MassiveSearchBundle Documentation

Release stable

March 03, 2015

Contents

1 Installation **3**

2 Mapping **5**

3 Indexing **7**

4 Searching **9**

5 Commands **11**
5.1 massive:search:query 11
5.2 massive:search:status 11

6 Extending **13**
6.1 Factory 13
6.2 Metadata Drivers 13
6.3 Hit Listeners 14

The MassiveSearchBundle provides:

- An abstraction for search engine libraries.
- A way to map classes which you want to index.

By default it is configured to use the Zend Lucene library, which must be installed (see the *suggests* and `require-dev` sections in `composer.json`.

CHAPTER 1

Installation

You can install the MassiveSearchBundle by adding it to *composer.json*:

```
"require": {
    ...
    "massive/search-bundle": "0.1"
}
```

And then include it in your `AppKernel`:

```
class AppKernel
{
    public function registerBundles()
    {
        return array(
            // ...
            new \Massive\Bundle\SearchBundle\MassiveSearchBundle(),
        );
    }
}
```

You will also need to include a search library. The search libraries are listed in the `suggests` section of `composer.json`, and exact package versions can also be found in the `require-dev` section (as all the libraries are tested).

For example, to enable the ZendLucene search library:

```
"require": {
    ...
    "zendframework/zend-stdlib": "2.3.1 as 2.0.0rc5",
    "zendframework/zendsearch": "2.*",
}
```

Mapping

The MassiveSearchBundle requires that you define which objects should be indexed through *mapping*. Currently only **XML mapping** supported:

This mapping will cause the fields `title` and `body` to be indexed into an index named `product` using the ID obtained from the objects `id` field. (We use the Symfony PropertyAccess component, so it works on properties and methods alike).

Note:

- This file MUST be located in `YourBundle/Resources/config/massive-search`
- It must be named after the name of your class (without the namespace) e.g. `Product.xml`
- Your `Product` class MUST be located in one of the following folders: - `YourBundle/Document` - `YourBundle/Entity` - `YourBundle/Model`

Note: This is an early version of the bundle, it will support explict non-magic mapping in the future.

CHAPTER 3

Indexing

Once you have created your mapping files you can index your objects, for example after saving it.

The bundle provides the `massive_search.search_manager` object which is the only service which you will need to access.

```
$product = new Product();

// ... populate the product, persist it, whatever.

$searchManager = $this->get('massive_search.search_manager');
$searchManager->index($product);
```

The SearchManager will know from the mapping how to index the product, and it will be indexed using the configured search library adapter.

Note: The bundle automatically removes existing documents with the same ID. The ID mapping is mandatory.

CHAPTER 4

Searching

As with the indexing, searching for results is also done with the SearchManager.

Currently only supported by query string is supported. The query string is passed directly to the search library:

```
$hits = $searchManager->createSearch('My Product')->index('product')->execute();

foreach ($hits as $hit) {
    echo $hit->getScore();

    // @var Massive\Bundle\SearchBundle\Search\Document
    $document = $hit->getDocument();

    // retrieve the indexed documents "body" field
    $body = $document->getField('body');

    // retrieve the indexed ID of the document
    $body = $document->getId();
}
```

Commands

The MassiveBuildBundle provides some commands.

5.1 massive:search:query

Perform a query from the command line:

```
$ php app/console massive:search:query "Foobar" --index="barfoo"
+------------------+--------------------------------------+-----------+-------------+-----------+---
| Score            | ID                                   | Title     | Description | Url       | Cla
+------------------+--------------------------------------+-----------+-------------+-----------+---
| 0.53148467371593 | ac984681-ca92-4650-a9a6-17bc236f1830 | Structure |             | structure | Ove
+------------------+--------------------------------------+-----------+-------------+-----------+---
```

5.2 massive:search:status

Display status information for the current search implementation:

```
$ php app/console massive:search:status
+-------------+-------------------------------------------------------------+
| Field       | Value                                                       |
+-------------+-------------------------------------------------------------+
| Adapter     | Massive\Bundle\SearchBundle\Search\Adapter\ZendLuceneAdapter |
| idx:product | {"size":11825,"nb_files":36,"nb_documents":10}              |
+-------------+-------------------------------------------------------------+
```

CHAPTER 6

Extending

You can extend the bundle by customizing the Factory class and with custom metadata drivers.

6.1 Factory

The factory service can be customized, enabling you to instantiate your own classes for use in any listeners which you register. For example, you want to add a "thumbnail" field to the Document object.

```
namespace My\Namespace;

use Massive\Bundle\SearchBundle\Search\Factory as BaseFactory;

class MyFactory extends BaseFactory
{
    public function makeDocument()
    {
        return MyCustomDocument();
    }
}
```

You must then register your factory as a service and register the ID of that service in your main application configuration:

```
massive_search:
    services:
        factory: my.factory.service
```

6.2 Metadata Drivers

Simply extend the `Metadata\Driver\DriverInterface` and add the tag `massive_search.metadata.driver` tag to your implementations service definition.

```
<service id="massive_search.metadata.driver.xml" class="%massive_search.metadata.driver.xml.class%">
    <argument type="service" id="massive_search.metadata.file_locator" />
    <tag type="massive_search.metadata.driver" />
</service>
```

6.3 Hit Listeners

The `SearchManager` will fire an event of type `HitEvent` in the Symfony EventDispatcher named `massive_search.hit`.

The `HitEvent` contains the hit object and the reflection class of the object which was originally indexed.

For example:

```php
<?php

namespace Sulu\Bundle\SearchBundle\EventListener;

use Massive\Bundle\SearchBundle\Search\Event\HitEvent;

class HitListener
{
    public function onHit(HitEvent $event)
    {
        $reflection = $event->getDocumentReflection();
        if (false === $reflection->isSubclassOf('MyClass')) {
            return;
        }

        $document = $event->getDocument();
        $docuemnt->setUrl('Foo' . $document->getUrl());
    }
}
```