Internationalization

Table of Contents

This page gives guidelines to I18n for:

- Plugin developers who would like to apply the i18n mechanism in their own plugin, so that this plugin can be available in several languages
- People who would like to help the community by making the platform available in a new language

Principles

Although the basics of the i18n mechanism are the same for every part of the ecosystem, the packaging differs depending on what you are developing:

- **Translations for SonarQube**: making SonarQube available in a new language requires you to develop and publish a new Language Pack plugin.
  - By default SonarQube embeds the English Pack.
  - All other Language Pack plugins, like the French Pack plugin, are hosted in the Plugins Forge, are maintained by the community, and are available through Marketplace (category "Localization").
- **Translations for the SonarQube Community Plugins**: open-source plugins from the SonarQube Community (hosted in the Plugins Forge) must embed only the bundles for the default locale (en). Translations will be done in the Language Pack plugins.
- **Translations for other Plugins**: closed-source/commercial/independent plugins must embed the bundles for the default locale **and** the translations for every language they want to support.

To sum up

- **SonarQube Platform and SonarQube Community Plugins rely on Language Pack plugins for translations**
- **Other independent SonarQube plugins must themselves embed all the translations they need**

Translation Bundles

Localized messages are stored in properties files:

- These are regular properties files with key/value pairs where you put most translations
- These files must be stored in the `org.sonar.l10n` package (usually in the *src/main/resources/org/sonar/l10n* directory)
- The names of these files must follow the convention "<key of the plugin to translate>_<language>.properties", for example "widgetlabs_fr.properties" or "core_fr.properties" for core bundle. See sonar-packaging-maven-plugin for details on plugin key derivation.
- Messages can accept arguments. Such entries would look like:

```
myplugin.foo=This is a message with 2 params: the first "{0}" and the second "{1}".
```

UTF-8 encoding

In the Java API, properties files are supposed to be encoded in ISO-8859 charset. Without good tooling, it can be quite annoying to write translations for languages that do not fit in this charset.
This is why we decided to encode the properties files in UTF-8, and let Maven turn them into ASCII at build time thanks to native2ascii-maven-plugin (check the French plugin pom.xml). This makes the process of writing translations with a standard editor far easier.

Naming conventions for keys

Here is what you need to know about conventions for keys:

Key	Description	Example
`metric.<key>.name`	Metric name	`metric.ncloc.name=Lines of code`
`metric.<key>.description`	Metric description	`metric.ncloc.description=Non Commenting Lines of Code`
`notification.channel.<channel key>`	Name of notification channel	`notification.channel.EmailNotificationChannel=Email`
`notification.dispatcher.<dispatcher key>`	Subscription to notification channel	`notification.dispatcher.ChangesInReviewAssignedToMeOrCreatedByMe=Changes in review assigned to me or created by me`
`rule.<repository>.<key>.name`	Rule name	`rule.pmd.StringInstantiation.name=String Instantiation`
`rule.<repository>.<key>.param.<param key>`	Description of rule parameter	`rule.pmd.VariableNamingConventions.param.memberSuffix=Suffix for member variables`
`dashboard.<key>.name`	Dashboard name, since 2.14.	`dashboard.Hotstpots.name=Point Chauds`

`qualifier.<key>`
 	
`qualifiers.<key>`
	Qualifier name, since 2.13.	`qualifier.TRK=Project`
		
		`qualifiers.TRK=Projects`

`widget.<key>.name`	Widget name	`widget.alerts.name=Alerts`
`widget.<key>.description`	Widget description	`widget.alerts.description=Display project alerts`
`widget.<key>.property.<property key>.name`	Name of widget property	`widget.hotspot_most_violated_rules.property.defaultSeverity.name=Default severity`
`widget.<key>.property.<property key>.desc`	Description of widget property	`widget.hotspot_most_violated_rules.property.defaultSeverity.desc=If selected, severity used to initialize the dropdown list of widget`
`widget.<key>.property.<property>.option.<option>.name`	Name of item of dropdown list	
`widget.<key>.*`	Any other widget message	`widget.alerts.tooltip=Threshold is raised`
`<page key>.page`	Page names shown in the left sidebar	`cloud.page=Cloud`
`<page key>.*`	Any other keys used in a page	`cloud.size=Size`
`property.category.<category key>`	Category name of properties, since 2.11	`property.category.General=Général`
`property.category.<category key>.description`	Short description of category of properties, since 3.6	`property.category.General.description=General properties of SonarQube`
`property.category.<category key>.<subcategory key>`	Subcategory name of properties, since 3.6	`property.category.exclusions.global=Global exclusions`
`property.category.<category key>.<subcategory key>.description`	Short description of subcategory of properties, since 3.6	`property.category.exclusions.global.description=Configuration of global exclusions`
`property.<key>.name`	Property name, since 2.11	`property.sonar.sourceEncoding.name=Source encoding`
`property.<key>.description`	Property description, since 2.11	`property.sonar.sourceEncoding.description=Source encoding`
`<plugin key>.*`	Any other keys used by plugin	

How to read localized messages from a plugin extension?

The component `org.sonar.api.i18n.I18n` is available for **web server extensions**. Scanner extensions can not load bundles.

Writing a Language Pack

A Language Pack defines bundles for SonarQube and/or plugins.

Creating a Language Pack

The easiest way to create a new pack is to copy the French Pack and adapt it to your language.

Maintaining a Language Pack

In the pom file, set the versions of SonarQube and of the plugins you want to translate.

When it's time to update your language pack for a new version of SonarQube or a plugin, the easiest way to see what keys are missing is to run:

```
mvn test
```

If the build fails, it means that some keys are missing. Go to target/l10n to check the reports for each bundle.

Missing keys are listed under 'Missing translations are:'

Report

`Missing translations are:`
`code_viewer.no_info_displayed_due_to_security=Due to security settings, no information can be displayed.`
`comparison.version.latest=LATEST`
`...`

Each time you add a new bundle or update an existing one, please create a JIRA ticket on the corresponding L10n component in order to track changes.

Localizing a Plugin

This section applies if you are developing a closed-source plugin, or an open-source plugin that is not part of the SonarSonarQube Community Plugins forge.

If your plugin falls in this category, it must embed its own bundles. Bundle must be defined in src/main/resources/org/sonar/l10n/<plugin key>_<language>. properties

The default bundle is mandatory, and must be English. For example the plugin with key "mysonarplugin" must define the following files in order to enable the French translation:

- `org/sonar/l10n/mysonarplugin.properties`
- `org/sonar/l10n/mysonarplugin_fr.properties`