Chapter 16

Multilanguage support

In a global world it's usual that we have to speak more than one language. Also software must support since years more than one language. A lot of different implemtations are made but I want to focus on the way which is used by the FPC/Lazarus team. I my opinion it makes sense to use established methods instead of implementing new methods on your own. Of course it would be possible to implement a laguages table structures in the database and to store there the information about the text used on our forms. But we are going for the established approach. What kind of text must be translated and loaded depending on the laguage setting of the users operating system? The answer is easy: Every text which appears readable for the user! In FPC/Lazarus we have powerful tools to realize such a functionality. We start with an easy project, just a hello world application. We make a new project and add only a labe and a button. The label gets the caption "Testlabel description" and the Button just "Button1". To get the

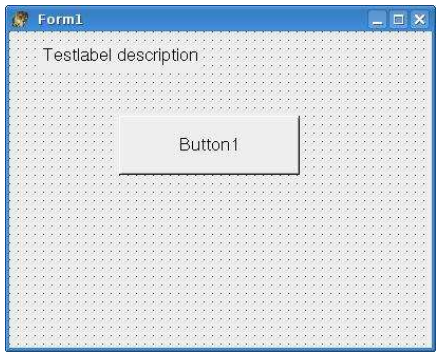

Figure 16.1: Hello World

multilanguage support we have to tell the IDE where to store the language files. This is done by the menu "Project-Project Options", this opens a dialog and on the tab "i18n" you can set the directory for the laguage files.

In the Lazarus project it's usual to use within the application directory a "languages" directory. So do we. In the code for the unit we must integrate

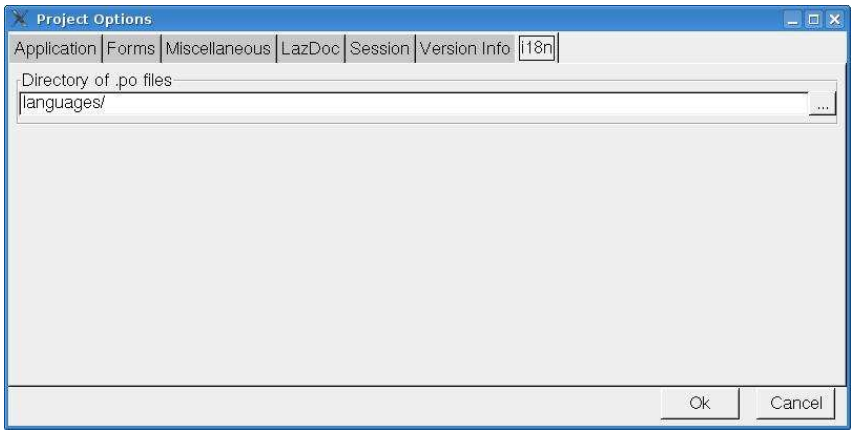

Figure 16.2: Project Options Dialog

into the uses clause the units "translations" and "gettext":

```
uses
  Classes, SysUtils, LResources, Forms, Controls, Graphics, Dialogs, Buttons, transl
  gettext, StdCtrls;
```

Next is to identify the text what must be translated. In our case this is the text for the caption of the label and the button. And to set the standard descriptions in the "resourcestring" section. This section must be added to the sourcecode after the "implementation" keyword. The standard resourcestrings are simply defined as string contants.

```
...

var
  Form1: TForm1;

implementation
resourcestring
   label1_text='Testdescription';
   button1_text='Button_1';


{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
begin
  showmessage('Hello world!');
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
var PODirectory, Lang, FallbackLang: String;
begin
     PODirectory:='./languages/';
     GetLanguageIDs(Lang,FallbackLang); // in unit gettext
     TranslateUnitResourceStrings('unit1',PODirectory+'unit1.%s.po',Lang,FallbackLa
     showmessage(Lang+' | '+FallbackLang);
end;

procedure TForm1.FormShow(Sender: TObject);
begin
  label1.Caption:=label1_text;
  button1.Caption:=button1_text;;
end;

initialization
  {$I unit1.lrs}

end.
```

OnCreate of the form the language settings are read and the resourcestrings are translated according the laguage setting on the computer. At the OnShow event we assign the resource strings to the label and to the button. With the first compile the IDE generates an .PO file, in our case the unit1.po. This file is an ASCCI file and looks like this: We copy the file

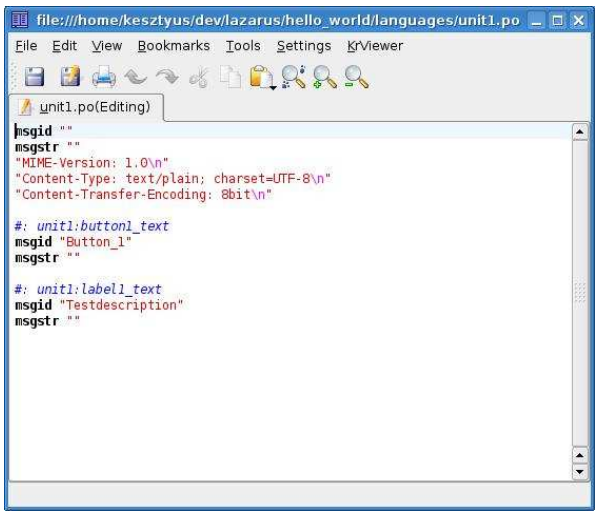

Figure 16.3: PO File Initial

to a “unit1.de.po” and to a “unit1.en.po”. In these two files we make the changes for the different languages. The language is defined by the “.de.” for german or the “.en.” for english. You must add a new file with the correct code for a new language. The files should look like this. For the English translation: And for the German translation: The result looks like

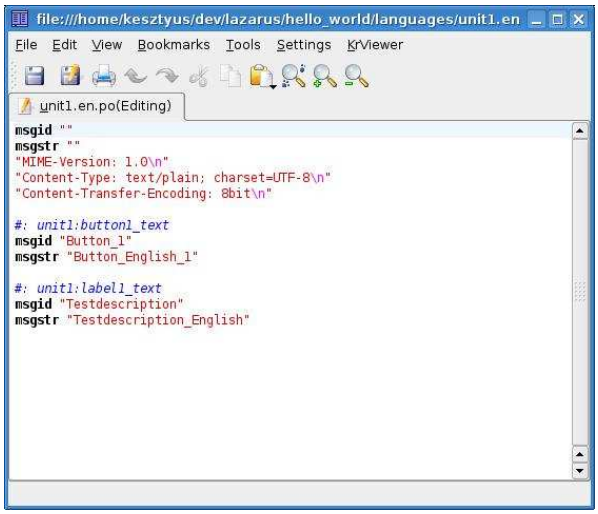

Figure 16.4: PO File English

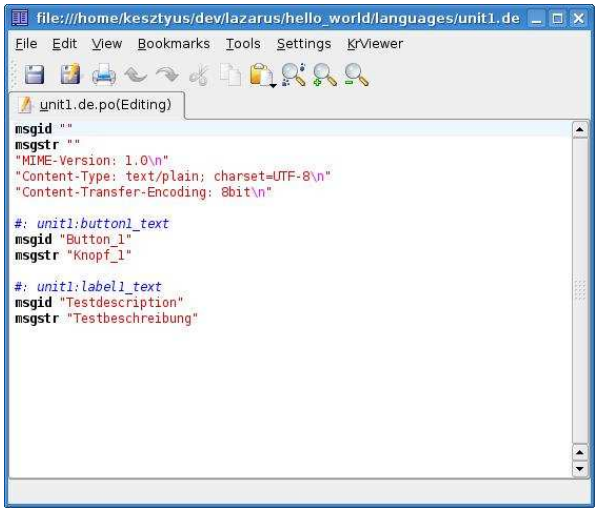

Figure 16.5: PO File German

this. The first screenshot is done on a machine with an English setting and the second with German settings.

Figure 16.6: Hello World English

Figure 16.7: Hello World German