

Rapid Application Development in Linux using Lazarus

By Michael Burton
Rimrock Software

<http://www.rimrocksoftware.com/>



March 2011

Table of Contents

Introduction.....	1
What is Lazarus?.....	1
Where to get Lazarus.....	1
Parts of Lazarus IDE.....	2
Object Inspector.....	3
Unit Files.....	3
Form Files.....	4
Component Palette.....	4
Creating a Text Editor Program.....	5
Naming/Caption.....	6
Add Editor Component.....	6
Saving units/project.....	8
Adding a Status Bar.....	8
Adding Non-visual Components.....	9
Adding Images.....	9
Creating a Main Menu.....	11
Adding the Main Menu.....	11
Connecting Program Code to Menu Items.....	12
Creating Toolbar.....	13
Adding The Toolbar.....	13
Hooking Up The Toolbar Buttons.....	13
Adding Final Menu Events.....	14
Adding text highlighter.....	15
Conclusion.....	15

Introduction

What is Lazarus?

The Lazarus Integrated Development Environment (IDE) is an open source implementation of Borland (now Embarcadero) Delphi IDE for Windows. Lazarus uses the open source Free Pascal compiler for all its compiling and linking chores. Lazarus is easy to use and the programs it creates are easy to deploy.

Free Pascal is an open source implementation of Borland's Object Oriented Pascal. Like Delphi, Lazarus implements an event-driven scheme to create programs. The languages and environments are close enough to one another that projects written in Delphi may be imported into Lazarus with minimal changes.

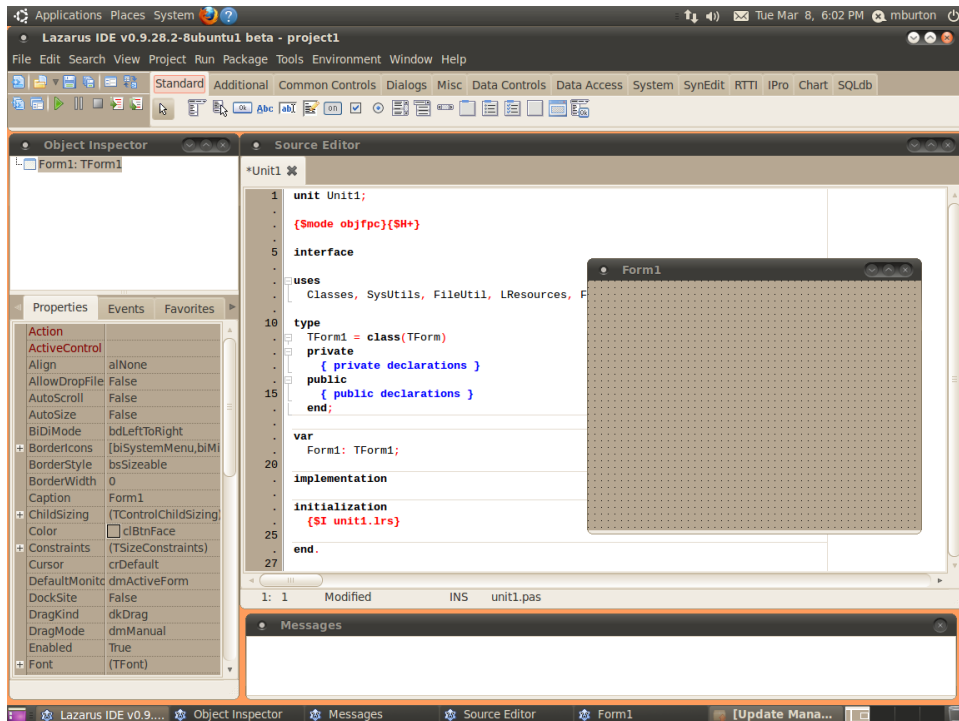
Object oriented Pascal is a high order language that looks somewhat like Java or C#. some of the operators are different from Java, but learning new operators is one of the things one always does when learning a new language. Pascal has the same basic programming constructs as most high level programming languages.

Where to get Lazarus

1. Lazarus can be obtained from the Ubuntu/Debian repositories. Search for Lazarus. Free Pascal will be included when Lazarus is selected and installed.
2. Lazarus and Free Pascal can be obtained directly from the web site <http://lazarus.freepascal.org/> Both Lazarus and Free Pascal must be downloaded. They are not easy to install, so we recommend that you use the repository if at all possible.

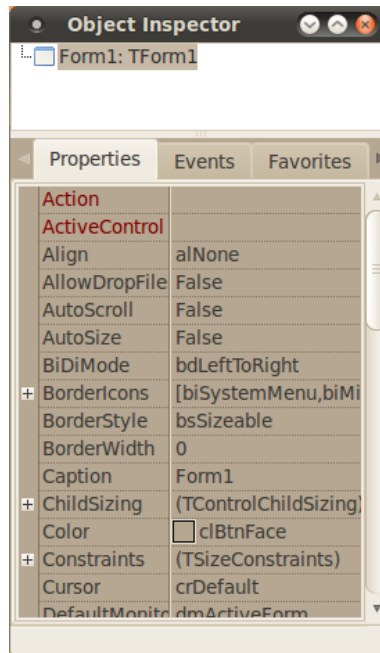
Parts of Lazarus IDE

- The Lazarus Integrated Development Environment is a multi-windowed environment that allows you to create and edit code, visually create graphical windows and compile and run your program.
- A Lazarus program consists of multiple files. They are all controlled using a file called the Lazarus project file (.lpr file extension). The project can contain one or more unit files (.pas) and zero or more form files (.lfm).



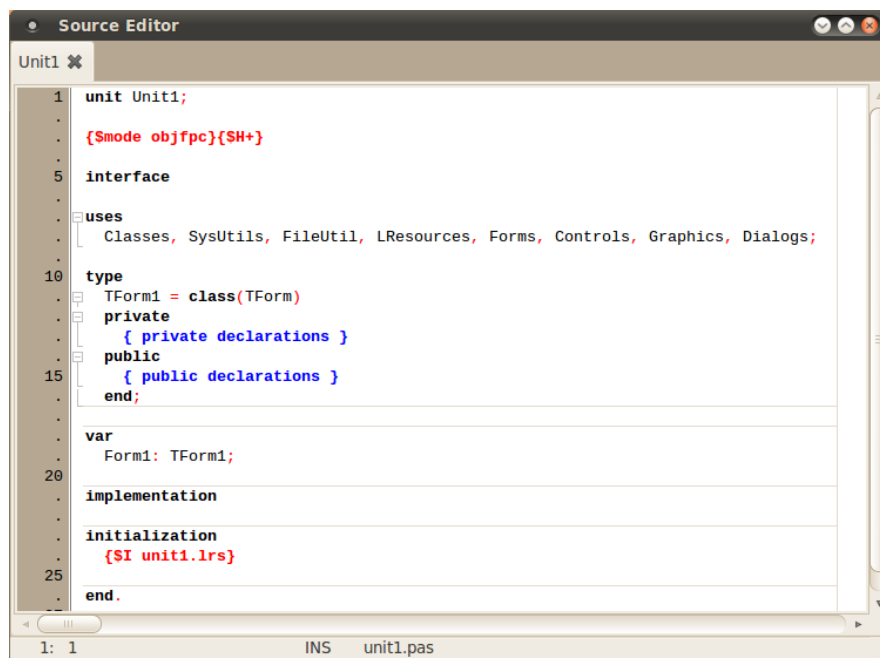
Object Inspector

- The Object Inspector is used to set property values for the components your program uses, and used for selecting/creating the event processing your program will use.



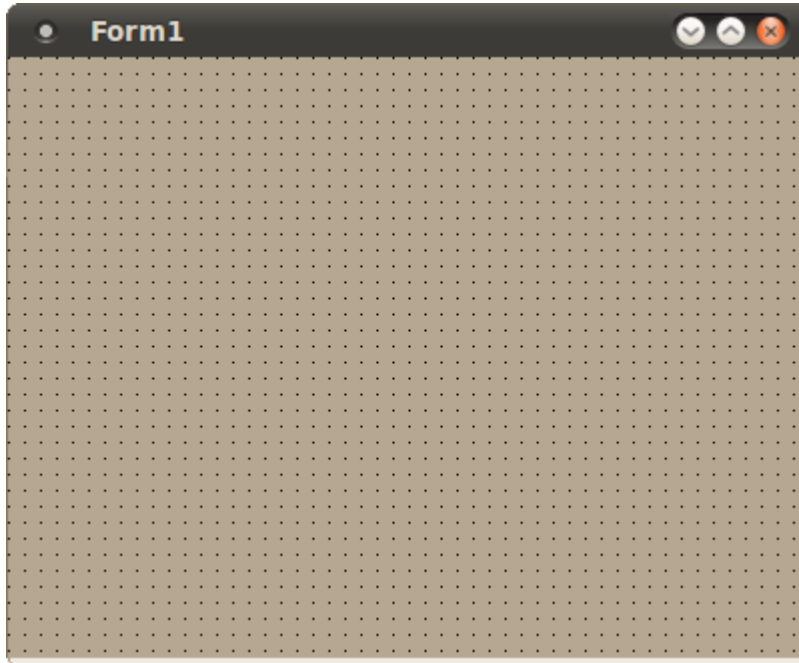
Unit Files

- Units are files that contain the object oriented Pascal program. When a new unit is created, Lazarus usually provides a code skeleton that can be filled in by the programmer or be Lazarus itself. A Lazarus unit skeleton is shown below.



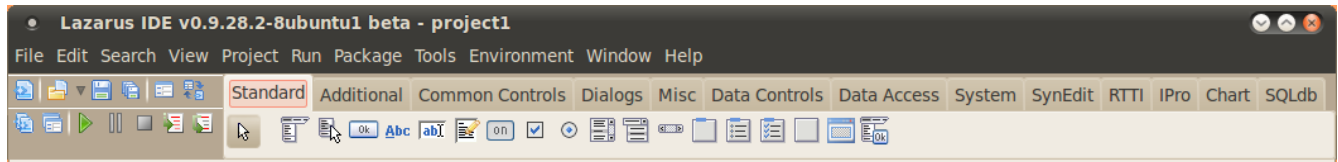
Form Files

- Forms are files that contain descriptions of program dialogs. A form is an instance of a Pascal class called TForm. Since a form is simply a textual description, a form can be displayed either as GUI or as text.



Component Palette

- Components are instances of Pascal class objects you add to your program to either display visual items or to implement non-visual items. Components can be chosen from the Component Palette



- The Message window displays information about the compilation of your program.



Creating a Text Editor Program

To demonstrate the power and ease with which we can create a program for Linux, we will create a text editor with menus, a toolbar and a status bar. The editing area will have line numbering and text highlighting. The nice thing about this editor is if it doesn't have a particular operation that you wish it did have, you can easily add the operation.

To begin creating the editor, the first thing you must do is to create a directory where you can save the program files. We have called the directory 'my_editor'.

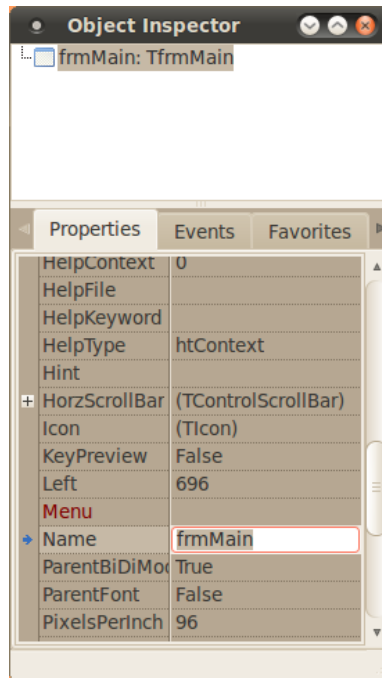
Next we must start up the Lazarus IDE. If an old project is displayed in Lazarus, close the old project, then select a New project - Application from the Project menu.

Naming/Caption

When Lazarus adds a Pascal object to a project, it gives the object a name. That name is unique and is usually not descriptive enough. That is fine for very small projects, but most projects will have many objects and you need to name them so you always know what they are, just from the name.

The new project we have just opened has a form whose name is Form1. We will rename that so we know what it really is. An object name should tell you two things: what kind of object it is, and what it is used for.

1. Click on the Form1 window. The Object Inspector will display the information about Form1.
2. Scroll down the Object Inspector to the Caption item and change it to be MyEditor. This is what we will call the editor and the caption will appear on the top of the editor window.
3. Scroll down to the Name item. Change it from Form1 to frmMain. This name indicates that this is a form and it is the main project form.

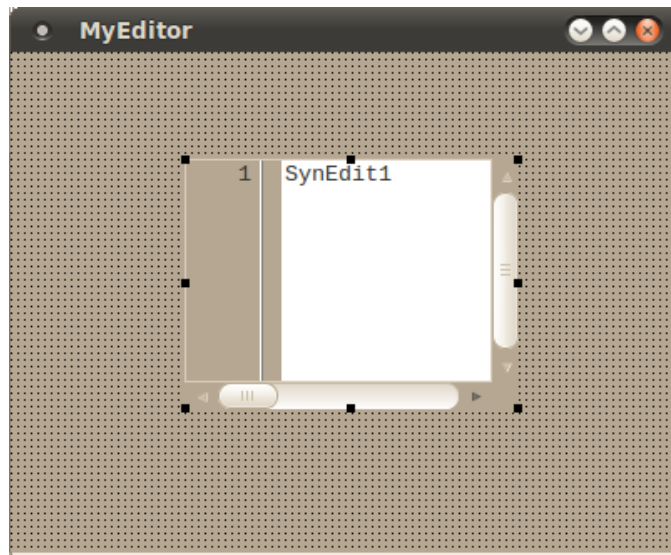


While we are in the Object Inspector for the form, we can change the default form size. Set Constraints | MinWidth to 600 and Constraints | MinHeight to 400.

Add Editor Component

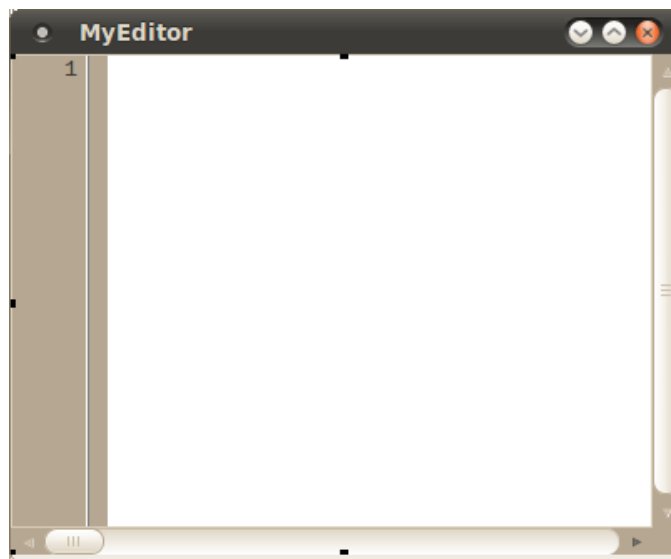
We need to add a control to the project that will hold the text and allow a user to edit the text. In Windows, we might choose one called RichEdit. We don't have that choice in Linux, so we will use a much more capable control called SynEdit.

Select the SynEdit tab in the Components bar. If you hover over the very first item in that tab, you will see it is called TsynEdit. Click on that control, Then move down to the form and click in the middle of the form.



To rename the control, click on it in the form and scroll down to the Name item in the Object Inspector. Change the name from SynEdit1 to sedtText. To get rid of the text inside the control, click on the Lines item, then on the button with the three dots. Remove the text and click OK.

Finally, to align the editing control so it will always be at its maximum size, change the Align item to be alClient.



Saving units/project

We should save our work now. To do that, you must be aware of a couple of things:

1. The project name will end up being the program name.
2. The name you save the unit as should not be the same as the we gave our form.

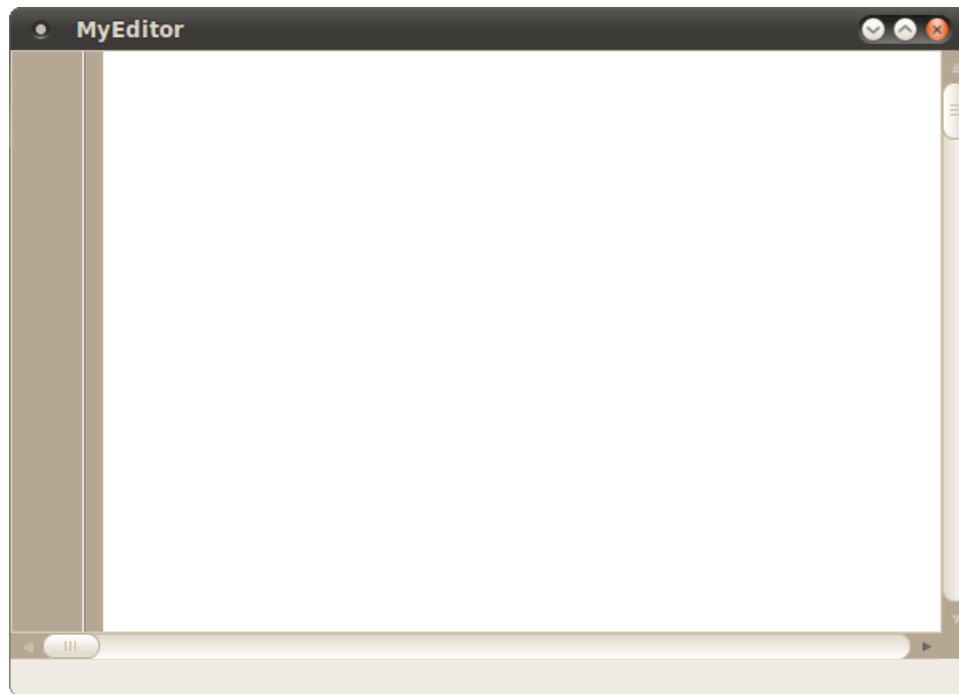
Click on the File | Save All item in the Lazarus menu. When asked for a unit name, change the name to dlgMain. Be sure to change to your project directory before clicking on the Save button.

When asked for a Project name, enter MyEditor, then click on Save. Save the main form as formEdit. Be sure to save your work as you add more components and make changes to the project.

Adding a Status Bar

Text editors normally have an area at the bottom of the screen that reflects the status of the editing process. We will put one at the bottom of the window that will display the name of the file we are working on. We can easily add more information to this area if we wish.

Click on Common Controls in the Component Palette and find the component with the help hint that says TStatusBar. Click on it, then click on the form. The status bar will be added to the bottom of the window. Rename it from StatusBar1 to sbarStatus.



We need to add a single line of code to support the status bar. It defines a global string where we can store the name of the file we are working on. At the the top of the dlgMain unit, in the public section, enter

```
sFileName : String;
```

Adding Non-visual Components

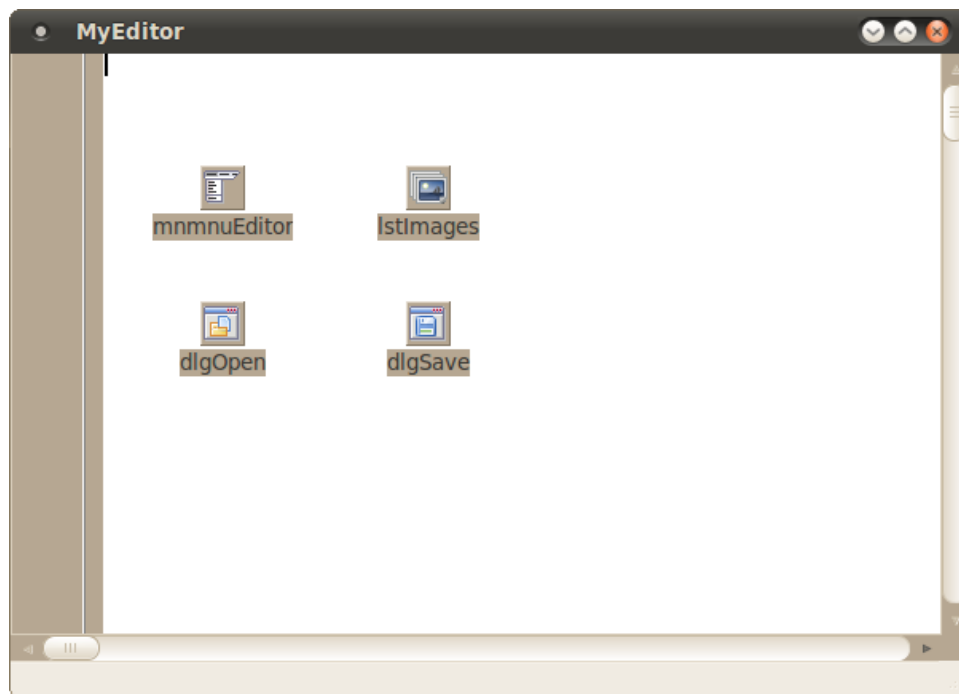
To help support some of the text editor' functionality, we need to add some support components. These components are not visual, but using the components will ultimately result in something visual being displayed. We need to add four components.

The TMainMenu component will allow us to create the main menu. To add it, click on the Standard tab of the Component Palette, click on the first component and then click on the form. A small icon with the label 'MainMenu1' will be added to the form. You can position this icon anywhere. When you execute the program, the icon will be hidden. Change the name item for this component to mnmnuEditor.

The TImageList component will allow use to add small images that we can use with the main menu and with the toolbar. Click on the Common Controls tab of the Component Palette Click on the next to last component, then click on the form. A small icon with the label 'ImageList1' will be added to the form. Change the name item for this component to lstImages.

The TOpenDialog component will allow a standard open file dialog to be displayed when we need to load a file. Click on the Dialogs tab of the Component Palette Change the name item for this component to dlgOpen.

The TSaveDialog component will allow a standard save file dialog to be displayed when we need to save our editing to a file. The component is right next to the one we just added for the open file dialog. Click on it, then click on the form. Change the name item for this component to dlgSave.



Adding Images

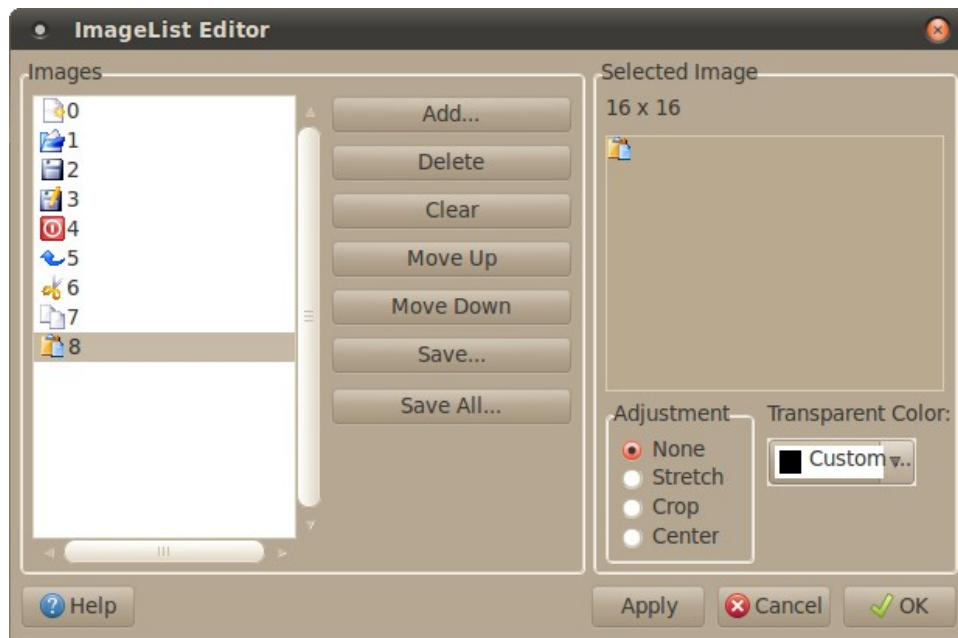
To add images to the image list component, right-click on the image list component on your form, then click on the ImageList Editor entry of the pop-up menu. Use the editor to add images to the image list component. The images we are using is on an Ubuntu system in /usr/share/icons/. Use one of the sets of icons there in the 16x16 subdirectory. We are using the icons in the default.kde directory. This set is

installed when you install Quanta Plus, the programmer's editor.

Before we pick up the images, we need to know what commands we will be implementing.

Command	Category	Uses Image
New	File	Yes
Open	File	Yes
Save	File	Yes
Save As	File	Yes
Exit	File	Yes
Undo	Edit	Yes
Cut	Edit	Yes
Copy	Edit	Yes
Paste	Edit	Yes
Line Numbers	View	No
Help Hints	View	No
Toolbar	View	No

Now we can go to the icons and pick up the ones we need. To make things easier, the images should be in the same order in the image list as they are in the above list.



Creating a Main Menu

Adding the Main Menu

In order for the editor to perform useful actions, it needs a menu. The same commands will be used in both the menu and the toolbar. You can centralize the code by using an action list, but we will do each command separately.

First let's hook up the image list to the main menu. Click on the `mnmnuEditor` component and then click its `Images` item. Pull down the list and select `lstImages`.

Here are the commands we will be adding to the menu. Note that this is only a starting point – more commands can be added at any time.

Command	Name	Toolbar	ImageIndex	ShortCut	Description
&File	N/A	N/A	-1	N/A	N/A
&New	<code>mnuNew</code>	X	0	<code>Ctrl+N</code>	New file
&Open	<code>mnuOpen</code>	X	1	<code>Ctrl+O</code>	Open file
&Save	<code>mnuSave</code>	X	2	<code>Ctrl+S</code>	Save file
Save &As	<code>mnuSaveAs</code>		3	<code>Shift+Ctrl+S</code>	Save file as
-	N/A	N/A	-1	N/A	N/A
E&xit	<code>mnuExit</code>	X	4	<code>Alt+F4</code>	Exit from editor
&Edit	N/A	N/A	-1	N/A	N/A
&Undo	<code>mnuUndo</code>	X	5	<code>Ctrl+Z</code>	Undo edit
-	N/A	N/A	-1	N/A	N/A
Cu&t	<code>mnuCut</code>	X	6	<code>Ctrl+X</code>	Cut to clipboard
&Copy	<code>mnuCopy</code>	X	7	<code>Ctrl+C</code>	Copy to clipboard
&Paste	<code>mnuPaste</code>	X	8	<code>Ctrl+V</code>	Paste from clipboard
&View	N/A	N/A	-1	N/A	N/A
&Line Numbers	<code>mnuLine</code>	N/A	-1	N/A	Toggle line numbering
&Help Hints	<code>mnuHints</code>	N/A	-1	N/A	Toggle help hints
&Toolbar	<code>mnuToolbar</code>	N/A	-1	N/A	Toggle Toolbar

The ampersand in each command causes an underline to be drawn under the next letter. The underlined letter is the hot key for that command, which means if you are running the editor and hit `Alt+F`, the File menu will be activated. Note also that some of the commands have no new names. Those commands are place markers in the menu.

Right-click on the `mnmnuEditor` Component and click on `Edit`. This will bring up the Menu Editor. It already has a new item which we will rename. Click on the menu item, then on the `Caption` item in the Object Inspector. Change the caption to `&File`.

Now we will add the File items. Right-click on the File item in the Menu Editor, then click on Create Submenu. A pull-down is created with one item. Right-click on the new item and click Insert New Item (after). Keep doing this until we have six items in the pull-down.

Now go back and select each of the menu items. Change the properties for each file menu item to agree with what we have in the chart above.

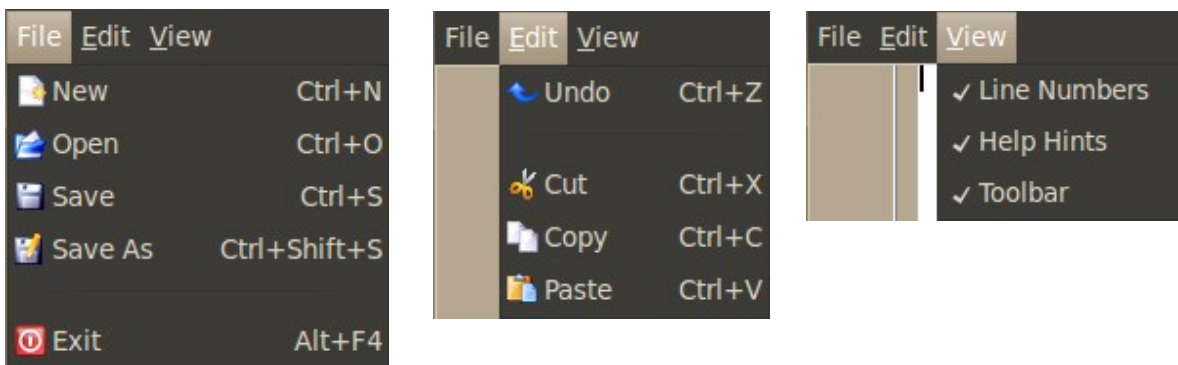
Next we need to add the Edit menu. Right-click on the &File item and click on Insert New Item (after). This will be the start of the Edit menu. Right-click on this new item in the Menu Editor, then click on Create Submenu. Add five new items to this submenu.

Fill out these Edit items the same way we did the File items.

Finally, we will add the View menu. Right-click on the &Edit item and click on Insert New Item (after). This will be the start of the View menu. Right-click on this new item in the Menu Editor, then click on Create Submenu. Add three new items to this submenu.

The View menu items are hooked up the same way as the other items, except they are toggles. This means that the Checked property for each menu item should be set to True (The three items all default to shown).

Close the menu editor. You should now have a program menu. Be sure to save your work.



Connecting Program Code to Menu Items

We mentioned earlier in this tutorial that Lazarus code is event-driven. To connect the menu commands so they actually perform actions, we need to create some of those events. Specifically, we will create an OnClick event for each of the commands.

Menu Item	OnClick Code
mnuNew	<pre>sFileName := ''; sedtText.Lines.Clear; sbarStatus.SimpleText := '';</pre>
mnuOpen	<pre>if dlgOpen.Execute then begin sFileName := dlgOpen.FileName; sedtText.Lines.LoadFromFile(sFileName); sbarStatus.SimpleText := ExtractFileName(sFileName); end;</pre>

mnuSaveAs	<pre> if dlgSave.Execute then begin sFileName := dlgSave.FileName; sedtText.Lines.SaveToFile(sFileName); sbarStatus.SimpleText := ExtractFileName(sFileName); end; </pre>
mnuSave	<pre> if Length(sFileName) = 0 then mnuSaveAsClick(Sender) else sedtText.Lines.SaveToFile(sFileName); </pre>
mnuExit	Close;
mnuUndo	sedtText.Undo;
mnuCut	sedtText.CutToClipboard;
mnuCopy	sedtText.CopyToClipboard;
mnuPaste	sedtText.PasteFromClipboard;
mnuLine	<pre> mnuLine.Checked := not mnuLine.Checked; sedtText.Gutter.Visible := mnuLine.Checked; </pre>

The final two commands, Help Hints and Toolbar, apply only to the toolbar, which we haven't created yet. We will do the code for them after we create the toolbar.

Creating Toolbar

Adding The Toolbar

A toolbar will provide the user with a quick method to perform some useful actions. To add a toolbar click on the Component tab called Common Controls. The toolbar component is in the middle. If you hover over it, the hint will say TToolBar. Click on it, then click on the form. The toolbar will be added to the top of the form. Click on the Images item in the Object Inspector and set it to lstImages. Set the Indent property to 4 and set the ShowHints property to True. Change the toolbar name to tbarMain.

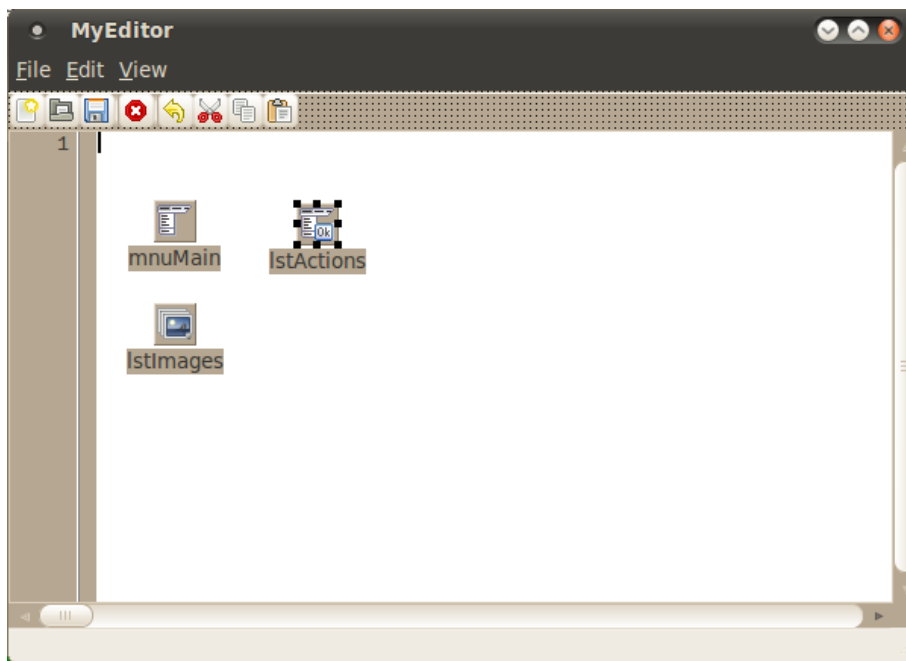
Next, we will add buttons and dividers to the toolbar. Right-click on the toolbar and click on New Button. Do this two more times to add three total buttons for New, Open and Save.

Now add a divider. Right-click on the toolbar and click on New Divider. Then add another button (Exit) and another divider. Finally, add four more buttons.

Hooking Up The Toolbar Buttons

Now that we have the buttons on the toolbar, we need to rename them and hook them up to the proper events. Here is a list of the buttons with the properties that need changing and the OnClick event to be used.

Button	Hint	ImgIdx	Name	OnClick Event
1	New file	0	tbtnNew	mnuNewClick
2	Open a file	1	tbtnOpen	mnuOpenClick
3	Save file	2	tbtnSave	mnuSaveClick
4	Exit from editor	4	tbtnExit	mnuExitClick
5	Undo edit	5	tbtnUndo	mnuUndoClick
6	Cut to clipboard	6	tbtnCut	mnuCutClick
7	Copy to clipboard	7	tbtnCopy	mnuCopyClick
8	Paste from clipboard	8	tbtnPaste	mnuPasteClick



Adding Final Menu Events

Now that we have a toolbar, we can now hook up the final two menu items. The Help Hints menu item will turn hints on and off, the Toolbar menu item will make the toolbar visible or invisible.

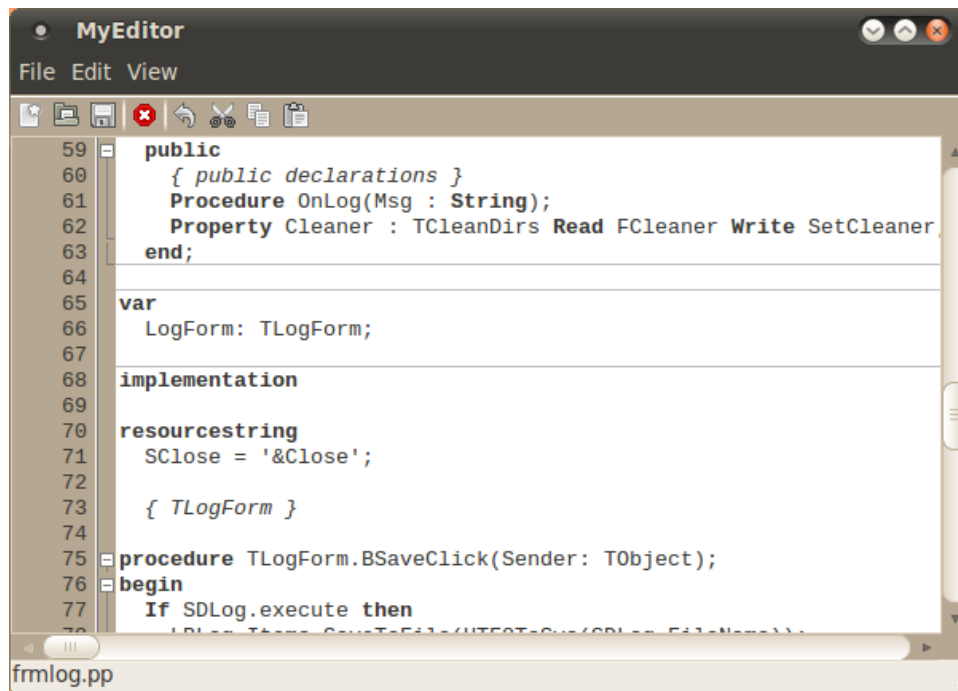
Menu Item	OnClick Code
mnuHints	<code>mnuHints.Checked := not mnuHints.Checked; tbarMain.ShowHint := mnuHints.Checked;</code>
mnuToolbar	<code>mnuToolbar.Checked := not mnuToolbar.Checked; tbarMain.Visible := mnuToolbar.Checked;</code>

Adding text highlighter

There are lots more items we can add to the text editor, but the final item we will add is a Pascal text highlighter. Click on the SynEdit tab of the Lazarus Components, then click on the TSynFreePascalSyn component and click on the form. Then click on the sedtText component and change it Highlighter item to SynFreePascalSyn1. Note that you could also add all the various highlighters and change the Highlighter item in the Open or Save As menu item, based on the file extension.

Conclusion

We now have a working text editor.



There are all kinds of things we can add to it. Here is a partial list of the kinds of things we can add to it.

1. Add an About box.
2. Add a help file.
3. Add a check for an edited file. If edited, ask to save before creating a new file.
4. Ask to save edited file before leaving the editor.
5. Add all the TSynHighLighter components. Hook up the proper highlighter based on file extension when doing an Open or Save As operation.
6. Add row and column information to the status bar.
7. Add caps lock and num lock to the status bar.
8. Add insert/overwrite mode to the status bar.
9. Add text search capability.

10. Add macro capability.
11. Add bookmarks capability.
12. Add code fold/unfold toggle.
13. Add editor property storage to restore it to its last size and position upon re-execution.
14. Add go to line command.