Shopware 6 Theme entwickeln – Snippets

Shopware 6 Theme entwickeln – Snippets

[expand title="mehr lesen…"]

To extend a language in Shopware 6 you can add your own snippets in your theme. You can also add a completely new language to Shopware 6.

General snippet structure

To organize your snippets you can add them to .json files, so structuring and finding snippets you want to change is very easy.

Adding snippets

You can add new snippets by adding files to the follow structure like this:

```
# move into your theme folder
$ cd custom/plugins/MyTheme

# structure of theme
└─ src
   ├─ Resources
   │  ├─ config
   │  │  └─ services.xml
   │  └─ snippet
   │     ├─ de_DE
```

```
    │     │  ├─ SnippetFile_de_DE.php
    │     │  └─ storefront.de-DE.json
    │     └─ en_GB
    │        ├─ SnippetFile_en_GB.php
    │        └─ storefront.en-GB.json
    └─ MyTheme.php
```

There is no explicit syntax for variables in the storefront. It is nevertheless recommended to encompass them with % symbols to be extra clear on what their purpose is. Pluralization works for any natural number. Just remember to explicitly define the intervals‘ amounts and ranges for that snippet.

Example of src/Resources/snippet/storefront.en-GB.json:

```
{
    "my-theme": {
        "productDetail": {
             "headLineText": "There are %count% discounts available for %product%:",
             "description": "Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam ..."
        },
        "cart": {
            "itemCounter": "{1} 1 discount item | ]1,Inf[ %count% discount items"
        }
    }
}
```

Adding Storefront snippets

Storefront snippets additionally require a class that extends the SnippetFileInterface. A suitable name would e.g. be SnippetFile_en_GB.php. Having created that file, you will have to implement the following five methods:

- getName: Returns the name of the snippet file as a string. By referring to this name, you can access the

translations later. It is **required** to use messages.en-GB, if you provide a whole new language. By default, an extension should call its Storefront extension storefront.en-GB. Otherwise a describing domain, like shopware's PayPal plugin using paypal.en-GB, is also okay.

- getPath: Each SnippetFile class has to point to the .json file, that contains the actual translations. Return its path here. We suggest using the name already chosen in getName for your file name.
- getIso: Return the ISO string of the supported locale here. This is important, because the Translator collects every snippet file with this locale and merges them to generate the snippet catalogue used by the storefront.
- getAuthor: Return your vendor name here. This can be used to distinguish your snippets from all the other available ones. The Administration snippet set module offers a filter, so users are able to easily find plugin specific snippets.
- isBase: Return true here, if your theme implements a whole new language, such as providing french snippets for the whole Shopware 6 system. Don't forget to watch your getName method then. Most of the time, you're just adding your own snippets to an existent language, then false will be your way to go.

Example for the language en-GB :

```
<?php declare(strict_types=1);

namespace MyTheme\Resources\snippet\en_GB;

use Shopware\Core\System\Snippet\Files\SnippetFileInterface;

class SnippetFile_en_GB implements SnippetFileInterface
{
    public function getName(): string
    {
        return 'storefront.en-GB';
```

```
    }

    public function getPath(): string
    {
        return __DIR__ . '/storefront.en-GB.json';
    }

    public function getIso(): string
    {
        return 'en-GB';
    }

    public function getAuthor(): string
    {
        return 'Enter developer name here';
    }

    public function isBase(): bool
    {
        return false;
    }
}
```

Registering your service

Lastly you have to register your snippet files by creating file src/Resources/config/services.xml.

Example:

```
<!-- src/Resources/config/services.xml -->
<?xml version="1.0" ?>

<container xmlns="http://symfony.com/schema/dic/services"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://symfony.com/schema/dic/services
http://symfony.com/schema/dic/services/services-1.0.xsd">

    <!-- Translations -->
    <services>
                                                <service
id="MyTheme\Resources\snippet\en_GB\SnippetFile_en_GB"
```

```
public="true">
            <tag name="shopware.snippet.file"/>
        </service>
    </services>

    <services>
                                                    <service
id="MyTheme\Resources\snippet\de_DE\SnippetFile_de_DE"
public="true">
            <tag name="shopware.snippet.file"/>
        </service>
    </services>
</container>
```

Using your snippets in your templates

With the trans filter you can use your snippets in the twig templates and they will be translated automatically. You also can pass values to replace the placeholders in the snippets.

```
<div class="product-detail-headline">
            {{ 'my-theme.productDetail.headLineText' |
trans({'%count%': count, '%product%': product}) }}
    </div>
```

[/expand]