How to translate VisualPlace

The international language support in VisualPlace is based on the 'Rosette' library. Rosette is a library that provides localization and internationalization support to applications via message catalogs. Together with the library, Rosette comes with a set of tools to aid translators in maintaining and verifying the catalogs with translations.

There are three sections in this guide. It starts with instructions for translation without any additional tools (except for a plain text editor, but your operating system is guaranteed to come with a suitable one). For translators, there are environments that are more efficient —the second section covers how the Rosette catalog can be converted to and from the formats used by these special environments. The third section contains general remarks that apply to both ways of working.

With no help

A Rosette catalog contains the translations for all languages in a single file. It is called the 'catalog' and it is a text file. It can be edited with a plain text editor, such as Notepad. Obviously, we advise you to use a more powerful text editor, but if Notepad is all that you have, it is suitable.

The catalog text file should be stored in the UTF-8 encoding (not the 16-bit 'Unicode' encoding). ASCII is fully compatible with UTF-8, but most languages require characters outside the ASCII range.

A translation block starts with a line in the 'key language' (for VisualPlace, this is English), followed by a list of translations in other languages. All languages are indicated by a 'language code'. This is a two-letter code, typically from the ISO 639-1 list. Examples are 'en' for English, 'nl' for Dutch, 'fr' for French, etc. Rosette does not require you to adhere to the ISO 639-1 list; it only requires that each code uniquely identifies a language.

Each message must be contained on a single line (or row). To break the line, insert the '`\n`' character pair at the location where the break must occur. If a backslash appears in the text, you should double it. This is called *escaping* a backslash in Rosette. The most common *special characters* in Rosette are:

Special characters (escape characters)

`\n`	A line break ('newline')
`\r`	A carriage return
`\t`	A TAB character
`\"`	A double-quote character
`\$`	The dollar character ('$')
`\\`	The backslash character itself ('\')
`\`*ddd*	A character code of 1 to 5 digits; the code must be in decimal
`\x`*dd*	A character code of 1 to 4 digits; the code must be in hexadecimal

For example, to force a line break after 'jumps' in the text

> the quick brown fox jumps
> over the lazy dog

use:

```
en: the quick brown fox jumps\nover the lazy dog
```

The '\$' escape code is only needed if a '$' appears at the beginning of a message (so immediately following the colon behind the language code). If a '$' appears somewhere in the middle of a message, there is no need to escape it. A '$' at the beginning of a message indicates a 'message tag', see page 6.

Double quotes do not need to be escaped. The \" escape code is merely supported for convenience when using the C/C++ language (or similar), where double quotes in strings must be escaped.

If a message is formulated differently depending on whether there are 'plural forms', you give both the 'general' message and the exceptions —the exceptions have a *count* in square brackets behind the language code. For European languages, the *general message* is usually the plural form and the exception is the *singular form*.

Plural forms

```
en     : %1d records are found
en[1]  : %1d record is found
de     : %1d Datensätze sind gefunden
de[1]  : %1d Datensatz ist gefunden
fr     : %1d lignes sont trouvées
fr[0,1]: %1d ligne est trouvée
```

In the example above, the strings that start with '`en`' and '`fr`' are the general messages. These will be used if no other string matches. If the count is 1, however, the strings starting with '`en[1]`' and '`fr[0,1]`' are selected.

There is no need to specify the same plural forms for all messages; in this instance, there are exceptions for the counts 0 and 1 in French, but only for 1 in English and German. The reason for the extra 'plural form' in French is that for a count of zero, you have to use the *singular* form, whereas English uses the plural form for zero. Thus, if the count is 0, for English the general message is selected, whilst the string starting with '`fr[0,1]`' is picked for French.

With some help

There exist various programs that are specifically designed for translating text. However, only very few are suitable for translating the strings in software products —mainly because of a lack of consensus on how internationalizing of software should be done.

There are no known Computer Aided Translations (CAT) tools that work directly on Rosette catalogs. You can use CAT tools, however, by first exporting part of the catalog in a new format, and importing it back into the catalog after translation. Rosette provides tools to perform the export, import and verification operations —and the essential tools are now freely available. These tools are available both as command line utilities, and as a windowed utility. This document only covers the windowed utility.

The windowed utility has four buttons for the main operations: opening a Rosette catalog, exporting a single translation (or template for translation) from that catalog in a selected format, importing a translated partial catalog/message file, and closing the utility.

Before starting, it is always a good idea to ensure that you have a backup of the original catalog, and of any other files that you work on. Since Rosette catalogs (and intermediate files that it uses) are all plain text files, a *better* idea is to use a version control system.

After opening a catalog, the utility shows some statistics about the catalog, among which the currently supported languages (the languages that are already in the catalog) and the key language.

When exporting a file, the utility shows a dialog where you need to set two options: the format for the output file and the language to export. The format depends on the translation editor (CAT tool) that you will use (more on this below). The language to export can either be one of the languages that are already in the catalog or, more likely, a new language. To export a new language, you need to type in the two

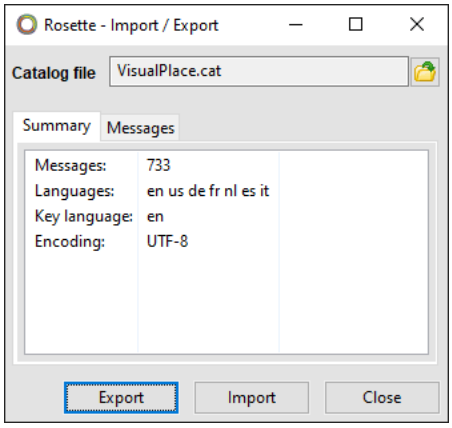

FIGURE 1: *The Rosette export/import utility*

letter code for the language. So for example, if you wish to make a translation for Spanish, you would specify '`es`' as the language to export.

When importing a translated file back into the catalog, no options need to be set. The Rosette utility detects the appropriate parameters.

Below are three example scenarios for translating Rosette catalogs: with a PO editor, an XLIFF-based CAT tool, and using Google's on-line 'Translator Toolkit'.

Using a PO editor

gettext is a library and toolkit for internationalizing software (the license restricts its use to 'open-source' programs). A few programs exist to edit the translation files used by gettext. These files are called 'PO' files, software to edit PO files are 'PO editors'.

A difference with Rosette catalogs is that a PO file contains a translation for only one language, whereas a catalog contains the translations for *all* supported languages. Therefore, a single language must be extracted from the catalog, before starting the translation.

With gettext, it is common practice to use the language code as the filename, with extension '`.po`' (for example '`fr.po`' or '`en-GB.po`'). For the Rosette 'import' functionality, this naming convention is mandatory, as the PO format does not have a header or message fields that encode the key and/or target languages.

When adding a translation for a new language, you have to type in the letter code for the new language and choose the format 'PO'. The export operation now creates a file with only the strings in the key language and empty translations. In gettext terminology, this is a PO Template, or 'POT' file. Rosette does not make the distinction between PO and POT files —a POT file is just a PO file that happens to contain zero translated strings.[1]

Issues like 'escaping' characters and handling plural forms (see the previous section) are often handled by the PO editor and/or the conversion tools. However, a literal '$' at the start of message may still need to be escaped to avoid it being mistaken as a message *tag* (see page 6).

[1] gettext makes this distinction because its 'xgettext' utility always creates an 'empty' POT file that contains only the messages in the key language (and translators then use 'msginit' to merge previously made translations to thus create a PO file. Rosette's rsexport utility, the equivalent of 'xgettext', extracts existing translated messages into the PO file. Therefore, what you send to the translators already contains the translations from previous versions.

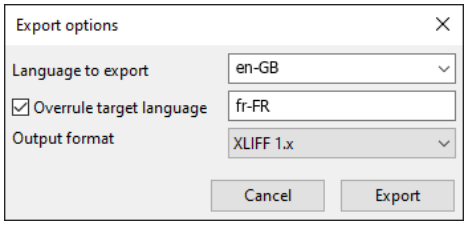

FIGURE 2: *Export language with a different language code*

Some PO editors offer automatic conversion (or compilation) to 'MO' format. This is not needed and not useful for Rosette. The MO format is a binary representation of the PO format —and it is incompatible with Rosette's binary format.

Using an XLIFF editor

The XLIFF format is an XML-based file format for translating general purpose texts. It is supported by several CAT tools and editors. Although not specifically designed for software translation, in practice it works well.

A popular and freely available CAT editor that supports XLIFF is OmegaT. It is used as an example in this section. Other XLIFF-capable editors have similar capabilities.

Although the XLIFF format can technically support multiple translated languages in the same file (in addition to the key language), the specification recommends that each file only contains a single translation.

An XLIFF file maps strings from a source language to a target language. When OmegaT imports an XLIFF file, it uses the 'source' string solely as a match target and it presents only the 'target' strings to the translator. You should therefore always extract the language that you wish to *translate from* into an XLIFF file (usually this is the *key language*). See also figure 2, in which the language that is exported is en-GB but the XLIFF will record it as fr-FR. It is furthermore advised to append the two-letter language code of the *intended target language* to the filename, separating it from the base name with a dash. For example, when the catalog 'myproduct' must be translated from English to German (via XLIFF), extract the language '`en`' into a file called '`myproduct-de.xliff`'.

The resulting XLIFF file will have the source and target strings both set to the key language. OmegaT will then present the 'target' strings (in the key language) for translation to the *real* target language.

An XLIFF file records both the 'source' and 'target' languages in a header of the file. However, following the above procedure, these are both set to the *key language*. OmegaT does not modify the header when writing the output XLIFF file, thereby making the language codes in the XLIFF header unreliable. This is why it is advised to record the *target* language code in the filename. When using the command line utility rsimport, the target language code of an XLIFF file should be specified on the command line (even though the documentation states that language codes are *optional* for XLIFF files[2]).

A limitation of the XLIFF format is that there is no inherent support for plural forms. The convention suggested by OASIS (the organization maintaining the XLIFF standard) is that a messages and its plural forms are grouped (the XLIFF standard describes a general purpose 'group' tag) and to add the plural count between square brackets to the 'id' of the translation unit. For example, a message with plural forms is grouped like:

[2] Language codes are optional if they are correctly recorded in the XLIFF file. When the XLIFF header is incorrect or unreliable (or absent), language codes should be explicitly given.

```
<group restype="rosette-plurals">
    <trans-unit id="37867a22">
        <source>The following %{number:d} components are deleted: %{list}</source>
        <target>Les %{number:d} composants suivants seront supprimés : %{list}</target>
    </trans-unit>
    <trans-unit id="37867a22[1]">
        <source>The following component is deleted: %{list}</source>
        <target>Le composant suivant sera supprimé : %{list}</target>
    </trans-unit>
</group>
```

OmegaT ignores the grouping and presents both strings individually. The group specifcation is used by the Rosette tools to correctly import a translated XLIFF file into a catalog.

Using Google Translator Toolkit

Google provides a 'Translator Toolkit' for translating texts or web pages —an on-line service that combines computer translations and translation memory. The Translator Toolkit uses a side-by-side editor for the original and translated texts. The editor requires that the source file contains strings in only a single language; the output file will also contain only a single language (i.e. the translation). These two files (source and output) are synchronized internally in the Translator Toolkit.

To use Google's Translator Toolkit with Rosette, the first step is to export the key language to a temporary file. This file is then uploaded to the Translator Toolkit and translated. The output from the translation can be imported back into the catalog.

To export the temporary file, choose the key language as the language to export and the 'HASH' format. This will give you a file with messages in the key language (usually English), where each message is preceded by an eight-digit hexadecimal code —the hash. This is the file that you will upload and translate.

After uploading, in the left panel of the editor of the Translator Toolkit, you will see lines with:

```
#127FA68E
en: This is not a pipe
```

In the right panel, there are the same strings, already machine translated into the target language. If the target language is French, the string might be:

```
#127FA68E
en: Ce n'est pas une pipe
```

You can correct 'Ce' into 'Ceci'. With a global search & replace operation, you should also replace all occurrences of 'en:' to 'fr:'. The comment with the hash code (#) should not be altered or removed; this code is required to to merge the translated file back. The corrected snippet is:

```
#127FA68E
fr: Ceci n'est pas une pipe
```

General remarks

After adding your translations, you can test the results immediately by launching VisualPlace and choosing your language. Strings that you have not yet translated will appear in the key language (e.g. English).[3]

A catalog provided with VisualPlace already contains a few translations. When starting a translation, you can learn by example, by looking how the existing translations handled things. Some background information, including tips and examples, can be found in a freely available white paper on Rosette, which you can find on the CompuPhase web site.[4]

[3] However, when importing translations from a PO editor, you may be required to complete the translation before Rosette is able to import it —some PO editors create *empty* translations for any message that is not yet handled.

[4] *Rosette — Internationalization through message catalogs*, available on the CompuPhase web site.

Placeholders

The text strings may contain placeholders for values, amounts or names. These placeholders have one of the forms:

- ◇ Numbered placeholders start with a percentage sign ('%') followed by a number, a letter, or both a number and a letter.
- ◇ Labeled placeholders start with a percentage sign ('%') followed by a label in curly braces ('{...}'). Type and format flags may follow the label, after a colon.

At run-time, the application replaces the placeholders by actual data, such as a name, a file path, or a number. When translating messages, you should copy the placeholders as they are. Notably, for labeled placeholders, you should *not* translate the label. In the translated messages, you may rearrange the placeholders in the text —that is, placeholder '`%2s`' may come before '`%1s`'.

If a message string contains a double percentage sign, '`%%`', it should be copied to the translations as is. The double percentage is translated to a single percentage symbol. This would only occur in messages that contain placeholders.

Message tags

Rosette knows a concept of 'tagging' a message for either specifying a context for the message, or for selecting a message by a *label* instead of by the text of the key language.

A 'tag' must be placed in front of the message. It starts with a '$' symbol, and a label follows. The label is a name; it may contain digits, underscores and dashes, but no punctuation or spaces. The first character (directly after the $) must be a letter.

In translations, the tag should be copied verbatim; it should *not* be translated.

The messages and menu text

All messages, menus and dialog texts are in the file '`VisualPlace.cat`'. This file is in the '`doc`' subdirectory below where VisualPlace is installed.

It is common that there is a message in the file that starts with the keyword '`$Languages`'. It is also common that this message is at the top of the catalog. The '`$Languages`' keyword is a *tag*, see the previous section. When adding a new language to the catalog, you should add its name and ISO code to this list. The ISO labguage code (between parentheses) may be a two-letter code, such as 'fr' or a combination of language and country codes, such as 'fr-CA'.

For example, if a catalog already contains English and German, and you wish to add a French translation, the original catalog will contain:

```
en: $Languages English (en); Deutsch (de)
de: *
```

You then adjust this to:

```
en: $Languages English (en); Deutsch (de); Français (fr)
de: *
fr: *
```

The reason for the lines '`de: *`' and '`fr: *`' is that for this particular message, VisualPlace will *only* use the message in the key language. There should be strings present for the other languages as well (to avoid warning messages from Rosette), but their contents are never used. It has therefore become common practice to translate this particular message to a '`*`'.

For all other messages in the file, you should add another message in the new language. The language code for the new language must be the same as the one that you gave at the `$Languages` list.

Never change the string in the key language (‘en’).

Apart from the escape characters described on page 1, the strings describing menu items and buttons for the user interface may contain a ‘&’ prefixed to the word or even in the middle of the word. This & character serves to indicate the *shortcut* letter. In the translation, there should preferably also be a shortcut letter, but it does not need to be on the same letter. For standard menu items and dialog fields, you may look at how other applications define the shortcuts, for application-specific menu items/fields, you can choose any letter that does not cause conflicts with other items/fields. If no suitable letter can be found, you can leave the shortcut out of the translation.

The tooltip help

Short help for each field in a dialog is provided via a call-out (or balloon). The texts in these call-outs comes from the file ‘VisualPlace.tips’. This file is in the ‘doc’ subdirectory below where VisualPlace is installed.

The format of this file is different from the ‘VisualPlace.cat’ file in various respects:

- The ‘key language’ is not ‘en’, but ‘00’ and the key that follows is a number instead of a string. You should not change this number.
- Some HTML commands may be used for formatting, specifically ‘<p>’, ‘
’, ‘’. . . ‘’, ‘’. . . ‘’ and ‘<u>’. . . ‘</u>’. The code used to display these help texts is based on the ‘DrawHTML’ function; for more information, see https://www.compuphase.com/drawhtml.htm.

Resources and references

The Rosette conversion tools are available from the CompuPhase site. The direct link is https://www.compuphase.com/software_rosette.htm.
This page also contains a white paper on Rosette, with more details on the catalog format.