Porting to Lazarus 2: Applications

Michaël Van Canneyt

June 24, 2007

Abstract

Since the last article, Lazarus has made big leaps of progress. On linux or BSD, it has become a stable development environment. On Windows, the IDE has become fully usable, and the most recent victory is the appearance of Lazarus on Mac OS X. In this second article on porting to Lazarus, 2 small Delphi applications will be ported to Lazarus: one that deals with graphics, the other one deals with Database access.

1 Introduction

In the previous article, porting of components written for Delphi to lazarus was discussed. In this article, the porting of programs will be discussed. Applications come in many flavours: from networking applications over graphics to Database applications. Or all combined. It require more pages than available in an issue of Toolbox to cover all ranges, so 2 small sample applications will be converted. Since not all application areas are supported equally well in Lazarus, some applications will port easier than others.

The first application is a small picture viewing application, designed to browse quickly through large collections of pictures: imagine a school, looking for a picture of one of its students on the CD-Rom, delivered by the photographer.

The second application is a small address databook. It saves all addresses in a single DBase file. The strength of this application lies not in it's usefullness, but in the fact that it shows that database applications can be ported (or built) and that components, written for Delphi, can be ported to Lazarus quite easily: the TDBF component is used.

2 Lazarus IDE tools

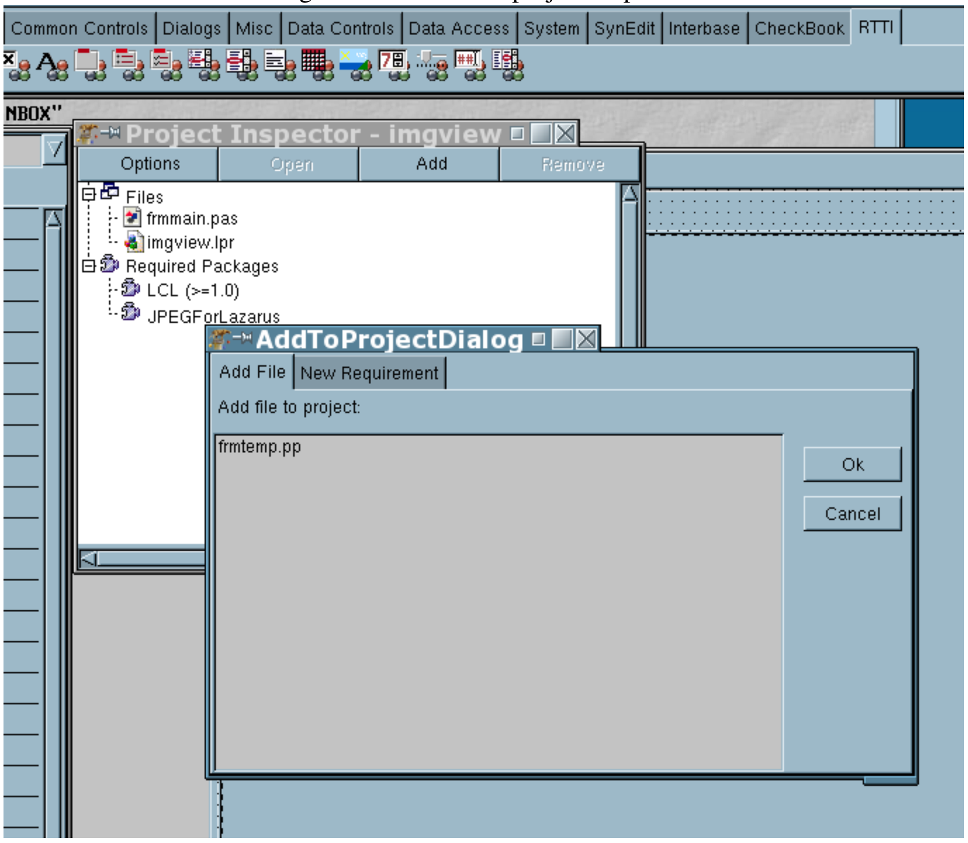
The lazarus IDE features 2 menu items, designed specially for the porting of Delphi programs or units to Lazarus. Both are located under the 'Tools' menu:

Convert DFM file to LFM This will take a Delphi .dfm form file and convert it to a lazarus (.lfm) file and the corresponding .lrs resource file. The unit file is not touched.

Convert Delphi unit to Lazarus unit This does the same as the previous point, but additionally changes the unit file: it includes the resource file in the initialization section of the unit file, and removes some units in the unit clause, while adding some others (lresources, among others).

The form must be in text format, not in binary format. To save the file in text format (in Delphi 5 and higher), it is sufficient to use the popup menu in the Delphi form designer, and check the 'Text dfm' menu item, and then saving the file.

Figure 1: The lazarus project inspector



Lazarus will remove unknown properties from the form file when loading it. It will display an error message, saying that unknown properties are encountered, but these can be dismissed with the 'Ignore' button.

3 Starting a port

To start a port, the best thing is to start a new project, and 'import' each unit in the new project. Lazarus does not use the Delphi project file as Delphi does: All project settings are not stored in the .lpr file (as delphi does in the .dpr file), but are stored in the project information (.lpi) file. Starting a new project makes sure that a .lpi file is created.

Any common units can be added in the 'Project inspector'. This dialog can be opened with the menuProject|Project Inspector menu: In it, project files and dependencies can be viewed and changed. To add a unit to the project, it must be opened in the source editor, and can then be added to the project using the 'Add' button in the project inspector, as shown in figure 1 on page 2. The dialog will show a list of editor files, one of which can be added to the project.

To add a Delphi form file to the new project, the following steps can be taken:

1. The file should be copied to the project directory.
2. The 'Convert Delphi unit to Lazarus unit' menu can be used to convert the unit.
3. Lazarus will open the unit in the source editor.

4. The ’Add’ button in the project inspector can be used to add the unit to the project.

After this, any compilation issues must of course still be dealt with.

4 A small graphical application

To demonstrate the above, a small graphical application will be ported from Delphi. It is a simple application, which allows to browse large collections of pictures in various directories: It maintains a list of pictures in a listbox at the left of the window, and the remainder of the window is used to show the currently selected picture.

There are some menu items to add files to the list: Single files can be selected, the contents of whole directories can be added, with the possibility to add the contents of subdirectories recursively. These options are available from the command-line as well.

Navigation can be done by selecting images from the filelist, but there are shortcuts to jump to the next or previous image, and to the next or previous directory in the list. Finally, the image size can be doubled or halved.

The sources are quite simple: A single form, containing a listbox for the filelist, a splitter to resize the filelist and an image component. A toolbar and menu complete the form. All toolbar buttons and menu items are connected to an Action: Most event handlers are connected to the actions.

The steps as outlined above were executed:

1. A new project was started: imgview and saved in a new directory. Lazarus added a new form file, it was saved too, and kept.
2. The main form file from the Delphi project was copied to the new project directory.
3. The ’Convert Delphi unit to Lazarus unit’ was used on the copied file. Lazarus opened the file in the editor. It was promptly saved.
4. The project inspector was used to add the new form to the project and to remove the form, created by lazarus, from the project.
5. Finally, in the project options, the newly added form was added to the list of forms which should be ’Auto-created’.

After this, compilation of the project was attempted. The first error came from the `Uses` clause in the implementation section:

```
uses filectrl,jpeg;
```

The jpeg unit is needed for JPEG support in Delphi. In Lazarus, this unit is called ’lazjpeg’ and resides in the standard delivered (but not installed) ’jpegforlazarus’ package. So the unit was added to the `uses` clause, and the ’jpegforlazarus’ package was added to the project dependencies in the project inspector: Doing this ensures that the lazjpeg unit can be found by the compiler: The lazarus IDE will automatically add the necessary directories to the compiler’s unit search path.

After this, a second compilation was attempted: This time, the compiler complained about the following form key handling method:

```
procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
```

```
begin
  if (shift=[ssShift]) or (shift=[ssAlt]) then
    begin
    if (key=VK_Prior) then
      begin
      RescaleImage(2.0);
      Key:=0;
      end
    else if (key=VK_Next) then
      begin
      RescaleImage(0.5);
      Key:=0;
      end
end;
```

The `VK_PRIOR` constant was unknown. The virtual key constants are in the LCLTypes unit, so this unit was added to the `Uses` clause. No dependency must be added, since this unit is part of the LCL (Lazarus Class Library) and all Lazarus projects automatically depend on it.

After this minor change, all compiled without a glitch. The application could be started and used - partially.

To get complete succes, there were only 2 quircks in this process which had to be fixed:

1. The images in the imagelist were correctly converted, but after the project was saved and re-opened in Lazarus, they were corrupted somehow. Reloading the images from their original bitmaps solved this problem.

2. The Listbox in Lazarus does not respond to the up and down arrow keys as in Delphi: the `OnClick` event handler is not triggered. The solution to this was to add the following code in to the `FormKeyDown` method:

   ```
   else if (shift=[]) then
     begin
     if Key=VK_UP then
       Previousimage
     else if Key=VK_DOWN then
       NextImage;
     end;
   ```

   As can be seen, this simply handles the up and down keys and calls the existing `PreviousImage` and `NextImage` methods.

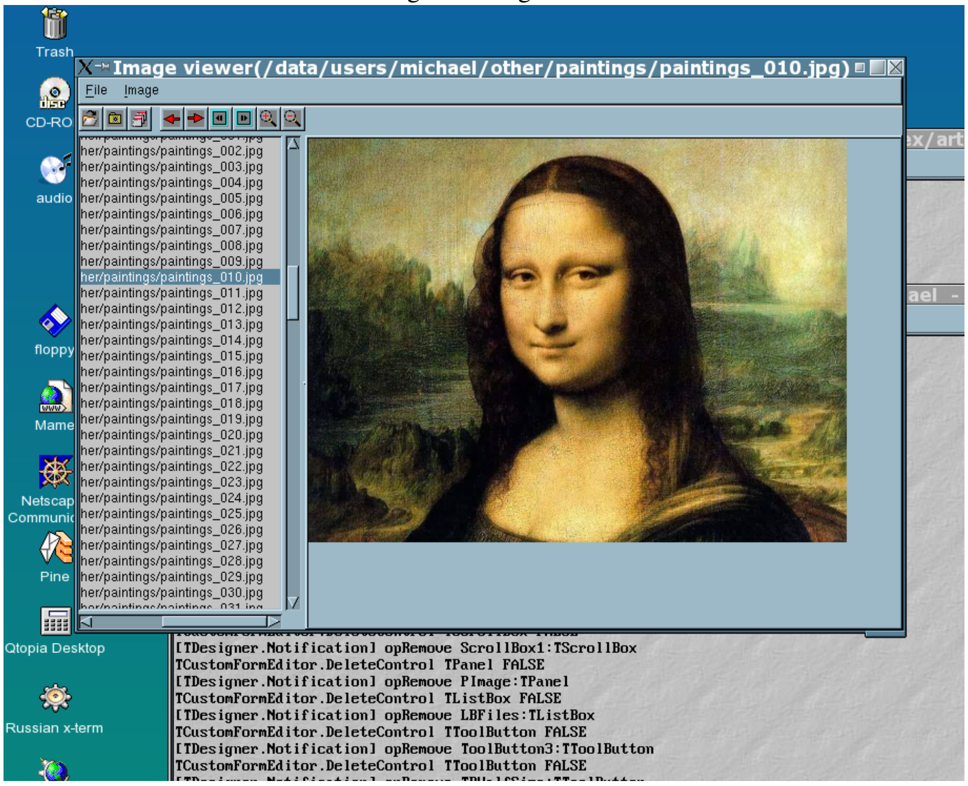
After this change, the application worked 100process took less than 20 minutes. The result can be seen in figure **??** on page **??**.

5 A small Database application: Addressbook

The graphical application was deceptively simple to convert. Sometimes a bit more work is needed. To demonstrate this, a small address book application was converted from Delphi to Lazarus. The application was developed specially for this purpose: it is not particularly useful in itself, it was made to make a point. The application allows to manage some DBase files which contain a list of addresses. It was kept quite simple, adding just some standard

Figure 2: The imgview program running on Linux

Figure 3: imgview



```
her/paintings/paintings_002.jpg
her/paintings/paintings_003.jpg
her/paintings/paintings_004.jpg
her/paintings/paintings_005.jpg
her/paintings/paintings_006.jpg
her/paintings/paintings_007.jpg
her/paintings/paintings_008.jpg
her/paintings/paintings_009.jpg
her/paintings/paintings_010.jpg
her/paintings/paintings_011.jpg
her/paintings/paintings_012.jpg
her/paintings/paintings_013.jpg
her/paintings/paintings_014.jpg
her/paintings/paintings_015.jpg
her/paintings/paintings_016.jpg
her/paintings/paintings_017.jpg
her/paintings/paintings_018.jpg
her/paintings/paintings_019.jpg
her/paintings/paintings_020.jpg
her/paintings/paintings_021.jpg
her/paintings/paintings_022.jpg
her/paintings/paintings_023.jpg
her/paintings/paintings_024.jpg
her/paintings/paintings_025.jpg
her/paintings/paintings_026.jpg
her/paintings/paintings_027.jpg
her/paintings/paintings_028.jpg
her/paintings/paintings_029.jpg
her/paintings/paintings_030.jpg
```

```
[TDesigner.Notification] opRemove ScrollBox1:TScrollBox
TCustomFormEditor.DeleteControl TPanel FALSE
[TDesigner.Notification] opRemove PImage:TPanel
TCustomFormEditor.DeleteControl TListBox FALSE
[TDesigner.Notification] opRemove LBFiles:TListBox
TCustomFormEditor.DeleteControl TToolButton FALSE
[TDesigner.Notification] opRemove ToolButton3:TToolButton
TCustomFormEditor.DeleteControl TToolButton FALSE
```

`TDataset` actions, which provide an easy way to add shortcuts to the program. The use of persistent fields was avoided, as the `TDataset` implementation of Free Pascal does not yet support them.

The addressbook application uses the Delphi `TDBF` component to create and manage the DBase files. This component is written 100does not need any external libraries such as the BDE, and is freely available from SourceForge:

```
http://tdbf.sourceforge.net/
```

The component also works with Free Pascal, and is distributed in the FCL (Free Component Library) that ships with Free Pascal. In fact, the man who is currently maintaining TDBF (Micha Nelissen) is also active in the Lazarus team.

The TDBF component is not available by default in Lazarus. The filetdbf folder in the components folder of Lazarus contains the 'DBFLaz' package. This package must be installed in order to get the `TDBF` component on the component palette, under the 'Data Access' tab. To install it, the 'dbflaz.lpk' file must be opened in the Lazarus IDE. When the 'Install' button in the package editor is pressed, the IDE will propose to recompile itself. After a succesful compile, the IDE can be restarted, and the TDBF component will be on the component palette.

Converting the application proved to be not so easy as the graphical application. Lazarus does not know the standard `TDataset` actions (in unit DBActns) and this, together with some of the unknown properties crashed the Lazarus IDE when attempting to convert the Delphi unit to a Lazarus unit.

Therefore the Delphi form file was first edited by hand, and the following 'unknown' properties were removed:

TDBGrid.TitleFont

TColumn.Expanded

TForm.OldCreateOrder

TPanel.DesignSize

As each property is on a line by itself in the Delpi form file, this was an easy task: All lines with an unknown property were simply removed using a simple text editor.

After this, the class references to the following standard dataset actions were changed to a simple 'TAction':

TDataSetFirst

TDataSetPrior

TDataSetNext

TDataSetLast

TDataSetInsert

TDataSetDelete

TDataSetEdit

TDataSetPost

TDataSetCancel

TDataSetRefresh

After removing the unknown properties and renaming the classes, the form loads properly in the Lazarus IDE.

To be able to compile the program, some editing of the unit was necessary: The form declaration contained published fields of the form:

```
    AFirst: TDataSetFirst;
    APrior: TDataSetPrior;
    ANext: TDataSetNext;
    ALast: TDataSetLast;
```

These are the declarations of the various dataset actions. Since the actions are unknown in the LCL they must be replaced with standard actions. To do this, a small trick was used. The following code was inserted just before the declaration of the main form:

```
type
  TDataSetFirst = TAction;
  TDataSetPrior = TAction;
  TDataSetNext = TAction;
  TDataSetLast = TAction;
  TDataSetInsert = TAction;
  TDataSetDelete = TAction;
  TDataSetEdit = TAction;
  TDataSetPost = TAction;
  TDataSetCancel = TAction;
  TDataSetRefresh = TAction;
```

This satisfied the compiler, and made the form compile, and the program could be started. Unfortunately, the Actions don't work properly. This is logical, since the standard dataset actions do all the work themselves. So the 'OnUpdate' and 'OnExecute' actions must be implemented specifically.

There are 2 kinds of actions: actions for navigation, actions for editing. The actions which perform navigational tasks should be enabled when the Dataset to which they are connected, is active. To do this, the 'OnUpdate' event of these actions is connected to the following 'DataOpen' event:

```
procedure TMainForm.DataOpen(Sender: TObject);
begin
  With DBA do
    (Sender as Taction).Enabled:=Active;
end;
```

In this method, `DBA` is the `TDBF` component. The varOnUpdate handler of the `TDataSetFirst`, `TDataSetPrior`, `TDataSetNext`, `TDataSetLast` and TDataSetRefresh actions should be connected to this method. The `TDataSetInsert` action was also connected to this method: Inserting is possible as soon as the dataset is open.

The `TDatasetEdit` action should be enabled when the dataset is open, and there is data present. Therefore, itss `OnUpdate` handler is connected to the following method:

```
procedure TMainForm.HaveDataNotEmpty(Sender: TObject);
begin
  With DBA do
```

```
    (Sender as Taction).Enabled:=Active and Not (EOF and BOF);
end;
```

Finally, the `TDatasetPost` and `TDatasetCancel` methods should only be active when the dataset is active and in edit mode. This is checked in the `InEditMode` procedure:

```
procedure TMainForm.InEditMode(Sender: TObject);
begin
  With DBA do
    (Sender As Taction).EnAbled:=State in dsEditModes;
end;
```

Now that the actions have an `OnUpdate` handler, they still need an `OnExecute` handler. Here, a different approach is taken: Each of the actions gets a unique value for their `Tag` property, and they are all connected to the `DoDataAction`:

```
procedure TMainForm.DoDataAction(Sender: TObject);
begin
  Case (Sender as TAction).Tag of
    0 : DBA.First;
    1 : DBA.Prior;
    2 : DBA.Next;
    3 : DBA.Last;
    4 : DBA.Insert;
    5 : DBA.Delete;
    6 : DBA.Edit;
    7 : DBA.Post;
    8 : DBA.Cancel;
    9 : DBA.Refresh;
  end;
end;
```

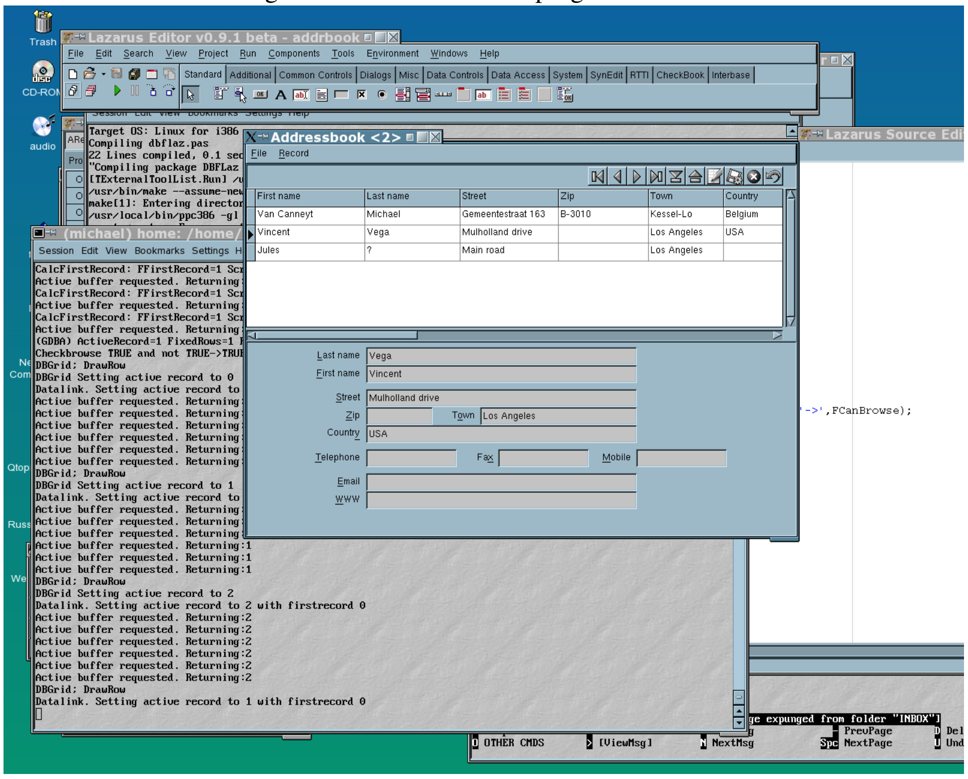
The value of the `Tag` property is used to decide what action to take.

After this change, the application compiles and runs as it was when compiled in Delphi. The result can be seen in figure 4 on page 9

6 Conclusion

Porting Delphi applications to Lazarus has definitely become an option for simple desktop applications. If the requirements are not too high or specialized, then chances are that porting to Lazarus is not too difficult. Of course, not all areas of application development have been covered in this article: network programming has not been treated yet, although there is a version of the Indy components which works with Free Pascal, and the Synapse component set works under FPC as well. Also more evolved database programming (like dbExpress or DataSnap) is not yet available. Web programming in Lazarus is possible, but not nearly at the level of Delphi. These issues will be solved in time: In a forthcoming contribution, Web programming will be discussed. Other areas where Lazarus has made some innovations will prove intesting subjects for contributions to Toolbox as well.

Figure 4: The addressbook program in action



```
Target OS: Linux for i386
Compiling dbflaz.pas
22 Lines compiled, 0.1 sec
"Compiling package DBFLaz
[TExternalToolList.Run] /u
/usr/bin/make --assume-new
make[1]: Entering director
/usr/local/bin/ppc386 -gl
```

```
CalcFirstRecord: FFirstRecord=1 Scr
Active buffer requested. Returning:
CalcFirstRecord: FFirstRecord=1 Scr
Active buffer requested. Returning:
CalcFirstRecord: FFirstRecord=1 Scr
Active buffer requested. Returning:
(GDBA) ActiveRecord=1 FixedRows=1 F
Checkbrowse TRUE and not TRUE->TRUE
DBGrid: DrawRow
DBGrid Setting active record to 0
Datalink. Setting active record to
Active buffer requested. Returning:
Active buffer requested. Returning:
Active buffer requested. Returning:
Active buffer requested. Returning:
Active buffer requested. Returning:
Active buffer requested. Returning:
DBGrid: DrawRow
DBGrid Setting active record to 1
Datalink. Setting active record to
Active buffer requested. Returning:
Active buffer requested. Returning:
Active buffer requested. Returning:
Active buffer requested. Returning:1
Active buffer requested. Returning:1
Active buffer requested. Returning:1
DBGrid: DrawRow
DBGrid Setting active record to 2
Datalink. Setting active record to 2 with firstrecord 0
Active buffer requested. Returning:2
Active buffer requested. Returning:2
Active buffer requested. Returning:2
Active buffer requested. Returning:2
Active buffer requested. Returning:2
Active buffer requested. Returning:2
DBGrid: DrawRow
Datalink. Setting active record to 1 with firstrecord 0
```

First name	Last name	Street	Zip	Town	Country
Van Canneyt	Michael	Gemeentestraat 163	B-3010	Kessel-Lo	Belgium
Vincent	Vega	Mulholland drive		Los Angeles	USA
Jules	?	Main road		Los Angeles	

Last name	Vega
First name	Vincent
Street	Mulholland drive
Zip	
Town	Los Angeles
Country	USA
Telephone	
Fax	
Mobile	
Email	
WWW	

```
'->',FCanBrowse);
```

```
ge expunged from folder "INBOX"]
                                   PrevPage        Del
                                   NextPage        Und
O OTHER CMDS   > [ViewMsg]   N NextMsg   Spc NextPage
```

Trash
CD-ROM
audio
Lazarus Editor v0.9.1 beta - addrbook
File Edit Search View Project Run Components Tools Environment Windows Help
Standard Additional Common Controls Dialogs Misc Data Controls Data Access System SynEdit RTTI CheckBook Interbase
Addressbook <2>
File Record
(michael) home: /home/
Session Edit View Bookmarks Settings
Lazarus Source Edi