Localisation mini guide

Purpose

Localisation is designed to provide a support in software for several langage. In the following text, a given langage supported is refered as « locale »

Localisation is provided by a set of files in a given folder designed in the software itself. For VSO products, the folder used for locale langages is often in lang subfolder of the software main executable folder. Other location are possible. The localisation file names always start with a custom prefix, designed in the software.

When the software start, it parses in this localisation folder for all the files having the matching prefix and if the file is of a given locale, it is added to the list of the available locales. Once the parsing is done, the software will apply a selected locale, or by default the locale of the user account, or perform no localisation if there is no matching locale files for the selected/default locale langage. In such case, the used langage will be the one hardcoded in the software.

Localisation files format

Localisation files are INI files, containing only text.
The localisation files can be edited with a regular **UNICODE** text editor, however this manual edition is not encouraged. A recommended locale editor is described later.

INI files contains a header composed by 2 sections (not designed to be modified manually):

```
[Locale]
Ident=VSO Unicode localisation file ← this must not be changed
version=1 ← format version number of this localisation file, current is 1
translation_date=11/12/2009 12:27:43 PM ← Date of the latest modification
translation_date_locale=jeudi 12 novembre 2009 12:27:43 ← localised date
email=http://www.colok-traductions.com ← optionnal contact email
loc_name=French (Standard) ← localised name of the langage
eng_name=French ← English name of the langage
comment='' ← optionnal comments
translator_name=Colok ← optionnal name of the person that did the translation
code_page=1252 ← windows code page for that locale, to convert unicode → mbcs
lang_code=1036 ← windows langage code for that locale
flag_file=flag_French.png ← name of a file containing the flag of the language
(png, gif, bmp, tiff + a few other formats supported)

[Global]
Tin_code_string_count=901 ← number of strings hardcoded in the source code
Tin_comp_string_count=708 ← number of strings / objects hardcoded in the visual
components
```


Following the headers, there is as many sections as there is strings to localise, either in code and in visual components. These sections can be modified manually

In the source code. This is the strings directly hardcoded in the source code.

```
[0039_INCODE] ← A string hardcoded in the source code
original='Thumbnail not used in this template.' ← The hardcoded text
revision=1 ← The revision number of the original text
tag='menu_ts' ← An optionnal tag given to the string
comment='in the menu' ← An optionnal comment given to the string
locale='Vignette non utilisée dans ce modèle de menu.' ← the localised text
locale_rev=1 ← The revision number of the localised text.
```

NOTES:

- original and revision entries are provided only for informative purpose, as they are hardcoded into the source of the software.
- A localised text will be applied only if **locale_rev** taken from this file is equal to **revision** taken from the hardcoded values.

In the visual components. This is the values directly set in the visual component properties.

```
[0005_INCOMP] ← a string hardcoded in one visual component
control=Frm_DivxtoDvdMain.MnuLoad.Caption ← location in components of the
string, under the form parent.[parent.]componentname.propertyname[.listindex]
original='Load Project' ← the hardcoded text
revision=5 ← revision number of the hardcoded text
locale='Ouvrir un projet' ← the localised text
locale_rev=5 ← the revision number of the localised text
```

A FONT can be included in the localisation, as it may be interesting to change some of its characteristics such as the default size for langage that may require bigger font size, such as Japan or Chinese. Here is a typical font entry

```
[0000_INCOMP]
control=Frm_DivxtoDvdMain.Font
original='Charset=1|Color=-16777208|Height=-11|Name=Tahoma|Orientation=0|
Pitch=fpDefault|Size=8|Style=' ← list of all the published properties in the
original object and their values, separated with |
revision=5
locale='Charset=1|Color=-16777208|Height=-11|Name=Tahoma|Orientation=0|
Pitch=fpDefault|Size=8|Style=' ← list of all the published properties to apply
in the locale, separated with |. The order/number of properties is not important
(if you need to modify only the Height and orientation, you can write
locale=Height=-15|Orientation=1
locale_rev=5
```


Translation editor software (editloc.exe)

A translation editor is recommended to edit/modify/update a locale langage. The editor maintain a sorted list of all the strings used, as well as a revision number for each string. It also provide dynamic searches throughout all the strings, WYSIWYG visualisation for Mini-HTML formatted strings, WYSIWYG edition of the fonts, …

To work, translation editor need a **original** file that is generated by the software to localise, using the string /co.
For instance, launch « **MySoftware.exe /co** » to regenerate an original file that is up to date with the launched version of the software.

To edit locales, put the translation editor software into the folder of the original file, and launch it.

In the string edition dialog, you see and ordered list of strings, with a status:

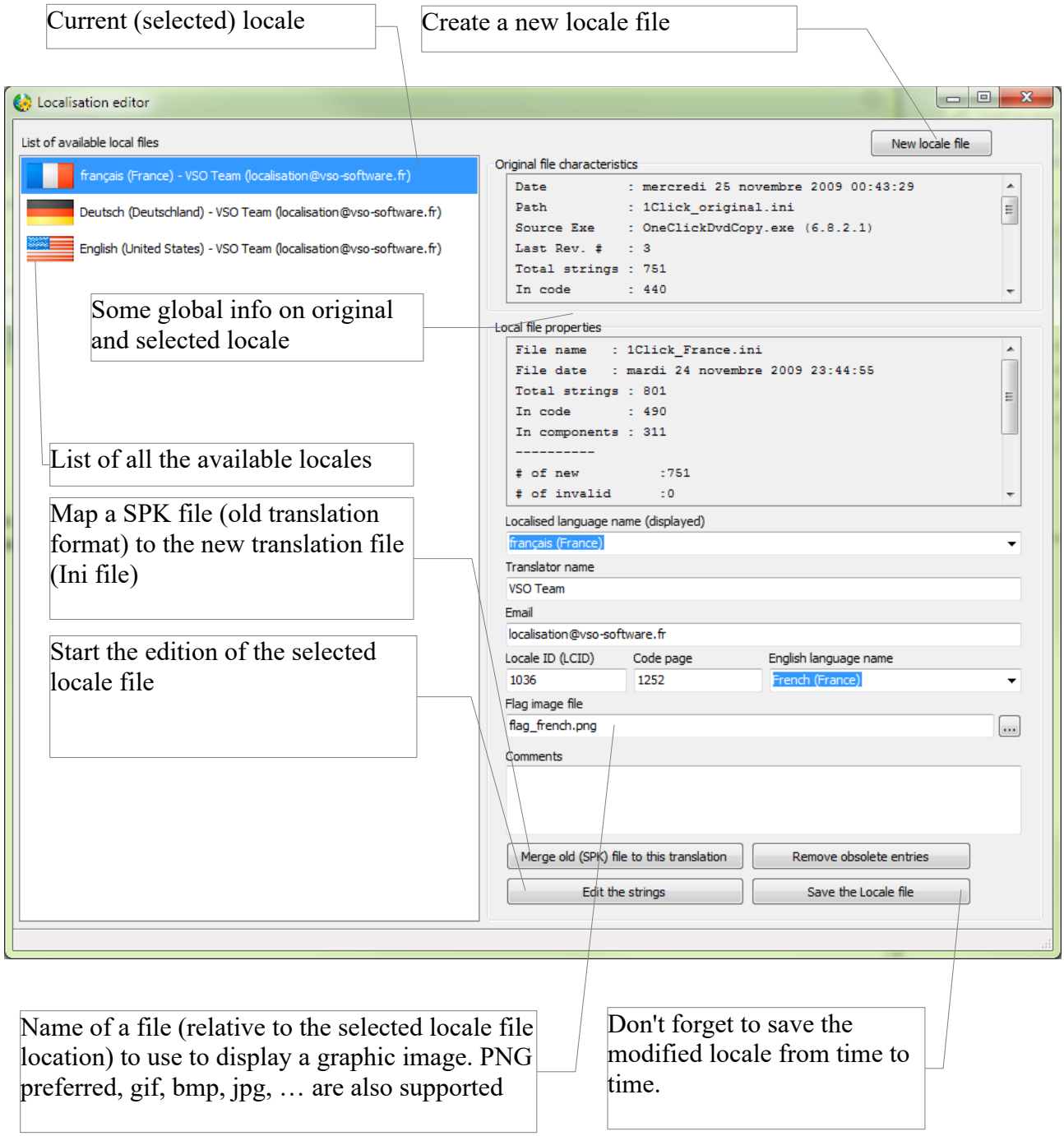
- new . The string is new
- need update. The string was already there, however it has been modified in the original version. So the localised version need to be checked/updated as well
- unverified. The string has been localised, however it's a sensitive string. It need to be validated before it is applied to the translation. Validation of such string is done with a specific version of the localisation editor.
- invalid If the original string is a format string (on the model of the delphi function Format('format string',[arg1,arg2,...]) or C function printf(« format string »,arg1,arg2,...)) and the list of arguments differ in number, order and / or type between the 2 strings
- up to date. The string is up to date, so it'll be applied when the matching locale langage is selected
- unused. The string is no longer used, however it's still in the translation for backward compatibility

NOTE: the unverified status is for the sensitive strings for which a validation by the software author is required before the localised version would apply. This is for instance to protect links to vendors web sites, and prevent from rogue translation redirecting to fake sites. To validate such strings, a special version of the translation editor is needed which can validate the string using a popup menu in the string list

SEARCHES:
This dialog also provide 3 editboxes at the bottom for the purpose of searching strings.
It searches in the matching section (Tag, localised text, original text) any string that contains at least one occurrence of the string typed in the editbox. Search is performed as text is typed , and non matching entries are immediately removed from the list.

CHECKBOXES:
- Hide unused – if checked, the unused (obsolete) strings are not displayed in the list
- Jump to next edit – If checked, when in edit mode hitting the <ENTER> key will switch to the next string to edit / update in the list

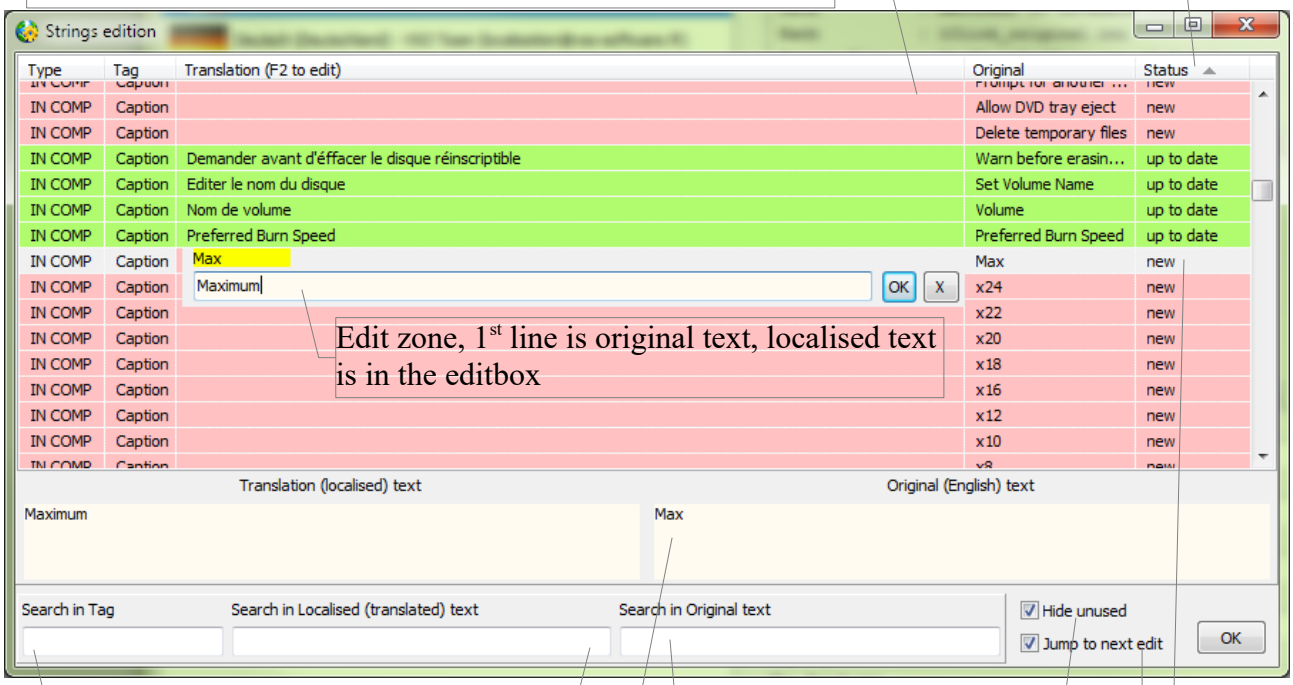
Main window of the locale editor:



the edit window

Columns are RESIZABLE and SORTABLE.
WARNING: To enlarge the size of the « original » column, click between translation and original column, and drag cursor in the right (→) direction

Translation (localised strings): Select one with the mouse or cursor, and hit **<F2>** key to start edit.
While editing, **<Enter>** or **<down arrow>** start to edit next string, **<up arrow>** edit previous string



Edit zone, 1st line is original text, localised text is in the editbox

Dynamic search in the tags of each strings

Dynamic search in the translated text (any string containing the typed text will match)

Dynamic search in the original text (any string containing the typed text will match)

WYSIWYG for Mini HTML formatted strings (if component / string is mini-html formatted)

Hide strings that are not in use in the software (obsolete strings …)

If checked, when ending an edited string, it'll automatically start with the next string to translate / update

String status: See text above for explanation

Important notes :

Modification of the translation file with a text editor :

Modifying directly the localisation file(s) with a text editor is possible but not recommended.

* The file must be saved in **UTF16 Little Endian**, which is the native unicode format of Microsoft Windows. Text editor such as NOTEPAD++ can save text file with this format (Preset under the Encoding Menu)

String formatting (Important : must read) :

Some strings requires to insert some variables values that are not known at the time of the translation. For instance, displaying a video resolution, or a time duration.
There is 3 different flavour of parametized strings. They are detailed here

1 – Delphi format() styled strings (roughly equivalent to C sprintf()):

Assemble a string with an array of arguments with the following form :

```
"%" [index ":"] ["-"] [width] ["." prec] type
```

A format specifier begins with a % character. After the percent sign come the following elements, in this order:

1. An optional argument zero-offset index specifier (that is, the first item has index 0), `[index ":"]`.
2. An optional left justification indicator, `["-"]`.
3. An optional width specifier, `[width]`.
4. An optional precision specifier, `["." prec]`.
5. The conversion type character, `type`.

Type character can be any of the following (case insensitive) :

`d`	Decimal. The argument must be an integer value.
`u`	Unsigned decimal. Similar to `d`, but no sign is output.
`e`	Scientific. The argument must be a floating-point value. The value is converted to a string of the form "-d.ddd...E+ddd".
`f`	Fixed. The argument must be a floating-point value. The value is converted to a string of the form "-ddd.ddd...".
`g`	General. The argument must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format.
`n`	Number. The argument must be a floating-point value. The value is converted to a string of the form "-d,ddd,ddd.ddd...".
`m`	Money. The argument must be a floating-point value.
`s`	String. The argument must be a character, a string,
`x`	Hexadecimal. The argument must be an integer value.

2 – composed string (custom formating)

This is an VSO custom string formating, designed to make translation more simple and self-explanatory.

Explicit argument list is provided at the beginning of the string, in a comma-separated list of value names enclosed with bracket {}. There is no value type, all are considered as string.

Each occurrence of the argument name enclosed with brackets will be replaced by its value in the final text.

Exemple :

« {height,width}Video resolution is {width}x{height} »

will output if the value of width is « 1920 » and value of height is « 1080 »

« Video resolution is 1920x1080 »

Such string can be fully translated into another langage, including the arguments names (all that is enclosed in {}). It's optional, they can be left untouched.

3 – Bash-Dos command line styled variables

Some strings embed explicit variables names surrounded with the percent sign (%) :

« %VARIABLE% »

the variable names can contains letters, numbers, dot

exemple :
« Your software version is %User.ExeVersion%. »

In the translation, the variable names must be left untouched.

Mini HTML reference

(taken from http://www.tmssoftware.com/site/minihtml.asp)

The mini HTML implementation to display text with HTML tags in various TMS components is a subset of the HTML standard and supports following tags :

B : Bold tag
<B> : start bold text
</B> : end bold text

Example : This is a <B>test</B>

U : Underline tag
<U> : start underlined text
</U> : end underlined text

Example : This is a <U>test</U>

I : Italic tag
<I> : start italic text
</I> : end italic text

Example : This is a <I>test</I>

S : Strikeout tag
<S> : start strike-through text
</S> : end strike-through text

Example : This is a <S>test</S>

A : anchor tag
<A href="value"> : text after tag is an anchor. The 'value' after the href identifier is the anchor. This can be an URL (with ftp,http,mailto,file identifier) or any text.
If the value is an URL, the shellexecute function is called, otherwise, the anchor value can be found in the OnAnchorClick event
</A> : end of anchor

Examples : This is a <A href= "mailto:myemail@mail.com ">test</A>
This is a <A href="http://www.tmssoftware.com">test</A>
This is a <A href="somevalue">test</A>

FONT : font specifier tag
<FONT face='facevalue' size='sizevalue' color='colorvalue' bgcolor='colorvalue'> : specifies font of text after tag.
with

- face : name of the font
- size : HTML style size if smaller than 5, otherwise pointsize of the font
- color : font color with either hexidecimal color specification or Borland style color name, ie clRed,clYellow,clWhite ... etc
- bgcolor : background color with either hexidecimal color specification or Borland style color name </FONT> : ends font setting

Examples : This is a <FONT face="Arial" size="12" color="clred">test</FONT>
This is a <FONT face="Arial" size="12" color="#FF0000">test</FONT>

P : paragraph
<P align="alignvalue" [bgcolor="colorvalue"] [bgcolorto="colorvalue"]> : starts a new paragraph, with left, right or center alignment. The paragraph background color is set by the optional bgcolor parameter. If bgcolor and bgcolorto are specified,
a gradient is displayed ranging from begin to end color.
</P> : end of paragraph

Example : <P align="right">This is a test</P>
Example : <P align="center">This is a test</P>
Example : <P align="left" bgcolor="#ff0000">This has a red background</P>
Example : <P align="right" bgcolor="clYellow">This has a yellow background</P>
Example : <P align="right" bgcolor="clYellow" bgcolorto="clred">This has a gradient background</P>*

HR : horizontal line
<HR> : inserts linebreak with horizontal line

BR : linebreak

 : inserts a linebreak

BODY : body color / background specifier
<BODY bgcolor="colorvalue" [bgcolorto="colorvalue"] [dir="v|h"] background="imagefile specifier"> : sets the background color of the HTML text or the background bitmap file

Example : <BODY bgcolor="clYellow"> : sets background color to yellow
<BODY background="file://c:\test.bmp"> : sets tiled background to file test.bmp
<BODY bgcolor="clYellow" bgcolorto="clWhite" dir="v"> : sets a vertical gradient from yellow to white

IND : indent tag
This is not part of the standard HTML tags but can be used to easily create multicolumn text
<IND x="indent"> : indents with "indent" pixels

Example :
This will be <IND x="75">indented 75 pixels.

IMG : image tag
<IMG src="specifier:name" [align="specifier"] [width="width"] [height="height"] [alt="specifier:name"] > : inserts an image at the location

specifier can be : idx : name is the index of the image in the associated imagelist
ssys : name is the index of the small image in the system imagelist or a filename for which the corresponding system imagelist is searched
lsys : same as ssys, but for large system imagelist image
file : name is the full filename specifier
res : name of a resource bitmap (not visible at design time)
no specifier : name of image in an PictureContainer

Optionally, an alignment tag can be included. If no alignment is included, the text alignment with respect to the image is bottom. Other possibilities are : align="top" and align="middle"

The width & height to render the image can be specified as well. If the image is embedded in anchor tags, a different image can be displayed when the mouse is in the image area through the Alt attribute.

Examples : This is an image <IMG src="idx:1" align="top">
This is an image <IMG src="ssys:1"> and another one <IMG src="ssys:worfile.doc">
This is an image <IMG src="file://c:\my documents\test.bmp">
This is an image <IMG src="res://BITMAP1">
This is an image <IMG src="name">

SUB : subscript tag
<SUB> : start subscript text
</SUB> : end subscript text

Example : This is ⁹/₁₆ looks like 9/16

SUP : superscript tag
<SUP> : start superscript text
</SUP> : end superscript text

BLINK : blink tag(supported in TAdvStringGrid and descendants and THTMListbox)
<BLINK> : start blinking text
</BLINK> : stop blinking text

Example : This is <FONT color="clred"><BLINK>blinking red</BLINK></FONT>text.

UL : list tag
<UL> : start unordered list tag
</UL> : end unordered list

Example : <UL>
<LI>List item 1
<LI>List item 2
<UL>
<LI> Sub list item A
<LI> Sub list item B
</UL>
<LI>List item 3
</UL>

LI : list item
<LI> : new list item

SHAD : text with shadow
<SHAD> : start text with shadow
</SHAD> : end text with shadow

Z : hidden text
<Z> : start hidden text
</Z> : end hidden text

HI : hilight
<HI> : start text hilighting
</HI> : stop text hilighting

E : Error marking
<E> : start error marker
</E> : stop error marker

Special characters
Following standard HTML special characters are supported :
< : less than : <
> : greater than : >
& : &
" : "
  : non breaking space
™ : trademark symbol
€ : euro symbol
§ : section symbol
© : copyright symbol
¶ : paragraph symbol

