



**Internationalize your Python project**

**Table of Content**

[I. Introduction 4](#_Toc468797321)

[II. Overall Process 4](#_Toc468797322)

[III. I18n support step by step 4](#_Toc468797323)

[1. Define all localization variables in the .pot files 4](#_Toc468797324)

[2. Generate .po files that contain all localized strings in the project using PoEdit. 5](#_Toc468797325)

[3. Convert .po file to binary version (.mo file) 6](#_Toc468797326)

[4. Load language into the Python application 6](#_Toc468797327)

[5. Specified the language in the workflow of testing application 7](#_Toc468797328)

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Quang Tran | Dec 05 2016 | Initialize document | 1.0 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Review and Approval**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Role** | **Signature** | **Action** |
|  |  |  |  |
|  |  |  |  |

# Introduction

**Internationalization (I18n)** is a module that help developers write software that is independent of language and locale by providing mechanisms for selecting a language to be used in program messages or by tailoring output to match local conventions. This document explains step by step how to add support for translations in a Python application (Python 3.x).

# Overall Process

1. We have one **.pot** file in a project. This is a plain text file that describes all strings that need translation. This file can be generated manually by command line or by a software named “PoEdit”.
2. With the **.pot** file, we generated the **.po** file for each language we want to have. There is one .po file per language, and it is the file that the translators need to complete. Before the program starts running, the .**po** need to be translate to binary version, call **.mo** file.
3. When the project starts, it calls the **gettext()** function to load the expected language from **.mo** file to memory/variable. Here, we just use those variables in source code and no need to worry about the language differences anymore.

Note: In this document, we only use the **PoEdit** software to manipulate the I18n files.

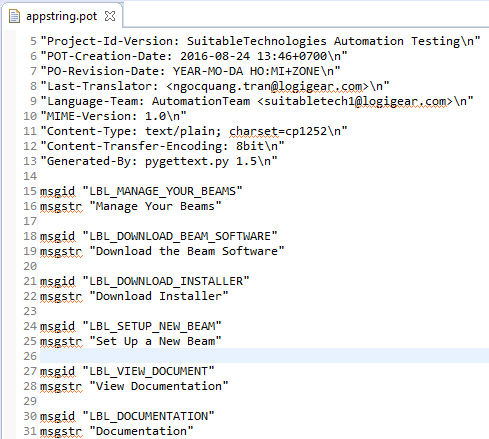
# I18n support step by step

## Define all localization variables in the .pot files

First, we generate a plain text file, named “**appstring.pot”.** We can use **{PYTHON\_DIR}\Tools\i18n\pygettext.py** to parse the Python source code and extract the string to the **pot** file. But I don’t recommend using this method because it extracts the string as the text-id. You will feel very confuse when looking at the file content.

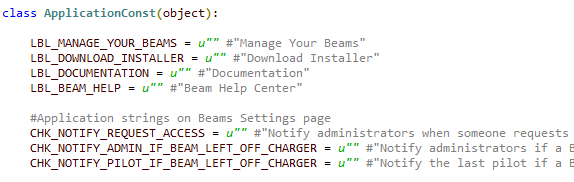
So in this document, I will create the **pot** file manualy from Notepad++. In this file, we define all the localized strings of application. Each localized string is defined by 2 values: **msgid** and **msgstr**.

Here is the sample:



Here, for easily reading purpose, we use the variable name from Python source code as the **msg-id** and give the default value in the **msgstr.**

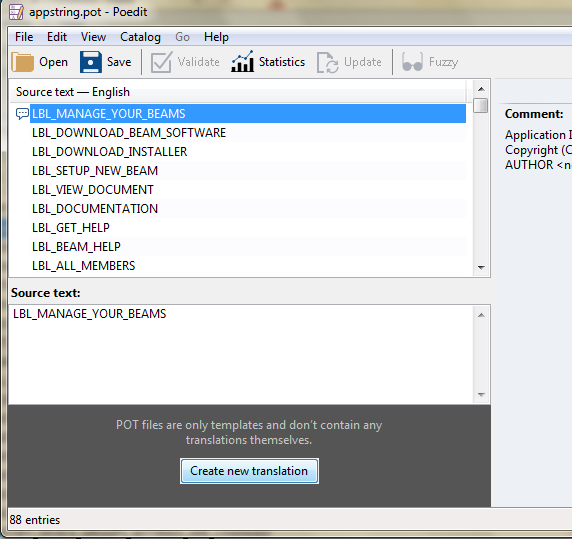
Here are the corresponding Python variables, defined in the ApplicationConst class.



## Generate .po files that contain all localized strings in the project using PoEdit.

Download the free **PoEdit** software from <http://poedit.net>.

Right click the **.pot** file and select **Open with > PoEdit.**



Click button “**Create new translation**”. Then select the language, choose **English (United States)**.

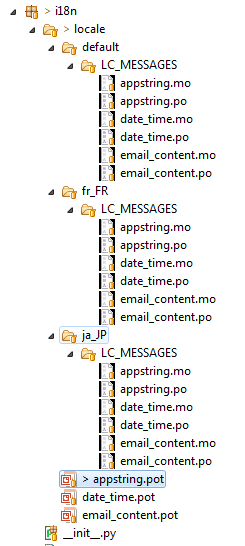
Put the translated string to be **Translation** section for each **Source text** item. Then click **Save** button.

**Poedit** will save the translation to **appstring.po** file and then convert it to **appstring**.**mo** file.

Save those two file to this location: **{current\_pot\_location}\default\LC\_MESSAGES.**

If you want to create the translation for Japanese or French, just do the above steps again and save to the location: **{current\_pot\_location}\ja\_JP\LC\_MESSAGES** or **{current\_pot\_location}\fr\_FR\LC\_MESSAGES**.

Here is the localization data structure in SuitableTechnologies project:



Note that you should put the .**po** and **.mo** in this structure: **locale/<language>/LC\_MESSAGES/** so that the **gettext** API can read/access them by default behavior.

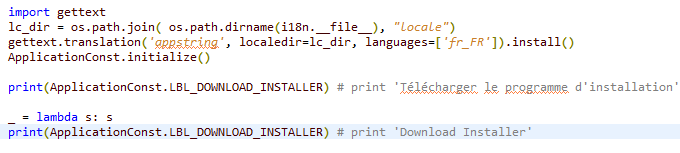
For faster modification purpose, instead using **PoEdit** application, you can copy the **appstring.po** file to a new language folder then open it by Notepad and modify the language string.

## Convert .po file to binary version (.mo file)

Just open the .**po** file by PoEdit application and click **Save** button. The new **.mo** file will be generated and replace the old one.

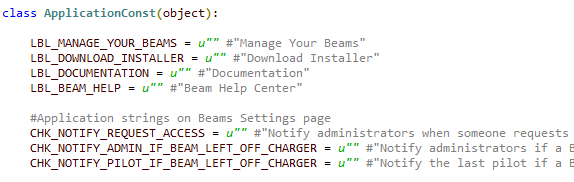
## Load language into the Python application

We use the built-in **install()** method of **gettext** module. The **install**() method will invoke all the **\_()** calls to return the translated string. If you want to go back to the default language, just assign a lambda function value to **\_** that returns the string it was passed:

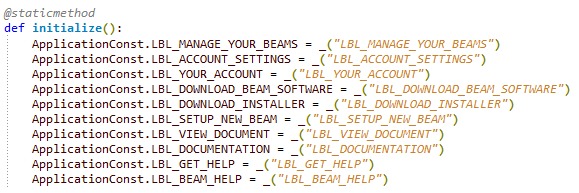


As you can see, we pass the name of .**mo** files (‘*appstring’*), language folder (‘*fr\_FR’)* and the parent folder (‘*locale*’) into the **gettext.translation()** method. It means that we want to load translated string from the **appstring.mo** in the folder **locale/fr\_FR/LC\_MESSAGES** folder.

We use constant variables to store the translated string after loading from **.mo** file. So we define the variables which named as the **msgid** value.



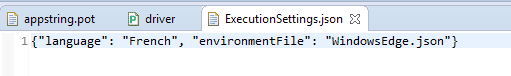
Last, let’s look at the **ApplicationConst.initialize()** method:

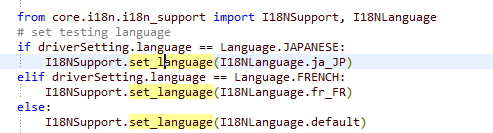


Here we load translated string from **.mo** file by the method: **\_(‘*msgid\_value*’)**. This is very important method. If you don’t call it and assign the translate string to the variable, it’s always empty value in the whole application.

## Specified the language in the workflow of testing application

In the SuitableTechnologies, each time running test suite, we define the language in the **data/setting/ExecutionSettings.json** file.



When the **Driver** object is initialized, it will read this setting and call the **set\_language(language)** and load the expected localized strings.