



# Understanding the Internationalization (I18n) and Translation System

Florian Gnägi

<gnaegi@frentix.com>

---

## Table of Contents

### 1. Introduction to the I18n Concepts

- 1.1. Brasato and application GUI messages
- 1.2. Language and country codes: Java Locale
- 1.3. Customizing overlays
- 1.4. Referencing other translation keys
- 1.5. Translation cache
- 1.6. The translation fallback chain
- 1.7. Load order of i18n files from source and jar files

### 2. I18n Configuration

- 2.1. Enabling active system languages
- 2.2. Default language (aka standard language)
- 2.3. Import languages from a language package
- 2.4. Export languages as language package
- 2.5. Delete installed language packages
- 2.6. Create a new language
- 2.7. Delete languages
- 2.8. Expert configuration
- 2.9. Translation server configuration

### 3. The Translation and Language Adaption Tool

- 3.1. Translation or customizing mode
- 3.2. Start screen
- 3.3. Translation item screen
- 3.4. Inline translation and language adaption mode

### 4. The Context Help System

- 4.1. Introduction
- 4.2. Rating feature
- 4.3. Adding a context help link
- 4.4. Creating a context help page
- 4.5. Adding the help page title and other translations
- 4.6. Tips and tricks
- 4.7. Context help dispatcher: reference from an external website

# Chapter 1. Introduction to the I18n Concepts

## 1.1. Brasato and application GUI messages

The Brasato application framework offers full internationalization (i18n) support. Every GUI message that appears in the user's browser, e.g. to navigate, error messages etc., are computed by the i18n system based on the *user's language configuration*.

*Translations*, as we call these GUI messages, are stored in Java `Properties` files on the file system as part of the source code of your application. Each Java package can contain a directory `_i18n`. There you will find the localization files, e.g. `LocalStrings_de.properties` for German translations. In the context of the translation infrastructure we call the package to which the translation belongs the *bundle*. Translation files in `org/olat/core/_i18n/` are the translation files for the bundle `org.olat.core`.

A translation message for a certain bundle and language consists of two parts: the *key* to identify the message, and the *translation* itself. They are separated by a "=".

```
1#Wed Dec 17 18:01:22 CET 2008
2error.noformpostdata=Attention\! Due to problems
3calendar.choose=Pick a date from the mini calend
4table.action=Action
5error.jspwrapper.renderfailed=This component can
6this.language.in.english=English
7welcome=Welcome
8page.title=Online Learning And Training
9form.mandatory.hover=This field is mandatory
```

The i18n system is fully UTF-8 capable. This means that every language and character set is supported: Chinese, Russian, German or English - the framework can deal with it. Note that some languages are written from *right to left*. This is another topic and has nothing to do with the i18n system. The RTL top is a layouting problem and is thus covered in the layout documentation "Understanding the Brasato Layout".

The i18n system features an easy to use *translation tool* that allows users to translate new languages or customize an existing language pack for a certain use case. The system does also provide easy to use workflows to enable and disable languages, import and export language packs and to create or delete languages.

Java properties files are stored in the ISO-8859-1 encoding. Therefore, everything different than US-ASCII must be properly encoded. Line breaks and other special characters like "!" must also be encoded as described in the [Java Properties spec](#). Best is to never write properties files yourself. Just use the integrated translation tool that takes care of everything for you.

## 1.2. Language and country codes: Java Locale

Technically, when a user configures his language, a Java `Locale` is stored for this user. A locale consists of three parts: `de_CH_ZH`

1. The *language* : e.g. German, only lower case letter
2. Optional: the *country* : e.g. Switzerland, only capital letters
3. Optional: the *variant* : e.g. Zürich (often used for a dialect) only capital letters

Since Java checks for valid languages it is mandatory to only use language and country codes that actually exist:

1. [Valid language codes](#)
2. [Valid country codes](#)

<input type="checkbox"/>	Albanian (sq)	71%	<input checked="" type="checkbox"/>	Italian (it)	93%
<input type="checkbox"/>	Arabic (ar)	3%	<input checked="" type="checkbox"/>	Korean (ko)	0%
<input checked="" type="checkbox"/>	Brazilian (pt_BR)	93%	<input type="checkbox"/>	Lithuanian (lt)	67%
<input type="checkbox"/>	Cape Dutch (af)	0%	<input type="checkbox"/>	Mongolian (mn)	1%
<input checked="" type="checkbox"/>	Chinese (zh_CN)	71%	<input type="checkbox"/>	Persian (fa)	47%
<input checked="" type="checkbox"/>	Czech (cs)	84%	<input checked="" type="checkbox"/>	Polish (pl)	85%
<input type="checkbox"/>	Danish (da)	55%	<input type="checkbox"/>	Portuguese (pt_PT)	77%
<input type="checkbox"/>	Dutch (nl_NL)	0%	<input checked="" type="checkbox"/>	Romansh (rm)	0%
<input checked="" type="checkbox"/>	English - customized (en)	100%	<input type="checkbox"/>	Russian (ru)	65%
<input checked="" type="checkbox"/>	French (fr)	93%	<input checked="" type="checkbox"/>	Spanish (es)	79%
<input checked="" type="checkbox"/>	Greek (el)	88%	<input checked="" type="checkbox"/>	Swissgerman (de)	100%
<input type="checkbox"/>	Hebrew (iw)	0%	<input type="checkbox"/>	Traditional Chinese (zh_TW)	35%
<input type="checkbox"/>	Hungarian (hu)	1%	<input type="checkbox"/>	Turkish (tr)	3%
<input type="checkbox"/>	Indonesian (in)	4%	<input type="checkbox"/>	Vietnamese (vi)	1%

It is very common to use only the language part. Sometimes it is handy to add the country part. The variant is almost never used.

The idea behind this concept is that it is possible to build a system that implements a hierarchy of translations: common elements are defined in the languages, those that are country specific are defined on country level and finally messages that are even more specific use the variant part:

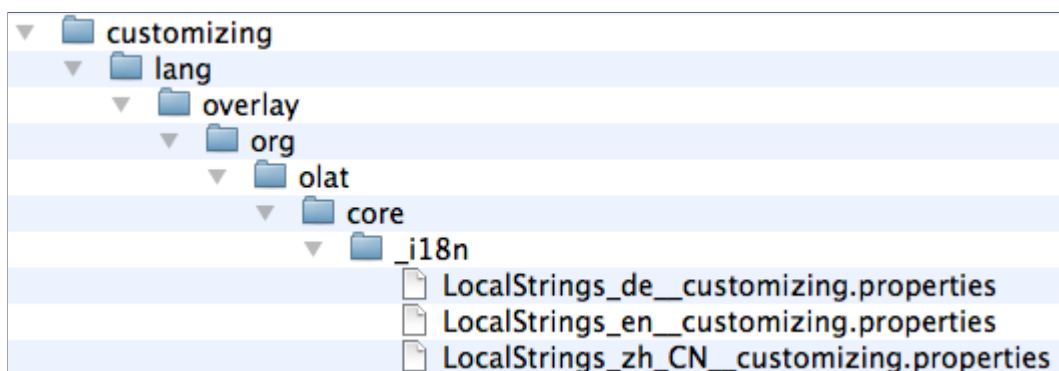
de: contains the translations that are valid in all German speaking countries

de\_CH contains translations that differ from the basic German translation, i.e. spellings and expressions unique to Switzerland. So does de\_DE contain specific translation for German users.

### 1.3. Customizing overlays

In addition to the three elements in the Locale we added a fourth element to be able to override the official translations from the application with a customized version. It is very common that you want to have some parts translated differently, for example on the login screen. You could change this translation in the OLAT translation files, but when updating to a new release all your modifications would be lost. This is where the Brasato i18n overlay mechanism comes in.

In your *user space directory* (normally this is the `olatdata` directory) you will find a directory `olatdata/customizing/lang/overlay`. In this directory you can add the language overlay files for each language. The file name consists of the locale that is overridden and the name of the overlay. The package/bundle structure must be the same as in the source code, e.g. you would add `olatdata/customizing/lang/overlay/org/olat/core/_i18n/` and add your customization file there.



`LocalStrings_de__customizing.properties` : this file would override the language `de`. In this example, the overlay is named `customizing` (default setting). This can be configured in the `I18nModule`. Note that you don't need to translate everything to use the overlay feature. It is perfectly okay to have only one overlay file with only one key in it. For all other keys the normal translation would be used.

No worries, you don't have to do this manually, just use the online language customization tool and everything happens automatically.

## 1.4. Referencing other translation keys

It is possible to reference within a translation message to keys from another translation message. This is useful to reduce redundant translations. For example you could have a button with the translation key "button.do.it=Do it now!" and then a little help text that says "help.buttons=To really do it, press \$:button.do.it". Instead of duplicating the "Do it now!" text, the help text automatically uses the correct button label even when the label changes.

Referencing other keys follows this syntax:

- `$:other.key` : a reference to a key in the same bundle (relative reference)
- `$another.bundle.name:other.key` : a reference to a key in another bundle (absolute reference)
- ``${another.bundle.name:other.key}`` : Add curly brackets to help the system to parse, sometimes needed when a reference is used within a word.

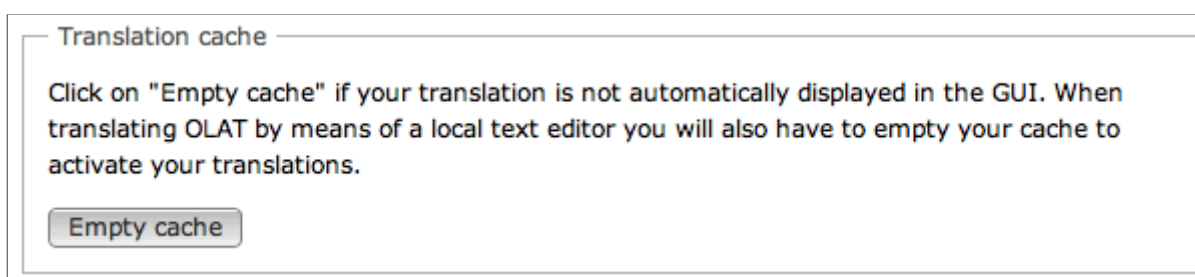
Be aware that using a lot of references can make your translations difficult to maintain. There is a tradeoff between redundancy and maintainability.

## 1.5. Translation cache

The i18n system loads properties files from disk only. This happens on demand when they are needed the first time. During this load procedure, the `I18nManager` solves all the relative and absolute references and then keeps these translations in an in-memory cache.

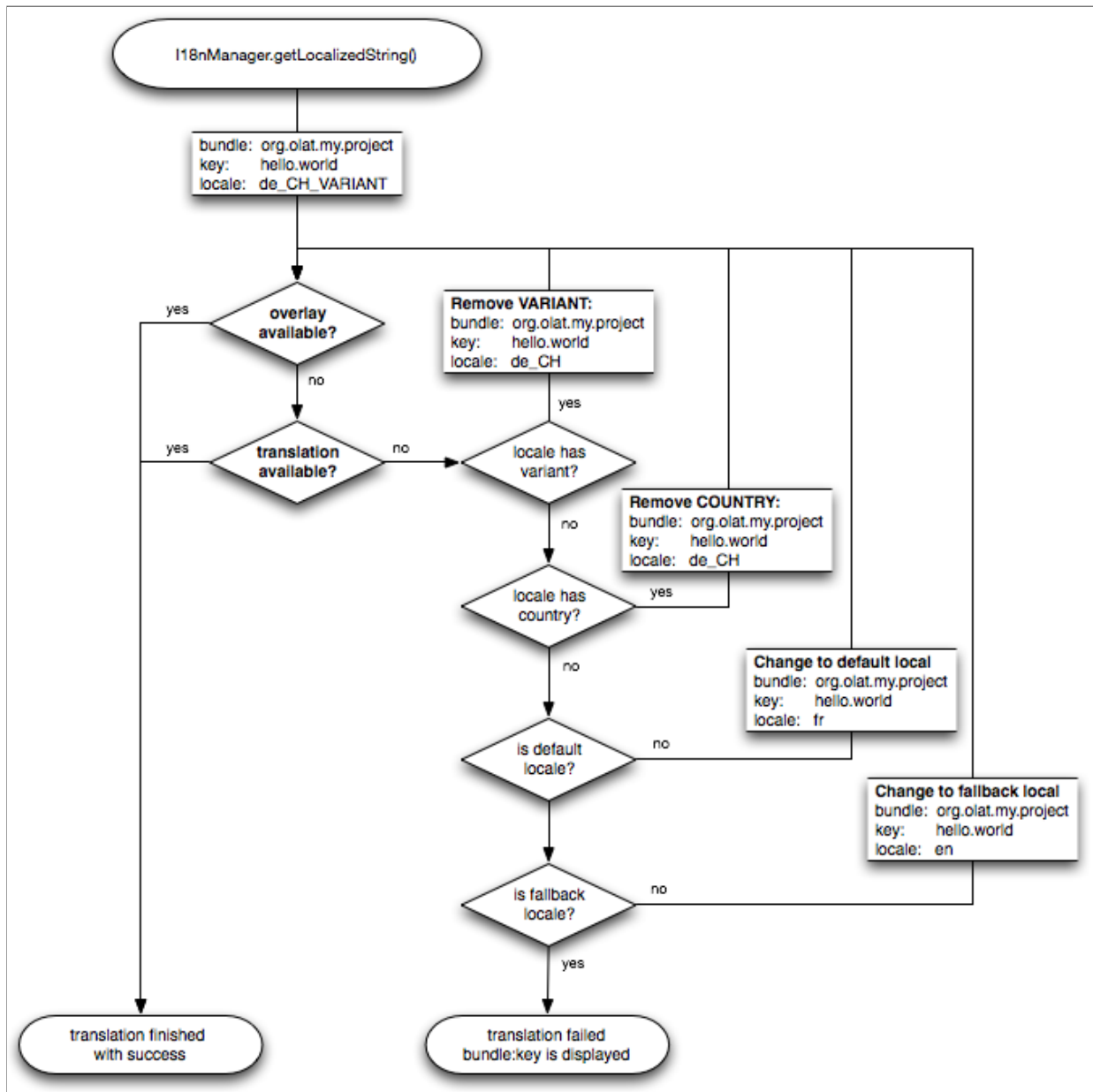
When modifying translations using the translation tool, the cache is updated automatically with the new version of the translation.

The cache can be disabled in the `I18nModule` for development purposes, however this will slow down the entire system. Even in a development environment I recommend not to disable the translation cache. Instead, when modifying a properties file directly on the file system you can press the "Empty cache" button from the administration menu where you also start the translation and language customizing tool. This will flush the entire cache and start rebuilding.



## 1.6. The translation fallback chain

It is very important to understand how Brasato decides which bundle and language should be used to translate a key. The following graphic will help you understand the translation fallback chain to find the right translation. Let's assume the system tries to translate the key "hello.world" from the bundle "org.olat.hello.world". The user has the language "de\_CH" enabled in his user session:



First, the language *overlay* is considered. If nothing is found there, then the system checks for the translation in the users *locale*. The locale is first checked against language, country and variant, then against language and country and finally only against the language. If still no translation has been found, the system tries it with the configured *default locale*. Note that the default locale is a configuration made by the system administrator. Since the chosen default locale might not be translated completely, the system uses the hard coded *fallback locale* as the ultimate fallback mechanism if everything else fails

## 1.7. Load order of i18n files from source and jar files

When the system starts up the `I18nManager` checks which languages are available. This is done in the method `doInitAvailableLanguages()`. The following sources are searched for i18n Files:

- In *development* mode: on the application and brasato source path (`brasato.src` in `build.properties` must be configured)
  - `WebappHelper.getBrasatoSourcePath()`: the brasato source path.
  - `WebappHelper.getSourcePath()`: the application source

The languages found there are considered to be *translatable*. Normally this is only DE and EN.

2. In *translation* mode: on the translation directories (`is.translation.server`, `i18n.application.src.dir` and `i18n.brasato.src.dir` must be configured accordingly)

The languages found there are considered to be *translatable*. This means that they can be translated with the translation tool with read/write access

3. In the *jar* libraries

- `webapp/WEB-INF/lib/*.jar`: each jar file is searched for `*/_i18n/*` directories, e.g. in `olatcore_i18n_all-SNAPSHOT.jar` and `olat3_i18n_all-SNAPSHOT.jar` but also in your custom jar files

The languages found here are *not translatable*, read only.

4. In the optional *language packs* directory (language packs uploaded in the language configuration)

- `olatdata/customizing/lang/packs/`

The languages found here are *not translatable*, read only.

5. In the language *customizing* directory (files created with the customizing tool)

- `olatdata/customizing/lang/overlay/`

The languages found here are *translatable*, read/write. However, these are only language overlays, not real languages.

The I18nManager processes the steps above and adds each found language file to the internal map. The first hit counts, a second hit for the same language and bundle in another source will be ignored.

## Chapter 2. I18n Configuration

The `I18nModule` configuration can be configured in the `webapp/WEB-INF/olat_config.xml` file. However, don't change things there if you don't know what you do. Please do also note that the configuration in this legacy configuration file will soon move to the spring file. You can consider this configuration as the default configuration.

Everything you normally have to change can be changed from the admin interface.

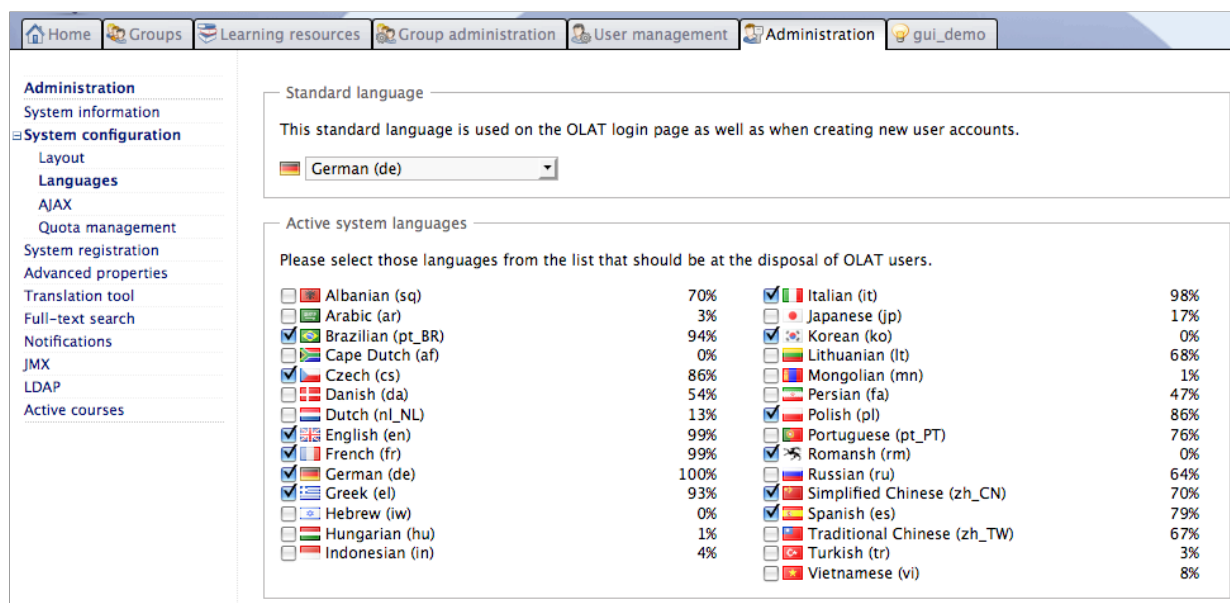
*NOTE: The i18n system does not automatically propagate config changes to other cluster nodes when running OLAT in cluster mode. In this case you manually have to press the `Empty Cache` button on the language adaption start screen.*

### 2.1. Enabling active system languages

During startup the system searches for all *available* languages. The system administrator can then decide which of those languages should be *enabled*. Enabled languages can then be selected by users in the personal settings or on the login screen.

To enable languages, go to Administration > System configuration > Languages > Active system languages

Select the languages you want to activate. Next to each language the translation status is displayed. It's no problem to activate a language that is not fully translated, the system will automatically fallback to the default language



Language	Translation Status
Albanian (sq)	70%
Arabic (ar)	3%
Brazilian (pt_BR)	94%
Cape Dutch (af)	0%
Czech (cs)	86%
Danish (da)	54%
Dutch (nl_NL)	13%
English (en)	99%
French (fr)	99%
German (de)	100%
Greek (el)	93%
Hebrew (iw)	0%
Hungarian (hu)	1%
Indonesian (in)	4%
Italian (it)	98%
Japanese (jp)	17%
Korean (ko)	0%
Lithuanian (lt)	68%
Mongolian (mn)	1%
Persian (fa)	47%
Polish (pl)	86%
Portuguese (pt_PT)	76%
Romansh (rm)	0%
Russian (ru)	64%
Simplified Chinese (zh_CN)	70%
Spanish (es)	79%
Traditional Chinese (zh_TW)	67%
Turkish (tr)	3%
Vietnamese (vi)	8%

### 2.2. Default language (aka standard language)

The default language is used for two things:

1. *as standard language* whenever no specific language is set: on the login screen, during registration etc.
2. *as a fallback language* in case a translation is missing in the language that the user has configured.

To change the default language, go to Administration > System configuration > Languages > Standard language

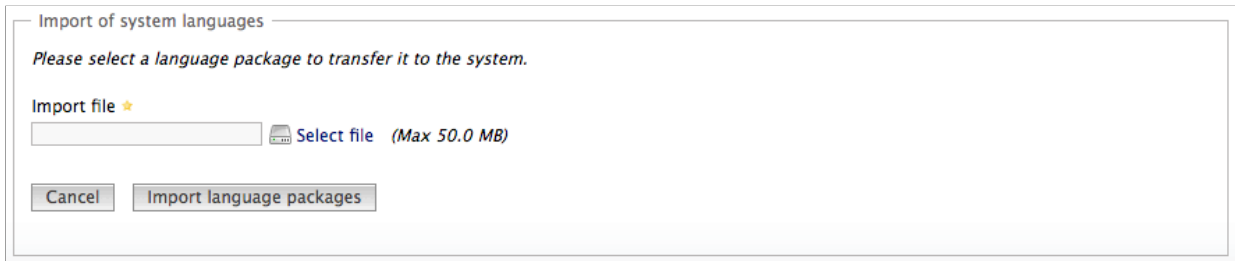
### 2.3. Import languages from a language package

Languages can be exported and imported as language packs. To import a language you must download a language pack from [olat.org](http://olat.org) or create one using the export mechanism explained later. Currently all available languages are bundles with OLAT, in the future we plan to make languages available for download

from our server.

To import a language pack, go to Administration > System configuration > Languages > Language administration and press the *Import languages packages* button. Select a language pack from your hard drive and start the upload.

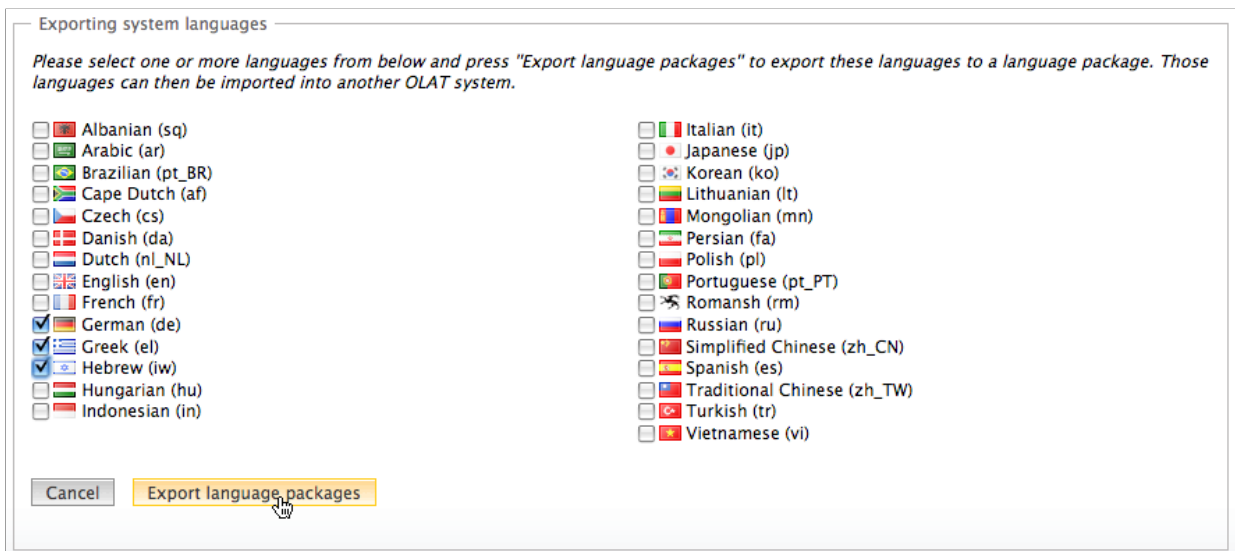
Note that languages are imported as language packs. Language packs can not be edited after import. There are plans to make imported languages editable when the server is configured as translation server, but currently imported languages are read-only.



## 2.4. Export languages as language package

If you configured your system as a translation server you can share your translations by exporting them to a language pack.

To export a language pack, go to Administration > System configuration > Languages > Language administration and press the *Export languages packages* button. Select the languages you want to export as a language pack and finish by pressing the appropriate button. The system creates the language pack for you and starts the download to your local hard drive. You can now share this language pack with others and import it to another system.



## 2.5. Delete installed language packages

Language packages that you import with the steps described above can be deleted.

To delete a language pack, go to Administration > System configuration > Languages > Language administration and press the *Delete language packs* button. This button is only visible when you installed a language using the import workflow. If you don't see the button, there is no language pack installed.

## 2.6. Create a new language

If you configured your system as a translation server you can create new languages.

To create a new language, go to Administration > System configuration > Languages > Language administration and press the *Generate new language* button.

A form appears where you must enter some basic information about the new language:

- The language code: two lowercase characters, must be a [valid language code](#)
- Optional: the country code: two uppercase characters, must be corresponding [valid country code](#)
- Optional: the variant
- The term of language in english
- The language's original term
- The name and institution of the translator

Details of new language

Please generate the new language here.

Language code ★   
de (German)

Country code   
CH (Switzerland)

Variant code   
ZH (Zurich region)

Term of language in English ★   
German

Language's original term   
Deutsch

Name of translator, institution   
Marion Weber, University of Zurich

## 2.7. Delete languages

If you configured your system as a translation server you can delete an existing language if you have the source for this language attached.

To delete an existing language, go to Administration > System configuration > Languages > Language administration and press the *Delete language* button. Select the languages you want to delete.

Note, deleted languages can not be restored! You have to reinstall your system to bring them back. Normally it is best to disable a language rather than deleting it! You can only delete languages that you have installed as source. It is not possible to delete languages that are in a bundled jar library.

Deletion of system languages

Please select one or more languages below and click on "Delete language" to delete these languages irrevocably.

<input type="checkbox"/> Albanian (sq)	<input type="checkbox"/> Italian (it)
<input type="checkbox"/> Arabic (ar)	<input type="checkbox"/> Japanese (jp)
<input type="checkbox"/> Brazilian (pt_BR)	<input type="checkbox"/> Korean (ko)
<input type="checkbox"/> Cape Dutch (af)	<input type="checkbox"/> Lithuanian (lt)
<input type="checkbox"/> Czech (cs)	<input type="checkbox"/> Mongolian (mn)
<input type="checkbox"/> Danish (da)	<input type="checkbox"/> Persian (fa)
<input type="checkbox"/> Dutch (nl_NL)	<input type="checkbox"/> Polish (pl)
<input checked="" type="checkbox"/> English (en)	<input type="checkbox"/> Portuguese (pt_PT)
<input type="checkbox"/> French (fr)	<input checked="" type="checkbox"/> Romansh (rm)
<input type="checkbox"/> German (de)	<input type="checkbox"/> Russian (ru)
<input type="checkbox"/> Greek (el)	<input type="checkbox"/> Simplified Chinese (zh_CN)
<input type="checkbox"/> Hebrew (iw)	<input type="checkbox"/> Spanish (es)
<input type="checkbox"/> Hungarian (hu)	<input type="checkbox"/> Traditional Chinese (zh_TW)
<input type="checkbox"/> Indonesian (in)	<input type="checkbox"/> Turkish (tr)
	<input type="checkbox"/> Vietnamese (vi)

## 2.8. Expert configuration

To make configurations without the admin GUI you can change the default parameters of the `I18nModule` directly in the `webapp/WEB-INF/olat_config.xml` file. The various options are well commented there.

From this config files the system reads the default configuration. Once an administrator uses the admin GUI to set the language parameters, the config is written into the `olatdata` user space. This user space configuration overwrites the default values. The configuration can be found in the file `olatdata/system/configuration/org.olat.core.util.i18n.I18nModule.properties`. This file can be deleted to restore the configuration to the default settings.

## 2.9. Translation server configuration

This special section describes how to configure a server that acts as a translation server. If you don't want to translate a language you can skip this section.

If you want to create a new language and translate it, please contact us first. The OLAT team operates a [public translation server](#) where all bundled languages are translated. You don't have to configure and operate your own translation server, we do this all for you!

So, this chapter is only relevant for people who want to translate their own languages and don't want to contribute the language to the OLAT project.

Here is what you have to do:

1. Check out the application project (`olat3`) from CVS or unzip the sources from a ZIP

Configure the application to run, if everything works, proceed with next

2. Check out the `olatcore` project (`olatcore`) from CVS or unzip the sources from a ZIP

Configure the `brasato` source path to this `olatcore` project in your `build.properties`:

```
brasato.src=/opt/olatcore/src/main/java
```

3. Check out the application `i18n` project (`olat3_i18n`) from CVS or unzip the sources from a ZIP

Maybe this project is stored in your private CVS and not named like the example below. You could also use any other path to an existing directory here, this directory will contain the new created languages from your server. Configure the `i18n` path for the application files in your `build.properties`:

```
i18n.application.src.dir = /opt/olat3_i18n/src/main/java
```

4. Check out the `olatcore i18n` project (`olatcore_i18n`) from CVS or unzip the sources from a ZIP

This could well point to the same directory as the `i18n` application source dir! Configure the `i18n` path for the `olatcore` files in your `build.properties`:

```
i18n.brasato.src.dir = /opt/olatcore_i18n/src/main/java
```

5. Configure the system as a translation server

Set `is.translation.server=true` in your `build.properties` file

6. Remove the bundled language jar files from your `webapp/lib` directory to reduce conflicts

```
/webapp/WEB-INF/lib/olatcore_i18n_all-SNAPSHOT.jar
```

```
/webapp/WEB-INF/lib/olat3_i18n_all-SNAPSHOT.jar
```

7. Run `ant config-all` and restart tomcat.

When logging in as administrator you will now see a menu item `Administration > Translation Tool`. Create a new language first as described above and then start the translation tool to translate your new language.

# Chapter 3. The Translation and Language Adaption Tool

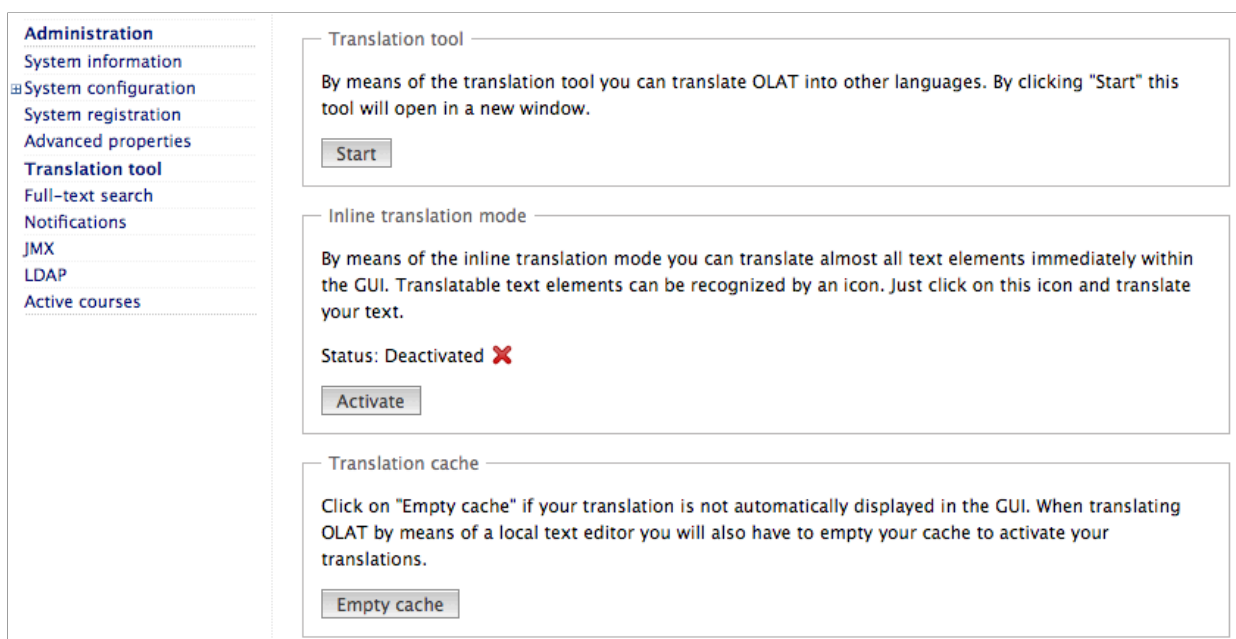
## 3.1. Translation or customizing mode

The brasato framework features a powerfull tool to translate system languages or to adapt the system languages to your need. Since both cases are done with the same tool, we just call it translation tool.

When the server is configured to be a translation server (see previous chapter), the tool acts as translation tool, in other cases (which is the normal case) it acts as a adaption tool.

To launch the translation/adaption tool go to Administration > Translation Tool Or Administration > Language Adaption Tool. On the launch screen you can:

- Start the language/adaption tool
- Activate/Deactivate the inline translation mode
- Flush the i18n cache.



If a translation you made does not appear on screen, especially when translating a context help page, *flushing* the i18n cache might help. This will re-read every translation file from disk and resolve all dependencies again. Normally this step is not necessary since the translation tool automatically updates cached values, but referenced values might not show the correct values. Flushing the cache is also needed when i18n files are modified directly on the filesystem, e.g. from within an IDE.

*NOTE: The i18n system does not automatically propagate adapted i18n keys to other cluster nodes when running OLAT in cluster mode. In this case you manually have to press the `Empty Cache` button on the language adaption start screen.*

## 3.2. Start screen

On the translation tool start screen you select

1. The reference and target *language*
2. Which *bundle* you want to translate

Additionally a *progress bar* indicates how many keys are already translated and how it compares to the reference language.

In customizing mode, the reference and target language are set to the same value. The target will be your

overlay locale.

The translation tool offers four ways to start with the translation work:

1. Show *missing* translation: when the missing translation count is 0 you are finished
2. Show *already translated* keys: review what you have already done
3. Show *all* translations, regardless of the translation status
4. *Search* for a specific key or value. You can use the \* as a wildcard in this search.

The screenshot shows the 'Start Translation tool' interface. At the top, it displays the reference language as German (de) and the target language as Simplified Chinese (zh\_CN). A status bar indicates that 71% (5243/7417) of translations are complete across 163 packages. Below this, a message instructs the user to select an option to begin. The interface is divided into four main sections: 'Missing translations' (2174 keys), 'Translated translation keys (revision)' (5243 keys), 'All translations' (7417 keys), and a 'Search' section. Each section includes a 'Show' button to view the keys and a 'Translate' button to start the translation process. The search section allows for searching within keys and translations, with options to search in the translation key or target language, and to consider sub-packages and sort by priority.

Normally, the system searches within all bundles. This can be *limited to a certain bundle*. Additionally, you can decide to include the child bundles as well or to search only within the specified bundle.

By default, the bundles are *sorted by priority*. This priority is set by the developers and indicates how *important a certain bundle* is. Bundles from the brasato framework are considered to be more important because they appear more often than other bundles. Example code is normally very unimportant and thus as a very low priority to be translated. This sorting behaviour can also be switched on or off.

Choose one of the options and press *Show* to see which keys are affected by your search or *Translate* to start translating right away.

### 3.3. Translation item screen

Translating a key is straight-forward. On top of the screen is the *bread crumb navigation* back to the translation tool start screen. Right below the *current package* and the *current translation key* is displayed. You can choose the pulldown menu to move fast forward within the packages and keys.

Next is the translation of the key in the *reference language*. This field is read-only and shows you how this key is translated in the reference language.

The reference language field is followed by a *comment field*. If no comment is available, a button allows you

to create a comment. In translation mode this field is read/write, in customizing mode it is read-only. In the comment field you will find hints about this translation key, how it is used or common pitfalls. You can help other translators by adding your notes there as well. This comment is only visible to the translators, it is not used for anything else.

After the comment field the *comparative language* can be activate or deactivated. Choose another language that you also understand and might help you to translate the key properly to your language. Deactivate this feature if not needed. This field is also read-only.

Finally, the input field for your language appears. Enter your translation and press *Save* to just save or *Save & next* to save and proceed with the next translation item in the list. You can enter any character you need, the i18n system is fully UTF-8 capable. But don't use single or double quotes unless also used in the reference language since this can break parts of the application.

Start Translation tool > Translation list > Translation key

Package: org.olat.core 1/119 Packages

Translation key: error.noformpostdata 19/66 Translation keys

Please translate your term from the reference language to the target language. Alternatively, you can skip that term and translate it later on. Activate another language to compare your translation.

Reference language: Deutsch

Achtung! Durch ein Problem Ihres Browsers wurden nicht alle Daten hochgeladen. Drücken Sie bitte den Zurück-Knopf und versuchen Sie es nochmals.

Comment

This message appears when the user submitted a form but no file was uploaded.

Comparative language: English (en)  Activate

Attention! Due to problems concerning your browser some data could not be transferred. Please use the "Back" button and try again.

Target language: 简体中文

注意! 由于IE问题, 一些数据无法传送. 请按 "返回"按钮重试.

Back Save Save & next Next

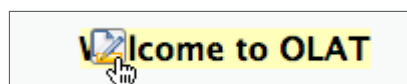
Go to top

### 3.4. Inline translation and language adaption mode

If this was not cool enough for you, check this out: the i18n system offers you a way to translate or adapt translations on-the-fly. We call this the *inline translation mode*. The inline translation mode is activated on the translation tool / language adaption tool launch screen.

Go to Administration > Translation Tool OR Administration > Language Adaption Tool and activate the inline translation / inline adaption mode. A red cross indicates that the mode is deactivated, a gree tick that it is activated. Now you have to log out and log in again to activate the change.

Once it is activated, every translatable element in the GUI will show a little edit icon when hovering over it. The text that is affected by this key will flash yellowish for a moment. Click the edit icon to translate this key.

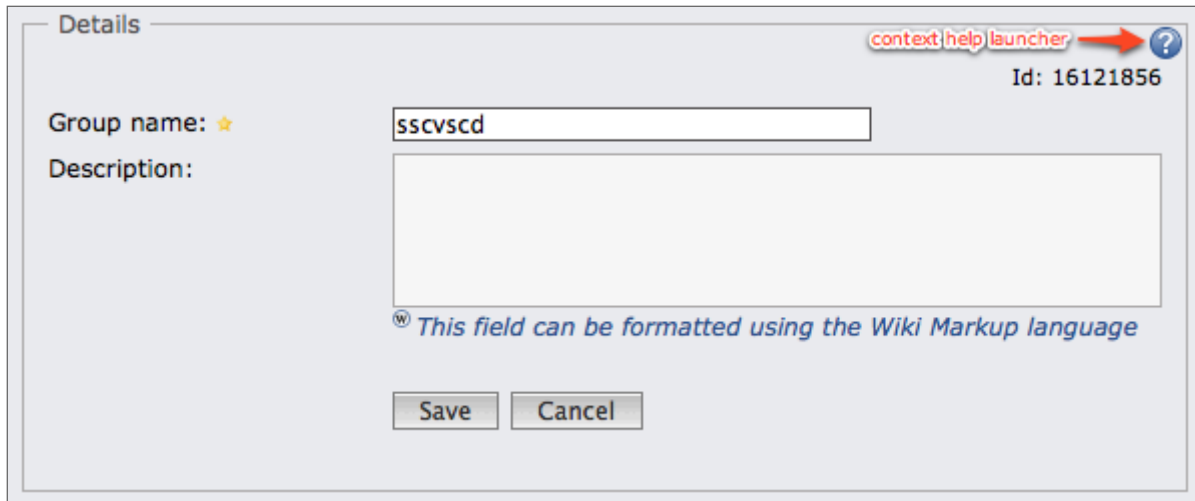


The inline translation mode is very helpfull when you start translating to catch the most important things first or when adapting the system to your local need and you only want to customize some strings.

# Chapter 4. The Context Help System

## 4.1. Introduction

The Brasato framework features a context sensitive help system. Whenever you as a developer decide that a user might need some information, you can place a context help icon that explains to the user what he can do. This is mainly used to give advice in forms or in some workflows that might need help.



A context help page is identified by the *Java package name* that the help page belongs and the *page name*:  
`org.olat.group.ui.edit:grp-BuddyGroup-des.html`

When context help page is displayed, the user has the possibility to change the language in case the help has not yet been translated to his language. There is a navigation element to go to a table of content that lists all available context help pages.

Note: Since OLAT 6.1 the context sensitive help is generated dynamically by the framework. There is no need to do any precompilation process anymore. Also, when keys in the users language are missing, the normal fallback mechanism is used and the context help from the default or fallback language is used instead. We also greatly reduced the copy-past translations that consisted only of key references.

## 4.2. Rating feature

A context help page can be rated by the user: was this a helpfull information or not? Currently both the community rating and the user rating is shown. The community rating equals to the average of all user ratings. Users can change their rating of a page at any time, the community rating is adjusted accordingly.



The *user rating* is stored in the user GUI `Preferences` available from the `UserSession`.

The *community rating* is stored in a simple XML file in the application user space: `olatdata/system/context_help_rating.xml`. The file contains a map with the language, package name and page name as key (e.g. `en:org.olat.group.ui.edit:grp-BuddyGroup-des.html`) and the summed up ratings and the number of ratings.

## 4.3. Adding a context help link

Adding a context help page to your `velocityContainer` view is easy. Simply use one of the two method calls

from the `VelocityRenderDecorator`:

```
<fieldset>
  <legend>My form</fieldset>
  ## Add a context help launcher with a wrapper container that makes
  ## it float at the top-right corner of your content

  $r.contextHelpWithWrapper("org.olat.group.ui.edit", "grp-BuddyGroup-des.html", "chelp.hover.bgDetail-BuddyGroup"

  [... your stuff here ...]

</fieldset>
```

Or...

```
## Add a context help launcher without a wrapper to place it
## somewhere in your HTML flow

For help press the $r.contextHelp("org.olat.group.ui.edit", "grp-BuddyGroup-des.html", "chelp.hover.bgDetail-Buddy
```

## 4.4. Creating a context help page

The next step is to create the actual help page. The help page is a normal `VelocityContainer` page that is used by the `ContextHelpPageCrumbController` to display the help page. Thus, you can use everything you can use in a normal Brasato velocity page.

The context help page must be stored in the directory `_chelp` under the name used in the launcher icon. In the example above, the help page would be located under `CLASSPATH/org/olat/group/ui/edit/_chelp/grp-BuddyGroup-des.html`.

Mostly you will use the translator to display the translatable elements on the screen. It is a good idea to split your help page into some paragraphs and use meaningful i18n keys.

By convention, all i18n keys that are used for a context help should have the `chelp.` prefix:

`chelp.grouphelp.intro`, `chelp.grouphelp.accessrights` ... Since the context help page is in the same Java package as the code where the help page is embedded, you have full access to all other i18n keys from your package. So you can refer to elements from your actual GUI very easily. There are also other methods you can use

Translations:

```
## Normal translation
## Example without and with arguments
##
$r.translate("chelp.grouphelp.intro") $r.translate("another.key.that.already.exists")
$r.translate("chelp.grouphelp.intro", ["hello", "world"])

## Translation that uses key from another package
## Example without and with arguments
##
$r.translateWithPackage("org.olat.core", "cancel")
$r.translateWithPackage("org.olat.core", "cancel", ["hello", "world"])
```

Links to other context help pages:

```
## Link to another context help page in another bundle with my custom link text
##
$r.contextHelpRelativeLink(String bundleName, String pageName, String linkText)

## Link to another context help page in another bundle with the page title as link text
##
$r.contextHelpRelativeLink(String bundleName, String pageName)

## Link to another context help page the same bundle
##
$r.contextHelpRelativeLink(String pageName)
```

It is also possible to use images and other static files in the context help. These files must be located in the `_static` directory of the context help directory. In our example an image would be located here: `CLASSPATH/org/olat/group/ui/edit/_chelp/_static/myimage_de_CH.png`

Note that the image must also contain the locale postfix. There must be an image for at least the fallback language `en`. If your image is localized, add an image for each language.

Using the image is easy since the context help system adds a special variable to velocity that gives you the path to the image folder. Note, the image name used in the velocity file is without the language code since this page is the same for each language! The system automatically checks if the image is available for the currently used locale and uses the fallbackchain to find the fallback image.

```
## Embedd an image. Or create a PDF download. Or add a movie...
##

```

## 4.5. Adding the help page title and other translations

Finally you must add the `i18n` keys to the package you used in your velocity page. There is one `i18n` key that you must add that you probably don't use in your help page but is required by the context help system: the help page title. The name for the key must follow this scheme:

```
chelp.pagename.title Example: chelp.grp-BuddyGroup-des.title
```

You must also add the `i18n` key you used in the context help launcher as hover text, in our example above this was `chelp.hover.bgDetail-BuddyGroup`.

In our example you have to add those to the files located under `CLASSPATH/org/olat/group/ui/edit/_i18n/*`.

## 4.6. Tips and tricks

To support the external referencing of help pages, the system checks for help pages during startup. Since the context help system prior to 6.1 used the page name as the only identifier, the external referencing mechanism will also work only with the page name and no bundle reference.

The problem is that this requires unique page names. So it is best to follow a naming scheme that creates unique page names. Don't use just `help.html` or `edit.html` as help page names, add something else to make it more clear. E.g. use `myprj-general.html` and `myprj-form-edit.html`. This makes it also easier to find your page in eclipse.

It is also good to know that the index of available context pages are cached. Thus, when you add a new context page and did not shutdown eclipse, you have to flush the translation cache. This will also flush the context help cache. This is found in the administration view where you launch the translation tool.

## 4.7. Context help dispatcher: reference from an external website

Content help pages can be referenced from external Websites. The URL must include the *language key*, the *Java package* and the *help page* name. To support legacy links to help pages prior to release 6.0, the system supports also requests that contains only the help page name. However, those requests will fail if there exists multiple pages with the same page name.

Links to a context help are handled by the `ContextHelpDispatcher` which is mounted under the path `/yourcontextpath/help/`.

Example link to the login screen help page:

```
<!-- Link to the olat context help page for the login screen -->
<a href="https://www.olat.uzh.ch/olat/help/en/org.olat.login/dmz.html">OLAT login help</a>
```