Migration Tutorial

Tutorial for migration of Delphi/C++Builder localization projects to Multilizer 6.0.

Background

This document is intended for Multilizer users that have been localizing software using component localization.

Because Multilizer components have been removed, Multilizer 6 users have to migrate to binary localization technology that was introduced in Multilizer 5.0.

Following table summarizes the differences between component and binary localization.

Feature	Component localization	Multilizer 6		
		Binary localization		
Basic features (also as end-users will experience it)				
Language change on the fly	Yes	Yes		
Translation of UI	Yes; requires Translator	Yes; no components/code needed.		
	component on every form.			
Translation of hard-coded strings	Yes; requires Multilizer's	Yes; requires use of Delphi		
	Translate() function.	resourcestring clause.		
Different UI layouts for each target	No.	Yes. Wysiwyg allows localization of		
language.		form layouts.		
Performance				
Storage for translations	MLD (binary file), text file, or	Resource DLL's or resources in the		
_	database.	EXE.		
Impact on localized EXE	Moderate; strings are translated	No impact when running software;		
performance	on run-time each time a form is	works just as any software		
	shown.	compiled with Delphi/C++Builder.		
Impact on localized EXE's code	A few hundred Kilobytes.	0 Kilobytes (without language		
size.		change on the fly).		
EXE startup performance.	Moderate impact, if there are lots	No impact.		
	of translations.			
Localization engineering specifics				
Required localization source.	Requires Delphi/C++Builder	Requires compiled binary (e.g.,		
	project, and executable.	EXE).		
Required software re-engineering.	Drop Multilizer Translator			
	component on every form in the			
	application, and Dictionary			
	component on the mainform.			
Support for visual form	No.	Yes. Requires access to source		
inheritance.		code, when creating localization		
		project.		

Table 1: Comparison of Multilizer 6 localization and Component Localization

1. Open project

Before opening existing Multilizer 5.x project, take a backup of it.

When opening an old Multilizer project, Multilizer will popup the following informative dialog.



Figure 1: Opening an old Multilizer project will make Multilizer 6 upgrade it,

Multilizer 6.0 - El\test\Component_L10N\d7_binarydic\Project1.mpr Fie Edit View Project Column Row Tools Heb				
D 🛩 🖬 🖻 🖀 🕺 🐿 🛙	· A = 5 7 7 7 ∕ 2	▶ Finnish	× •	
E- 🐌 Project 1. exe	Context	Native	Finnish 🔺	
🖶 🧰 Bitmaps 🗄 🛄 Cursors	Cursors.2.Cursor	R Ticon		
B D Forms B D Icons	Cursors.3.Cursor	Tioon Tioon		
🗄 🚞 Strings	Cursors.4.Cursor	÷ Tioon		
	Cursors.5.Cursor	+∥ + Ticon		
	Cursors.6.Cursor	Ticon Ticon		
	Cursors.7.Cursor	Ø Tioon		
	Forms.TForm1.Caption	Form1	Lomake1 ^	
	Forms.TForm1.Color	clBtnFace		
	Forms.TForm1.Font	(MS Sans Serif, 8, cfw/indowText)		
	Forms.TForm1.Label1.Caption	Label	Form1_Etiketti1 ^	
	Forms.TForm1.Button1.Caption	Button1	Form1_Button1	
	Forms.TForm1.Memo1.Text	Memo1		
	Forms.TForm1.Button2.Caption	Button2		
	Forms.TForm1.WBinaryDictions	Dictionary1		
	Forms.TForm1.IvBinaryDictions	E:\test\Component_L10N\d7_b	*	
Native Row count: 292 String count: 269 Word count: 774	Finnish Translated rows: 6 Untenslated rows: 286 2%	Diffees Do not translate" rows: D lew rows: 292 Jnueed rows: D Jsed rows: D	Row Native length: 11	
E Log Validate ∑ Status 26: 292		Not ready	New	

After this, Multilizer opens the project and displays the translation grid as usually. Before going further, save the project (Don't choose Save As...).

Figure 2: Multilizer 6 UI with an old (ML 5.x) project opened.

When opening a Multilizer 5.x project, Multilizer 6 will import all languages with accompanying translations. These strings are shown in 'Forms' and 'Strings' nodes in Project Tree.

If there were hard-coded strings in ML 5 project (in 'source' node in project), they were not imported yet. If there were project strings in ML 5 project, there weren't imported either. See chapter **5** to import these.

As shown in picture above, there is a lot of non-string data in project. In order to hide them, select appropriate filter (View \rightarrow Filter...).

Grid Filter	×
Data Types Translation Statuses Row Statuses Dthess Boolean values Colors Colors Floating point numbers Forsts INE modes INE modes Reading orders Status Short cuts Short cuts Strings	
OK Cancel Default He	qle

Figure 3: Defining a filter to show strings only.

2. Define type of localized files

Before going further, the type of localized files needs to be specified; right-click target-node in project tree (Currently selected in previous image) and choose properties.

×
idicVProject1.exe
nts IME Resources
Resource file options Copy all resources Write to root directory Enbedded resource file
eydo
Cancel Default Help

Figure 4: Output settings for binary-localized files.

Binary localization supports three kinds of localization output file types:

- Localized file produces one exe for each language.
- **Resource files** produces files with compiled resources, one for each target language. This is the format that Borland® promotes. Using this format enables run-time language switch; implementation is described in chapter **4**.
- **Multilingual file** includes all translated resources in one executable. It starts in the language matching Windows default locale.

3. Configurations that affect localization work

Configurations that affect the work with Multilizer project are found on project tab. Read this chapter to know, the reason for specifying Delphi project (and source) location, and Delphi DRC-file location.

Delphi Target 🔀
Ele E-Ver//Component_L10N1d7_binarpdic/Project1.eve
Options Project Encodings Output Fonts INE Resources
Project file name: E:/vect/Component_L10N/d7_binarydic/Project1.dpr
E:\test\Component_L10N\d7_binaydic\Project1.drc
Options Scan images from the form data
Check gcaled property Set BDMode property
Wate wide stangs
DK Cancel Default Help

Figure 5: Delphi target, project options.

Enable display of visual inheritance

Multilizer 6 is capable of showing visual inheritance of forms. This reduces time spent on translation; translations populate inherited forms just as the same code is propagated to inherited classes.

Turn on this setting by specifying project name.

Multilizer 6.0 - E:\test\bin_localization_frametest\Project1.mpr		
<u>File E</u> dit <u>V</u> iew <u>P</u> roject <u>C</u> olumn <u>R</u> ow <u>T</u> ools <u>H</u> elp		
🗅 😂 🔳 🖻 🚭 👗 🛍 🖷 🛤 🗮 🖬 🖷 🖬 🖷 🖓 🚱 🕨 Finnish		
Project1.exe Forms The TBaseFrame ToverFrame ToverFrame TivSelectResourceL Strings		

Figure 6: Visual Inheritance: TCheckFrame is inherited from TBaseFrame. Translations done in ancestor are inherited to child.

Enable full context for resource strings

Delphi compiler generates numeric id's for all strings declared with resourcestring clause in Delphi source code. These numeric id's may change when recompiling the software, which changes the context of the string in

Multilizer. In order to keep the context the same, Multilizer needs to have access to the DRC file generated by Delphi compiler. Using this Multilizer is able to use a context that is the same as resourcestring constant name.

Turn on this setting by specifying DRC file name.

E- 🎾 Project1.exe	Context	Native
🕀 🚞 Bitmapz	4084.65340	- Dock zone has no control
E Cursors	4084.65339	- Dock zone not found
E- Forme	 4093.65472 	%s (%s, line %d)
- TBaseFrame	+ 4090.65432	"%s" is not a valid component name
TOverFrame	+ 4096.65520	%s' is not a valid integer value
TivSelectResourceL	4039.65416	%s not in a class registration group
	+ 4089.65414	%s.Seek not implemented
	 4096.65373 	5.Abort
4081	 4085.65344 	LA.8
- 22 4082	 4087.65390 	LA.S
- 🎎 4083	 4087.65386 	&Close
- 24 4084	4087.65385	8.Help

Figure 7: Strings with resource id's as context; Delphi compiler creates the id's, and on each compilation resourcestrings may get different id's.

E- 🥻 Project1.exe	Γ	Context	Native A
🖻 🖼 Forms	ł,	Consts.SDockZoneHasNoCtl	- Dock zone has no control
🖻 🔚 TBaseFrane		Consts.SDockZoneNotFound	- Dock zone not found
TCheckFrame		SysConst. SAssertError	%s (%s, line %d)
- TForm1		Consts.SinvalidName	"%s" is not a valid component name
TivSelectResourceL		SysConst. SinvalidInteger	%s' is not a valid integer value
Constr	-	Consts.SMsgDlgAbort	&Abort
- SusConst	•	Consts.SMsgDlgAll	8,All
- 👪 Unit1		Consts.SMsgDlgHelp	&Help
		Consts. SMsqDlqlqnore	&lgnore

Figure 8: Strings with full context; context is formed out of unit name and resourcestring constant name.

4. Changes in code

While Multilizer 6 takes care of upgrading the project to the new format, Developers have to take care of the changes in Delphi/C + +Builder project.

Generally there are two things to do:

- 1. Remove Multilizer components
- 2. Stop using Multilizer's Translate() function. These strings were placed in "source" node in ML 5.x projects, after doing the change below the strings will be placed in "strings" node in ML6 project.

```
Example: if you earlier had Delphi code like this:
var
   s : String;
begin
   s:=IvDictionary1.Translate(`Hello world!');
change it to
resourcestring
   strHello = `Hello world!';
var
   s : String;
begin
   s:= strHello;
```

Dynamic language change can be added if output files are "resource files". Use SelectResourceLocale function (located in IvResLangD unit) to show a language selection dialog. Add IvResDLL in mainform's uses clause to enable runtime language switch.

Translation of Common Dialogs is enabled, if you add IvDialogs.pas to the Project Manager, or add IvDialogs unit to the uses clause anywhere in the application.

5. Finalize migration

After changes in code, build the new executable(s). Once the native executable(s) are working properly, rescan them in Multilizer project.

Now Multilizer will find new strings, if you used resourcestring clause as described in previous chapter. Translations for these can be imported from Multilizer 5 project; choose File \rightarrow Import... and specify location of the Multilizer 5 project.

Finally build localized items in Multilizer and test them.

Other migration paths

Migrate from Multilizer 5 binary localization projects

It's easy to migrate to Multilizer 6; simply open ML 5 project in Multilizer 6. No changes in Delphi/C++Builder source code are required.

Migrate from Multilizer 4 or older projects

Best results in this migration path are achieved by first creating a new Multilizer 6 project, and then importing (File \rightarrow Import...) translations to it.

In addition the same code changes as in migration from Multilizer 5 component localization project needs to be done.