Localising With Delphi 5

by Marco Cantù

If you take a quick look at Delphi 5 Enterprise to see what's new, and if you are living in or working for a non-English-speaking country, you will immediately find what I think is the best new feature: the Integrated Translation Environment, or ITE for short. Actually, this is not really totally new: Delphi 4 provided a bare Resource DLL Wizard, which is still in Delphi 5 but has been completely updated. Veteran Delphi developers will also remember the very expensive and quite unfinished Borland Translation Suite, and the cut-down Delphi Translations Pack, which provided ready-to-use translations of the VCL messages.

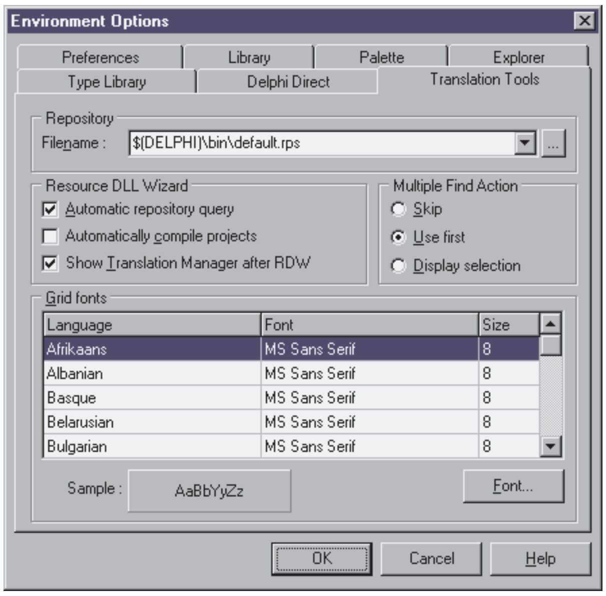
The not-so-great technologies of the past have now been turned into an interesting tool. I will cover the ITE briefly and then I'll delve into the core of the article: the built-in VCL support for multi-language resources.

Welcome To The ITE

The ITE is a collection of tools and forms part of the Delphi IDE. The tools exploit some VCL features which I'll describe later. Most of the ITE features can be activated from the `Project | Language` menu of the Delphi IDE. This menu has four commands: you can add a new translation (opening the Resource DLL Wizard), remove a translation, set the active language (for debugging purposes), and update the resource DLL (refresh the current data). You can also perform all these operations from the Translation Manager, which you can activate from the `View` menu.

Another menu item related to the ITE is `Tools | Translation Repository`, which opens a repository of default translations for words or sentences appearing multiple times in your application, or translations you want to share amongst multiple applications.

Finally, for customising the ITE you can move to the new `Translation Tool` page of the `Environment Options` dialog you can see in Figure 1. Some of the options are not intuitive, but they can be figured out. I'm not sure why you are supposed to use different fonts for the various languages in the Translation Manager. Notice that the repository is based on a file, so that you can have different repositories for different projects, or (more interestingly I guess) share the repository amongst multiple programmers.


Environment Options

Tabs: Preferences | Library | Palette | Explorer | Type Library | Delphi Direct | Translation Tools

Repository
Filename: $(DELPHI)\bin\default.rps

Resource DLL Wizard
- [✓] Automatic repository query
- [ ] Automatically compile projects
- [✓] Show Translation Manager after RDW

Multiple Find Action
- ( ) Skip
- (●) Use first
- ( ) Display selection

Grid fonts

Language	Font	Size
Afrikaans	MS Sans Serif	8
Albanian	MS Sans Serif	8
Basque	MS Sans Serif	8
Belarusian	MS Sans Serif	8
Bulgarian	MS Sans Serif	8

Sample: AaBbYyZz

➤ *Figure 1: The Translation Tool page of the environment.*

The Resource DLL Wizard

This is the starting point for developing a localised version of a program. In the first page of the wizard you select the projects of the active group you want to translate. In the second page you choose one or more target languages (you can always add more languages later on). In the third page you can change the directory where the wizard will store the files it generates. By default it uses the three letter code of the Windows language or 'locale'.

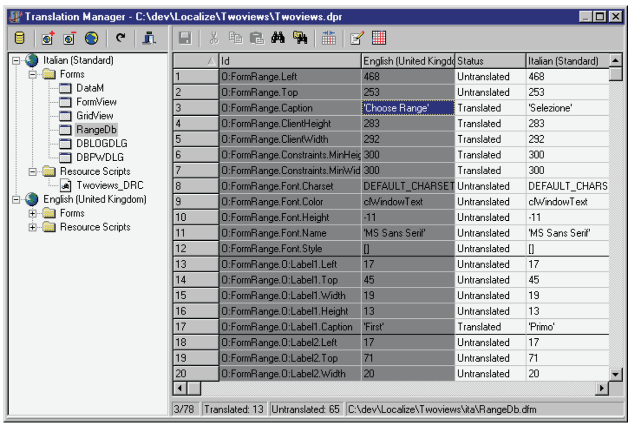
The fourth page of the wizard prompts you for the action to perform (adding a new language or updating). Considering that you cannot add a language already defined, as this will be removed from the list, I've had a hard time figuring out when this page can be used. The fifth and final page of the wizard provides a summary of the operations, and allows you to view a short statistical summary after the wizard is done.

Notice that the wizard requires you to save the current project when it starts and will automatically perform a compilation before generating the translated projects (there is apparently a Delphi bug preventing this process if the `Todo List` window is open). The translation involves extracting the form and string resources of the application. Form resources are the DFM files included in the project. String resources are the string constants declared with the `resourcestring` keyword.

While completing its work, the wizard generates a directory for each of the languages you have selected. Each of these directories contains a Delphi project and the DFM files and string resource files to be translated. The project is a library, with an extension corresponding to the three-letter locale code, including the various resources, but not the original code. The code will be only in the main executable file, while the translated resources will be in these libraries automatically loaded by the main program, as I'll describe later.

The Translation Manager

The Resource DLL Wizard was already available in Delphi 4, even if in a slightly different form. What is new in Delphi 5 is the support for



Translation Manager - C:\dev\Localize\Twoviews\Twoviews.dpr

- Italian (Standard)
  - Forms
    - DataM
    - FormView
    - GridView
    - RangeDb
    - DBLOGDLG
    - DBPWDLG
  - Resource Scripts
    - Twoviews_DRC
- English (United Kingdom)
  - Forms
  - Resource Scripts

	Id	English (United Kingdo	Status	Italian (Standard)
1	0:FormRange.Left	468	Untranslated	468
2	0:FormRange.Top	253	Untranslated	253
3	0:FormRange.Caption	'Choose Range'	Translated	'Selezione'
4	0:FormRange.ClientHeight	283	Translated	283
5	0:FormRange.ClientWidth	292	Translated	292
6	0:FormRange.Constraints.MinHeig	300	Translated	300
7	0:FormRange.Constraints.MinWid	300	Translated	300
8	0:FormRange.Font.Charset	DEFAULT_CHARSET	Untranslated	DEFAULT_CHARS
9	0:FormRange.Font.Color	clWindowText	Untranslated	clWindowText
10	0:FormRange.Font.Height	-11	Untranslated	-11
11	0:FormRange.Font.Name	'MS Sans Serif'	Untranslated	'MS Sans Serif'
12	0:FormRange.Font.Style	[]	Untranslated	[]
13	0:FormRange.0:Label1.Left	17	Untranslated	17
14	0:FormRange.0:Label1.Top	45	Untranslated	45
15	0:FormRange.0:Label1.Width	19	Untranslated	19
16	0:FormRange.0:Label1.Height	13	Untranslated	13
17	0:FormRange.0:Label1.Caption	'First'	Translated	'Primo'
18	0:FormRange.0:Label2.Left	17	Untranslated	17
19	0:FormRange.0:Label2.Top	71	Untranslated	71
20	0:FormRange.0:Label2.Width	20	Untranslated	20

3/78 | Translated: 13 | Untranslated: 65 | C:\dev\Localize\Twoviews\ita\RangeDb.dfm

➤ *Figure 2: The Delphi 5 Translation Manager.*

managing the translation task with two support tools, the Translation Manager and the Translation Repository.

The Translation Manager is the main window you'll work with, or you will get a professional translator to work with. This window lists all the available translations of the project and, for each translation, all the resources you can modify.

The main window (Figure 2) lists the elements you can modify in the translation and provides the original and translated text, but also tracks past versions of the original and translation, the change dates, and the translation status. Of course, you can filter this table and choose the columns you want to see, using its shortcut menu.

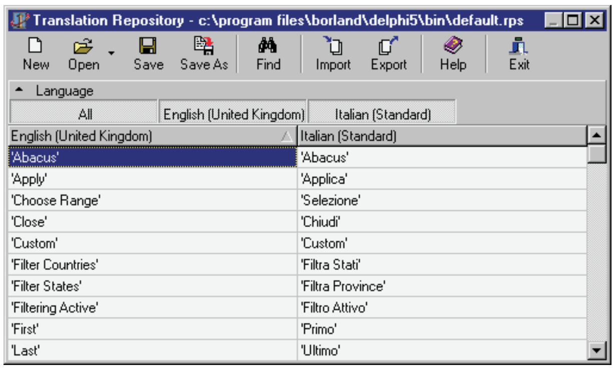
The resources you can modify include all the strings and all the string properties of the DFM files, plus a few other properties you might want to modify for the translation. For example, you can change the positions and fonts of most controls. This might be required when the different length of the translated text affects the user interface. Of course, it is not easy to determine whether the size and position of the controls is correct by looking at these numbers. What you can do is close the Translation Manager, select the new translation project (which is automatically added to the current project group), move to the form, and open it (Figure 3). The interesting feature is that you can freely edit the translated form (as long as you don't add any components to it), moving controls, changing fonts and so on.

When you re-open the Translation Manager (by selecting the main project and issuing the non-obvious command `Project | Languages | Update Resource DLL`), this will read the current values from the DFM files (the original and the translated ones) and refresh its structure accordingly. This way, if you update the original form, you don't have to translate it again, but only provide the translation for the new elements. At the same time, you can modify the translated DFM file and see the changes reflected in the translation system. All the information generated by the Translation Manager is saved in a series of files with the DFN extension. There is one DFN file for each form of each translation.

The Translation Repository

The Translation Manager works in conjunction with another tool, the Translation Repository (see Figure 4), where you can store the standard translation of a frequent term. Borland has promised to provide pre-defined repositories with the translation of VCL error messages in French, German, and Japanese.

You can update the repository manually (using the `Tools | Translation Repository` command to open it) or use the `Repository | Add strings to repository` command from the shortcut menu of the Translation Manager. You can use another command from the same shortcut submenu (`Get strings from repository`) to automatically translate all of the terms available in the repository. Notice that the repository can handle multiple translations for the same word, and other advanced elements.



Translation Repository - c:\program files\borland\delphi5\bin\default.rps

English (United Kingdom)	Italian (Standard)
'Abacus'	'Abacus'
'Apply'	'Applica'
'Choose Range'	'Selezione'
'Close'	'Chiudi'
'Custom'	'Custom'
'Filter Countries'	'Filtra Stati'
'Filter States'	'Filtra Province'
'Filtering Active'	'Filtro Attivo'
'First'	'Primo'
'Last'	'Ultimo'

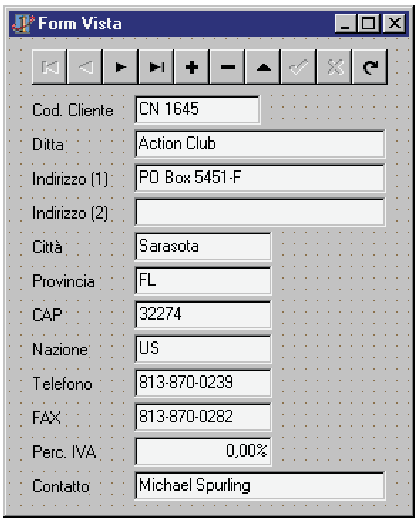
➤ *Figure 4: The Translation Repository.*

The data of the translation repository is saved in files with the RPS extension. You can handle multiple repositories and even export and import them to and from an XML format.

VCL Support

Once you've compiled the translated project, you can use the `Project | Languages | Set Active` command to activate it, so that running the project in the debugger will load the active language extension. This is only a test: usually you will simply need to run the main executable file and it will automatically use the translated DLL corresponding to the current regional locale selected on the computer (via the Control Panel's Regional Settings).

➤ *Figure 3: A translated form at design-time. You can resize components and even correct the translation in the form designer, and the changes will be incorporated back into the Translation Manager files.*



Form Vista

Cod. Cliente	CN 1645
Ditta	Action Club
Indirizzo (1)	PO Box 5451-F
Indirizzo (2)	
Città	Sarasota
Provincia	FL
CAP	32274
Nazione	US
Telefono	813-870-0239
FAX	813-870-0282
Perc. IVA	0,00%
Contatto	Michael Spurling

As I've already mentioned, the wizard creates a DLL project which includes the translated DFM file and strings, and has an extension that corresponds to the three-letter code which identifies the locale. This structure is visible in the project source code. In Listing 1 the `$R` directives determine the resources to include in the project (six forms and a set of strings), and are followed by the comments required by the ITE (you should be careful not to modify them). Notice that some of the forms, such as the database login dialog, are not part of the application but part of the VCL, and are stored in a specific subdirectory. The `$E` directive determines the extension of the executable file.

As you compile the resource DLL, its output is placed in the parent directory (thanks to a project option set up by the wizard), the same one hosting the project, so that it will become immediately available. As you start this executable file, the VCL will actually load the resources either from the main EXE file or from the DLL matching the current regional settings. This takes place in the `LoadResourceModule` function of Delphi's `System` unit. This function calls the `GetLocaleInfo` API and looks in the registry for locale overrides to determine the required locale, then loads either the language and country translation (using the three-letter extension) or the language-only translation (using the first two letters only). The library to use for loading the resource is determined at startup, before forms or strings are loaded, and is saved in the `ResInstance` field of the `TLibModule` data structure. A list of these data structures is referenced by the `LibModuleList` global pointer of the `System` unit.

➤ *Listing 1: The source code of a sample project generated by the Resource DLL Wizard.*

```
// Do not edit.This file is machine generated by the Resource DLL Wizard.
library Twoviews;
{ITE} {DFMFileType} {vcl\DBLOGDLG.dfm}
{ITE} {DFMFileType} {vcl\DBPWDLG.dfm}
{ITE} {DFMFileType} {DataM.dfm}
{ITE} {DFMFileType} {FormView.dfm}
{ITE} {DFMFileType} {GridView.dfm}
{ITE} {DFMFileType} {RangeDb.dfm}
{ITE} {RCFileType} {Twoviews_DRC.rc}
{$R 'vcl\DBLOGDLG.dfm' LoginDialog:TForm(Form)}
{$R 'vcl\DBPWDLG.dfm' PasswordDialog:TForm(Form)}
{$R 'DataM.dfm' DataModule2:TDataModule}
{$R 'FormView.dfm' Form3:TForm}
{$R 'GridView.dfm' Form1:TForm}
{$R 'RangeDb.dfm' FormRange:TForm}
{$R 'Twoviews_DRC.res' 'Twoviews_DRC.rc'}
{$E ita}
begin
end.
```

Hidden Gems Of The RichEdit Demo

This translation support offered by Delphi 5 is certainly powerful and can be handy when you need to provide multiple versions of your application. However, it has a definite limitation: to change the language within the program you need to close and restart it.

An alternative approach provided by Borland is the ability to modify the resource instance handle at runtime, reload the DFM files, and recreate the forms. However, not all of these features are part of the VCL, but are provided in a separate unit, `ReInit`, part of the RichEdit demo.

You can embed this `ReInit` unit in your programs and call its two routines to change the language dynamically:

```
function LoadNewResourceModule(
  Locale: LCID): Longint;
procedure ReinitializeForms;
```

The first routine changes the `ResInstance` field of the library module, after loading the proper translation library. The second reloads the resource file for every form used by the program. You can see the code of this function in the unit itself, so I won't describe it here. An example of its usage is given in the `SwitchLanguage` method of the main form of the RichEdit program.

The problem with this approach is that while recreating the forms you might lose the current editing and database connections. As an example, I've translated the TwoViews program from my book *Mastering Delphi 5*, to show you how to apply the `ReInit` unit to a simple database program (the program is a simple database example based on a data module and two different views of the same data, one grid-based and the other form-based, with some filtering and range setting capabilities).

In the sample program the database connection is kept because it is handled by a separate data module, and so the database fields' input is preserved while changing the language dynamically. The secondary form, where you can select ranges and filters, loses its current content when the language changes. You should make it modal, close it, or read in the current settings and re-apply them after switching the language.

Dynamic Language Loading

As you can see in Figure 5, the ViewGrp project group contains the original TwoViews program plus two localised versions, UK English and Italian. I've added the UK English version to highlight the fact that you might want differences from the US to the UK version of a program, to account for different spelling and more. The second reason was to better support dynamic linking: by adding a locale corresponding to the original program you can reload it in the same way you load a translation. Otherwise you would have to modify the code to load the resources from the main EXE for a standard language.

In the example, I've done a very fast translation of the main resources into Italian. I haven't translated the VCL error messages or the pre-defined VCL forms, but only the main forms. If you run the Italian version, or press the corresponding button, you'll realise there is an obvious error: the grid columns take their captions from the field names, which are not and should not be localised. The solution is to use the `Columns` property of the grid to make these strings explicit and include them in the translation.

The real feature of the program is its dynamic loading support. When it starts it gets the current user locale (which in this case should be either Italian or English), as you can seen in the `FormCreate` method of Listing 2. A specific button allows a user to change the language on the fly, with the code of the `LanguageSpeedButtonClick` method, which calls the two routines of the `ReInit` unit I've borrowed from the RichEdit Delphi demo.

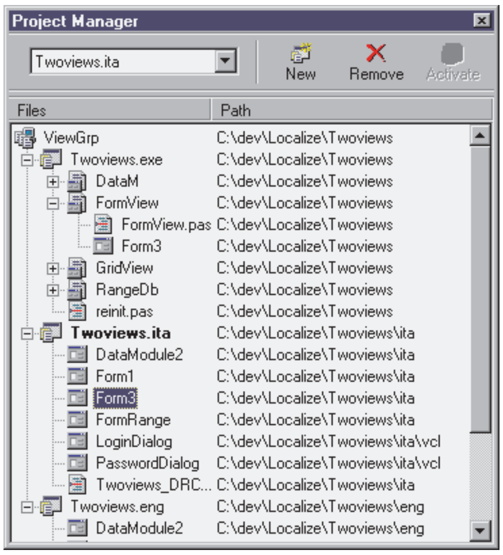
```
type
  TForm1 = class(TForm)
    ...
  private
    CurrLocale: LCID;
  end;
implementation
uses
  ReInit;
const
  ENGLISH = (SUBLANG_ENGLISH_UK shl 10) or LANG_ENGLISH;
  ITALIAN = LANG_ITALIAN;
procedure TForm1.LanguageSpeedButtonClick(Sender: TObject);
var Locale: LCID;
begin
  if CurrLocale = ENGLISH then
    Locale := ITALIAN
  else
    Locale := ENGLISH;
  if LoadNewResourceModule(Locale) <> 0 then begin
    ReinitializeForms;
    CurrLocale := Locale;
  end;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  CurrLocale := GetUserDefaultLCID;
end;
```

➤ *Listing 2: Portions of the code from the TwoViews example.*

Prepare To Translate

Although I have only a limited experience with the ITE, I've prepared a set of rules you can follow to help avoid translation problems. First, avoid hard-coded strings. Even your message boxes and exception messages should be based on resource string constants. Second, beware automatic strings, such as the automatic `DBGrid` headers and so on. You should make all VCL strings explicit at design-time, so they'll be included in the set of strings to translate. Lastly, to support dynamic language changes use data modules, which don't have to be re-created dynamically, instead of placing the dataset components in the forms.

➤ *Figure 5: A sample project group with two translations. Notice that only the main program includes the Pascal code.*



Project Manager

Twoviews.ita

New Remove Activate

Files	Path
ViewGrp	C:\dev\Localize\Twoviews
Twoviews.exe	C:\dev\Localize\Twoviews
DataM	C:\dev\Localize\Twoviews
FormView	C:\dev\Localize\Twoviews
FormView.pas	C:\dev\Localize\Twoviews
Form3	C:\dev\Localize\Twoviews
GridView	C:\dev\Localize\Twoviews
RangeDb	C:\dev\Localize\Twoviews
reinit.pas	C:\dev\Localize\Twoviews
Twoviews.ita	C:\dev\Localize\Twoviews\ita
DataModule2	C:\dev\Localize\Twoviews\ita
Form1	C:\dev\Localize\Twoviews\ita
Form3	C:\dev\Localize\Twoviews\ita
FormRange	C:\dev\Localize\Twoviews\ita
LoginDialog	C:\dev\Localize\Twoviews\ita\vcl
PasswordDialog	C:\dev\Localize\Twoviews\ita\vcl
Twoviews_DRC...	C:\dev\Localize\Twoviews\ita
Twoviews.eng	C:\dev\Localize\Twoviews\eng
DataModule2	C:\dev\Localize\Twoviews\eng

Conclusion

Although I haven't covered in detail all of the features of the Integrated Translation Environment provided by Delphi 5, I hope I've given you a feeling for its capabilities. Also, I've explored the technical support offered by the VCL to manage localised versions, including the dynamic loading provided by Borland in the RichEdit demo.

Programmers have devised many schemes for handling static or dynamic translations and localisation of Delphi programs, but the ITE support changes the picture. Although some third-party solutions are still viable, the built-in support is at last good enough for professional development. The only drawback is that is it very Delphi-centric. You can move all the strings to a repository, convert it to XML, let a professional translator work with an XML editor, import the XML-based translation, and apply the strings in the repository to the Translation Manager. But the translator needs Delphi Enterprise to check his work.

Marco Cantù (www.marcocantu.com) is the author of a number of books, which include *Mastering Delphi 5*, *Delphi Developer's Handbook*, and *Essential Pascal*, as well as articles. Marco lives in Italy, teaches Delphi classes and speaks at conferences around the world.

Editor's Note. *Together with Bob Swart, Marco Cantù recently received the 1999 Spirit of Delphi award from Inprise. Congratulations to them both!*