

---

# R4.01

# Architecture Logicielle

---

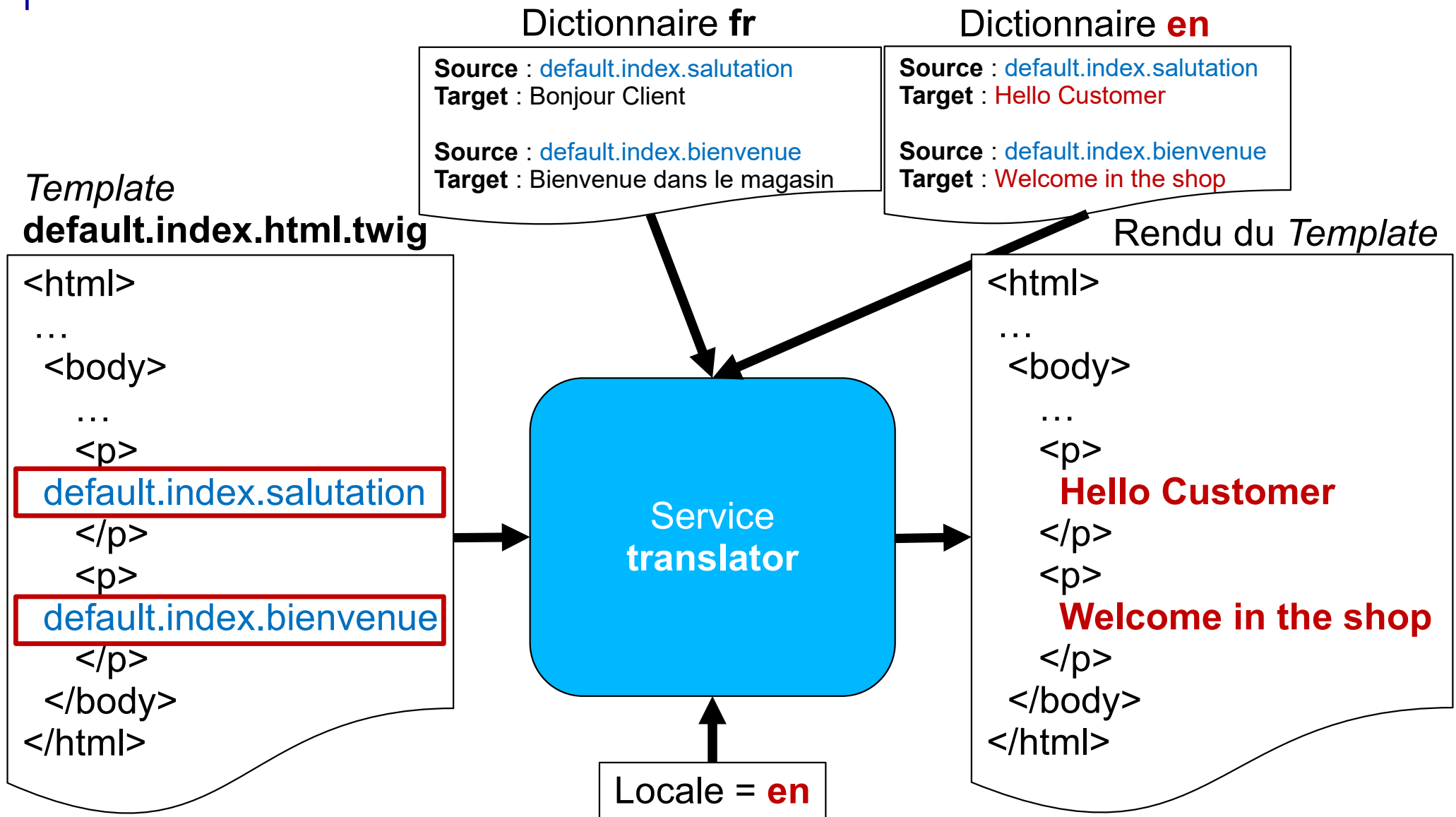
**Traduction (i18n : internationalisation)**

**<https://symfony.com/doc/current/translation.html>**

# Mise en Place de la Traduction

- La traduction consiste à « abstraire » les chaînes de caractères pour qu'elles puissent être traduites selon la **locale** (langue/pays, par exemple **fr**) de l'utilisateur de l'application
- Dans Symfony, la traduction est réalisée grâce au **service translator**
- Travail à réaliser pour pouvoir la mettre en place sur une application :
  - Être capable sur une page de déterminer la langue cible : cela s'obtient avec la **locale**, définie dans l'URL
  - Marquer les éléments à traduire :
    - Dans les vues : grâce à une **balise** ou un **filtre** Twig
    - Dans les contrôleurs : en faisant explicitement appel au service **translator**
  - Mettre en place des dictionnaires (appelés **catalogues** dans Symfony)
- Bonnes pratiques :
  - Il existe plusieurs formats possibles pour les catalogues, XLIFF est le plus répandu
  - Il faut bien organiser ses catalogues

# Principe de la traduction : substitution !



# Déterminer la langue cible (1)

- La **locale** de l'utilisateur est transmise à chaque requête HTTP (*cf* transparent suivant)
- L'application va disposer d'une variable **\_locale** pour connaître la langue à utiliser pour la traduction
- Pour pouvoir toujours être en mesure de proposer une traduction, il faut :
  - définir une **valeur par défaut** pour cette variable dans la configuration de l'application
  - Il faut définir la **valeur de repli** qui sera utilisée si la locale est fournie mais n'est pas supportée par l'application
- Ces valeurs sont définies dans le fichier de configuration **translation.yaml** :

```
# config/packages/translation.yaml
```

```
framework:
```

```
    default_locale: 'fr'
```

```
    translator:
```

```
        default_path: '%kernel.project_dir%/translations'
```

```
        fallbacks: ['fr']
```

Valeur par défaut

Valeur de repli

## Déterminer la langue cible (2)

- Une application ne va proposer des traductions que pour certaines langues
- Il faut définir les langues supportées par votre application et faire en sorte que seules celles-ci puissent être « demandées » par l'utilisateur
- Les langues supportées sont définies dans un paramètre global de Symfony, défini dans le fichier de configuration **services.yaml** :

```
# config/services.yaml
parameters:
    app.supported_locales: 'fr|en'
```

Expression régulière (**regex**) qui sera utilisée pour vérifier si la locale fournie fait partie des locales supportées par l'application :  
'fr | en' signifie « fr ou en »

## Déterminer la langue cible (3)

- Pour rendre un site multilingue et que ses pages, disponibles dans plusieurs langues, soient bien référencées par les moteurs de recherche, la bonne pratique consiste à définir la **locale** comme un paramètre de l'URL de **chacune** des pages du site :  
Exemple : [http://www.mon\\_site.com/fr/accueil/](http://www.mon_site.com/fr/accueil/)
- Il va donc falloir rajouter un paramètre **{\_locale}** (reconnu automatiquement par le contrôleur frontal de symfony) dans les routes de toutes les pages
- Pour chaque route, il faudra aussi contraindre ce paramètre pour qu'il ne puisse prendre pour valeur que les locales supportées par l'application

# Déterminer la langue cible (5)

- Pour des routes définies par annotations dans le code des contrôleurs :

```
// src/Controller/DefaultController.php
namespace App\Controller;

// ...
class DefaultController extends AbstractController {
    #[Route(
        path: '{_locale}',
        name: 'app_default_index',
        requirements: ['_locale' => '%app.supported_locales%'],
        defaults: ['_locale' => 'fr']
    )]
    public function index() {
        // ...
    }

    #[Route(
        path: '{_locale}/contact',
        name: 'app_default_contact',
        requirements: ['_locale' => '%app.supported_locales%']
    )]
    public function contact() {
        // ...
    }
}
```

Ajouter le paramètre `_locale` à chaque route

Contraindre le paramètre `_locale` en fonction des locales supportées (regex)

Définir une valeur par défaut pour `_locale` (uniquement sur la page d'accueil)

# Marquer les éléments à traduire (1)

- Dans une application web, l'essentiel des éléments d'interface utilisateur à traduire sont présents dans les *templates Twig*
- Lors de l'élaboration de ces *templates* (ou *a posteriori*) , il faut :
  - Déterminer les mots / phrases / paragraphes qui doivent être traduits
  - Remplacer ces éléments par un **identifiant unique** qui correspondra à une **source** (une « entrée ») dans les catalogues de traduction
  - Pour s'y retrouver, une bonne pratique consiste à construire un identifiant qui a en préfixe le chemin du template dans lequel il est défini, par exemple :  
**default.index.message\_de\_bienvenue**
  - Et enfin il faut « marquer » cet identifiant comme étant à traduire.  
Deux syntaxes sont possibles :

- Délimiter l'identifiant par une **balise** `{% trans %} ... {% endtrans %}` :

```
<p>
    {% trans %} default.index.message_de_bienvenue {% endtrans %}
</p>
```

- Ou appliquer le **filtre trans** à l'**identifiant** à l'intérieur d'une balise `{{ ... }}` :

```
<p>
    {{ 'default.index.message_de_bienvenue' | trans }}
</p>
```



# Utiliser le service translator dans son code PHP

- Certains éléments d'interface utilisateur sont parfois définis dans le code PHP d'une application (en général pour être ensuite transmis à un *template* !)
- Il est dans ce cas possible de faire appel au service de traduction **translator** pour réaliser la substitution depuis le code PHP, à l'intérieur d'une méthode de contrôleur ou dans un service
- Par exemple, dans un **contrôleur**

```
// ...  
use Symfony\Contracts\Translation\TranslatorInterface;  
// ...  
  
public function index(TranslatorInterface $translator) {  
  
    $traduction = $translator->trans('default.index.un_element');  
  
    // ...  
}
```

Appel au service translator

# Mettre en place des dictionnaires (catalogues)

- **Translator** ne fait pas de la traduction « intelligente », il fait de la **substitution**
- Il faut définir, pour chaque langue à proposer, un ou des **catalogues** qui contiennent des entrées (trans-unit) composées :
  - D'une **source** : une entrée du catalogue qui est doit être un identifiant unique
  - D'une **cible (target)** : le texte qui sera substitué à la source, dans la langue du catalogue
- Les fichiers **catalogue** peuvent être au format **yaml**, **xliff (par défaut)**, ...
- Le catalogue par défaut pour une locale **LL** doit être défini dans le fichier **messages+intl-cu.LL.xlf** du répertoire **translations** de votre projet :

```
<?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">
    <header>
      <tool tool-id="symfony" tool-name="Symfony"/>
    </header>
    <body>
      <trans-unit id="CVcZ3ir" resname="default.index.hello">
        <source>default.index.hello</source>
        <target>__default.index.hello</target>
      </trans-unit>
      <trans-unit id="1eq4n5m" resname="default.index.welcome_message">
        <source>default.index.welcome_message</source>
        <target>__default.index.welcome_message</target>
      </trans-unit>
    </body>
  </file>
</xliff>
```

Il va falloir définir « à la main »  
le texte de substitution (**target**)  
pour chaque **source**, dans la  
langue du catalogue

## Mettre en place des dictionnaires (catalogues)

- Il est possible de produire, pour une locale donnée, un catalogue avec toutes les sources déjà définies à partir des éléments marqués comme « à traduire » dans les *templates* de votre application (et partout où vous utilisez le service translator dans votre code PHP) :

```
php bin/console translation:extract --force fr
```

```
php bin/console translation:extract --force en
```

=> génère deux fichiers **messages+intl-icu.en.xlf** et **messages+intl-icu.fr.xlf** dont il ne reste plus qu'à compléter les balises **<target>**

# Produire un site multilingue « professionnel »

- Dans les vues, il faut utiliser **un identifiant unique** comme source des textes à traduire.
- Utiliser une notation hiérarchique pointée qui indique l'emplacement de la source dans la structure de vos templates
- Par exemple :
  - **default.index.message\_bienvenue**
  - **default.index.devise\_du\_site**

```
{# Dans la vue Default/index.html.twig #}  
{% trans %}default.index.message_bienvenue{% endtrans %}  
{{ 'default.index.devise_du_site' | trans }}
```

- Dans un contexte « pro », après avoir fabriqué les catalogues pour les différentes langues supportées, le développeur complète uniquement le catalogue français et envoie les autres catalogues à des **traducteurs professionnels**

# Notion de Domaines de Traduction

- Pour un site important, le catalogue dans une langue peut être très (trop) important
- Il est alors possible d'utiliser plusieurs catalogues pour une même langue, afin de regrouper les traductions par **domaine** (sémantique ou fonctionnel).
  - **domaine1+intl-icu.fr.yml, domaine2+intl-icu.fr.xlf, ...**
- Si plusieurs catalogues (domaines) sont définis, il faut préciser à chaque traduction le domaine que l'on veut utiliser, sinon c'est le domaine **messages** qui est utilisé par défaut
- Syntaxe pour choisir le domaine :
  - **Twig - Balise** : `{% trans from 'domaine' %}chaîne{% endtrans %}`
  - **Twig - Filtre** : `{{ 'chaîne' | trans({}, 'domaine') }}`  
 **Tableau vide en Twig : { }**
  - **Contrôleur - Service** : `$translator->trans('chaîne', array(), 'domaine')`
- Il est aussi possible, au début d'un template Twig, de définir le domaine qui sera utilisé pour toutes les traductions de ce template :
  - **Twig** : `{% trans_default_domain 'domaine' %}`

# Paramètres dans une cible de traduction

- On peut définir un ou des « placeholders » (paramètres) dans la cible d'une traduction
- Exemple : traduire un message de bienvenue personnalisé paramétré avec un nom
  - Dans le catalogue français : il faut un espace avant le '!'

```
<source>Default.index.hello</source>
<target>Bonjour {nom} !</target>
```

Français : espace avant le !

- Dans le catalogue anglais : **pas d'espace** avant le '!'

```
<source>Default.index.hello</source>
<target>Hello {nom}!</target>
```

Anglais : pas d'espace avant le !

- Il faut alors, au moment de la traduction, préciser la valeur du ou des paramètres :

```
{# templates/Default/index.html.twig #}
{{ 'Default.index.hello' | trans({'{nom}': name}) }}
```

Syntaxe d'un tableau associatif en Twig :  
{'cle':valeur, ...}

- Syntaxe :


- Balise : {% trans with {'{nom}': name} %}Default.index.hello{% endtrans %}
- Filtre : {{ 'Default.index.hello' | trans({'{nom}': name}) }}
- Service : \$translator->trans('Default.index.hello', array('{nom}' => \$name))

# En TP : Traduire votre site

- Mettez en place le service de traduction dans votre projet
- Faites-en sorte que toutes les pages développées jusqu'à maintenant soient disponibles en Français (**fr**) et en Anglais (**en**), y compris la barre de navigation !
- Rajouter dans la barre de navigation un menu pour changer la locale (fr/en) :
  - « injecter » la variable `supported_locales` dans twig pour pouvoir l'utiliser dans vos templates `#config/packages/twig.yml`

```
twig:
  globals:
    supported_locales: '%app.supported_locales%'
```
  - Dans votre « navbar », vous pourrez alors proposer un menu pour choisir la locale :

```
{% for uneLocale in supported_locales | split('|') %}
  ...
{% endfor %}
```


  - Pour récupérer la valeur courante de la locale en twig :

```
app.request.attributes.get('_locale')
```
  - Pour fabriquer une url qui permet de recharger la page actuelle en changeant sa locale :

```
{% set route = app.request.attributes.get('_route') %}
{% set params = app.request.attributes.get('_route_params') %}
{{ path(route, params | merge({'_locale': uneLocale})) }}
```
- **A partir du TP3, on ne traduira plus les nouvelles pages ajoutées au site mais nous continuerons à intégrer la locale aux nouvelles routes ajoutées**