i18n
Internationalization

SensioLabs

Basic concepts

Internationalization

The process of **abstracting strings** and other locale-specific pieces out of your application into a layer where they can be **translated** and **converted** based on the **user's locale**.

Locale

Locale = Language + Country

- ISO 639-1 defines language codes
  https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
- ISO-3166-1 alpha-2 defines country codes
  https://en.wikipedia.org/wiki/ISO_3166-1

Locale examples

	Language	Country
en_AU	English	Australia
en_GB	English	United Kingdom
en_US	English	United States

	Language	Country
fr_FR	French	France
fr_BE	French	Belgium

It's common for the **locale** to only define the first language part (**en**, **fr**, etc.)

Internationalization workflow

Workflow

1. Enable and configure translation.
2. Extract content strings.
3. Create/update translation files.
4. Manage user locale.

Steps **1** and **4** are one-time tasks. Steps **2** and **3** are repeated continuously as long as the application grows and evolves.

Configuration

Enable and configure translation

```yaml
# app/config/config.yml
framework:
    translator:
        fallbacks:  ['fr', 'en']
```

By default, **translation** is disabled to avoid any impact in the application performance.

If a content is not available in the current locale, it is translated into the **fallback locales** (you can define more than one).

Enable and configure translation

```
framework:
    translator:
        fallbacks:  ['fr', 'en']
```

Define the default locale

```yaml
# app/config/config.yml
framework:
    default_locale: 'en'
```

This is the **default locale** used when no locale is explicitly defined by the given user. You can only define one default locale which is applied to all users.

Complete translation configuration

```yaml
# app/config/config.yml
framework:
    default_locale: 'en'
    translator:
        fallbacks:  ['fr', 'en']
```

Extract content strings

Translating contents outside templates

```
public function indexAction()
{
    $title = $this->get('translator')
                  ->trans('Contact us');
}
```

If the user's locale is **fr_FR** and there is a catalogue of french translations, **$title** value will be **Contactez-nous**.

Translating template contents

```
{% trans %}
  Contact us
{% endtrans %}
```

Twig tags are mostly used to translate large blocks of static contents.

```
{{ 'Contact us'|trans }}
```

Twig filters are used to translate variables and expressions.

Main difference between filters and tags

```
{% trans %}
  <h1>Contact us</h1>
{% endtrans %}
```

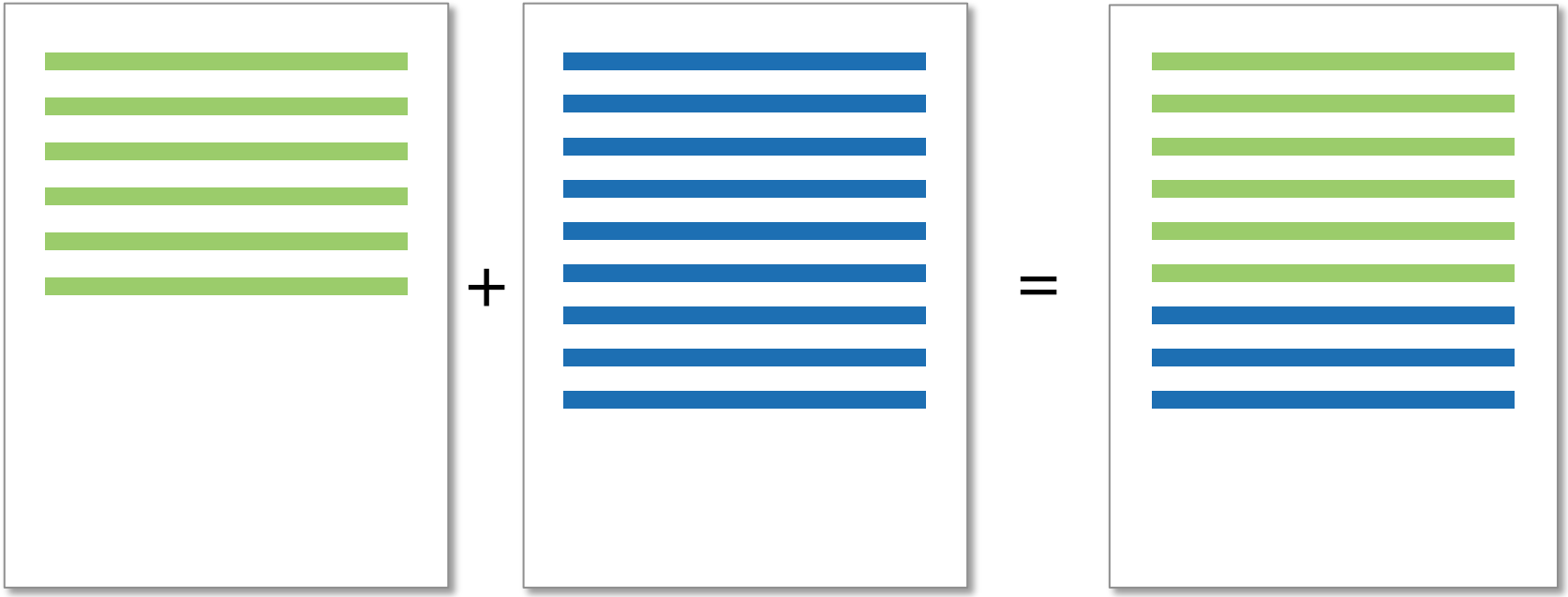
OUTPUT `<h1>Contactez-nous</h1>`

```
{{ '<h1>Contact us</h1>'|trans }}
```

OUTPUT `<h1>Contactez-nous</h1>`

Create translation files

How does Symfony get the translation



User's locale
translations

Fallback locale
translations

Complete translation file
used by Symfony

Translation files naming syntax

messages.fr_FR.xlf

Domain
(explained later)

Locale
(it's common to define just the language without the country)

File format
(XLIFF, YAML, PHP, PO/MO, etc.)

Translation files location

```
your-project/
├── app
│   └── Resources
│       ├── translations/
│       │   └── messages.fr.xlf
│       └── AcmeBlogBundle/
│           └── translations/
│               └── messages.fr.xlf
├── ...
└── src
    └── Acme/
        └── BlogBundle/
            └── Resources
                └── translations/
                    └── messages.fr.xlf
```

Symfony applies an **overriding mechanism** to select the catalogue to use.

This allows to override any bundle translation, including third-party bundles.

Translation files priority

`app/Resources/translations/messages.fr.xlf`

HIGHEST priority. It **OVERRIDES** any other catalogue with the same name and locale, regardless of where it's defined originally.

`app/Resources/AcmeBlogBundle/translations/messages.fr.xlf`

MEDIUM priority. It **OVERRIDES** any catalogue with the same name and locale defined by a bundle with the same name as this directory.

`src/Acme/BlogBundle/Resources/translations/messages.fr.xlf`

LOWEST priority. It can be **OVERRIDDEN** by any catalogue with the same name and locale defined in the **app/** directory.

The XLIFF translation format

- Symfony Best Practices recommend to use this format.
- **Pro**: it's the standard format in the translation industry.
- **Con**: it's very verbose (it's based on XML)

An example of XLIFF translation file

```
<!-- app/Resources/translations/messages.fr_FR.xlf -->
<?xml version="1.0" encoding="utf-8"?>
<xliff xmlns="urn:oasis:names:tc:xliff:document:1.2" version="1.2">
<file source-language="en" target-language="fr" datatype="plaintext" original="file.ext">
    <body>
        <trans-unit id="1">
            <source>Login</source>
            <target>Identifiez-vous</target>
        </trans-unit>
        <trans-unit id="2">
            <source>Username</source>
            <target>Nom d'utilisateur</target>
        </trans-unit>
        <trans-unit id="3">
            <source>Password</source>
            <target>Mot de passe</target>
        </trans-unit>
    </body>
</file>
</xliff>
```

The YAML translation format

- Lots of Symfony developers use it.
- **Pro**: it's easy to read/write and supports nested messages.
- **Con**: it's not standard and its syntax is very strict (spaces vs. tabs, etc.)

An example of YAML translation file

```yaml
# app/Resources/translations/messages.fr_FR.yml
Login: Identifiez-vous
Username: Nom d'utilisateur
Password: Mot de passe
```

Symfony supports lots of translation formats

- PHP Arrays
- CSV
- ICU (Data & RES)
- INI
- MO / PO
- Plain PHP
- QT
- XLIFF
- JSON
- YAML

Translation strings vs Translation keys

```
<!-- messages.en.xlf -->
<trans-unit id="1">
    <source>An authentication exception occurred.</source>
    <target>An authentication exception occurred.</target>
</trans-unit>

<!-- messages.fr.xlf -->
<trans-unit id="1">
    <source>An authentication exception occurred.</source>
    <target>Une exception d'authentification s'est produite.</target>
</trans-unit>
```

Translation strings make catalogues easier to read, but any change in the original contents forces you to update the catalogues for all locales.

Translation strings vs Translation keys

```
<!-- messages.en.xlf -->
<trans-unit id="1">
    <source>error.auth_exception</source>
    <target>An authentication exception occurred.</target>
</trans-unit>

<!-- messages.fr.xlf -->
<trans-unit id="1">
    <source>error.auth_exception</source>
    <target>Une exception d'authentification s'est produite.</target>
</trans-unit>
```

Symfony's Best Practices recommend to use keys.

Translation keys simplify translation management because you can change the original contents without updating the rest of catalogues.

Manage user locale

Getting the user's locale

```
use Symfony\Component\HttpFoundation\Request;

public function indexAction(Request $request)
{
    $locale = $request->getLocale();
}
```

The locale is stored in the **Request**, which means that **it's not "sticky"** and you must get its value for every request.

Setting the user's locale via the URL

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

class DefaultController
{
    /**
     * @Route("/{_locale}/contact", name="contact")
     */
    public function contactAction(Request $request)
    {
        $locale = $request->getLocale();
        // ...
    }
}
```

_locale (with a leading underscode) is a special routing parameter used by Symfony to set the user's locale.

Setting the user's locale via the session

```
public function onKernelRequest(GetResponseEvent $event)
{
    $request = $event->getRequest();

    // some logic to determine the $locale ...

    $request->getSession()->set('_locale', $locale);
}
```

This solution requires the use of **events** and **listeners**, which is out of the scope of this workshop.

Full details: http://symfony.com/doc/current/cookbook/session/locale_sticky_session.html

Forcing the translation locale in the controller

```
public function indexAction()
{
  $title = $this->get('translator')
    ->trans(
      'Contact us',
      array(),
      'messages',
      'fr_FR'
    );
}
```

Avoid this technique as much as possible and rely on the other natural ways of setting and getting the user's locale.

Forcing the translation locale in the template

```
{{ 'Contact us'|trans(
  { }, 'messages', 'fr_FR')
}}

{% trans into 'fr_FR' %}
  Contact us
{% endtrans %}
```

Avoid this technique as much as possible and rely on the other natural ways of setting and getting the user's locale.

Translation domains

Translation domains

- An **optional** way to organize messages into groups.
- By default, all messages are grouped in a domain called "**messages**".
- In most applications there is no need or justification for using several domains.

Selecting the domain in the controller

```
$this->get('translator')->trans(
  'Contact us',
  array(),
  'admin'
);
```

The translation is stored in the **admin.fr_FR.<format>** file

If different from "**messages**", set the translation domain as the third optional argument of the **trans()** method.

Selecting the domain in the template

```
{{ 'Contact us'|trans({ }, 'admin') }}

{% trans from 'admin' %}
  Contact us
{% endtrans %}
```

The translation is stored in the **admin.fr_FR.<format>** file.

Selecting the default domain in the template

```
{% trans_default_domain 'admin' %}

{# ... template contents ... #}
```

Note that this only influences **the current template**, not any "included" template (in order to avoid side effects).

Translating variable contents

Translating messages that include variables

```
$message = "Hello $name";
```

Messages which contain **the value of some variable** are very common in web applications. How can you translate them?

Translating variable messages in controllers

```
public function indexAction()
{
  $title = $this->get('translator')->trans(
    'Hello %name%',
    array('%name%' => 'John')
  );
}
```

Variable parts are called **placeholders**. The wrapping % ... % characters are optional but used by convention.

Translating variable messages in templates

```
{{ 'Hello %name%'|trans({
  '%name%': 'John'
}) }}

{% trans with {'%name%': 'John'} %}
  Hello %name%
{% endtrans %}
```

Variable parts are called **placeholders**. The wrapping % ... % characters are optional but used by convention.

Translating XLIFF messages with variable parts

```
<!-- app/Resources/translations/messages.fr_FR.xlf -->
<?xml version="1.0"?>
<xliff version="1.2"
       xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="en" target-language="fr"
          datatype="plaintext" original="file.ext">
        <body>
            <trans-unit id="1">
                <source>Hello %name%</source>
                <target>Bonjour %name%</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

Translating YAML messages with variable parts

```yaml
# app/Resources/translations/messages.fr_FR.yml
'Hello %name%': Bonjour %name%
```

Translations based on variables

Translating plural messages

```
$singular = 'There is one product left.';
$plural = 'There are %count% products left.';
```

Most languages have simple **pluralization rules**, but some of them (e.g. Russian) define very complex rules.

Symfony abstracts this issue and provides out-of-the-box pluralization support for most of the world's languages.

Translating plural messages in controllers

```
public function indexAction()
{
  $title = $this->get('translator')->transChoice(
    'There is one product left.|There are %count%
products left.',
    10,
    array('%count%' => 10)
  );
}
```

Message alternatives are separated with a pipe (|)

This is the value considered to decide which message to pick (singular or plural).

Translating plural messages in templates

```
{% transchoice 10 %}
  There is one product left.|There are %count%
products left.
{% endtranschoice %}

{{ 'There is one product left.|There are %count%
products left.'|transchoice(10) }}
```

Understanding the pluralization rules

```
// English
'There is one product left.
|There are %count% products left.'
```

If **count = 0**, Symfony selects ...

```
// French
'Il y a %count% produit.
|Il y a %count% produits.'
```

Understanding the pluralization rules

```
// English
'There is one product left.
|There are %count% products left.'
```

If **count = 1**, Symfony selects ...

```
// French
'Il y a %count% produit.
|Il y a %count% produits.'
```

Understanding the pluralization rules

```
// English
'There is one product left.
|There are %count% products left.'
```

If **count** > **1**, Symfony selects ...

```
// French
'Il y a %count% produit.
|Il y a %count% produits.'
```

Explicit interval pluralization

```
// English
'{0} There is one product left.|[1,Inf] There
are %count% products left.'

// French
'{0, 1} Il y a %count% produit.|]1,Inf] Il y
a %count% produits.'
```

It's **optional**, but most of the times it helps to better understand which message will be selected.

Explicit interval pluralization

```
]-Inf, 0] C'est fini, vous n'avez plus d'essai !
|{1} Attention, c'est votre dernière chance !
|[2,5] Méfiez-vous, il vous reste %count% essais restants !
|[6,8] Pas de panique, vous avez encore %count% essais restants !
|[9, +Inf[ Vous avez encore %count% essais restants !
```

Intervals are defined using the **ISO 31-11** standard.

Full Details: https://en.wikipedia.org/wiki/Interval_(mathematics)#Notations_for_intervals

Full reference of trans() and transchoice()

```
$this->get('translator')->trans(
    'Hello %name%',
    array('%name%' => 'John'),
    'admin',
    'fr_FR'
);

$this->get('translator')->transChoice(
    'There is one product left.|There are %count% products left.',
    10,
    array('%count%' => 10),
    'admin',
    'fr_FR'
);
```

Full reference of |trans and |transchoice

```
{{ message|trans }}
{{ message|trans({'%name%': 'John'}, 'admin', 'fr') }}

{{ message|transchoice(10) }}
{{ message|transchoice(10, {'%name%': 'John'},
                        'admin', 'fr') }}
```

Full reference of {% trans %} and {% transchoice %}

```
{% trans with {'%name%': 'John'} from 'admin' into 'fr_FR' %}
  Hello %name%
{% endtrans %}

{% transchoice count with {'%name%': 'John'} from 'admin'
   into 'fr_FR' %}
  'There is one product left.|There are %count% products left.'
{% endtranschoice %}
```

Form and database translation

Translating form validation messages

```php
// src/AppBundle/Entity/User.php
use Symfony\Component\Validator\Constraints as Assert;

class User {
    /**
     * @Assert\NotBlank(message = "user.name.not_blank")
     */
    public $name;
}
```

```xml
<!-- messages.en.xlf -->
<trans-unit id="1">
    <source>user.name.not_blank</source>
    <target>Please enter the name of the user.</target>
</trans-unit>
```

Translating database contents

- This feature is not provided by the translation component.
- Install Stof**DoctrineExtensions**Bundle
  https://github.com/stof/StofDoctrineExtensionsBundle
- Use **Translatable** extension.

Creating and updating translation files

Log missing translations

```yaml
# app/config/config.yml
translator:
    logging: true
```

```
# app/logs/dev.log
[201X-04-20 15:06:43] translation.WARNING: Translation not found.
{"id":"Title","domain":"messages","locale":"en"}
[201X-04-20 15:06:43] translation.WARNING: Translation not found.
{"id":"Summary", "domain":"messages","locale":"en"}
[201X-04-20 15:06:43] translation.WARNING: Translation not found.
{"id":"Content", "domain":"messages","locale":"en"}
[201X-04-20 15:06:43] translation.WARNING: Translation not found.
{"id":"Author email", "domain":"messages","locale":"en"}
```

Show unused or missing translations

```
$ php bin/console debug:translation fr AppBundle

+----------+--------------------+-------------------------+
| State(s) | Id                 | Message Preview (fr)    |
+----------+--------------------+-------------------------+
|          | title.post_list    | Liste des articles      |
|          | action.show        | Voir                    |
|          | action.edit        | Editer                  |
|          | action.create_post | Créer un nouvel article |
+----------+--------------------+-------------------------+

Legend:
 x Missing message
 o Unused message
 = Same as the fallback message
```

Create the translation catalogues

```
$ php bin/console translation:update en --dump-messages

Generating "en" translation files for "app/ folder"
Parsing templates
Loading translation files

Displaying messages for domain messages:
  title.post_list
  action.show
  action.edit
  action.create_post
```

Create the translation catalogues

```
$ php bin/console translation:update en --force

Generating "en" translation files for "app/ folder"
Parsing templates
Loading translation files
Writing files
```

```yaml
# app/Resources/translations/messages.en.yml
title.post_list: __title.post_list
action.show: __action.show
action.edit: __action.edit
action.create_post: __action.create_post
```

Create the translation catalogues

```
$ php bin/console translation:update en --force --prefix=new_

Generating "en" translation files for "app/ folder"
Parsing templates
Loading translation files
Writing files
```

```yaml
# app/Resources/translations/messages.en.yml
title.post_list: new_title.post_list
action.show: new_action.show
action.edit: new_action.edit
action.create_post: new_action.create_post
```

Create the translation catalogues

```
$ php bin/console translation:update en --force --prefix=new_
                  --output-format=xlf

Generating "en" translation files for "app/ folder"
Parsing templates
Loading translation files
Writing files
```

```
<!-- app/Resources/translations/messages.en.xlf -->
<trans-unit id="04a6524e12dc0bad0a3146c8" resname="title.post_list">
    <source>title.post_list</source>
    <target>new_title.post_list</target>
</trans-unit>
```

Create the translation catalogues

```
$ php bin/console translation:update en --force --prefix=new_
                  --output-format=xlf AppBundle

Generating "en" translation files for "AppBundle"
Parsing templates
Loading translation files
Writing files
```

```
<!-- src/AppBundle/Resources/translations/messages.en.xlf -->
<trans-unit id="04a6524e12dc0bad0a3146c8" resname="title.post_list">
    <source>title.post_list</source>
    <target>new_title.post_list</target>
</trans-unit>
```

SensioLabs