# Introduction to Lazarus/FreePascal

### *Contents*


Installation and first launch ..... 1

Graphical Applications developing environment ..... 2

Instructions execution sequence ..... 4

Inputs and outputs ..... 4

Graphical Outputs ..... 6

Animated Graphical Outputs ..... 7

Comments ..... 9


### *Installation and first launch*

First of all you need to download Lazarus' installation files from this page: http://sourceforge.net/projects/lazarus/files/.

There are many packages to be downloaded because many operating systems and cpu architecture are supported, chose the most suitable for your system.

In Windows launch the self-installer and follow the procedure, usually there is no need to modify any parameter.
In Linux you can use your RPM or DEB package manager to install package. If Lazarus is available from your distribution repository installing it directly is probably the best choice.

Once launched, Lazarus is ready to edit your first project. But let's define what Lazarus' Project is.

Rarely the traditional Pascal program, where all the code is written in a single file with .pas extension, is used in Lazarus, and you can do nothing with a .pas file until you have not defined a Lazarus project.
The project is a collection of files needed by Lazarus to manage the multiplatform compilation and to integrate all library units eventually used.

There are many types of project, most interesting are "Pascal program" and "Application". The "Pascal program" project will manage the development of a single .pas file program, in the editor you'll find the file starting with a line like the following

```
Program program_name;
```

In the web there are many good guides, manuals, and tutorials to write a Pascal program so we are going to talk about the "application" project type.

Most important differences between a Pascal program and a FreePascal Graphical Application can be found in input and output methods and in the instruction execution sequence.

### *Graphical Applications developing environment*

Launching Lazarus we have the environment ready to develop a graphical application.

The IDE (Integrated Development Environment) appears as a set of windows whose function will be explained as follows.

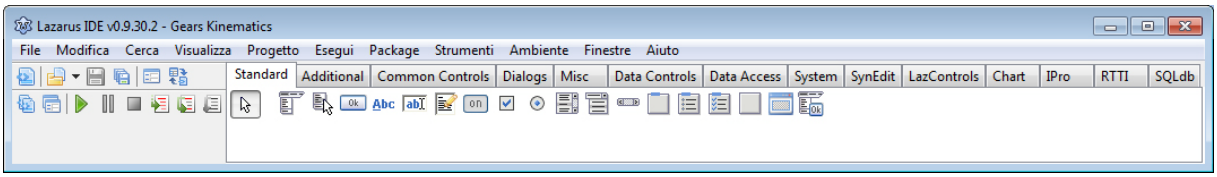


*Figure 1*

The main Window contains menus, start and stop buttons of debugger, the collection of objects to be used in the form composition and other common functions.

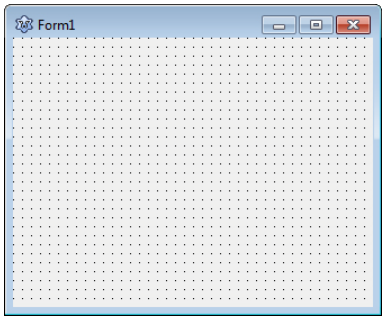


*Figure 2*

This one is the window to edit the final aspect of your application interface. Edit the interface by adding and modifying objects chosen from the main window's collection. You'll find it simple and fast.

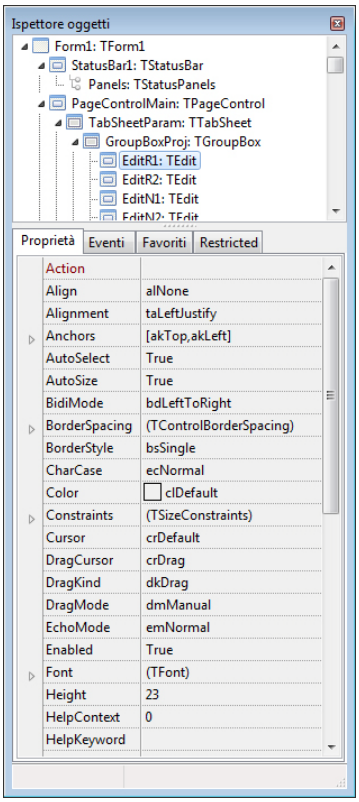


*Figure 3*

The "Object Inspector" window is divided in two parts. The upper part shows a list of objects used in the form, the lower part shows the list of available properties and events of selected object. You can select an object from the list in this window or in the form editing window by a single clicking on the object.

In the tab "Events" there is a list of available events for the selected object.

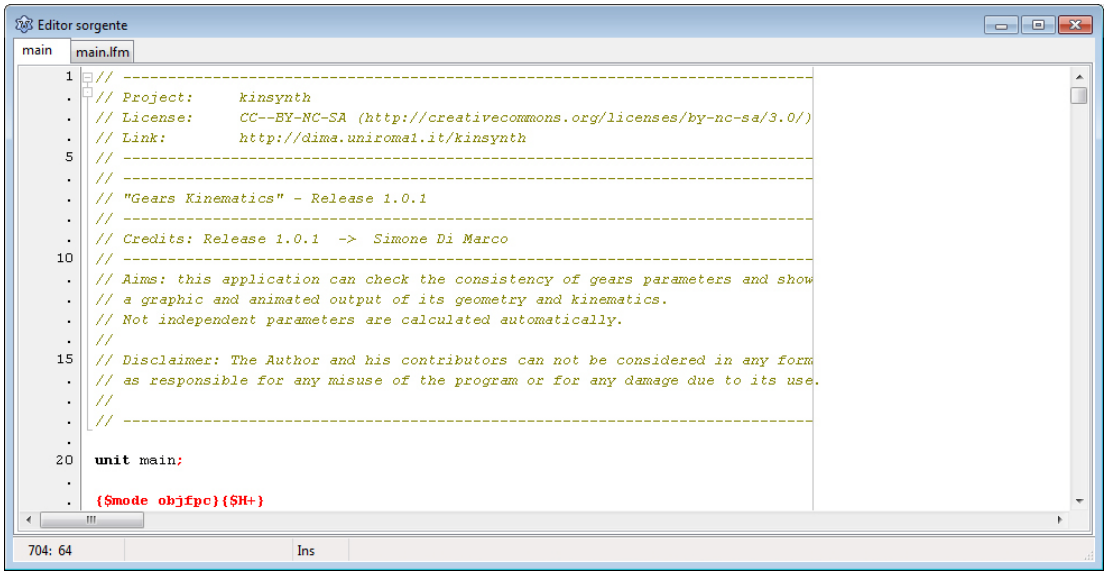


*Figure 4*

The code editing windows is nothing else than a simple text editor where can be edited each source file of your project.

Note that an application project consist in a minimum of 4 files with the following meanings:

- One file with .lpi extension that contains information about the project, such as the file list, the compiling options and all is needed by Lazarus to open the project. Opening a .lpi file Lazarus loads the whole related project.
- One file with .lpr extension, it is a .pas file with renamed extension and is the main Pascal program of the project, it starts with the following line

  ```
  Program application_name;
  ```

  The programmer doesn't need to open or modify this file that is automatically generated by Lazarus.
- One file with .pas extension that starts with the following line:

  ```
  Unit unit_name;
  ```

  Is a single file that contains all the procedures associated with events in the application form or defined by the programmer.
  Each instruction must be included in a procedure that will be called during the program execution.
  Later an example will clarify these concepts.
- One file with .lfm extension that contains all the information about the graphical interface structure. It's automatically generated during the graphical editing and there's no need to edit the code of this file.

### *Instructions execution sequence*

In a graphical application each instruction is given as a part of a procedure. During the application runtime the user or the operating system or other procedures are able to call a procedure in anytime. Procedures are executed in the same order they was called and each procedure can start only when the previous one has completed its execution. Note that a procedure that contains an infinite loop cannot be interrupted by an input because any input is a call for a following procedure.

### *Inputs and outputs*

A basic example of a possible input and output managing will introduce some characteristics of a Lazarus/FreePascal application.

We will create a simple application that reads two values from two edit fields and returns the result of a mathematical operation when a button is clicked.

At first let's create an interface with two edit fields (objects defined in the TEdit class), a text label (defined in the TLabel class) and a button (defined in the TButton class) like this

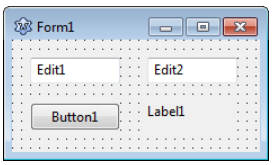


*Figure 5*

Edit1, Edit2, Label1 and Button1 are the names of variables automatically generated in the code when the graphical object has been added in the form.

Select the object "Button1", open the Event list tab in the Object Inspector window and select the OnClick event, an empty procedure will be automatically generated in the source code editor as shown in

Figure 6.

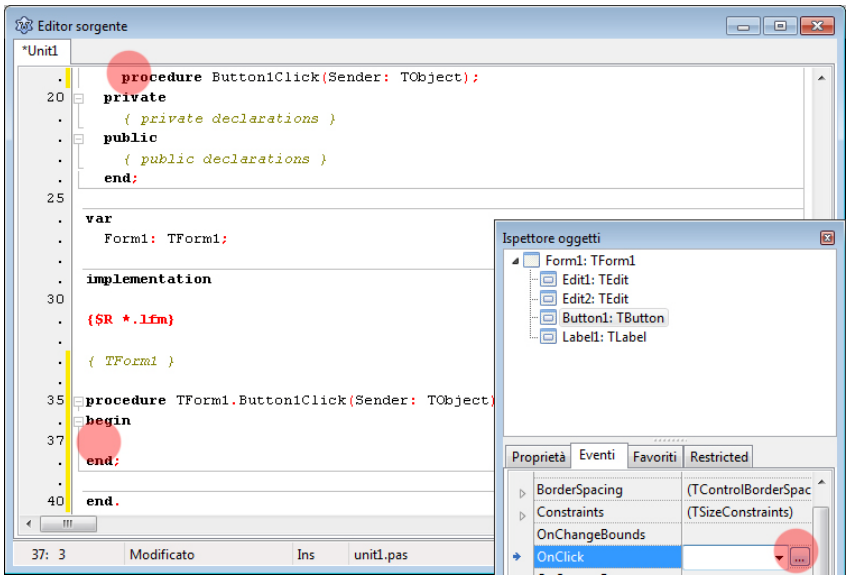


*Figure 6*

In the space highlighted with the bottom left red circle all the instructions can be defined to be executed when the button is clicked. For instance:

a) Input
b) Calculus
c) Output

Each object, that belongs to a class, is defined by a set o variables. These variables can be used in each procedure without previous declaration because have been already implicitly declared in the object declaration.
To access this variables is used the following syntax:

```
Edit1.Text
```

Where the first part identifies the object and the second part, separated by a dot, identifies the property variable, in the case of the example the variable contains the string shown in the edit box of the object called Edit1.

**To obtain an input** could be necessary a type conversion from string format to numerical format. The following instruction converts the string in the edit box Edit1 and assigns its value to a real type variable called "a".

```
a := StrToFloat(Edit1.Text);
```

**To obtain an output** from a real type variable the reverse conversion is needed, in the following instruction the string generated by the conversion is assigned to the Text variable of Edit1.

```
Edit1.Text := FloatToStr(a);
```

The following code shows a possible implementation of steps a) b) c) in the interface of Figure 5

```
procedure TForm1.Button1Click(Sender: TObject);
      var a,b,c: Real;
begin
//a):
      a := StrToFloat(Form1.Edit1.Text);
      b := StrToFloat(Form1.Edit2.Text);
//b):
      c := a*b;
//c):
      Form1.Label1.Caption := FloatToStr(c);
end;
```

### *Graphical Outputs*

Lazarus integrates many libraries with functions of any kind. Among these functions there are drawing functions provided by "Canvas" library. Canvas is already integrated in Lazarus but many other graphical libraries can be downloaded from the internet.

Drawing functions, provided by Canvas library, need a space to draw everything, the space can be defined with a TPaintbox object that can be positioned in the Form area. The PaintBox space determinates the origin of x and y axis that is positioned in its top left corner, x-axis is oriented as usually from left to right and *y-axis is oriented from top to bottom*.

The unit of this coordinates system is the pixel so coordinates are given in integer o longinteger formats.

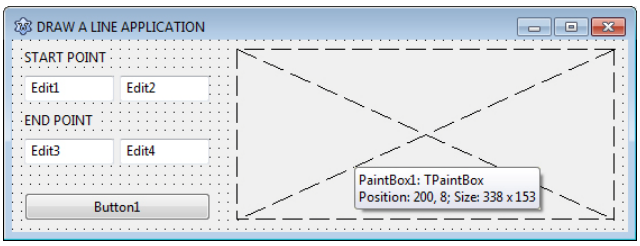


*Figure 7*

For example the following instruction draws a line from point $(x_1, y_1)$ to point $(x_2, y_2)$:

```
PaintBox1.Canvas.Line(x1,y1,x2,y2);
```

The first part of this line defines the space of drawing (PaintBox1), the second part refers to Canvas library, third part calls the drawing function (Line).

With reference to interface of Figure 7, if we assign the following procedure to OnClick event of the button, the output will be like the one shown in Figure 8.

```
procedure TForm1.Button1Click(Sender: TObject);
  var x1, y1, x2, y2 : longint;
begin
  x1 := round( StrToFloat(Edit1.Text) ) ;
  y1 := round( StrToFloat(Edit2.Text) ) ;
  x2 := round( StrToFloat(Edit3.Text) ) ;
  y2 := round( StrToFloat(Edit4.Text) ) ;
  PaintBox1.Canvas.Line( x1, y1, x2, y2);
end;
```

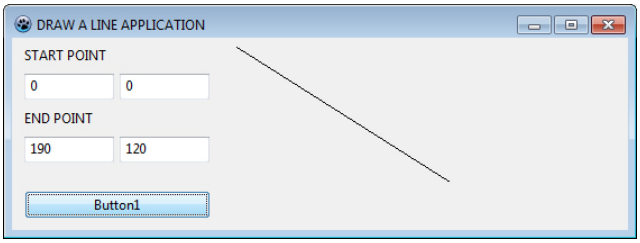


*Figure 8*

Note that non integer values must be rounded with an instruction like the following

```
Round(a)
```

### *Animated Graphical Outputs*

To animate a graphical output an auxiliary object called TTimer must be added in the form.

Among properties of a TTimer there is the boolean variable "Enabled". When the "Enabled" variable is set on "True" the TTimer generates an event each interval of presetted duration.

```
Form1.Timer1.Enabled := 'True'
```

To this event can be associated a procedure that draws a single frame of an animation and the instructions needed to modify the drawing configuration to obtain the animation.

As example is shown part of the code for the animation of "Gear Kinematics" application available on KinSynth.

```
//start/stop animation button procedure
procedure TForm1.ButtonStartStopClick(Sender: TObject);
begin
//...
    //instruction to start the timer procedure to draw on each interval
    Timer1.Enabled := True;
//...
end;

//animating procedure (Timer1 draws one bitmap frame per clock)
procedure TForm1.Timer1Timer(Sender: TObject);
  var r1, r2, n1, n2, tau, m, phi, omega_t: real;
begin
  //read parameters
  r1 := StrToFloat(EditR1.Text);
  r2 := StrToFloat(EditR2.Text);
  n1 := StrToFloat(EditN1.Text);
  n2 := StrToFloat(EditN2.Text);
  tau := StrToFloat(EditRatio.Text);
  m := StrToFloat(EditM.Text);
  theta := StrToFloat(EditTheta.Text);
  //draw
```

```
  omega_t := (FloatSpinEditSpeed.Value)*ButtonStartStop.Tag/100 ;
  DrawGears (r1, r2, n1, n2, tau, m, theta, omega_t);
  if abs(omega_t) >= 2*pi/n1 then
    ButtonStartStop.Tag := 0;
  ButtonStartStop.Tag := ButtonStartStop.Tag + 1 ;
end;
```

### *Comments*

*This document is a translation of part of my thesis work.*

*This document is not part of any official Lazarus/FreePascal documentation, it is just a collection of personal experiences gained during the thesis work with the support of online documentation, Lazarus' community support and Pascal programming schoolbooks.*

*This material is published with the hope of helping who wants to start programming with Lazarus/FreePascal or read code written in this language.*

*A special thank is for Professor Belfiore who encouraged me to apply my interests and my curiosity in this work.*

*Simone Di Marco*