```

(continues on next page)

(continued from previous page)

```

    private function validateRuleConfiguration(array $configuration)
    {
        $resolver = new OptionsResolver();
        $this->configureOptions($resolver);

        try {
            return $resolver->resolve($configuration);
        } catch (\Exception $e) {
            $this->logger->warning($e->getMessage());
        }

        return [];
    }

    private function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([$this->supportedKeys[1] => null]);
        $resolver->setDefined($this->supportedKeys[0]);
    }

    private function isAllowedType(RuleSubjectInterface $subject)
    {
        if (!$subject instanceof ArticleInterface) {
            $this->logger->warning(sprintf(
                '"%s" is not supported by "%s" rule applicator!',
                is_object($subject) ? get_class($subject) : gettype($subject),
                get_class($this)
            ));

            return false;
        }

        return true;
    }
}
```

### Configuring the Rule Applicator

To register your new rule applicator, simply add a definition to your services file and tag it with a special name: `applicator.rule_applicator`, it will be automatically added to the chain of rule applicators:

```yaml
# Resources/config/services.yml
acme_my_custom_rule_applicator:
    class: 'Acme\DemoBundle\Applicator\ArticleRuleApplicator'
    arguments:
        - '@swp.provider.route'
        - '@logger'
    tags:
        - { name: applicator.rule_applicator }
```

### How to Create and Enable Custom Rule Entity

In some cases you would want to extend the default `Rule` model to add some extra properties etc. To do this you need to create a custom class which extends the default one.

```
<?php
// ..
namespace Acme\DemoBundle\Entity;

use SWP\Component\Rule\Model\Rule as BaseRule;

class Rule extends BaseRule
{
    protected $something;

    public function getSomething()
    {
        return $this->something;
    }

    public function setSomething($something)
    {
        $this->something = $something;
    }
}
```

Add class's mapping file:

```
# Acme\DemoBundle\Resources\config\doctrine\Rule.orm.yml
Acme\DemoBundle\Entity\Rule:
    type: entity
    table: custom_rule
    fields:
        something:
            type: string
```

The newly created class needs to be now added to the bundle's configuration:

```
# app/config/config.yml
swp_rule:
    persistence:
        orm:
            # ..
            classes:
                rule:
                    model: Acme\DemoBundle\Entity\Rule
```

That's it, a newly created class will be used instead.

**Note:** You could also provide your own implementation for Rule Factory and Rule Repository. To find out more about it check /bundles/SWPStorageBundle/register_storage

### How rules are processed?

You can create Event Subscriber which can listen on whatever event is defined. If the event is dispatched, the subscriber should run Rule Processor which will process all rules.

Example subscriber:

```php
<?php

namespace SWP\Bundle\ContentBundle\EventListener;

use SWP\Bundle\ContentBundle\ArticleEvents;
use SWP\Bundle\ContentBundle\Event\ArticleEvent;
use SWP\Component\Rule\Processor\RuleProcessorInterface;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;

class ProcessArticleRulesSubscriber implements EventSubscriberInterface
{
    /**
     * @var RuleProcessorInterface
     */
    private $ruleProcessor;

    /**
     * ProcessArticleRulesSubscriber constructor.
     *
     * @param RuleProcessorInterface $ruleProcessor
     */
    public function __construct(RuleProcessorInterface $ruleProcessor)
    {
        $this->ruleProcessor = $ruleProcessor;
    }

    /**
     * {@inheritdoc}
     */
    public static function getSubscribedEvents()
    {
        return [
            ArticleEvents::PRE_CREATE => 'processRules',
        ];
    }

    /**
     * @param ArticleEvent $event
     */
    public function processRules(ArticleEvent $event)
    {
        $this->ruleProcessor->process($event->getArticle());
    }
}
```

### Enabling separate Monolog channel

It is possible to enable a separate Monolog channel to which all Rule Bundle related logs will be forwarded. An example log entry might be logged when the rule can not be evaluated properly etc. You could have then a separate log file for (which will log everything related to that bundle) which will be saved under the directory `app/logs/` in your application and will be named, for example: `swp_rule_<env>.log`. By default, a separate channel is not enabled. You can enable it by adding an extra channel in your Monolog settings (in one of your configuration files):

```yaml
# app/config/config.yml
monolog:
    handlers:
        swp_rule:
            level:    debug
            type:     stream
            path:     '%kernel.logs_dir%/swp_rule_%kernel.environment%.log'
            channels: swp_rule
```

For more details see the Monolog documentation.

### 4.6.4 Models

#### Rule

Rule is to check if your "rule aware" objects are allowed to be processed and if some rule's configuration can be applied to it.

A rule is configured using the `configuration` attribute which is an array serialized into database. Your custom Rule Applicator should define which configuration key-value pair should be applied. For example, you could configure the *route* key to define which route should be applied to an object if the given rule evaluates to true. You could also apply `templateName` or any other keys.

See *Usage* section for more details.

#### RuleSubjectInterface

To make use of the Rule bundle and allow to apply rule to an object, the entity must be "rule aware", it means that "subject" class needs to implement `SWP\Component\Rule\Model\RuleSubjectInterface` interface.

If you make your custom entity rule aware, the Rule Processor will automatically process all rules for given object.

By implementing `SWP\Component\Rule\Model\RuleSubjectInterface` interface, your object will have to define the following method:

- `getSubjectType()` - should return the name of current object, for example: `article`.

Rule Evaluator is using this method to evaluate rule on an object.

If the `getSubjectType()` returns `article` the rule expression should be related to this object using `article` prefix. For example: `article.getSomething('something') > 1`.

### 4.6.5 Configuration Reference

The SWPRuleBundle can be configured under the `swp_rule` key in your configuration file. This section describes the whole bundle's configuration.

#### Full Default Configuration

```yaml
# app/config/config.yml
swp_rule:
    persistence:
        orm:
            enabled: true
            classes:
                rule:
                    model: SWP\Component\Rule\Model\Rule
                    repository: SWP\Bundle\RuleBundle\Doctrine\ORM\RuleRepository
                    factory: SWP\Bundle\StorageBundle\Factory\Factory
                    object_manager_name: ~
```

### persistence

#### `persistence`

##### `orm`

```yaml
# app/config/config.yml
swp_rule:
    # ..
    persistence:
        orm:
            enabled: true
```

###### `enabled`

**type**: `boolean` **default**: `false`

If `true`, ORM is enabled in the service container.

ORM can be enabled by multiple ways such as:

```yaml
orm: ~ # use default configuration
# or
orm: true # straight way
# or
orm:
    enabled: true ... # or any other option under 'orm'
```

###### `classes`

```yaml
# app/config/config.yml
swp_rule:
    # ..
    persistence:
        orm:
            # ..
            classes:
                rule:
                    model: SWP\Component\Rule\Model\Rule
```

(continues on next page)

(continued from previous page)

```
                        repository: SWP\Bundle\RuleBundle\Doctrine\ORM\RuleRepository
                        factory: SWP\Bundle\StorageBundle\Factory\Factory
                        object_manager_name: ~
```

**type**: `string` **default**: `null`

The name of the object manager. If set to null it defaults to *default*. If Doctrine ORM persistence backend is enabled it will register `swp.object_manager.rule` service which is an alias for `doctrine.orm.default_entity_manager`.

#### `rule.model`

**type**: `string` **default**: `SWP\Component\Rule\Model\Rule`

The FQCN of the Rule model class which is of type `SWP\Component\Rule\Model\RuleInterface`.

#### `rule.factory`

**type**: `string` **default**: `SWP\Bundle\StorageBundle\Factory\Factory`

The FQCN of the Rule Factory class.

#### `rule.repository`

**type**: `string` **default**: `SWP\Bundle\RuleBundle\Doctrine\ORM\RuleRepository`

The FQCN of the Rule Repository class.

#### `rule.object_manager_name`

**type**: `string` **default**: `null`

The name of the object manager. If set to null it defaults to *default*. If Doctrine ORM persistence backend is enabled it will register `swp.object_manager.rule` service which is an alias for `doctrine.orm.default_entity_manager`.

## 4.7 ContentListBundle

This bundle gives you an ability to create powerful content lists where lists' items can be of any type. For example, imagine a list which contains a lot of articles which you can add, reorder, drag and drop etc.

***The Content List Component*, which is used by this bundle, provides classes to build powerful** and flexible content lists.

### 4.7.1 Prerequisites

This version of the bundle requires Symfony >= 2.8 and PHP version >=7.0

### 4.7.2 Installation

#### Install the Bundle with Composer

In your project directory execute the following command to download the latest stable version:

```
composer require swp/content-list-bundle
```

This command requires you to have Composer installed globally. If it's not installed globally, download the `.phar` file locally as explained in Composer documentation.

#### Enable the bundle and its dependencies

Enable the bundle by adding the following lines in the `app/AppKernel.php` file:

```php
// app/AppKernel.php

// ...
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            new SWP\Bundle\ContentListBundle\SWPStorageBundle(),
            new Stof\DoctrineExtensionsBundle\StofDoctrineExtensionsBundle(),
            // ...
            new SWP\Bundle\ContentListBundle\SWPContentListBundle(),
        );

        // ...
    }

    // ...
}
```

**Note:** All dependencies will be installed automatically. You will just need to configure the respective bundles if needed.

#### Configure the bundle

- *YAML*

```yaml
# app/config/config.yml
swp_content_list:
    persistence:
        orm:
            # if true, ORM is enabled as a persistence backend
            enabled: true
```

**Note:** By default this bundle supports only Doctrine ORM as a persistence backend.

**Note:** If this bundle is used together with *ContentBundle*, configuration will be automatically pre-pended and enabled, so there is no need to configure it in your config file.

Configure Doctrine extensions which are used by this bundle:

```yaml
# app/config/config.yml
stof_doctrine_extensions:
    orm:
        default:
            timestampable: true
            softdeleteable: true
            loggable: true
```

Using your custom list item content class:

- *YAML*

```yaml
# app/config/config.yml
swp_content_list:
    persistence:
        orm:
            # if true, ORM is enabled as a persistence backend
            enabled: true
            classes:
                # ..
                list_content:
                    model: Acme\MyBundle\Entity\Post
```

**Note:** `Acme\MyBundle\Entity\Post` must implement `SWP\Component\ContentList\Model\ListContentInterf`
interface.

### Update Database Schema

Run the following command:

```
$ php app/console doctrine:schema:update --force
```

That's it, the bundle is configured properly now!

## 4.7.3 Usage

### Creating new content lists

Here is an example on how to create a new content list:

```php
// src/AppBundle/Controller/MyController.php

use SWP\Component\ContentList\Model\ContentListInterface;
// ...
public function createAction()
{
```

(continues on next page)

(continued from previous page)

```php
    $repository = $this->container->get('swp.repository.content_list');

    /* @var ContentListInterface $contentList */
    $contentList = $this->get('swp.factory.content_list')->create();
    $contentList->setName('my content list');
    $contentList->setDescription('description');
    $contentList->setLimit(10);
    $contentList->setType(ContentListInterface::TYPE_AUTOMATIC);
    // ...

    $repository->add($contentList);

    // ...
}
```

### Adding new items to content list

```php
// src/AppBundle/Controller/MyController.php

use SWP\Component\ContentList\Model\ContentListInterface;
use SWP\Component\ContentList\Model\ContentListItemInterface;
use Acme\AppBundle\Entity\Article;
// ...
public function createAction()
{
    $repository = $this->container->get('swp.repository.content_list');

    /* @var ContentListInterface $contentList */
    $contentList = $this->get('swp.factory.content_list')->create();
    $contentList->setName('my content list');
    $contentList->setDescription('description');
    $contentList->setLimit(10);
    $contentList->setType(ContentListInterface::TYPE_AUTOMATIC);
    // ...

    /* @var ContentListItemInterface $contentListItem */
    $contentListItem = $this->get('swp.factory.content_list_item')->create();
    $contentListItem->setPosition(6);
    $contentListItem->setContent(new Article());
    $contentListItem->setSticky(true);
    $contentList->addItem($contentListItem);

    $repository->add($contentList);

    // ...
}
```

**Note:** `Article` class must implement `SWP\Component\ContentList\Model\ListContentInterface`.

### Deleting content lists

```php
// src/AppBundle/Controller/MyController.php

use SWP\Component\ContentList\Model\ContentListInterface;
use Acme\AppBundle\Entity\Article;
// ...
public function deleteAction($id)
{
    $repository = $this->container->get('swp.repository.content_list');

    /* @var ContentListInterface $contentList */
    $contentList = $repository->findOneBy(['id' => $id]);
    // ...

    $repository->remove($contentList);

    // ...
}
```

### Deleting content lists items

```php
// src/AppBundle/Controller/MyController.php

use SWP\Component\ContentList\Model\ContentListItemInterface;
use Acme\AppBundle\Entity\Article;
// ...
public function deleteAction($id)
{
    $repository = $this->container->get('swp.repository.content_list_item');

    /* @var ContentListItemInterface $contentListItem */
    $contentListItem = $repository->findOneBy(['id' => $id]);
    // ...

    $repository->remove($contentListItem);

    // ...
}
```

## Forms

### Content list type selector

If you want to use content list type selector inside your custom form you can do it by adding `SWP\Bundle\ContentListBundle\Form\Type\ContentListTypeSelectorType` form field type to your form:

```php
namespace Acme\AppBundle\Form\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;
```

(continues on next page)

(continued from previous page)

```php
use Symfony\Component\Validator\Constraints\NotBlank;


class MyListType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('name', TextType::class, [
            'constraints' => [
                new NotBlank(),
            ],
            'description' => 'List name',
        ])
        ->add('type', ContentListTypeSelectorType::class, [
            'constraints' => [
                new NotBlank(),
            ],
            'description' => 'List type',
        ])
    }
}
```

Alternatively, you could also extend from the default `SWP\Bundle\ContentListBundle\Form\Type\ContentListType` class if you would only add more fields on top of the existing form.

**Note:** For more details on how to register custom factory, repository, object manager, forms using custom classes see SWPStorageBundle *Usage* section.

## Getting content lists from repository

To get single or all content lists from the repository you can use default Doctrine ORM `SWP\Bundle\ContentListBundle\Doctrine\ORM\ContentListRepository` repository. It has the same methods as Doctrine ORM `EntityRepository`, but it contains an extra method to get content lists by its type:

- `findByType(string $type): array` - it gets many content lists by its type, type can be either: automatic or manual.

```php
// src/AppBundle/Controller/MyController.php

use SWP\Component\ContentList\Model\ContentListInterface;
// ...
public function getAction()
{
    $repository = $this->container->get('swp.repository.content_list');

    $lists = $repository->findByType(ContentListInterface::TYPE_AUTOMATIC);
    var_dump($lists);die;
    // ...
}
```

**Note:** This repository is automatically registered as a service for you and can be accessible under service id: `swp.`

`repository.content_list` in Symfony container.

#### Getting content lists items from repository

To get content list items you can use default repository which is registered as a service under the `swp.repository.content_list_item` key in Symfony container. It extends default Doctrine ORM EntityRepository.

### 4.7.4 Models

#### ContentList

ContentList model is a main class which defines default list properties. This includes list's items, name, description and more:

- list can be limited to display certain number of items
- list can have cache life time defined in seconds (which is useful, for example, if you want to cache list for some time when rendering it on frontend)
- **list can be of one of the types:**
    - automatic
    - manual

Automatic list is meant to be created manually but the items in that list should not be draggable and droppable. It just a flat list that you can add items and simply render list with it's items. Whatever content you want to place in this list you should be able to do it. An example can be that if some part of your business logic is able to decide where the article should go, if it matches some criteria, you can use that logic and add an article to the list automatically - this list will be then called automatic list.

As in the case of Automatic lists, the Manual list is meant to be created manually but you should be able to add, remove, drag and drop, sort items in this list manually by simply linking items to lists.

#### ContentListItem

ContentListItem model represents an item which can be placed inside content list. It has a reference to content list, position at which it should be placed inside the list and a content. Content can be of any type, but it should implement `SWP\Component\ContentList\Model\ListContentInterface`.

#### ListContentInterface

This interface should be implemented by your class if you want objects of that type to be a content of the list. For example, if you have `Article` class in your project and you want to make objects of this type to be a content of list item, it needs to implement `SWP\Component\ContentList\Model\ListContentInterface`.

### 4.7.5 Configuration Reference

The SWPContentListBundle can be configured under the `swp_content_list` key in your configuration file. This section describes the whole bundle's configuration.

### Full Default Configuration

```yaml
# app/config/config.yml
swp_content_list:
    persistence:
        orm:
            enabled: true
            classes:
                content_list:
                    model: SWP\Component\ContentList\Model\ContentList
                    interface: SWP\Component\ContentList\Model\ContentListInterface
                    repository: SWP\Bundle\ContentListBundle\Doctrine\ORM\ContentListRepository
                    factory: SWP\Bundle\StorageBundle\Factory\Factory
                    object_manager_name: ~
                content_list_item:
                    model: SWP\Component\ContentList\Model\ContentListItem
                    interface: SWP\Component\ContentList\Model\ContentListItemInterface
                    repository: SWP\Bundle\StorageBundle\Doctrine\ORM\EntityRepository
                    factory: SWP\Bundle\StorageBundle\Factory\Factory
                    object_manager_name: ~
                list_content:
                    model: ~
                    interface: SWP\Component\ContentList\Model\ListContentInterface
                    repository: SWP\Bundle\StorageBundle\Doctrine\ORM\EntityRepository
                    factory: SWP\Bundle\StorageBundle\Factory\Factory
                    object_manager_name: ~
```

### persistence

#### `persistence`

##### `orm`

```yaml
# app/config/config.yml
swp_content_list:
    # ..
    persistence:
        orm:
            enabled: true
```

##### `enabled`

**type**: `boolean` **default**: `false`

If `true`, ORM is enabled in the service container.

ORM can be enabled by multiple ways such as:

```yaml
orm: ~ # use default configuration
# or
orm: true # straight way
```

(continues on next page)

(continued from previous page)

```yaml
# or
orm:
    enabled: true ... # or any other option under 'orm'
```

### classes

```yaml
# app/config/config.yml
swp_content_list:
    # ..
    persistence:
        orm:
            # ..
            classes:
                content_list:
                    model: SWP\Component\ContentList\Model\ContentList
                    interface: SWP\Component\ContentList\Model\ContentListInterface
                    repository: SWP\Bundle\ContentListBundle\Doctrine\ORM\ContentListRepository
                    factory: SWP\Bundle\StorageBundle\Factory\Factory
                    object_manager_name: ~
                content_list_item:
                    model: SWP\Component\ContentList\Model\ContentListItem
                    interface: SWP\Component\ContentList\Model\ContentListItemInterface
                    repository: SWP\Bundle\StorageBundle\Doctrine\ORM\EntityRepository
                    factory: SWP\Bundle\StorageBundle\Factory\Factory
                    object_manager_name: ~
                list_content:
                    model: ~
                    interface: SWP\Component\ContentList\Model\ListContentInterface
                    repository: SWP\Bundle\StorageBundle\Doctrine\ORM\EntityRepository
                    factory: SWP\Bundle\StorageBundle\Factory\Factory
                    object_manager_name: ~
```

**type**: `string` **default**: `null`

The name of the object manager. If set to null it defaults to *default*. If Doctrine ORM persistence backend is enabled it will register `swp.object_manager.content_list` service which is an alias for `doctrine.orm.default_entity_manager`.

### content_list.model

**type**: `string` **default**: `SWP\Component\ContentList\Model\ContentList`

The FQCN of the ContentList model class which is of type `SWP\Component\ContentList\Model\ContentListInterface`

### content_list.interface

**type**: `string` **default**: `SWP\Component\ContentList\Model\ContentListInterface`

The FQCN of your custom interface which is used by your model class.

#### `content_list.factory`

**type**: `string` **default**: `SWP\Bundle\StorageBundle\Factory\Factory`

The FQCN of the ContentList Factory class.

#### `content_list.repository`

**type**: `string` **default**: `SWP\Bundle\StorageBundle\Doctrine\ORM\EntityRepository`

The FQCN of the ContentList Repository class.

#### `content_list.object_manager_name`

**type**: `string` **default**: `null`

The name of the object manager. If set to null it defaults to *default*. If Doctrine ORM persistence backend is enabled it will register `swp.object_manager.content_list` service which is an alias for `doctrine.orm.default_entity_manager`.

#### `content_list_item.model`

**type**: `string` **default**: `SWP\Component\ContentList\Model\ContentListItem`

The FQCN of the ContentListItem model class which is of type `SWP\Component\ContentList\Model\ContentListItemIr`

#### `content_list_item.interface`

**type**: `string` **default**: `SWP\Component\ContentList\Model\ContentListItemInterface`

The FQCN of your custom interface which is used by your model class.

#### `content_list_item.factory`

**type**: `string` **default**: `SWP\Bundle\StorageBundle\Factory\Factory`

The FQCN of the ContentListItem Factory class.

#### `content_list_item.repository`

**type**: `string` **default**: `SWP\Bundle\StorageBundle\Doctrine\ORM\EntityRepository`

The FQCN of the ContentListItem Repository class.

#### `content_list_item.object_manager_name`

**type**: `string` **default**: `null`

The name of the object manager. If set to null it defaults to *default*. If Doctrine ORM persistence backend is enabled it will register `swp.object_manager.content_list_item` service which is an alias for `doctrine.orm.default_entity_manager`.

#### `list_content.model`

**type**: `string` **default**: `null`

The FQCN of the model class which must be of type `SWP\Component\ContentList\Model\ContentListInterface`. This is the content of the list item. You can use your custom classes here so for example, `ACME\DemoBundle\Entity\Post` could be your content.

#### `list_content.interface`

**type**: `string` **default**: `SWP\Component\ContentList\Model\ListContentInterface`

The FQCN of your custom interface which is used by your model class.

#### `list_content.factory`

**type**: `string` **default**: `SWP\Bundle\StorageBundle\Factory\Factory`

The FQCN of the List Item's content Factory class.

#### `list_content.repository`

**type**: `string` **default**: `SWP\Bundle\StorageBundle\Doctrine\ORM\EntityRepository`

The FQCN of the List Item's content Repository class.

#### `list_content.object_manager_name`

**type**: `string` **default**: `null`

The name of the object manager. If set to null it defaults to *default*. If Doctrine ORM persistence backend is enabled it will register `swp.object_manager.content_list` service which is an alias for `doctrine.orm.default_entity_manager`.

## 4.8 FacebookInstantArticlesBundle

This bundle integrates Facebook Instant Articles PHP SDK with Symfony application.

### 4.8.1 Authentication

To work with Facebook Instant Articles REST API you need to have valid access token. Bundle provides controller for `authentication url generation` and `handling authorization callback requests from Facebook`.

Authorization procedure requires providing Facebook Application and Page (token will be generated for that combination) ID's. Bundle provide entities for both (Application and Page) and will look for matching rows in database.

**Note:** Authentication controller checks if provided page and application are in your storage (database). But bundle doesn't provide controllers for adding them (there are only pre-configured factories and repositories) - You need to implement it manually in your application.

#### Authentication flow

Assuming that in your database you have Application with id `123456789` and Page with id `987654321` (and both it exists on Facebook platform), You need to call this url `(route: swp_fbia_authorize): /facebook/instantarticles/authorize/123456789/987654321`

In response You will be redirected to Facebook where You will need allow for all required permissions.

After that Facebook will redirect You again to application where (in background - provided by Facebook `code` will be exchanged for access token and that access) you will get JSON response with `pageId` and `accessToken` (never expiring access token).

### 4.8.2 Article Parsing

#### Instant Article view

For Instant Article rendering this template file is used `/platforms/facebook_instant_article.html.twig`. Basic version is provided by Publisher but you can (should) override it with your theme.

To control how exactly look pushed to Facebook Instant Articles API article you can use preview url.

`/facebook/instantarticles/preview/{articleId}` - shows how article template was converted into FBIA article. Parser is removing all not recognized tags. Read more about allowed rules here: https://developers.facebook.com/docs/instant-articles/sdk/transformer.

By default we use standard SDK rules: https://github.com/facebook/facebook-instant-articles-sdk-php/blob/master/src/Facebook/InstantArticles/Parser/instant-articles-rules.json

## 4.9 SettingsBundle

This bundle provides tools to define settings and save user changes.

### 4.9.1 Installation

#### Install the Bundle with Composer

In your project directory execute the following command to download the latest stable version:

```
composer require swp/settings-bundle
```

This command requires you to have Composer installed globally. If it's not installed globally, download the `.phar` file locally as explained in Composer documentation.

### Enable the bundle and its dependencies

Enable the bundle and its dependencies by adding the following lines in the `app/AppKernel.php` file:

```php
// app/AppKernel.php

// ...
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            // ...

            new SWP\Bundle\SettingsBundle\SWPSettingsBundle(),
        );

        // ...
    }

    // ...
}
```

Enable the bundle in application configuration (`config.yml`):

```yaml
swp_settings:
    persistence:
        orm:
            enabled: true
```

**Note:** All dependencies will be installed automatically. You will just need to configure the respective bundles if needed. To see all required dependencies - check bundle `composer.json` file.

That's it, the bundle is configured properly now!

## 4.9.2 Settings definitions

In your application or bundle configuration add settings definitions:

```yaml
swp_settings:
    settings:
        registration_confirmation.template:
            value: "example value"
            scope: user
            type: string
```

Minimal definition looks like that:

```yaml
swp_settings:
    settings:
        registration_confirmation.template: ~
```

**Note:**

**Default values:**

- value: `null`
- scope: `global` (possible options: `global`, `user`)
- type: `string` (possible options: `string`, `array`)

### Settings scopes

Scope defines level for custom changes. If setting have scope `user` then every user will have his own value for this setting.

### Settings value types

Setting value can be `string` or `array` (it will be saved as json).

Example with array as value:

```yaml
parameters:
    array_value:
        a: 1
        b: 2

swp_settings:
    settings:
        custom_setting_1:
            value: "%array_value%"
        custom_setting_2:
            value: '{"a":1, "b": 2}'
```

## 4.9.3 Usage

### REST API

### List settings

You can list all settings (with values loaded for scope) by API.

`GET /api/{version}/settings/`

### Update settings

You can update settings with API call:

```
curl -X "PATCH" -d "settings[name]=setting_name&settings[value]=setting_value" -H
↪"Content-type:\ application/x-www-form-urlencoded" /api/v1/settings
```

### In code

#### Get all settings

```
$settingsManager = $this->get('swp_settings.manager.settings');
$settings = $settingsManager->all();
```

#### Get single setting

```
$settingsManager = $this->get('swp_settings.manager.settings');
$setting = $settingsManager->get('setting_name');
```

#### Set setting value

```
$settingsManager = $this->get('swp_settings.manager.settings');
$setting = $settingsManager->set('setting_name', 'setting value');
```

For details check `SWP\Bundle\SettingsBundle\Manager\SettingsManager` class.

### 4.9.4 Scope context

**Hint:** `Scope` defines level for custom changes. If setting have scope `user` then every user will have his own value for this setting.

`ScopeContext` class defines available scopes (in `getScopes()` method). Every setting must have scope and can have setting owner. Scope Context collects owners for defined scopes from current application state.

```
...
$scopeContext = new \SWP\Bundle\SettingsBundle\Context\ScopeContext();
...

// Set user in scope
$scopeContext->setScopeOwner(ScopeContextInterface::SCOPE_USER, $user);
```

**Note:** Owner object set to scope context must implement `SettingsOwnerInterface`. Scope owner allows for system to fill settings with correct custom set values kept in storage.

Bundle already register event subscriber responsible for setting currently logged user in scope context - `SWP\Bundle\SettingsBundle\EventSubscriber\ScopeContextSubscriber`.

## 4.10 WebhookBundle

This bundle provides tools to build a webhook's system in Symfony application.

This bundle uses the Doctrine ORM persistence backend.

## 4.10.1 Installation

### Install the Bundle with Composer

In your project directory execute the following command to download the latest stable version:

```
composer require swp/webhook-bundle
```

This command requires you to have Composer installed globally. If it's not installed globally, download the `.phar` file locally as explained in Composer documentation.

### Enable the bundle and its dependencies

This bundle require StorageBundle to be installed and configured. Enable the bundle by adding the following lines in the `app/AppKernel.php` file:

```php
// app/AppKernel.php

// ...
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(

            new SWP\Bundle\StorageBundle\SWPStorageBundle()
            // ...
            new SWP\Bundle\WebhookBundle\SWPWebhookBundle(),
        );

        // ...
    }

    // ...
}
```

**Note:** All dependencies will be installed automatically.

That's it, the bundle is configured properly now!

## 4.10.2 Usage

### Webhooks Management

Bundle provides abstract Controller class for CRUD webhook actions. It cane be used in Your own controllers.

### Sending Webhooks

Bundle allows to create new webhooks, and provides repository for searching them by event name. Sending events need to be implemented in end application. We recommend to do that with dispatcher and event listeners/subscribers.

Example implementation can be found in `SWP\Bundle\CoreBundle\EventSubscriber\WebhookEventsSubscriber` class (in Superdesk Publisher project).

## 4.11 OutputChannelBundle

This bundle provides tools to build output channels abstraction in Symfony application.

This bundle uses the Doctrine ORM persistence backend.

### 4.11.1 Installation

#### Install the Bundle with Composer

In your project directory execute the following command to download the latest stable version:

```
composer require swp/output-channel-bundle
```

This command requires you to have Composer installed globally. If it's not installed globally, download the `.phar` file locally as explained in Composer documentation.

#### Enable the bundle and its dependencies

This bundle requires StorageBundle to be installed and configured. Enable the bundle by adding the following lines in the `app/AppKernel.php` file:

```
// app/AppKernel.php

// ...
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(

            new SWP\Bundle\StorageBundle\SWPStorageBundle()
            // ...
            new SWP\Bundle\OutputChannelBundle\SWPOutputChannelBundle(),
        );

        // ...
    }

    // ...
}
```

**Note:** All dependencies will be installed automatically.

That's it, the bundle is configured properly now!

### 4.11.2 Usage

This bundle provide the JMS Serializer mapping files, Doctrine ORM mapping files, form types and the basic configuration.

#### OutputChannelType

This bundle comes with the `OutputChannelType` to build forms using Symfony.

#### WordpressOutputChannelConfigType

This bundle contains the `WordpressOutputChannelConfigType` form type and defines the configuration for the Wordpress output channel.

## 4.12 Paywall Bundle

This bundle provides tools to build paywall solution in Symfony application.

### 4.12.1 Installation

#### Install the Bundle with Composer

In your project directory execute the following command to download the latest stable version:

```
composer require swp/paywall-bundle
```

This command requires you to have Composer installed globally. If it's not installed globally, download the `.phar` file locally as explained in Composer documentation.

#### Enable the bundle and its dependencies

Enable the bundle by adding the following lines in the `app/AppKernel.php` file:

```php
// app/AppKernel.php

// ...
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            // ...
            new SWP\Bundle\PaywallBundle\SWPPaywallBundle(),
        );

        // ...
    }

    // ...
}
```

Import config to into your `config.yml` file:

```yaml
imports:
    - { resource: "@SWPPaywallBundle/Resources/config/app/config.yml" }
```

**Note:** All dependencies will be installed automatically.

That's it, the bundle is configured properly now!

## 4.12.2 Usage

This bundle provide the services which help to interact with PaymentsHub. It also allows you to create your custom implementations to interact with different/custom subscriptions systems.

### How to add a new adapter

Adapters are used to retrieve the subscriptions data from the external subscription system. It is possible to implement your custom adapter and use it to fetch the subscriptions data from the 3rd party subscription system.

1. Create your custom adapter class which uses GuzzleHttp client

```php
// src/AcmeBundle/Adapter/CustomAdapter.php

namespace AcmeBundle\Adapter;

use GuzzleHttp\ClientInterface;
use GuzzleHttp\Exception\RequestException;
use Psr\Http\Message\ResponseInterface;
use SWP\Component\Paywall\Factory\SubscriptionFactoryInterface;
use SWP\Component\Paywall\Model\SubscriberInterface;
use SWP\Component\Paywall\Model\SubscriptionInterface;

// ...
final class CustomAdapter implements PaywallAdapterInterface
{
    public function __construct(array $config, SubscriptionFactoryInterface
↪$subscriptionFactory, ClientInterface $client)
    {
        $this->config = $config;
        $this->subscriptionFactory = $subscriptionFactory;
        $this->client = $client;
    }

    public function getSubscriptions(SubscriberInterface $subscriber, array $filters␣
↪= []): array
    {
        // custom logic here to get subscriptions
        // ...

        $subscription = $this->subscriptionFactory->create();

        // ...
    }

    public function getSubscription(SubscriberInterface $subscriber, array $filters =␣
↪[]): ?SubscriptionInterface
    {
        // custom logic here to get a single subscription
        // ...
```

(continues on next page)

(continued from previous page)

```
    }

    // ...
}
```

2. Register your adapter as a service

```yaml
# services.yml
AcmeBundle\Adapter\CustomAdapter:
    arguments:
        -
            serverUrl: "%env(resolve:PAYWALL_SERVER_URL)%"
            credentials:
                username: "%env(resolve:PAYWALL_SERVER_USERNAME)%"
                password: "%env(resolve:PAYWALL_SERVER_PASSWORD)%"
        - '@SWP\Component\Paywall\Factory\SubscriptionFactory'
        - '@GuzzleHttp\Client'
```

3. Enabled newly created adapter in bundle’s config

```yaml
# config.yml
swp_paywall:
    adapter: AcmeBundle\Adapter\CustomAdapter
```

CHAPTER 5

# Developer Guide

## 5.1 Contributing

**Note:** This section is based on Symfony2 documentation.

### 5.1.1 Contributing Code

#### Reporting a Bug

Whenever you find a bug, we kindly ask you to report it. It helps us make a better Superdesk Publisher.

**Caution:** If you think you've found a security issue, please use the special *procedure* instead.

Before submitting a bug:

- Double-check the official *documentation* to see if you're not misusing the project;

- Ask for assistance on StackOverflow or on the official bug tracker if you're not sure if your issue is really a bug.

If your problem definitely looks like a bug, report it using the bug tracker and follow some basic rules:

- Use the title field to clearly describe the issue;
- Describe the steps needed to reproduce the bug with short code examples (providing a unit test that illustrates the bug is best);
- Give as much detail as possible about your environment (OS, PHP version, Superdesk Publisher version, enabled extensions, . . . );
- *(optional)* Attach a *patch*.

### Submitting a Patch

Patches are the best way to provide a bug fix or to propose enhancements to Superdesk Publisher.

### Step 1: Setup your Environment

### Install the Software Stack

Before working on Superdesk Publisher, setup a friendly environment with the following software:

- Git;
- PHP version 5.5.9 or above;
- PHPUnit 4.2 or above.
- PostgreSQL

### Configure Git

Set up your user information with your real name and a working email address:

```
git config --global user.name "Your Name"
git config --global user.email you@example.com
```

**Tip:** If you are new to Git, you are highly recommended to read the excellent and free ProGit book.

**Tip:** If your IDE creates configuration files inside the project's directory, you can use a global `.gitignore` file (for all projects) or a `.git/info/exclude` file (per project) to ignore them. See GitHub's documentation.

**Tip:** Windows users: when installing Git, the installer will ask what to do with line endings, and suggests replacing all LF with CRLF. This is the wrong setting if you wish to contribute to Superdesk Publisher! Selecting the as-is method is your best choice, as Git will convert your line feeds to the ones in the repository. If you have already installed Git, you can check the value of this setting by typing:

```
git config core.autocrlf
```

This will return either "false", "input" or "true"; "true" and "false" being the wrong values. Change it to "input" by typing:

```
git config --global core.autocrlf input
```

Replace –global by –local if you want to set it only for the active repository

### Get the Superdesk Publisher Source Code

Get the Superdesk Publisher source code:

- Create a GitHub account and sign in;
- Fork the Superdesk Publisher repository (click on the "Fork" button);
- After the "forking action" has completed, clone your fork locally (this will create a `web-publisher` directory):

```
git clone git@github.com:USERNAME/web-publisher.git
```

- Add the upstream repository as a remote:

```
cd web-publisher
git remote add upstream git://github.com/superdesk/web-publisher.git
```

### Check that the current Tests Pass

Now that Superdesk Publisher is installed, check that all tests pass for your environment as explained in the dedicated *document*.

## Step 2: Work on your Patch

### The License

Before you start, you must know that all the patches you are going to submit must be released under the GNU AGPLv3 license, unless explicitly specified in your commits.

### Create a Topic Branch

Each time you want to work on a patch for a bug or an enhancement, create a topic branch:

```
git checkout -b BRANCH_NAME master
```

**Tip:** Use a descriptive name for your branch, containing the ticket number from the bug tracker.

The above checkout commands automatically switch the code to the newly created branch (check the branch you are working on with `git branch`).

### Work on your Patch

Work on the code as much as you want and commit as much as you want; but keep in mind the following:

- Read about the Superdesk Publisher *conventions* and follow the coding *standards* (use `git diff --check` to check for trailing spaces – also read the tip below);
- Add unit tests to prove that the bug is fixed or that the new feature actually works;
- Try hard to not break backward compatibility (if you must do so, try to provide a compatibility layer to support the old way) – patches that break backward compatibility have less chance to be merged;
- Do atomic and logically separate commits (use the power of `git rebase` to have a clean and logical history);
- Never fix coding standards in some existing code as it makes the code review more difficult;
- Write good commit messages (see the tip below).

**Tip:** When submitting pull requests, StyleCI checks your code for common typos and verifies that you are using the PHP coding standards as defined in PSR-1 and PSR-2.

A status is posted below the pull request description with a summary of any problems it detects or any Travis CI build failures.

**Tip:** A good commit message is composed of a summary (the first line), optionally followed by a blank line and a more detailed description. The summary should start with the Component you are working on in square brackets (`[MultiTenancy]`, `[MultiTenancyBundle]`, `...`). Use a verb (`fixed ...`, `added ...`, `...`) to start the summary and don't add a period at the end.

### Prepare your Patch for Submission

When your patch is not about a bug fix (when you add a new feature or change an existing one for instance), it must also include the following:

- An explanation of the changes in the relevant `CHANGELOG` file(s) (the `[BC BREAK]` or the `[DEPRECATION]` prefix must be used when relevant);
- An explanation on how to upgrade an existing application in the relevant `UPGRADE` file(s) if the changes break backward compatibility or if you deprecate something that will ultimately break backward compatibility.

### Step 3: Submit your Patch

Whenever you feel that your patch is ready for submission, follow the following steps.

### Rebase your Patch

Before submitting your patch, update your branch (needed if it takes you a while to finish your changes):

```
git checkout master
git fetch upstream
git merge upstream/master
```

(continues on next page)

(continued from previous page)

```
git checkout BRANCH_NAME
git rebase master
```

When doing the `rebase` command, you might have to fix merge conflicts. `git status` will show you the *unmerged* files. Resolve all the conflicts, then continue the rebase:

```
git add ... # add resolved files
git rebase --continue
```

Check that all tests still pass and push your branch remotely:

```
git push --force origin BRANCH_NAME
```

### Make a Pull Request

You can now make a pull request on the `superdesk/web-publisher` GitHub repository.

To ease the core team work, always include the modified components in your pull request message, like in:

```
[MultiTenancy] fixed something
[Common] [MultiTenancy] [MultiTenancyBundle] added something
```

The pull request description must include the following checklist at the top to ensure that contributions may be reviewed without needless feedback loops and that your contributions can be included into Superdesk Publisher as quickly as possible:

```
| Q             | A
| ------------- | ---
| Bug fix?      | [yes|no]
| New feature?  | [yes|no]
| BC breaks?    | [yes|no]
| Deprecations? | [yes|no]
| Tests pass?   | [yes|no]
| Fixed tickets | [comma separated list of tickets fixed by the PR]
| License       | AGPLv3
```

An example submission could now look as follows:

```
| Q             | A
| ------------- | ---
| Bug fix?      | no
| New feature?  | no
| BC breaks?    | no
| Deprecations? | no
| Tests pass?   | yes
| Fixed tickets | #12, #43
| License       | AGPLv3
```

The whole table must be included (do **not** remove lines that you think are not relevant). For simple typos, minor changes in the PHPDocs, or changes in translation files, use the shorter version of the check-list:

```
| Q             | A
| ------------- | ---
| Fixed tickets | [comma separated list of tickets fixed by the PR]
| License       | GPLv3
```

Some answers to the questions trigger some more requirements:

- If you answer yes to "Bug fix?", check if the bug is already listed in the Superdesk Publisher bug tracker and reference it/them in "Fixed tickets";
- If you answer yes to "New feature?", you must submit a pull request to the documentation and reference it under the "Doc PR" section;
- If you answer yes to "BC breaks?", the patch must contain updates to the relevant `CHANGELOG` and `UPGRADE` files;
- If you answer yes to "Deprecations?", the patch must contain updates to the relevant `CHANGELOG` and `UPGRADE` files;
- If you answer no to "Tests pass", you must add an item to a todo-list with the actions that must be done to fix the tests;
- If the "license" is not as AGPLv3 here, please don't submit the pull request as it won't be accepted anyway.

If some of the previous requirements are not met, create a todo-list and add relevant items:

```
- [ ] fix the tests as they have not been updated yet
- [ ] submit changes to the documentation
- [ ] document the BC breaks
```

**Caution:** When submitting pull requests which require some documentation changes, please also update the documentation where appropriate, as it is kept in the same repository (documentation dir)

If the code is not finished yet because you don't have time to finish it or because you want early feedback on your work, add an item to the todo-list:

```
- [ ] finish the code
- [ ] gather feedback for my changes
```

As long as you have items in the todo-list, please prefix the pull request title with "[WIP]".

In the pull request description, give as much detail as possible about your changes (don't hesitate to give code examples to illustrate your points). If your pull request is about adding a new feature or modifying an existing one, explain the rationale for the changes. The pull request description helps the code review and it serves as a reference when the code is merged (the pull request description and all its associated comments are part of the merge commit message).

### Rework your Patch

Based on the feedback on the pull request, you might need to rework your patch. Before re-submitting the patch, rebase with `upstream/master`, don't merge; and force push to the origin:

```
git rebase -f upstream/master
git push --force origin BRANCH_NAME
```

**Note:** When doing a `push --force`, always specify the branch name explicitly to avoid messing with other branches in the repo (`--force` tells Git that you really want to mess with things, so do it carefully).

If moderators asked you to "squash" your commits, this means you will need to convert many commits to one commit.

### Security Issues

This document explains how security issues affecting code in the main `superdesk/web-publisher` Git repository are handled by the Superdesk Publisher core team.

#### Reporting a Security Issue

If you think that you have found a security issue in Superdesk Publisher, don't use the bug tracker and don't publish it publicly. Instead, all security issues must be sent to **security [at] superdesk.org**. Emails sent to this address are forwarded to the Superdesk Publisher core-team private mailing list.

#### Resolving Process

For each report, we first try to confirm the vulnerability. When it is confirmed, the core team works on a solution following these steps:

1. Send an acknowledgement to the reporter;
2. Work on a patch;
3. Write a security announcement for the official Superdesk Publisher blog about the vulnerability. This post should contain the following information:
    - a title that always include the "Security release" string;
    - a description of the vulnerability;
    - the affected versions;
    - the possible exploits;
    - how to patch/upgrade/workaround affected applications;
    - credits.
4. Send the patch and the announcement to the reporter for review;
5. Apply the patch to all maintained versions of Superdesk Publisher;
6. Package new releases for all affected versions;
7. Publish the post on the official Superdesk Publisher blog

**Note:** Releases that include security issues should not be made on a Saturday or Sunday, except if the vulnerability has been publicly posted.

**Note:** While we are working on a patch, please do not reveal the issue publicly.

### Running Superdesk Publisher Tests

The Superdesk Publisher project uses a third-party service which automatically runs tests for any submitted *patch*. If the new code breaks any test, the pull request will show an error message with a link to the full error details.

In any case, it's a good practice to run tests locally before submitting a *patch* for inclusion, to check that you have not broken anything.

### Before Running the Tests

To run the Superdesk Publisher test suite, install the external dependencies used during the tests, such as Doctrine, Twig and Monolog. To do so, *install Composer* and execute the following:

```
composer install
```

**Note:** For unit tests we use PHPSpec, for functional tests PHPUnit and Behat for integration.

### Running the PHPUnit Tests

Then, run the test suite from the Superdesk Publisher root directory with the following command:

```
bin/phpunit -c app/
```

The output should display `OK`. If not, read the reported errors to figure out what's going on and if the tests are broken because of the new code.

**Tip:** The entire Superdesk Publisher suite can take up to several minutes to complete. If you want to test a single component/bundle, type its path after the `phpunit` command, e.g.:

```
bin/phpunit src/SWP/Bundle/MultiTenancyBundle/
```

**Tip:** On Windows, install the ConEmu, ANSICON or Mintty free applications to see coloured test results.

### Running the PHPSpec specs

**Note:** This section is based on Sylius documentation.

PHPSpec is a PHP toolset to drive emergent design by specification. It is not really a testing tool, but a design instrument, which helps structuring the objects and how they work together.

The Superdesk Publisher approach is to always describe the behaviour of the next object you are about to implement.

As an example, we'll write a service, which sets the current tenant in the context. To initialize a new spec, use the `desc` command.

We just need to tell **PHPSpec** we will be working on the *TenantContext* class.

```
bin/phpspec desc "SWP\Component\MultiTenancy\Context\TenantContext"
Specification for TenantContext created in spec.
```

What have we just done? **PHPSpec** has created the spec for us. You can navigate to the spec folder and see the spec there:

```php
<?php

namespace spec\SWP\Component\MultiTenancy\Context;

use PhpSpec\ObjectBehavior;
use Prophecy\Argument;

class TenantContextSpec extends ObjectBehavior
{
    function it_is_initializable()
    {
        $this->shouldHaveType('SWP\Component\MultiTenancy\Context\TenantContext');
    }
}
```

The object behaviour is made of examples. Examples are encased in public methods, started with `it_` or `its_`.

**PHPSpec** searches for such methods in your specification to run. Why underscores for example names? `just_because_its_much_easier_to_read` than `someLongCamelCasingLikeThat`.

Now, let's write the first example, which will set the current tenant:

```php
<?php

namespace spec\SWP\Component\MultiTenancy\Context;

use PhpSpec\ObjectBehavior;
use SWP\Component\MultiTenancy\Model\TenantInterface;

class TenantContextSpec extends ObjectBehavior
{
    function it_is_initializable()
    {
        $this->shouldHaveType('SWP\Component\MultiTenancy\Context\TenantContext');
    }

    function it_should_set_tenant(TenantInterface $tenant)
    {
        $tenant->getId()->willReturn(1);
        $tenant->getSubdomain()->willReturn('example1');
        $tenant->getName()->willReturn('example1');

        $this->setTenant($tenant)->shouldBeNull();
    }
}
```

The example looks clear and simple, the `TenantContext` service should obtain the tenant id, name, subdomain and call the method to set the tenant.

Try running the example by using the following command:

```
bin/phpspec run

> spec\SWP\Component\MultiTenancy\Context\TenantContext

   it should set tenant
      Class TenantContext does not exists.

         Do you want me to create it for you? [Y/n]
```

Once the class is created and you run the command again, PHPSpec will ask if it should create the method as well. Start implementing the initial version of the TenantContext.

```php
<?php

namespace SWP\Component\MultiTenancy\Context;

use SWP\Component\MultiTenancy\Model\TenantInterface;

/**
 * Class TenantContext.
 */
class TenantContext implements TenantContextInterface
{
    /**
     * @var TenantInterface
     */
    protected $tenant;

    /**
     * {@inheritdoc}
     */
    public function setTenant(TenantInterface $tenant)
    {
        $this->tenant = $tenant;
    }
}
```

Done! If you run PHPSpec again, you should see the following output:

```
bin/phpspec run

> spec\SWP\Component\MultiTenancy\Context\TenantContext

  ✓ it should set tenant

1 examples (1 passed)
123ms
```

This example is greatly simplified, in order to illustrate how we work. More examples might cover errors, API exceptions and other edge-cases.

A few tips & rules to follow when working with PHPSpec & Superdesk Publisher:

- RED is good, add or fix the code to make it green;
- RED-GREEN-REFACTOR is our rule;
- All specs must pass;
- When writing examples, **describe** the behaviour of the object in the present tense;
- Omit the `public` keyword;
- Use underscores (_) in the examples;
- Use type hinting to mock and stub classes;
- If your specification is getting too complex, the design is wrong. Try decoupling a bit more;
- If you cannot describe something easily, probably you should not be doing it that way;
- shouldBeCalled or willReturn, never together, except for builders;

- Use constants in assumptions but strings in expected results;

### Coding Standards

When contributing code to Superdesk Publisher, you must follow its coding standards. To make a long story short, here is the golden rule: **Imitate the existing Superdesk Publisher code**. Most open-source Bundles and libraries used by Superdesk Publisher also follow the same guidelines, and you should too.

Remember that the main advantage of standards is that every piece of code looks and feels familiar, it's not about this or that being more readable.

Superdesk Publisher follows the standards defined in the PSR-0, PSR-1, PSR-2 and PSR-4 documents.

Since a picture - or some code - is worth a thousand words, here's a short example containing most features described below:

```php
<?php

/**
 * This file is part of the Superdesk Publisher package.
 *
 * Copyright 2016 Sourcefabric z.ú. and contributors.
 *
 * For the full copyright and license information, please see the
 * AUTHORS and LICENSE files distributed with this source code.
 *
 * @copyright 2016 Sourcefabric z.ú.
 * @license http://www.superdesk.org/license
 */

namespace Acme;

/**
 * Coding standards demonstration.
 */
class FooBar
{
    const SOME_CONST = 42;

    /**
     * @var string
     */
    private $fooBar;

    /**
     * @param string $dummy Some argument description
     */
    public function __construct($dummy)
    {
        $this->fooBar = $this->transformText($dummy);
    }

    /**
     * @return string
     *
     * @deprecated
     */
    public function someDeprecatedMethod()
```

(continues on next page)

(continued from previous page)

```php
    {
        @trigger_error(sprintf('The %s() method is deprecated since version 1.0 and will be removed in 2.0. Use Acme\Baz::someMethod() instead.', __METHOD__), E_USER_DEPRECATED);

        return Baz::someMethod();
    }

    /**
     * Transforms the input given as first argument.
     *
     * @param bool|string $dummy   Some argument description
     * @param array       $options An options collection to be used within the transformation
     *
     * @return string|null The transformed input
     *
     * @throws \RuntimeException When an invalid option is provided
     */
    private function transformText($dummy, array $options = array())
    {
        $defaultOptions = array(
            'some_default' => 'values',
            'another_default' => 'more values',
        );

        foreach ($options as $option) {
            if (!in_array($option, $defaultOptions)) {
                throw new \RuntimeException(sprintf('Unrecognized option "%s"', $option));
            }
        }

        $mergedOptions = array_merge(
            $defaultOptions,
            $options
        );

        if (true === $dummy) {
            return;
        }

        if ('string' === $dummy) {
            if ('values' === $mergedOptions['some_default']) {
                return substr($dummy, 0, 5);
            }

            return ucwords($dummy);
        }
    }

    /**
     * Performs some basic check for a given value.
     *
     * @param mixed $value     Some value to check against
     * @param bool  $theSwitch Some switch to control the method's flow
     *
```

(continues on next page)

(continued from previous page)

```
     * @return bool|null The resultant check if $theSwitch isn't false, null otherwise
     */
    private function reverseBoolean($value = null, $theSwitch = false)
    {
        if (!$theSwitch) {
            return;
        }

        return !$value;
    }
}
```

## Structure

- Add a single space after each comma delimiter;
- Add a single space around binary operators (==, `&&`, `...`), with the exception of the concatenation (`.`) operator;
- Place unary operators (`!`, `--`, `...`) adjacent to the affected variable;
- Always use identical comparison unless you need type juggling;
- Use Yoda conditions when checking a variable against an expression to avoid an accidental assignment inside the condition statement (this applies to ==, `!=`, ===, and `!==`);
- Add a comma after each array item in a multi-line array, even after the last one;
- Add a blank line before `return` statements, unless the return is alone inside a statement-group (like an `if` statement);
- Use just `return;` instead of `return null;` when a function must return void early;
- Use braces to indicate control structure body regardless of the number of statements it contains;
- Define one class per file - this does not apply to private helper classes that are not intended to be instantiated from the outside and thus are not concerned by the PSR-0 and PSR-4 autoload standards;
- Declare class properties before methods;
- Declare public methods first, then protected ones and finally private ones. The exceptions to this rule are the class constructor and the `setUp` and `tearDown` methods of PHPUnit tests, which should always be the first methods to increase readability;
- Use parentheses when instantiating classes regardless of the number of arguments the constructor has;
- Exception and error message strings should be concatenated using `sprintf`.
- Calls to `trigger_error` with type `E_USER_DEPRECATED` should be switched to opt-in via `@` operator. Read more at *Deprecations*;

## Naming Conventions

- Use camelCase, not underscores, for variable, function and method names, arguments;
- Use underscores for option names and parameter names;
- Use namespaces for all classes;
- Prefix abstract classes with `Abstract`.

- Suffix interfaces with `Interface`;
- Suffix traits with `Trait`;
- Suffix exceptions with `Exception`;
- Use alphanumeric characters and underscores for file names;
- For type-hinting in PHPDocs and casting, use `bool` (instead of `boolean` or `Boolean`), `int` (instead of `integer`), `float` (instead of `double` or `real`);
- Don't forget to look at the more verbose *Conventions* document for more subjective naming considerations.

### Service Naming Conventions

- A service name contains groups, separated by dots;
- The DI alias of the bundle is the first group (e.g. `swp_user`);
- Use lowercase letters for service and parameter names;
- A group name uses the underscore notation.

### Documentation

- Add PHPDoc blocks for all classes, methods, and functions;
- Group annotations together so that annotations of the same type immediately follow each other, and annotations of a different type are separated by a single blank line;
- Omit the `@return` tag if the method does not return anything;
- The `@package` and `@subpackage` annotations are not used.

### License

- Superdesk Publisher is released under the license, and the license block has to be present at the top of every PHP file, before the namespace.

### Conventions

The *Coding Standards* document describes the coding standards for the Superdesk Publisher projects and the internal and third-party bundles. This document describes coding standards and conventions used in the core project to make it more consistent and predictable.

### Method Names

When an object has a "main" many relation with related "things" (objects, parameters, . . . ), the method names are normalized:

- `get()`
- `set()`
- `has()`
- `all()`

- `replace()`
- `remove()`
- `clear()`
- `isEmpty()`
- `add()`
- `register()`
- `count()`
- `keys()`

The usage of these methods are only allowed when it is clear that there is a main relation:

- a `CookieJar` has many `Cookie` objects;
- a Service `Container` has many services and many parameters (as services is the main relation, the naming convention is used for this relation);
- a Console `Input` has many arguments and many options. There is no "main" relation, and so the naming convention does not apply.

For many relations where the convention does not apply, the following methods must be used instead (where `XXX` is the name of the related thing):

| Main Relation | Other Relations |
|---|---|
| `get()` | `getXXX()` |
| `set()` | `setXXX()` |
| n/a | `replaceXXX()` |
| `has()` | `hasXXX()` |
| `all()` | `getXXXs()` |
| `replace()` | `setXXXs()` |
| `remove()` | `removeXXX()` |
| `clear()` | `clearXXX()` |
| `isEmpty()` | `isEmptyXXX()` |
| `add()` | `addXXX()` |
| `register()` | `registerXXX()` |
| `count()` | `countXXX()` |
| `keys()` | n/a |

**Note:** While "setXXX" and "replaceXXX" are very similar, there is one notable difference: "setXXX" may replace, or add new elements to the relation. "replaceXXX", on the other hand, cannot add new elements. If an unrecognized key is passed to "replaceXXX" it must throw an exception.

### Deprecations

From time to time, some classes and/or methods are deprecated in the project; that happens when a feature implementation cannot be changed because of backward compatibility issues, but we still want to propose a "better" alternative. In that case, the old implementation can simply be **deprecated**.

A feature is marked as deprecated by adding a `@deprecated` phpdoc to relevant classes, methods, properties, and so on:

```
/**
 * @deprecated Deprecated since version 1.0, to be removed in 2.0. Use XXX instead.
 */
```

The deprecation message should indicate the version when the class/method was deprecated, the version when it will be removed, and whenever possible, how the feature was replaced.

A PHP `E_USER_DEPRECATED` error must also be triggered to help people with the migration starting one or two minor versions before the version where the feature will be removed (depending on the criticality of the removal):

```
@trigger_error('XXX() is deprecated since version 1.0 and will be removed in 2.0. Use␣
↪XXX instead.', E_USER_DEPRECATED);
```

Without the @-silencing operator, users would need to opt-out from deprecation notices. Silencing swaps this behaviour and allows users to opt-in when they are ready to cope with them (by adding a custom error handler like the one used by the Web Debug Toolbar or by the PHPUnit bridge).

### Git

This document explains some conventions and specificities in the way we manage the Superdesk Publisher code with Git.

### Pull Requests

Whenever a pull request is merged, all the information contained in the pull request (including comments) is saved in the repository.

You can easily spot pull request merges as the commit message always follows this pattern:

```
merged branch USER_NAME/BRANCH_NAME (PR #11)
```

The PR reference allows you to have a look at the original pull request on GitHub: https://github.com/superdesk/web-publisher/pull/11. But all the information you can get on GitHub is also available from the repository itself.

The merge commit message contains the original message from the author of the changes. Often, this can help understand what the changes were about and the reasoning behind the changes.

### The License

Superdesk Publisher is released under the GNU Affero General Public License, version 3.

## 5.1.2 Contributing Documentation

### Contributing to the Documentation

One of the essential principles of the Superdesk Publisher project is that **documentation is as important as code**. That's why a great amount of resources are dedicated to documenting new features and to keeping the rest of the documentation up-to-date.

### Before Your First Contribution

**Before contributing**, you should consider the following:

- Superdesk Publisher documentation is written using reStructuredText markup language. If you are not familiar with this format, read *this article* for a quick overview of its basic features.
- Superdesk Publisher documentation is hosted on GitHub. You'll need a GitHub user account to contribute to the documentation.
- Superdesk Publisher documentation is published under a *Creative Commons BY-SA 3.0 License* and all your contributions will implicitly adhere to that license.

### Your First Documentation Contribution

In this section, you'll learn how to contribute to the Superdesk Publisher documentation for the first time. The next section will explain the shorter process you'll follow in the future for every contribution after your first one.

Let's imagine that you want to improve the installation chapter of the Superdesk Publisher. In order to make your changes, follow these steps:

**Step 1.** Go to the official Superdesk Publisher repository located at github.com/superdesk/web-publisher and fork the repository to your personal account. This is only needed the first time you contribute to Superdesk Publisher.

**Step 2. Clone** the forked repository to your local machine (this example uses the `projects/web-publisher/` directory to store the Superdesk Publisher; change this value accordingly):

```
cd projects/
git clone git://github.com/<YOUR GITHUB USERNAME>/web-publisher.git
```

**Step 3.** Create a dedicated **new branch** for your changes. This greatly simplifies the work of reviewing and merging your changes. Use a short and memorable name for the new branch:

```
git checkout -b improve_install_chapter
```

**Step 4.** Now make your changes in the documentation. Add, tweak, reword and even remove any content, but make sure that you comply with the *Documentation Standards*.

**Note:** Documentation is stored under the `docs` directory in the root Superdesk Publisher folder.

**Step 5. Push** the changes to your forked repository:

```
git commit book/installation.rst
git push origin improve_install_chapter
```

**Step 6.** Everything is now ready to initiate a **pull request**. Go to your forked repository at `https//github.com/<YOUR GITHUB USERNAME>/web-publisher` and click on the **Pull Requests** link located in the sidebar.

Then, click on the big **New pull request** button. As GitHub cannot guess the exact changes that you want to propose, select the appropriate branches where changes should be applied.

The **base fork** should be `superdesk/web-publisher` and the **base** branch should be the `master`, which is the branch that you selected to base your changes on. The **head fork** should be your forked copy of `web-publisher` and the **compare** branch should be `improve_install_chapter`, which is the name of the branch you created and where you made your changes.

**Step 7.** The last step is to prepare the **description** of the pull request. To ensure that your work is reviewed quickly, please add the following table at the beginning of your pull request description:

```
| Q             | A
| ------------- | ---
| Doc fix?      | [yes|no]
| New docs?     | [yes|no] (PR # on superdesk/web-publisher if applicable)
| Applies to    | [Superdesk Publisher version numbers this applies to]
| Fixed tickets | [comma separated list of tickets fixed by the PR]
```

In this example, this table would look as follows:

```
| Q             | A
| ------------- | ---
| Doc fix?      | yes
| New docs?     | no
| Applies to    | all
| Fixed tickets | #10575
```

**Step 8.** Now that you've successfully submitted your first contribution to the Superdesk Publisher documentation, **go and celebrate!** The documentation managers will carefully review your work in short time and they will let you know about any required change.

In case you need to add or modify anything, there is no need to create a new pull request. Just make sure that you are on the correct branch, make your changes and push them:

```
cd projects/web-publisher/
git checkout improve_install_chapter

# ... do your changes

git push
```

**Step 9.** After your pull request is eventually accepted and merged in the Superdesk Publisher, you will be included in the Superdesk Publisher Contributors list.

### Your Second Documentation Contribution

The first contribution took some time because you had to fork the repository, learn how to write documentation, comply with the pull requests standards, etc. The second contribution will be much easier, except for one detail: given the furious update activity of the Superdesk Publisher documentation repository, odds are that your fork is now out of date with the official repository.

Solving this problem requires you to sync your fork with the original repository. To do this, execute this command first to tell git about the original repository:

```
cd projects/web-publisher/
git remote add upstream https://github.com/superdesk/web-publisher.git
```

Now you can **sync your fork** by executing the following command:

```
cd projects/web-publisher/
git fetch upstream
git checkout master
git merge upstream/master
```

This command will update the `master` branch, which is the one you used to create the new branch for your changes. If you have used another base branch, e.g. `testing`, replace the `master` with the appropriate branch name.

Great! Now you can proceed by following the same steps explained in the previous section:

```
# create a new branch to store your changes based on the master branch
cd projects/web-publisher/
git checkout master
git checkout -b my_changes

# ... do your changes

# submit the changes to your forked repository
git add xxx.rst     # (optional) only if this is a new content
git commit xxx.rst
git push origin my_changes

# go to GitHub and create the Pull Request
#
# Include this table in the description:
# | Q             | A
# | ------------- | ---
# | Doc fix?      | [yes|no]
# | New docs?     | [yes|no]
# | Applies to    | [Superdesk Publisher version numbers this applies to]
# | Fixed tickets | [comma separated list of tickets fixed by the PR]
```

Your second contribution is now complete, so **go and celebrate again!** You can also see how your ranking improves in the list of Superdesk Publisher Contributors.

### Your Next Documentation Contributions

Now that you've made two contributions to the Superdesk Publisher documentation, you are probably comfortable with all the Git-magic involved in the process. That's why your next contributions would be much faster. Here you can find the complete steps to contribute to the Superdesk Publisher documentation, which you can use as a **checklist**:

```
# sync your fork with the official Superdesk Publisher repository
cd projects/web-publisher/
git fetch upstream
git checkout master
git merge upstream/master

# create a new branch from the maintained version
git checkout master
git checkout -b my_changes

# ... do your changes

# add and commit your changes
git add xxx.rst     # (optional) only if this is a new content
git commit xxx.rst
git push origin my_changes

# go to GitHub and create the Pull Request
#
# Include this table in the description:
# | Q             | A
```

(continues on next page)

(continued from previous page)

```
# | ------------- | ---
# | Doc fix?      | [yes|no]
# | New docs?     | [yes|no]
# | Applies to    | [Superdesk Publisher version numbers this applies to]
# | Fixed tickets | [comma separated list of tickets fixed by the PR]

# (optional) make the changes requested by reviewers and commit them
git commit xxx.rst
git push
```

You guessed right: after all this hard work, it's **time to celebrate again!**

### Review your changes

Every GitHub Pull Request when merged, is automatically deployed to http://superdesk-publisher.readthedocs.io/en/latest/

### Minor Changes (e.g. Typos)

You may find just a typo and want to fix it. Due to GitHub's functional frontend, it is quite simple to create Pull Requests right in your browser while reading the docs on http://superdesk-publisher.readthedocs.io/en/latest/. To do this, just click the **edit this page** button on the upper right corner. Beforehand, please switch to the right branch as mentioned before. Now you are able to edit the content and describe your changes within the GitHub frontend. When your work is done, click **Propose file change** to create a commit and, in case it is your first contribution, also your fork. A new branch is created automatically in order to provide a base for your Pull Request. Then fill out the form to create the Pull Request as described above.

### Frequently Asked Questions

### Why Do my Changes Take so Long to Be Reviewed and/or Merged?

Please be patient. It can take up to several days before your pull request can be fully reviewed. After merging the changes, it could take again several hours before your changes appear on the http://superdesk-publisher.readthedocs.io/en/latest/ website.

### What If I Want to Submit my Work without Fully Finishing It?

You can do it. But please use one of these two prefixes to let reviewers know about the state of your work:

- `[WIP]` (Work in Progress) is used when you are not yet finished with your pull request, but you would like it to be reviewed. The pull request won't be merged until you say it is ready.
- `[WCM]` (Waiting Code Merge) is used when you're documenting a new feature or change that hasn't been accepted yet into the core code. The pull request will not be merged until it is merged in the core code (or closed if the change is rejected).

### Would You Accept a Huge Pull Request with Lots of Changes?

First, make sure that the changes are somewhat related. Otherwise, please create separate pull requests. Anyway, before submitting a huge change, it's probably a good idea to open an issue in the Superdesk Web Publisher repository to ask the managers if they agree with your proposed changes. Otherwise, they could refuse your proposal after you put all that hard work into making the changes. We definitely don't want you to waste your time!

### Documentation Format

The Superdesk Publisher documentation uses reStructuredText as its markup language and Sphinx for generating the documentation in the formats read by the end users, such as HTML and PDF.

### reStructuredText

reStructuredText is a plaintext markup syntax similar to Markdown, but much stricter with its syntax. If you are new to reStructuredText, take some time to familiarize with this format by reading the existing Superdesk Publisher documentation source code.

If you want to learn more about this format, check out the reStructuredText Primer tutorial and the reStructuredText Reference.

**Caution:** If you are familiar with Markdown, be careful as things are sometimes very similar but different:

- Lists starts at the beginning of a line (no indentation is allowed);
- Inline code blocks use double-ticks (``` ``like this`` ```).

### Sphinx

Sphinx is a build system that provides tools to create documentation from reStructuredText documents. As such, it adds new directives and interpreted text roles to the standard reST markup. Read more about the Sphinx Markup Constructs.

### Syntax Highlighting

PHP is the default syntax highlighter applied to all code blocks. You can change it with the `code-block` directive:

```
.. code-block:: yaml

    { foo: bar, bar: { foo: bar, bar: baz } }
```

**Note:** Besides all of the major programming languages, the syntax highlighter supports all kinds of markup and configuration languages. Check out the list of supported languages on the syntax highlighter website.

### Configuration Blocks

Whenever you include a configuration sample, use the `configuration-block` directive to show the configuration in all supported configuration formats (`PHP`, `YAML` and `XML`). Example:

```
.. configuration-block::

    .. code-block:: yaml

        # Configuration in YAML

    .. code-block:: xml

        <!-- Configuration in XML -->

    .. code-block:: php

        // Configuration in PHP
```

The previous reST snippet renders as follow:

- *YAML*

```
# Configuration in YAML
```

- *XML*

```
<!-- Configuration in XML -->
```

- *PHP*

```
// Configuration in PHP
```

The current list of supported formats are the following:

| Markup Format | Use It to Display |
|---|---|
| `html` | HTML |
| `xml` | XML |
| `php` | PHP |
| `yaml` | YAML |
| `twig` | Pure Twig markup |
| `html+twig` | Twig markup blended with HTML |
| `html+php` | PHP code blended with HTML |
| `ini` | INI |
| `php-annotations` | PHP Annotations |

### Adding Links

The most common type of links are **internal links** to other documentation pages, which use the following syntax:

```
:doc:`/absolute/path/to/page`
```

The page name should not include the file extension (`.rst`). For example:

```
:doc:`/book/controller`

:doc:`/components/event_dispatcher/introduction`

:doc:`/cookbook/configuration/environments`
```

The title of the linked page will be automatically used as the text of the link. If you want to modify that title, use this alternative syntax:

```
:doc:`Spooling Email </cookbook/email/spool>`
```

**Note:** Although they are technically correct, avoid the use of relative internal links such as the following, because they break the references in the generated PDF documentation:

```
:doc:`controller`

:doc:`event_dispatcher/introduction`

:doc:`environments`
```

**Links to the API** follow a different syntax, where you must specify the type of the linked resource (`namespace`, `class` or `method`):

```
:namespace:`SWP\\Component\\MultiTenancy`

:class:`SWP\\Component\\MultiTenancy\\Context\\TenantContext`

:method:`SWP\\Component\\MultiTenancy\\Context\\TenantContext::setTenant`
```

**Links to the PHP documentation** follow a pretty similar syntax:

```
:phpclass:`SimpleXMLElement`

:phpmethod:`DateTime::createFromFormat`

:phpfunction:`iterator_to_array`
```

## New Features or Behaviour Changes

If you're documenting a brand new feature or a change that's been made in Superdesk Publisher, you should precede your description of the change with a `.. versionadded:: 2.X` directive and a short description:

```
.. versionadded:: 2.3
    The ``askHiddenResponse`` method was introduced in Superdesk Publisher 2.3.

You can also ask a question and hide the response. This is particularly [...]
```

If you're documenting a behaviour change, it may be helpful to *briefly* describe how the behaviour has changed:

```
.. versionadded:: 2.3
    The ``include()`` function is a new MultiTenancy feature that's available in
    Superdesk Publisher 2.3. Prior, the ``{% include %}`` tag was used.
```

At this point, all the `versionadded` tags for Superdesk Publisher versions that have reached end-of-maintenance will be removed. For example, if Superdesk Publisher 2.5 were released today, and 2.2 had recently reached its end-of-life, the 2.2 `versionadded` tags would be removed from the new `2.5` branch.

### Testing Documentation

When submitting new content to the documentation repository or when changing any existing resource, an automatic process will check if your documentation is free of syntax errors and is ready to be reviewed.

Nevertheless, if you prefer to do this check locally on your own machine before submitting your documentation, follow these steps:

- Install Sphinx;
- Install the Sphinx extensions using git submodules: `git submodule update --init`;
- Run `make html` and view the generated HTML in the `_build/html` directory.

### Documentation Standards

In order to help the reader as much as possible and to create code examples that look and feel familiar, you should follow these standards.

### Sphinx

- The following characters are chosen for different heading levels: level 1 is = (equal sign), level 2 - (dash), level 3 ~ (tilde), level 4 `.` (dot) and level 5 `"` (double quote);
- Each line should break approximately after the first word that crosses the 72nd character (so most lines end up being 72-78 characters);
- The `::` shorthand is *preferred* over `.. code-block:: php` to begin a PHP code block (read the Sphinx documentation to see when you should use the shorthand);
- Inline hyperlinks are **not** used. Separate the link and their target definition, which you add on the bottom of the page;
- Inline markup should be closed on the same line as the open-string;

### Example

```
Example
=======

When you are working on the docs, you should follow the
`Superdesk Publisher Documentation`_ standards.

Level 2
-------

A PHP example would be::

    echo 'Hello World';

Level 3
~~~~~~~

.. code-block:: php

```

(continues on next page)

(continued from previous page)

```
    echo 'You cannot use the :: shortcut here';

.. _`Superdesk Publisher Documentation`: http://superdesk-publisher.readthedocs.io/en/
↪latest
```

### Code Examples

- The code follows the *Superdesk Publisher Coding Standards* as well as the Twig Coding Standards;
- The code examples should look real for a web application context. Avoid abstract or trivial examples (`foo`, `bar`, `demo`, etc.);
- The code should follow the Symfony Best Practices. Unless the example requires a custom bundle, make sure to always use the `AppBundle` bundle to store your code;
- Use `Acme` when the code requires a vendor name;
- Use `example.com` as the domain of sample URLs and `example.org` and `example.net` when additional domains are required. All of these domains are reserved by the IANA.
- To avoid horizontal scrolling on code blocks, we prefer to break a line correctly if it crosses the 85th character;
- When you fold one or more lines of code, place `...` in a comment at the point of the fold. These comments are: `// ...` (php), `# ...` (yaml/bash), `{# ... #}` (twig), `<!-- ... -->` (xml/html), `; ...` (ini), `...` (text);
- When you fold a part of a line, e.g. a variable value, put `...` (without comment) at the place of the fold;
- Description of the folded code: (optional)
  - If you fold several lines: the description of the fold can be placed after the `...`;
  - If you fold only part of a line: the description can be placed before the line;
- If useful to the reader, a PHP code example should start with the namespace declaration;
- When referencing classes, be sure to show the `use` statements at the top of your code block. You don't need to show *all* `use` statements in every example, just show what is actually being used in the code block;
- If useful, a `codeblock` should begin with a comment containing the filename of the file in the code block. Don't place a blank line after this comment, unless the next line is also a comment;
- You should not put a `$` in front of every bash line, as it doesn't work when readers copy and paste into the console.

### Formats

Configuration examples should show all supported formats using *configuration blocks*. The supported formats (and their orders) are:

- **Configuration** (including services): YAML, XML, PHP
- **Routing**: Annotations, YAML, XML, PHP
- **Validation**: Annotations, YAML, XML, PHP
- **Doctrine Mapping**: Annotations, YAML, XML, PHP
- **Translation**: XML, YAML, PHP

### Example

```php
// src/Foo/Bar.php
namespace Foo;

use Acme\Demo\Cat;
// ...

class Bar
{
    // ...

    public function foo($bar)
    {
        // set foo with a value of bar
        $foo = ...;

        $cat = new Cat($foo);

        // ... check if $bar has the correct value

        return $cat->baz($bar, ...);
    }
}
```

**Caution:** In YAML you should put a space after `{` and before `}` (e.g. `{ _controller: ... }`), but this should not be done in Twig (e.g. `{'hello' : 'value'}`).

### Files and Directories

- When referencing directories, always add a trailing slash to avoid confusions with regular files (e.g. "execute the `console` script located at the `app/` directory").
- When referencing file extensions explicitly, you should include a leading dot for every extension (e.g. "XML files use the `.xml` extension").
- When you list a Superdesk Publisher file/directory hierarchy, use `your-project/` as the top level directory. E.g.

```
your-project/
├─ app/
├─ src/
├─ vendor/
└─ ...
```

### English Language Standards

Superdesk Publisher documentation uses the English dialect, sometimes called British English by people who don't know any better. Collins Dictionary is used as the vocabulary reference.

In addition, documentation follows these rules:

- **Section titles**: use a variant of the title case, where the first word is always capitalized and all other words are capitalized, except for the closed-class words (read Wikipedia article about headings and titles).

  E.g.: The Vitamins are in my Fresh California Raisins
- **Punctuation**: avoid the use of Serial (Oxford) Commas;
- **Pronouns**: avoid the use of nosism and always use *you* instead of *we*. (i.e. avoid the first person point of view: use the second instead);
- **Gender-neutral language**: when referencing a hypothetical person, such as *"a user with a session cookie"*, use gender-neutral pronouns (they/their/them). For example, instead of:
  - he or she, use they
  - him or her, use them
  - his or her, use their
  - his or hers, use theirs
  - himself or herself, use themselves

## Contributed Documentation License

The Superdesk Publisher documentation is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License (CC BY-SA 3.0).

**You are free:**

- to *Share* — to copy, distribute and transmit the work;
- 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 or similar license to this one.

**With the understanding that:**

- *Waiver* — Any of the above conditions can be waived if you get permission from the copyright holder;
- *Public Domain* — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license;
- *Other Rights* — In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  - The author's moral rights;
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- *Notice* — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

This is a human-readable summary of the Legal Code (the full license).

#### Translations

The Superdesk Publisher documentation has not been translated yet.

If you would like to help translate this documentation, please email us at contact@sourcefabric.org or find us on GitHub (https://github.com/superdesk/web-publisher).

## 5.1.3 Community

### Community Reviews

Superdesk Publisher is an open-source project driven by its community. If you don't feel ready to contribute code or patches, reviewing issues and pull requests (PRs) can be a great start to get involved and give back. In fact, people who "triage" issues are the backbone to Superdesk Publisher's success!

#### Why Reviewing Is Important

Community reviews are essential for the development of the Superdesk Publisher project. On the Superdesk Publisher JIRA bug tracker and GitHub, you can find many items to work on:

- **Bug Reports**: Bug reports need to be checked for completeness. Is any important information missing? Can the bug be *easily* reproduced?
- **Pull Requests**: Pull requests contain code that fixes a bug or implements new functionality. Reviews of pull requests ensure that they are implemented properly, are covered by test cases, don't introduce new bugs and maintain backwards compatibility.

Note that **anyone who has some basic familiarity with Symfony and PHP can review bug reports and pull requests**. You don't need to be an expert to help.

#### Be Constructive

Before you begin, remember that you are looking at the result of someone else's hard work. A good review comment thanks the contributor for their work, identifies what was done well, identifies what should be improved and suggests a next step.

#### Create a GitHub Account

Superdesk Publisher uses GitHub to manage pull requests. If you want to do reviews, you need to create a GitHub account and log in.

#### Create a JIRA Account

Superdesk Publisher uses JIRA to manage bug reports. If you want to report a bug, you need to create a JIRA account and log in.

### The Pull Request Review Process

Reviews of pull requests usually take a little longer since you need to understand the functionality that has been fixed or added and find out whether the implementation is complete.

It is okay to do partial reviews! If you do a partial review, comment how far you got and leave the PR in "needs review" state.

Pick a pull request from the PRs in need of review and follow these steps:

1. **Is the PR Complete**? Every pull request must contain a header that gives some basic information about the PR. You can find the template for that header in the *Contribution Guidelines*.
2. **Is the Base Branch Correct?** GitHub displays the branch that a PR is based on below the title of the pull request. Is that branch correct?
3. **Reproduce the Problem** Read the issue that the pull request is supposed to fix. Reproduce the problem on a clean Superdesk Web Publisher project and try to understand why it exists. If the linked issue already contains such a project, install it and run it on your system.
4. **Review the Code** Read the code of the pull request and check it against some common criteria:
   - Does the code address the issue the PR is intended to fix/implement?
   - Does the PR stay within scope to address *only* that issue?
   - Does the PR contain automated tests? Do those tests cover all relevant edge cases?
   - Does the PR contain sufficient comments to easily understand its code?
   - Does the code break backwards compatibility? If yes, does the PR header say so?
   - Does the PR contain deprecations? If yes, does the PR header say so? Does the code contain `trigger_error()` statements for all deprecated features?
   - Are all deprecations and backwards compatibility breaks documented in the latest UPGRADE-X.X.md file? Do those explanations contain "Before"/"After" examples with clear upgrade instructions?

   **Note:** Eventually, some of these aspects will be checked automatically.

5. **Test the Code**
6. **Update the PR Status**

   At last, add a comment to the PR. **Thank the contributor for working on the PR**.

**Example**

Here is a sample comment for a PR that is not yet ready for merge:

```
Thank you @takeit for working on this! It seems that your test
cases don't cover the cases when the counter is zero or smaller.
Could you please add some tests for that?
```

### Other Resources

In order to follow what is happening in the community you might find helpful these additional resources:

- List of open pull requests

- List of recent commits
- List of open bugs and enhancements
- **Code**
  - *Bugs*
  - *Patches*
  - *Reviewing Issues and Patches*
  - *Security*
  - *Tests*
  - *Coding Standards*
  - *Code Conventions*
  - *Git*
  - *License*
- **Documentation**
  - *Overview*
  - *Format*
  - *Documentation Standards*
  - *License*
- **Community**
  - *Community Reviews*
  - *Other Resources*

- **Code**
  - *Bugs*
  - *Patches*
  - *Reviewing Issues and Patches*
  - *Security*
  - *Tests*
  - *Coding Standards*
  - *Code Conventions*
  - *Git*
  - *License*
- **Documentation**
  - *Overview*
  - *Format*
  - *Documentation Standards*
  - *License*
- **Community**

## 5.2 Components

**Note:** This section is based on Sylius documentation released under a Creative Commons Attribution-Share Alike 3.0 Unported License.

The Components are a set of decoupled PHP libraries, which are the foundation of the Superdesk Web Publisher project, but they can also be used in other PHP projects which don't rely on the Symfony framework itself.

The goal of these components is to solve the problems related to content publishing and content rendering.

### 5.2.1 How to Install and Use the Components

**Note:** This section is based on Symfony2 documentation.

If you're starting a new project (or already have a project) that will use one or more components, the easiest way to integrate everything is with Composer. Composer is smart enough to download the component(s) that you need and take care of autoloading so that you can begin using the libraries immediately.

This article will take you through using *The MultiTenancy Component*, though this applies to using any component.

#### Using the MultiTenancy Component

**1.** If you're creating a new project, create a new empty directory for it.

**2.** Open a console and use Composer to grab the library.

```
composer require swp/multi-tenancy
```

The name `swp/multi-tenancy` is written at the top of the documentation for the component.

**Tip:** Install composer if you don't have it already present on your system. Depending on how you install, you may end up with a `composer.phar` file in your directory. In that case, no worries! Just run `php composer.phar require swp/multi-tenancy`.

**3.** Write your code!

Once Composer has downloaded the component(s), all you need to do is include the `vendor/autoload.php` file that was generated by Composer. This file takes care of autoloading all of the libraries so that you can use them immediately.

```
<?php
// File example: src/script.php

// update this to the path to the "vendor/"
// directory, relative to this file
```

(continues on next page)

(continued from previous page)

```php
require_once __DIR__.'/../vendor/autoload.php';

use SWP\Component\MultiTenancy\Context\TenantContext;
use SWP\Component\MultiTenancy\Model\Tenant;

$tenant = new Tenant();
$tenantContext = new TenantContext();

$tenantContext->setTenant($tenant);

var_dump($tenantContext->getTenant());

// ...
```

### Now what?

Now that the component is installed and autoloaded, read the specific component's documentation to find out more about how to use it.

And have fun!

## 5.2.2 The MultiTenancy Component

This component provides basic services, models and interfaces to build multi-tenant PHP apps.

### Installation

You can install the component in two different ways:

- *Install it via Composer* (`swp/multi-tenancy` on Packagist);
- Use the official Git repository (https://github.com/SuperdeskWebPublisher/multi-tenancy).

**Note:** This section is based on Symfony2 documentation.

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Symfony component.

### Usage

### TenantContext

The **TenantContext** allows you to manage the currently used tenant.

```php
<?php

// ..
use SWP\Component\MultiTenancy\Context\TenantContext;
use SWP\Component\MultiTenancy\Model\Tenant;

```

(continues on next page)

(continued from previous page)

```php
$tenant = new Tenant();
$tenantContext = new TenantContext();

$tenantContext->setTenant($tenant);

var_dump($tenantContext->getTenant());
```

**Note:** This service implements *TenantContextInterface*.

### TenantProvider

The **TenantProvider** allows you to get all available tenants.

```php
<?php

 // ..
use SWP\Component\MultiTenancy\Provider\TenantProvider;

$tenantProvider = new TenantProvider(/*␣
↪SWP\Component\MultiTenancy\Repository\TenantRepositoryInterface repository */);

var_dump($tenantProvider->getAvailableTenants());
```

The `getAvailableTenants` method retrieves all tenants which have the `enabled` property set to true and have been inserted in the given repository.

**Note:** This service implements the *TenantProviderInterface*.

### TenantResolver

The **TenantResolver** allows you to resolve the current tenant from the request.

```php
<?php

// ..
use SWP\Component\MultiTenancy\Resolver\TenantResolver;

$tenantResolver = new TenantResolver('example.com', /*␣
↪SWP\Component\MultiTenancy\Repository\TenantRepositoryInterface repository */);

var_dump($tenantResolver->resolve('tenant.example.com')); // will return an instance␣
↪of TenantInterface.
```

The `resolve` method resolves the tenant based on the current subdomain name. For example, when the host is `tenant.example.com`, it will resolve the subdomain (`tenant`) and then it will search for the tenant in the given repository, by the resolved subdomain name. If the subdomain `tenant` is not found, it always returns the default tenant.

**Note:** This service implements the *TenantResolverInterface*.

### TenantAwarePathBuilder

The **TenantAwarePathBuilder** responsibility is to build PHPCR base paths which are tenant-aware. This can build whatever path is needed by the provided paths' names and the given context.

```php
<?php

// ..
use SWP\Component\MultiTenancy\PathBuilder\TenantAwarePathBuilder;
use SWP\Component\MultiTenancy\Context\TenantContext;
use SWP\Component\MultiTenancy\Model\Tenant;

$tenant = new Tenant();
$tenant->setSubdomain('example');
$tenant->setName('Example tenant');
$tenantContext = new TenantContext();
$tenantContext->setTenant($tenant);

$pathBuilder = new TenantAwarePathBuilder($tenantContext, '/swp');

var_dump($pathBuilder->build('routes')); // will return: /swp/example/routes.
var_dump($pathBuilder->build(['routes', 'content'])); // will return an array: ['/swp/
↪example/routes', '/swp/example/routes']
var_dump($pathBuilder->build('/')); // will return: /swp/default
```

The `build` method method builds the PHPCR path. It accepts as a first argument, a string or an array of routes' names. The second argument is the context for the given path(s) name(s).

In order to build the base paths, the TenantAwarePathBuilder's construct requires an object of type *TenantResolver-Interface* to be provided as a first argument, the object of type *TenantContextInterface* as a second argument, and the root base path as a third argument.

**Note:** This service implements the *TenantResolverInterface*.

### TenantFactory

The **TenantFactory** allows you to create an objects of type *TenantInterface*.

```php
<?php

 // ..
use SWP\Component\MultiTenancy\Model\Tenant;
use SWP\Component\MultiTenancy\Factory\TenantFactory;

$tenantFactory = new TenantFactory(Tenant::class);
$tenant = $tenantFactory->create();

var_dump($tenant);
```

### OrganizationFactory

The **OrganizationFactory** allows you to create objects of type *OrganizationInterface*.

```php
<?php

 // ..
use SWP\Component\MultiTenancy\Model\Organization;
use SWP\Component\MultiTenancy\Factory\OrganizationFactory;
use SWP\Component\Common\Generator\RandomStringGenerator;

$organizationFactory = new OrganizationFactory(Organization::class, new
↪RandomStringGenerator());
$organization = $organizationFactory->create();
$organizationWithCode = $organizationFactory->createWithCode();

var_dump($organization);
```

## Models

### Tenant

Every tenant is represented by a **Tenant** model which by default has the following properties:

| Method | Description |
|---|---|
| id | Unique identifier |
| subdomain | Tenant’s subdomain |
| name | Tenant’s name |
| code | Tenant’s code |
| enabled | Indicates whether the tenant is enabled |
| createdAt | Date of creation |
| updatedAt | Date of last update |
| deletedAt | Indicates whether the tenant is deleted |

**Note:** This model implements *TenantInterface*.

### Organization

Every organization is represented by a **Organization** model which by default has the following properties:

| Method | Description |
|---|---|
| id | Unique identifier |
| name | Tenant’s name |
| code | Tenant’s code |
| enabled | Indicates whether the tenant is enabled |
| createdAt | Date of creation |
| updatedAt | Date of last update |
| deletedAt | Indicates whether the tenant is deleted |

**Note:** This model implements *OrganizationInterface*.

## Interfaces

### Model Interfaces

#### TenantInterface

This interface should be implemented by every tenant.

**Note:** This interface extends TimestampableInterface, EnableableInterface and SoftDeleteableInterface.

#### OrganizationInterface

This interface should be implemented by every organization.

#### TenantAwareInterface

This interface should be implemented by models associated with a specific tenant.

### Service Interfaces

#### TenantContextInterface

This interface should be implemented by a service responsible for managing the currently used *Tenant*.

#### TenantProviderInterface

This interface should be implemented by a service responsible for providing all available tenants.

#### TenantResolverInterface

This interface should be implemented by a service responsible for resolving the current *Tenant*.

### Repository Interfaces

#### TenantRepositoryInterface

This interface should be implemented by repositories responsible for storing the *Tenant* objects.

#### OrganizationRepositoryInterface

This interface should be implemented by repositories responsible for storing the *OrganizationInterface* objects.

### Factory Interfaces

#### TenantFactoryInterface

This interface should be implemented by tenant factories which are responsible for creating objects of type *TenantInterface*.

#### OrganizationFactoryInterface

This interface should be implemented by organization factories which are responsible for creating objects of type *OrganizationInterface*.

## 5.2.3 The Storage Component

This component provides basic models and interfaces to build persistence-agnostic storage.

### Installation

You can install the component in two different ways:

- *Install it via Composer* (`swp/storage` on Packagist);
- Use the official Git repository (https://github.com/SuperdeskWebPublisher/storage).

**Note:** This section is based on Symfony2 documentation.

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Symfony component.

### Usage

#### Creating storage-agnostic objects

The **Factory** allows you to create objects. Let's say you need to instantiate a new **Article** object, which you would want to persist in the persistence storage.

You could use the following code to do that:

```php
<?php

// ..
use SWP\Component\Storage\Factory\Factory;
use Acme\DemoBundle\Article;

$factory = new Factory(Article::class);
```

(continues on next page)

(continued from previous page)

```php
$article = $factory->create();

var_dump($article); // dumps Article object
```

By passing the fully qualified class name in Factory's construct we allow for a flexible way to create storage-agnostic objects. For example, depending on the persistence backend, such as PHPCR or MongoDB, you can instantiate new objects very easily using the same factory class.

### Creating storage-agnostic repositories

In some cases you would need to implement different repositories for your persistence backend. Let's say you are using Doctrine PHPCR and you want to have a generic way of adding and removing objects from the storage, even if you decide to change Doctrine PHPCR to Doctrine MongoDB or something else. By implementing **RepositoryInterface** in your new repository, you will be able to achive that.

Here's the example PHPCR repository implementation:

```php
<?php

namespace SWP\Bundle\StorageBundle\Doctrine\ODM\PHPCR;

use Doctrine\ODM\PHPCR\DocumentRepository as BaseDocumentRepository;
use SWP\Component\Storage\Model\PersistableInterface;
use SWP\Component\Storage\Repository\RepositoryInterface;

class DocumentRepository extends BaseDocumentRepository implements RepositoryInterface
{
    /**
     * {@inheritdoc}
     */
    public function add(PersistableInterface $object)
    {
        $this->dm->persist($object);
        $this->dm->flush();
    }

    /**
     * {@inheritdoc}
     */
    public function remove(PersistableInterface $object)
    {
        if (null !== $this->find($object->getId())) {
            $this->dm->remove($object);
            $this->dm->flush();
        }
    }
}
```

In this case, all objects that need to be persisted should implement **PersistableInterface** which extends Doctrine's default `Doctrine\Common\Persistence\ObjectRepository` interface. This component gives you simple interfaces to create storage-agnostic repositories.

### 5.2.4 The Bridge Component

This component provides tools which help to make a connection between Superdesk and Superdesk Web Publisher.

#### Installation

You can install the component in 2 different ways:

- *Install it via Composer* (`swp/bridge` on Packagist);
- Use the official Git repository (https://github.com/SuperdeskWebPublisher/bridge).

**Note:** This section is based on Symfony2 documentation.

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Symfony component.

#### Usage

#### Validators

This component provides one type of validator by default:

- ninjs validator

This is a simple implementation of JSON format validation against a concrete schema. There is also a custom implementation of the ninjs validator which validates the specific Superdesk ninjs schema.

#### Superdesk Ninjs Validator

This validator validates against specific Superdesk ninjs schema.

Usage:

```php
<?php
// example.php
// ..

use SWP\Component\Bridge\Validator\NinjsValidator;

$validator = new NinjsValidator();
if ($validator->isValid('{some ninjs format value}')) {
    // valid
}

// not valid
```

The Superdesk ninjs schema:

```
{
    "$schema": "http://json-schema.org/draft-03/schema#",
    "id" : "http://www.iptc.org/std/ninjs/ninjs-schema_1.1.json#",
    "type" : "object",
    "title" : "IPTC ninjs - News in JSON - version 1.1 (approved, 2014-03-12) /
↪document revision of 2014-11-15: geometry_* moved under place",
```

(continues on next page)

(continued from previous page)

```
    "description" : "A news item as JSON object -- copyright 2014 IPTC -␣
↪International Press Telecommunications Council - www.iptc.org - This document is␣
↪published under the Creative Commons Attribution 3.0 license, see  http://
↪creativecommons.org/licenses/by/3.0/  $$comment: as of 2014-03-13 ",
    "additionalProperties" : false,
    "patternProperties" : {
        "^description_[a-zA-Z0-9_]+" : {
            "description" : "A free-form textual description of the content of the␣
↪item. (The string appended to description_ in the property name should reflect the␣
↪format of the text)",
            "type" : "string"
        },
        "^body_[a-zA-Z0-9_]+" : {
            "description" : "The textual content of the news object. (The string␣
↪appended to body_ in the property name should reflect the format of the text)",
            "type" : "string"
        }
    },
    "properties" : {
        "guid" : {
            "description" : "The identifier for this news object",
            "type" : "string",
            "format" : "guid",
            "required" : true
        },
        "type" : {
            "description" : "The generic news type of this news object",
            "type" : "string",
            "enum" : ["text", "audio", "video", "picture", "graphic", "composite"]
        },
        "slugline" : {
            "description" : "The slugline",
            "type" : "string",
            "required" : true
        },
        "mimetype" : {
            "description" : "A MIME type which applies to this news object",
            "type" : "string"
        },
        "representationtype" : {
            "description" : "Indicates how complete this representation of a news␣
↪item is",
            "type" : "string",
            "enum" : ["complete", "incomplete"]
        },
        "profile" : {
            "description" : "An identifier for the kind of content of this news object
↪",
            "type" : "string"
        },
        "version" : {
            "description" : "The version of the news object which is identified by␣
↪the uri property",
            "type" : "string"
        },
        "versioncreated" : {
            "description" : "The date and time when this version of the news object␣
↪was created",
```

(continues on next page)

(continued from previous page)

```
            "type" : "string",
            "format" : "date-time"
        },
        "embargoed" : {
            "description" : "The date and time before which all versions of the news␣
↪object are embargoed. If absent, this object is not embargoed.",
            "type" : "string",
            "format" : "date-time"
        },
        "pubstatus" : {
            "description" : "The publishing status of the news object, its value is␣
↪*usable* by default.",
            "type" : "string",
            "enum" : ["usable", "withheld", "canceled"]
        },
        "urgency" : {
            "description" : "The editorial urgency of the content from 1 to 9. 1␣
↪represents the highest urgency, 9 the lowest.",
            "type" : "number"
        },
        "priority" : {
            "description" : "The editorial priority of the content from 1 to 9. 1␣
↪represents the highest priority, 9 the lowest.",
            "type" : "number"
        },
        "copyrightholder" : {
            "description" : "The person or organisation claiming the intellectual␣
↪property for the content.",
            "type" : "string"
        },
        "copyrightnotice" : {
            "description" : "Any necessary copyright notice for claiming the␣
↪intellectual property for the content.",
            "type" : "string"
        },
        "usageterms" : {
            "description" : "A natural-language statement about the usage terms␣
↪pertaining to the content.",
            "type" : "string"
        },
        "language" : {
            "description" : "The human language used by the content. The value should␣
↪follow IETF BCP47",
            "type" : "string"
        },
        "service" : {
            "description" : "A service e.g. World Photos, UK News etc.",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of a service",
                        "type" : "string"
                    },
                    "code" : {
```

(continues on next page)

(continued from previous page)

```
                        "description": "The code for the service in a scheme (= controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    }
                }
            }
        },
        "person" : {
            "description" : "An individual human being",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of a person",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news object to the person",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled vocabulary) which includes a code for the person",
                        "type" : "string",
                        "format" : "uri"
                    },
                    "code" : {
                        "description": "The code for the person in a scheme (= controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    }
                }
            }
        },
        "organisation" : {
            "description" : "An administrative and functional structure which may act as as a business, as a political party or not-for-profit party",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of the organisation",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news object to the organisation",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled vocabulary) which includes a code for the organisation",
```

(continues on next page)

(continued from previous page)

```
                        "type" : "string",
                        "format" : "uri"
                    },
                    "code" : {
                        "description": "The code for the organisation in a scheme (=␣
↪controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    },
                    "symbols" : {
                        "description" : "Symbols used for a finanical instrument␣
↪linked to the organisation at a specific market place",
                        "type" : "array",
                        "items" : {
                            "type" : "object",
                            "additionalProperties" : false,
                            "properties" : {
                                "ticker" : {
                                    "description" : "Ticker symbol used for the␣
↪financial instrument",
                                    "type": "string"
                                },
                                "exchange" : {
                                    "description" : "Identifier for the marketplace␣
↪which uses the ticker symbols of the ticker property",
                                    "type" : "string"
                                }
                            }
                        }
                    }
                }
            }
        },
        "place" : {
            "description" : "A named location",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "patternProperties" : {
                    "^geometry_[a-zA-Z0-9_]+" : {
                        "description" : "An object holding geo data of this place.␣
↪Could be of any relevant geo data JSON object definition.",
                        "type" : "object"
                    }
                },
                "properties" : {
                    "name" : {
                        "description" : "The name of the place",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news␣
↪object to the place",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled␣
↪vocabulary) which includes a code for the place",
```

(continues on next page)

(continued from previous page)

```
                        "type" : "string",
                        "format" : "uri"
                    },
                    "qcode" : {
                        "description": "The code for the place in a scheme (=␣
↪controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    },
                    "state" : {
                        "description" : "The state for the place",
                        "type" : "string"
                    },
                    "group" : {
                        "description" : "The place group",
                        "type" : "string"
                    },
                    "name" : {
                        "description" : "The place name",
                        "type" : "string"
                    },
                    "country" : {
                        "description" : "The country name",
                        "type" : "string"
                    },
                    "world_region" : {
                        "description" : "The world region",
                        "type" : "string"
                    }
                }
            }
        },
        "subject" : {
            "description" : "A concept with a relationship to the content",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of the subject",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news␣
↪object to the subject",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled␣
↪vocabulary) which includes a code for the subject",
                        "type" : "string",
                        "format" : "uri"
                    },
                    "code" : {
                        "description": "The code for the subject in a scheme (=␣
↪controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
```

(continues on next page)

(continued from previous page)

```
                    }
                }
            }
        },
        "event" : {
            "description" : "Something which happens in a planned or unplanned manner
↪",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of the event",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news␣
↪object to the event",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled␣
↪vocabulary) which includes a code for the event",
                        "type" : "string",
                        "format" : "uri"
                    },
                    "code" : {
                        "description": "The code for the event in a scheme (=␣
↪controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    }
                }
            }
        },
        "object" : {
            "description" : "Something material, excluding persons",
            "type" : "array",
            "items" : {
                "type" : "object",
                "additionalProperties" : false,
                "properties" : {
                    "name" : {
                        "description" : "The name of the object",
                        "type" : "string"
                    },
                    "rel" : {
                        "description" : "The relationship of the content of the news␣
↪object to the object",
                        "type" : "string"
                    },
                    "scheme" : {
                        "description" : "The identifier of a scheme (= controlled␣
↪vocabulary) which includes a code for the object",
                        "type" : "string",
                        "format" : "uri"
                    },
```

(continues on next page)

(continued from previous page)

```
                    "code" : {
                        "description": "The code for the object in a scheme (= controlled vocabulary) which is identified by the scheme property",
                        "type" : "string"
                    }
                }
            }
        },
        "byline" : {
            "description" : "The name(s) of the creator(s) of the content",
            "type" : "string"
        },
        "headline" : {
            "description" : "A brief and snappy introduction to the content, designed to catch the reader's attention",
            "type" : "string"
        },
        "located" : {
            "description" : "The name of the location from which the content originates.",
            "type" : "string"
        },
        "renditions" : {
            "description" : "Wrapper for different renditions of non-textual content of the news object",
            "type" : "object",
            "additionalProperties" : false,
            "patternProperties" : {
                "^[a-zA-Z0-9]+" : {
                    "description" : "A specific rendition of a non-textual content of the news object.",
                    "type" : "object",
                    "additionalProperties" : false,
                    "properties" : {
                        "href" : {
                            "description" : "The URL for accessing the rendition as a resource",
                            "type" : "string",
                            "format" : "uri"
                        },
                        "mimetype" : {
                            "description" : "A MIME type which applies to the rendition",
                            "type" : "string"
                        },
                        "title" : {
                            "description" : "A title for the link to the rendition resource",
                            "type" : "string"
                        },
                        "height" : {
                            "description" : "For still and moving images: the height of the display area measured in pixels",
                            "type" : "number"
                        },
                        "width" : {
                            "description" : "For still and moving images: the width of the display area measured in pixels",
```

(continues on next page)

(continued from previous page)

```
                                "type" : "number"
                            },
                            "sizeinbytes" : {
                                "description" : "The size of the rendition resource in␣
↪bytes",
                                "type" : "number"
                            }
                        }
                    }
                }
            },
            "associations" : {
                "description" : "Content of news objects which are associated with this␣
↪news object.",
                "type" : "object",
                "additionalProperties" : false,
                "patternProperties" : {
                    "^[a-zA-Z0-9]+" :  { "$ref": "http://www.iptc.org/std/ninjs/ninjs-
↪schema_1.0.json#" }
                }
            }
        }
}
```

## Validator Chain

You could also use Validator Chain to validate the json value with many validators at once:

```php
<?php
// example.php
// ..

use SWP\Component\Bridge\ValidatorChain;
use SWP\Component\Bridge\Validator\NinjsValidator;

$validatorChain = ValidatorChain();
$validatorChain->addValidator(new NinjsValidator(), 'ninjs');
$validatorChain->addValidator(new CustomValidator(), 'custom');

if ($validatorChain->isValid('{json value}')) {
    // valid
}

// not valid
```

## Data Transformers

Transformers are meant to transform an incoming value to an object representation.

This component supports one transformer by default:

- JSON to Package

### JSON to Package Data Transformer

This transforms a JSON string which is first validated by the Validator Chain. If the validation is a success, it serializes the JSON value to a `Package` object.

The `Package` object is a one-to-one representation of Superdesk Package.

Usage:

```php
<?php
// example.php
// ..

use SWP\Component\Bridge\Transformer\JsonToPackageTransformer;

$transformer = new JsonToPackageTransformer();
$package = $transformer->transform('{json value}');

var_dump($package);die; // will dump an instance of␣
↪``SWP\Component\Bridge\Model\Package`` object.
```

This transformer could support reverse transform, but it is not supported at the moment.

The example below will throw an `SWP\Component\Bridge\Exception\MethodNotSupportedException` exception:

```php
<?php
// example.php
// ..

use SWP\Component\Bridge\Model\Package;
use SWP\Component\Bridge\Transformer\JsonToPackageTransformer;

$package = new Package();
// ..
$transformer = new JsonToPackageTransformer();
$package = $transformer->reverseTransform($package);
```

**Note:** If the transformation fails for some reason, an exception `SWP\Component\Bridge\Exception\TransformationFail` will be thrown.

### Data Transformer Chain

You can use Transformer Chain to transform any value by many transformers at once:

```php
<?php
// example.php
// ..

use SWP\Component\Bridge\Transformer\DataTransformerChain;
use SWP\Component\Bridge\Transformer\JsonToPackageTransformer;

$transformerChain = DataTransformerChain(new JsonToPackageTransformer(), /* new␣
↪CustomTransformer() */);
$result = $transformer->transform(/* some value or object */);
```

(continues on next page)

(continued from previous page)

```

var_dump($result); // result of transformation
```

**Note:** If the transformation fails for some reason an exception `SWP\Component\Bridge\Exception\TransformationFail` will be thrown.

To reverse transform, use the `reverseTransform` method:

```
<?php
// example.php
// ..

use SWP\Component\Bridge\Transformer\DataTransformerChain;
use SWP\Component\Bridge\Transformer\JsonToPackageTransformer;

$transformerChain = DataTransformerChain(new JsonToPackageTransformer(), /* new␣
↪CustomTransformer() */);
$result = $transformer->reverseTransform(/* some value or object */);

var_dump($result); // result of transformation
```

### 5.2.5 The Templates System Component

This component provides useful features for templates and themes systems based on Twig templating engine.

#### Installation

You can install the component in two different ways:

- *Install it via Composer* (`swp/templates-system` on Packagist);
- Use the official Git repository (https://github.com/SuperdeskWebPublisher/templates-system) and autoload files manually.

#### Features

#### Containers and widgets

Containers are intended to give editors more control over templates. Properly implemented, they can transform a theme.

#### What is a Container?

A template file can optionally have one or more containers, *block* elements that can be overriden in specific places. For example, article sidebar content, footer, or front page content blocks. This same container can be placed in many different templates, or even many times in the same template.

Every container can have default parameters and content, and can be hidden when not needed. The container twig tag keeps HTML syntax always up to date with JavaScript live management expectations. Container default values can be overridden by widgets.

### What is a Widget?

Widgets can be attached to a container in any order. Many types of widget can represent different features, for example:

- Newsletter signup form
- Social media components, like a page widget or comments widget
- Simple HTML widget with your own custom HTML rendered by widget
- Airtime player widget

### How to create a new type of widget?

To create a new type of widget, you create a new class in `/src/SWP/Component/TemplatesSystem/Gimme/Widget` which extends the `AbstractWidgetHandler.php` in that same directory.

As well as having to implement the render function, you can define what parameters you're expecting in the widget model by adding a static variable to your class called `$expectedParameters`.

For example:

```
protected static $expectedParameters = array(
    'parameter_name' => [
        'type' => 'string',            // or bool, int, float
        'default' => 'default_value'   // if no default is provided, the parameter␣
↪must be set
    ]
);
```

## Context and Meta objects

### Why Meta objects?

Meta objects provides extra layer between your internal documents/entities and this what is available for theme developer (Templator). Thanks to this feature you can make more changes in Your project code and data structures without breaking templates.

### How to create Meta?

Every Meta object requires `Context`, `Value` and `Configuration`.

- Context - it's special service class used for collecting all meta's and resolving meta objects inside other meta's
- Value - object or array with data
- Configuration - array with configuration definitions for provided object.

Create `Meta` manually:

```
<?php

...
use SWP\Component\TemplatesSystem\Gimme\Meta\Meta;

return new Meta($context, $value, $configuration);
```

Create `Meta` with `MetaFactory`:

```php
<?php

...
use SWP\Component\TemplatesSystem\Gimme\Factory\MetaFactory;
use SWP\Component\TemplatesSystem\Gimme\Meta\Meta;

$metaFactory = new MetaFactory($context);

return $metaFactory->create($value, $configuration);
```

### What is Context?

`Context` is a special service class used for collecting all meta's and resolving meta objects inside other meta's. It can collect all available configurations for meta's, and convert provided objects into meta's when there is configuration for it.

**Note:** When property of `Meta` object can be itself a `Meta` instance (there is configuration for it) Context will automatically process it.

Example yaml configuration file for object (context can read config from `.yml` files).

```yaml
name: article
class: "SWP\\Component\\TemplatesSystem\\Tests\\Article"
description: Article Meta is representation of Article in Superdesk Web Publisher.
properties:
    title:
        description: "Article title, max 160 characters, can't have any html tags"
        type: text
    keywords:
        description: "Article keywords"
        type: text
to_string: title
```

**Note:** Configurations are used to manually expose properties from provided data, and create documentation for templators.

All objects passed to template should be Meta's.

### Loading Meta from template

### Meta Loaders

`Meta Loader` provides easy way for fetching data directly from template file. Loader can return single meta or collection of meta's (with `MetaCollection` class).

Library provides `ChainLoader` class witch can simplify work with many loaders.

`Meta Loader` must implement Loader Interface.

## How to load single Meta?

### gimme

The tag `gimme` has one required parameter and one optional parameter:

- (required) Meta object type (and name of variable available inside block), for example: *article*
- (optional) Keword `with` and parameters for Meta Loader, for example: `{ param: "value" }`

```
{% gimme article %}
    {# article Meta will be available under "article" variable inside block #}
    {{ article.title }}
{% endgimme %}
```

Meta Loaders sometimes require special parameters - like the article number, language of the article, user id, etc..

```
{% gimme article with { articleNumber: 1 } %}
    {# Meta Loader will use provided parameters to load article Meta #}
    {{ article.title }}
{% endgimme %}
```

## How to load Meta collection?

### gimmelist

The `gimmelist` tag has two required parameters and two optional parameters:

- (required) Name of variable available inside block: `article`
- (required) Keyword `from` and type of requested Metas in collection: `from articles` with filters passed to Meta Loader as extra parameters (`start`, `limit`, `order`)
- (optional) Keyword `with` and parameters for Meta Loader, for example: `with {foo: 'bar', param1: 'value1'}`
- (optional) Keyword `without` and parameters for Meta Loader, for example: `without {source: 'AAP'}`
- (optional) Keyword `if` and expression used for results filtering
- (optional) Keyword `ignoreContext` and optional array of selected meta to be ignored

Example of the required parameters:

```
{% gimmelist article from articles %}
    {{ article.title }}
{% endgimmelist %}
```

Example with ignoring selected context parameters:

```
{% gimmelist article from articles ignoreContext ['route', 'article'] %}
...
```

Example with ignoring whole context

```
{% gimmelist article from articles ignoreContext [] %}
...
```

Or even without empty array

```
{% gimmelist article from articles ignoreContext %}
...
```

Example with filtering articles by metadata:

```
{% gimmelist article from articles with {metadata: {byline: "Karen Ruhiger", located:
↪"Sydney"}} %}
    {{ article.title }}
{% endgimmelist %}
```

The above example will list all articles by metadata which contain `byline` equals to `Karen Ruhiger` **AND** `located` equals to `Sydney`.

To list articles by authors you can also do:

```
{% gimmelist article from articles with {author: ["Karen Ruhiger", "Doe"]} %}
    {{ article.title }}
    Author(s): {% for author in article.authors %}<img src="{{ url(author.avatar) }}"␣
↪/>{{ author.name }} ({{ author.role }}) {{ author.biography }} - {{ author.jobTitle.
↪name }},{% endfor %}
{% endgimmelist %}
```

It will then list all articles written by `Karen Ruhiger` **AND** `Doe`.

To list articles from the `Forbes` source but without an `AAP` source you can also do:

```
{% gimmelist article from articles with {source: ["Forbes"]} without {source: ["AAP"]}
↪ %}
    {% for source in article.sources %} {{ source.name }} {% endfor %}
{% endgimmelist %}
```

It will then list all articles **with** source `Forbes` and **without** `AAP`.

Listing article's custom fields:

```
{% gimmelist article from articles %}
    {{ article.title }}
    {{ article.extra['my-custom-field'] }}
{% endgimmelist %}
```

Example with usage of all parameters:

```
{% gimmelist article from articles|start(0)|limit(10)|order('id', 'desc')
    with {foo: 'bar', param1: 'value1'}
    contextIgnore ['route', 'article']
    if article.title == "New Article 1"
%}
    {{ article.title }}
{% endgimmelist %}
```

## Custom Twig tags

Gimme allows you to fetch the Meta object you need in any place of your template. It supports single Meta objects (with `gimme` ) and collections of Meta objects (with `gimmelist`).

### container

The `container` tag has one required parameter and one optional parameter:

- (required) container unique name, for example: *frontpage_sidebar*
- (optional) keyword `with` and default parameters for containers (used to create the container on theme installation).

Here is an example of a container tag:

```
{% container 'frontpage_sidebar' with {
    'width': 400,
    'height': 500,
    'styles': 'border: solid 1px red',
    'cssClass': 'css_class_name',
    'data': {'custom-key': value}
} %}
{% endcontainer %}
```

This container tag will render the HTML code:

```
<div id="frontpage_sidebar" class="swp_container css_class_name" style="width: 400px;␣
↪height: 500px; border: solid 1px red;" data-custom-key="value"></div>
```

The available container parameters are:

- [integer] width - container width
- [integer] height - container height
- [string] styles - container inline styles
- [string] cssClass - container class string
- [string] data - JSON object string with html-data properties (keys and values)

### gimme

The tag `gimme` has one required parameter and one optional parameter:

- (required) Meta object type (and name of variable available inside block), for example: *article*
- (optional) Keword `with` and parameters for Meta Loader, for example: `{ param:  "value" }`

```
{% gimme article %}
    {# article Meta will be available under "article" variable inside block #}
    {{ article.title }}
{% endgimme %}
```

Meta Loaders sometimes require special parameters - like the article number, language of the article, user id, etc..

```
{% gimme article with { articleNumber: 1 } %}
    {# Meta Loader will use provided parameters to load article Meta #}
    {{ article.title }}
{% endgimme %}
```

### gimmelist

The `gimmelist` tag has two required parameters and two optional parameters:

- (required) Name of variable available inside block: `article`
- (required) Keyword `from` and type of requested Metas in collection: `from articles` with filters passed to Meta Loader as extra parameters (`start`, `limit`, `order`)
- (optional) Keyword `with` and parameters for Meta Loader, for example: `with {foo: 'bar', param1: 'value1'}`
- (optional) Keyword `without` and parameters for Meta Loader, for example: `without {source: 'AAP'}`
- (optional) Keyword `if` and expression used for results filtering
- (optional) Keyword `ignoreContext` and optional array of selected meta to be ignored

Example of the required parameters:

```
{% gimmelist article from articles %}
    {{ article.title }}
{% endgimmelist %}
```

Example with ignoring selected context parameters:

```
{% gimmelist article from articles ignoreContext ['route', 'article'] %}
...
```

Example with ignoring whole context

```
{% gimmelist article from articles ignoreContext [] %}
...
```

Or even without empty array

```
{% gimmelist article from articles ignoreContext %}
...
```

Example with filtering articles by metadata:

```
{% gimmelist article from articles with {metadata: {byline: "Karen Ruhiger", located:
↪"Sydney"}} %}
    {{ article.title }}
{% endgimmelist %}
```

The above example will list all articles by metadata which contain `byline` equals to `Karen Ruhiger` **AND** `located` equals to `Sydney`.

To list articles by authors you can also do:

```
{% gimmelist article from articles with {author: ["Karen Ruhiger", "Doe"]} %}
    {{ article.title }}
    Author(s): {% for author in article.authors %}<img src="{{ url(author.avatar) }}"␣
↪/>{{ author.name }} ({{ author.role }}) {{ author.biography }} - {{ author.jobTitle.
↪name }},{% endfor %}
{% endgimmelist %}
```

It will then list all articles written by `Karen Ruhiger` **AND** `Doe`.

To list articles from the `Forbes` source but without an `AAP` source you can also do:

```
{% gimmelist article from articles with {source: ["Forbes"]} without {source: ["AAP"]}
↪ %}
    {% for source in article.sources %} {{ source.name }} {% endfor %}
{% endgimmelist %}
```

It will then list all articles **with** source `Forbes` and **without** `AAP`.

Listing article's custom fields:

```
{% gimmelist article from articles %}
    {{ article.title }}
    {{ article.extra['my-custom-field'] }}
{% endgimmelist %}
```

Example with usage of all parameters:

```
{% gimmelist article from articles|start(0)|limit(10)|order('id', 'desc')
    with {foo: 'bar', param1: 'value1'}
    contextIgnore ['route', 'article']
    if article.title == "New Article 1"
%}
    {{ article.title }}
{% endgimmelist %}
```

### How to work with `gimmelist` pagination?

`gimmelist` is based on Twig `for` tag, like in Twig there is loop variable available. In addition to default loop properties there is also `totalLength`. It's filled by loader with number of total elements in storage which are matching criteria. Thanks to this addition we can build real pagination.

`TemplateEngine` Bundle provides simple default pagination template file: `pagination.html.twig`.

**Note:** You can override that template with `SWPTemplatesSystemBundle/views/pagination.html.twig` file in Your theme. Or You can use own file used for pagination rendering.

Here is commented example of pagination:

```
{# Setup list and pagination parameters #}
{% set itemsPerPage, currentPage = 1, app.request.get('page', 1) %}
{% set start = (currentPage / itemsPerPage) - 1 %}

{# List all articles from route '/news' and limit them to `itemsPerPage` value␣
↪starting from `start` value #}
{% gimmelist article from articles|start(start)|limit(itemsPerPage) with {'route': '/
↪news'} %}

    <li><a href="{{ url(article) }}">{{ article.title }} </a></li>

    {# Render pagination only at end of list #}
    {% if loop.last  %}
        {#
```

(continues on next page)

(continued from previous page)

```
            Use provided by default pagination template

            Parameters:
            * currentFilters (array) : associative array that contains the current␣
↪route-arguments
            * currentPage (int) : the current page you are in
            * paginationPath (Meta|string) : the route name (or supported by router␣
↪Meta object) to use for links
            * lastPage (int) : represents the total number of existing pages
            * showAlwaysFirstAndLast (bool) : Always show first and last link (just␣
↪disabled)
        #}
        {% include '@SWPTemplatesSystem/pagination.html.twig' with {
            currentFilters: {}|merge(app.request.query.all()),
            currentPage: currentPage,
            paginationPath: gimme.route,
            lastPage: (loop.totalLength/itemsPerPage)|round(1, 'ceil'),
            showAlwaysFirstAndLast: true
        } only %}
    {% endif %}
{% endgimmelist %}
```

For referrence, see original *pagination.html.twig* template (if you want to customize it and use instead of default one):

```
{#
  Source: http://dev.dbl-a.com/symfony-2-0/symfony2-and-twig-pagination/
  Updated by: Simon Schick <simonsimcity@gmail.com>

  Parameters:
    * currentFilters (array) : associative array that contains the current route-
↪arguments
    * currentPage (int) : the current page you are in
    * paginationPath (string) : the route name to use for links
    * showAlwaysFirstAndLast (bool) : Always show first and last link (just disabled)
    * lastPage (int) : represents the total number of existing pages
#}
{% spaceless %}
    {% if lastPage > 1 %}

        {# the number of first and last pages to be displayed #}
        {% set extremePagesLimit = 3 %}

        {# the number of pages that are displayed around the active page #}
        {% set nearbyPagesLimit = 2 %}

        <nav class="pagination">
            <div class="numbers">
                <ul>
                    {% if currentPage > 1 %}
                        <li><a href="{{ path(paginationPath, currentFilters|merge(
↪{page: currentPage-1})) }}">Previous</a></li>

                        {% for i in range(1, extremePagesLimit) if ( i < currentPage -
↪ nearbyPagesLimit ) %}
                            <li><a href="{{ path(paginationPath, currentFilters|merge(
↪{page: i})) }}">{{ i }}</a></li>
                        {% endfor %}
```

(continues on next page)

(continued from previous page)

```

                        {% if extremePagesLimit + 1 < currentPage - nearbyPagesLimit %}
                            <span class="sep-dots">...</span>
                        {% endif %}

                        {% for i in range(currentPage-nearbyPagesLimit, currentPage-1) if ( i > 0 ) %}
                            <li><a href="{{ path(paginationPath, currentFilters|merge({page: i})) }}">{{ i }}</a></li>
                        {% endfor %}
                    {% elseif showAlwaysFirstAndLast %}
                        <span class="disabled">Previous</span>
                    {% endif %}

                    <li class="current"><a href="{{ path(paginationPath, currentFilters|merge({ page: currentPage })) }}">{{ currentPage }}</a></li>

                    {% if currentPage < lastPage %}
                        {% for i in range(currentPage+1, currentPage + nearbyPagesLimit) if ( i <= lastPage ) %}
                            <li><a href="{{ path(paginationPath, currentFilters|merge({page: i})) }}">{{ i }}</a></li>
                        {% endfor %}

                        {% if  (lastPage - extremePagesLimit) > (currentPage + nearbyPagesLimit) %}
                            <li><span class="sep-dots">...</span></li>
                        {% endif %}

                        {% for i in range(lastPage - extremePagesLimit+1, lastPage) if ( i > currentPage + nearbyPagesLimit ) %}
                            <li><a href="{{ path(paginationPath, currentFilters|merge({page: i})) }}">{{ i }}</a></li>
                        {% endfor %}

                        <li><a href="{{ path(paginationPath, currentFilters|merge({page: currentPage+1})) }}">Next</a></li>
                    {% elseif showAlwaysFirstAndLast %}
                        <li><span class="disabled">Next</span></li>
                    {% endif %}
                </ul>
            </div>
        </nav>
    {% endif %}
{% endspaceless %}
```

### How to work with Meta objects

On the template level, every variable in Context and fetched by `gimme` and `gimmelist` is a representation of Meta objects.

**dump**

```
{{ dump(article) }}
```

**print**

```
{{ article }}
```

If the meta configuration has the `to_string` property then the value of this property will be printed, otherwise it will be represented as JSON.

**access property**

```
{{ article.title }}
{{ article['title']}}
```

**generate url**

```
{{ url(article) }}    // absolute url
{{ path(article) }}   // relative path
```

Here's an example using gimmelist:

```
{% gimmelist article from articles %}
    <li><a href="{{ url(article) }}">{{ article.title }} </a></li>
{% endgimmelist %}
```

### 5.2.6 The Rule Component

This component provides basic business rules engine tools.

#### Installation

You can install the component in two different ways:

- *Install it via Composer* (`swp/rule` on Packagist);
- Use the official Git repository (https://github.com/SuperdeskWebPublisher/rule).

---

**Note:** This section is based on Symfony2 documentation.

---

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Symfony component.

#### Usage

#### Using Rule Applicator Chain Service

The Rule Applicator Chain service is used to execute all rule applicators added to the chain.

Usage:

```
<?php
 // example.php
 // ..

 use Acme\DemoBundle\Model\Subject;
```

(continues on next page)

(continued from previous page)

```php
use SWP\Component\Rule\Applicator\RuleApplicatorChain;
use SWP\Component\Rule\Model\Rule;
// ..

$applicatorChain = new RuleApplicatorChain();
$applicatorChain->addApplicator(/* instance of RuleApplicatorInterface */)

$subject = new Subject(); // an instance of RuleSubjectInterface
$applicatorChain->isSupported($subject); // return true or false

$rule = new Rule();
// ..

$applicatorChain->apply($rule, $subject);
```

#### What is Rule Evaluator?

Rule evaluators are used to evaluate rule on an given object and make sure it matches the rule's criteria. By default, the Symfony Expression Language Component is used which perfectly fits into the business rules engine concept.

#### Adding new Rule Evaluator

There is a possibility to create your custom implementation for Rule Evaluator. All you need to do is to create a new class and implement `SWP\Component\Rule\Evaluator\RuleEvaluatorInterface` interface.

#### What is Rule Processor?

Rule processor is responsible for processing all rules and apply them respectively to an object, based on the defined rule priority. The greater the priority value, the higher the priority.

Rule Processor implements `SWP\Component\Rule\Processor\RuleProcessorInterface` interface.

#### What is Rule Applicator?

Rule applicators are used to apply given rule's configuration to an object. You create your custom rule applicators and register them in Rule Applicator Chain service which triggers `apply` method on them if the given applicator is supported by the rule subject.

#### Adding new Rule Applicator

There is a possibility to create your custom implementation for Rule Applicator. All you need to do is to create a new class and implement `SWP\Component\Rule\Applicator\RuleApplicatorInterface` interface.

## 5.2.7 The Content List Component

This component provides basic classes to build dynamic and flexible content lists.

### Installation

You can install the component in two different ways:

- *Install it via Composer* (`swp/content-list` on Packagist);
- Use the official Git repository (https://github.com/SuperdeskWebPublisher/content-list).

**Note:** This section is based on Symfony2 documentation.

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Symfony component.

### Models

### ContentList

Every content list is represented by a **ContentList** model which by default has the following properties:

| Method | Description |
|---|---|
| id | Unique identifier |
| description | List's description |
| name | List's name |
| type | List's type (`automatic` or `manual`) |
| cacheLifeTime | List cache life time in seconds |
| limit | List limit |
| items | Collection of list items |
| enabled | Indicates whether the list is enabled |
| createdAt | Date of creation |
| updatedAt | Date of last update |
| deletedAt | Indicates whether the list is deleted |

**Note:** This model implements `SWP\Component\ContentList\Model\ContentListInterface`.

### ContentListItem

Every content list item is represented by a **ContentListItem** model which by default has the following properties:

| Method | Description |
|---|---|
| id | Unique identifier |
| position | List item position |
| content | Object of type `ListContentInterface` |
| enabled | Indicates whether the item is enabled |
| sticky | Defines whether content is sticky or not |
| filters | An array of list criteria/filters |
| createdAt | Date of creation |
| updatedAt | Date of last update |
| deletedAt | Indicates whether the item is deleted |

**Note:** Read more about *ListContentInterface*.

**Note:** This model implements `SWP\Component\ContentList\Model\ContentListItemInterface`.

### Repository Interface

This component contains `SWP\Component\ContentList\Repository\ContentListRepositoryInterface` interface which can be used by your custom entity repository, in order to make it working with *ContentListBundle*.

## 5.2.8 The Output Channel Component

This component provides the implementation of output channels.

### Installation

You can install the component in two different ways:

- *Install it via Composer* (`swp/output-channel` on Packagist);
- Use the official Git repository (https://github.com/SuperdeskWebPublisher/OutputChannel).

**Note:** This section is based on Symfony2 documentation.

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Symfony component.

### Models

### OutputChannel

Every output channel is represented by a **OutputChannel** model which by default has the following properties:

| Method | Description |
|---|---|
| id | Unique identifier |
| type | Output channel's type (`wordpress` etc.) |
| config | Output channel configuration per type |

**Note:** This model implements `SWP\Component\OutputChannel\Model\OutputChannelInterface`.

### Output Channel Aware Interface

This component contains `SWP\Component\OutputChannel\Model\OutputChannelAwareInterface` interface to make your custom entity/model output channel aware.

## 5.2.9 The Paywall Component

This component provides the paywall implementation.

### Installation

You can install the component in two different ways:

- *Install it via Composer* (`swp/paywall` on Packagist);
- Use the official Git repository (https://github.com/SuperdeskWebPublisher/Paywall).

**Note:** This section is based on Symfony2 documentation.

Then, require the `vendor/autoload.php` file to enable the autoloading mechanism provided by Composer. Otherwise, your application won't be able to find the classes of this Symfony component.

### Models

### Subscription

Every subscription is represented by a **Subscription** model which by default has the following properties:

| Method | Description |
|---|---|
| id | Unique identifier |
| type | Subscription's type (`recurring` etc.) |
| details | Subscription details |
| code | Subscription's unique code |
| active | Subscription's status |
| updatedAt | Subscription updated at datetime |
| createdAt | Subscription created at datetime |

**Note:** This model implements `SWP\Component\Paywall\Model\SubscriptionInterface`.

### Interfaces

### Subscriber Interface

This component contains `SWP\Component\Paywall\Model\SubscriberInterface` interface which should be implemented by your user class.

### Paywall Secured Interface

This component contains `SWP\Component\Paywall\Model\PaywallSecuredInterface` interface which should be implemented by classes that must be flagged as "paywall secured".

#### Paywall Secured Trait

This component contains `SWP\Component\Paywall\Model\PaywallSecuredTrait` trait which adds a special property along with getter and setter. By using this trait it is possible to flag your objects as paywall secured.

### Factories

#### Subscription Factory

A factory class to create new instances of `SWP\Component\Paywall\Model\Subscription` class.

| Method | Description |
|---|---|
| create() | Create a new instance of Subscription |

### Adapters

Adapters are used to interact with the external subscription systems to fetch existing subscriptions.

#### PaymentsHub Adapter

This adapter interacts with the Payments Hub API to fetch the subscriptions data.

**Note:** This model implements `SWP\Component\Paywall\Adapter\PaywallAdapterInterface`.

#### Paywall Adapter Interface

This interface should be implemented by your custom adapter.

## 5.3 (Un)publishing content

Deciding if content is visible for user in Publisher is done dynamically - based on user roles.

**Note:** By default all users can see only published content, but if user have role `ROLE_CAN_VIEW_NON_PUBLISHED` then also not published content will be fetched and rendered.

### 5.3.1 How published content is marked in database?

Published content need to match those criteria:

- `status` property set to `published`
- `publishedAt` property filled with date and time
- `isPublishable` property set to `true`

### 5.3.2 How to Publish content?

*By REST API call*

```
curl -X "PATCH" -d "article[status]=published" -H "Content-type:\ application/x-www-
↪form-urlencoded" /api/v1/content/articles/get-involved
```

*By code*

```
// $this->container - instance of Service Container
// $article - ArticleInterface implementation
$articleService = $this->container->get('swp.service.article');
$articleService->publish($article);
```

### 5.3.3 How to check (in code) if content is published?

```
use Symfony\Cmf\Bundle\CoreBundle\PublishWorkflow\PublishWorkflowChecker;
....
$publishWorkflowChecker = $this->serviceContainer->get('swp.publish_workflow.checker
↪');

if ($publishWorkflowChecker->isGranted(PublishWorkflowChecker::VIEW_ATTRIBUTE,
↪$article)) {
    // give access for article
}
```

**Note:** Content assigned to routes is automatically checked (if it's publishable for current user) by event listener.

### 5.3.4 How to Un-publish Content?

If an article is published you can easily un-publish it via API as described above. Another way to un-publish already published article is by `killing` the article. It can be achieved by setting the value of `pubStatus` property to `canceled` in the JSON (Ninjs) content according to IPTC standards. Once that status will be set, and the content will be send to Publisher (/content/push API endpoint), article will be un-published immediately and its status will be set to `canceled`.

## 5.4 Publisher API

### 5.4.1 API authentication

Internal API endpoints require user authentication (user need to have `ROLE_INTERNAL_API` role assigned).

Authentication data (token) must be attached to every request with `Authorization` header or `auth_token` query parameter.

#### Get authentication token for registered user

To get authentication token you need to call `/api/v1/auth` with your `username` and `password` - in response you will get your user information's and token data.

Example:

```
curl 'http://publisher.dev/api/v1/auth' -d 'auth%5Busername%5D=username&auth
↪%5Bpassword%5D=password' --compressed
```

**Note:** Publisher token will be valid for 48 hours

## Get authentication token for superdesk user

To get authentication token you need to call `/api/v1/auth/superdesk` with superdesk legged in user `session_id` and `token` - in response you will get your user information's and token data.

Example:

```
curl 'http://publisher.dev/api/v1/auth/superdesk' -d 'auth_superdesk%5Bsession_id
↪%5D=5831599634d0c100405d84c7&auth_superdesk%5Btoken%5D=Basic␣
↪YTRmMWMzMTItODlkNS00MzQzLTkzYjctZWMyMmM5ZGMzYWEwOg==' --compressed
```

Publisher in background will ask authorized superdesk server for user session (and user data). If Superdesk will confirm session information then Publisher will get internal user (or create one if not exists) and create token for him.

**Note:** Publisher token will be this same as the one from superdesk (provided in `/api/v1/auth/superdesk` request).

## Generate Authentication URL for Livesite Editor

You can create with API special authentication URL for tenant website. To do that you need to call `/api/v1/livesite/auth/livesite_editor` as authorized user (with token in request header or url).

```
curl 'http://publisher.dev/api/v1/livesite/auth/livesite_editor' -H 'Authorization:␣
↪d6O3UorCHZ2Pd8PRs/0aXGg1qnT0bKUPWW43dgKqYm3CI4U4Og==' --compressed
```

In response you will get JSON with Your token details and special URL which can be used for authentication and Livesite Editor activation.

After following that url you will be redirected to tenant homepage. Meantime special cookie with name `activate_livesite_editor` will be set. This cookie will have API token set as it's value. It would best if you will set token value in browser local storage and remove cookie (so it will not be send to server with every request).

CHAPTER 6

# Contributing to Web Publisher

# Index