Beginners' Guide to Lazarus IDE

by Adnan Shameem

Conquering Lazarus in bite-size chunks!

Your first guide to reach the planet of Lazarus Users

Beginners' Guide to Lazarus IDE
Conquering Lazarus in bite-size chunks!
An E-book on Lazarus IDE to help newbies start quickly
by Adnan Shameem
from LazPlanet

First Edition: 15 December 2017

Source files for this E-book can be found at:
https://github.com/adnan360/lazarus-beginners-guide
Fork and remix. Have fun!

Contents

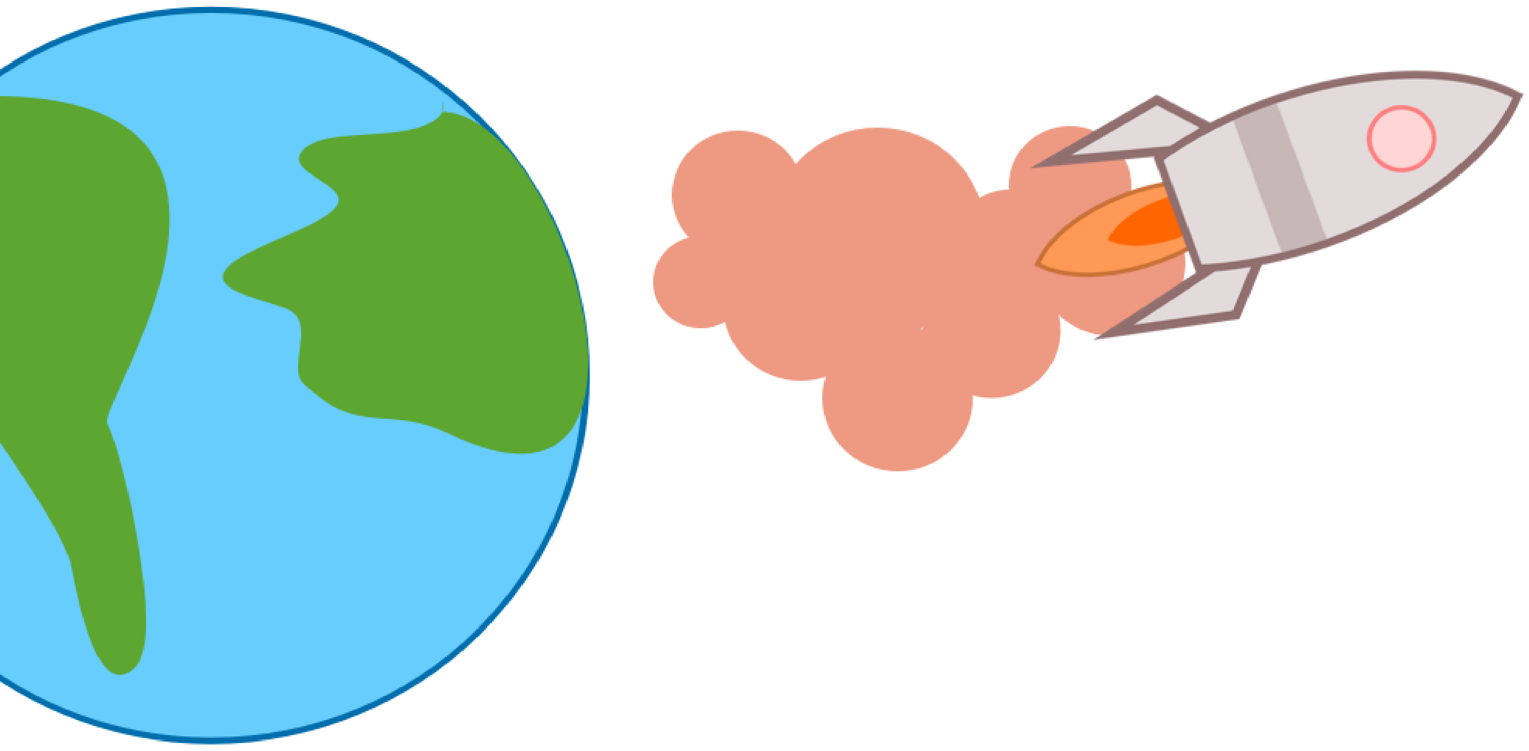
1: What is Lazarus

Lazarus is a Cross Platform IDE which lets you easily create programs you want. It has an easy-to-use interface and works out of the box so that you can start right away!

IDE means Integrated Development Environment – which means it is an interface to create software fast. So fast that it is called a **RAD** or Rapid Application Development IDE.

Cross Platform means you can run the IDE in any Operating System you use, Windows, Linux, Mac, BSD, Raspberry Pi – you name it! Your programs will run there too!

Let's not spend more words on this. Let's jump right in...!



2: How to Install

Installing Lazarus is easy and is similar to installing any other software. Just make sure your platform is supported and off you go!

Any system you use – Windows, Linux, Mac or a Raspberry Pi – chances are, you would be able to use Lazarus. Let's take a minute and install away...!

On Windows

Just download the exe file, then double click it and follow the on screen instructions.

Download the file from here: http://www.lazarus-ide.org/index.php?page=downloads

On Ubuntu

Open a terminal (Ctrl+Alt+T), then run these commands:

```
sudo aptitude install lazarus
sudo aptitude install fpc fpc-source lcl
```

On Arch Linux

Normally this should do it:

```
sudo pacman -S lazarus lazarus-gtk2 gdb
```

If you are using KDE or lxqt, go ahead and run this instead:

```
sudo pacman -S lazarus lazarus-qt gdb
```

On Fedora / RedHat / Cent OS etc.

You can go ahead and run:

```
sudo yum install lazarus gtk2-devel
```

On a Raspberry Pi (Raspbian)

See here for details: http://wiki.freepascal.org/Lazarus_on_Raspberry_Pi

On a Mac

See here for details: http://wiki.freepascal.org/Installing_Lazarus_on_MacOS_X

On other Operating Systems

Follow the instructions described here for your OS:
http://wiki.freepascal.org/Installing_Lazarus#Installing_Lazarus

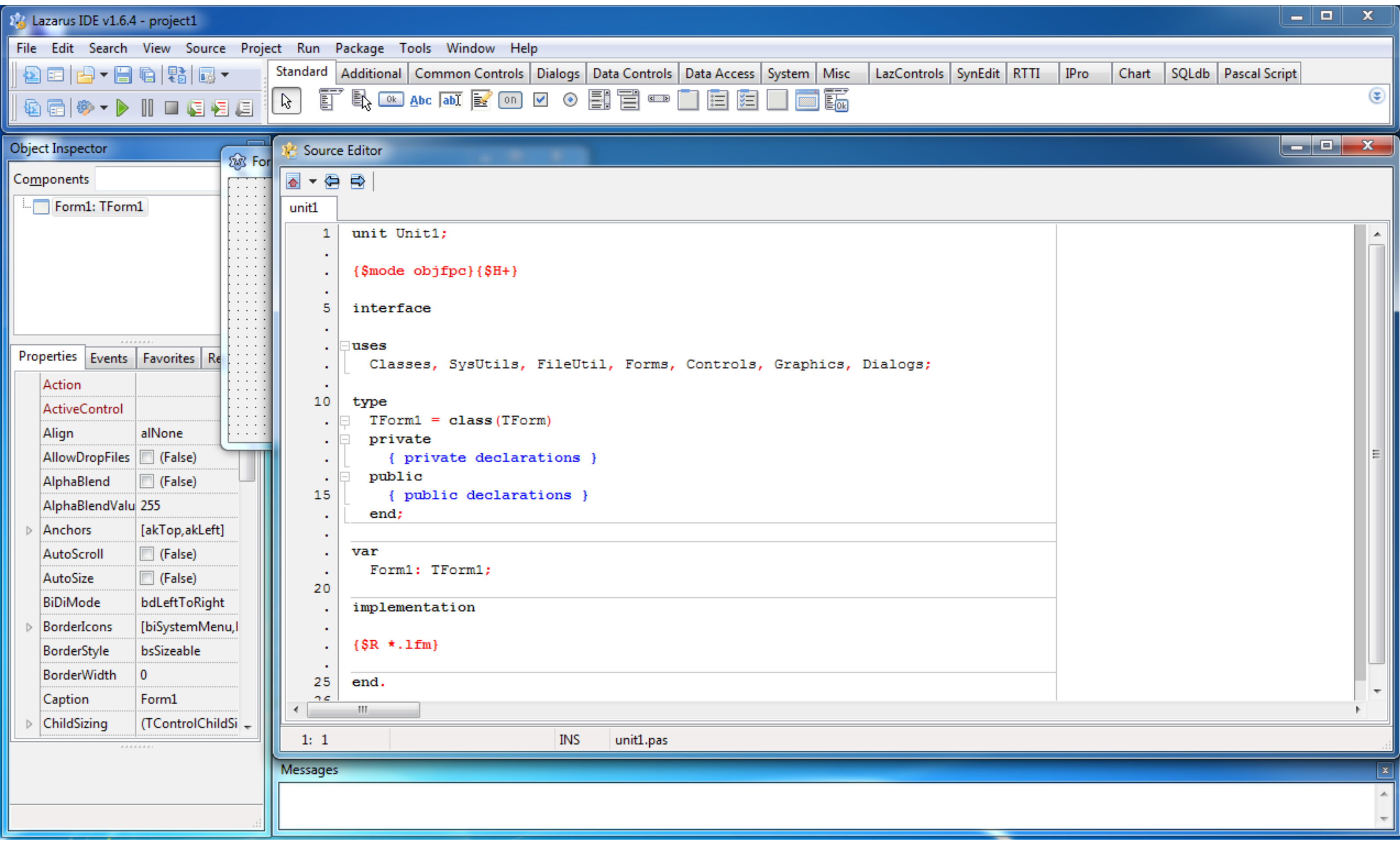
3: How to Create Your First Program

Creating with Lazarus is very easy. With just some quick clicks and a bit of typing you can achieve what others take hours and even days.

Start Lazarus – just like any other program.

If this is your first time, you should see a dialog titled **Welcome to Lazarus IDE 1.6.4**. Click **Start IDE**.

It should start the Lazarus IDE. Now move and resize the windows a little bit and it would look something like this:



Lazarus IDE v1.6.4 - project1
File Edit Search View Source Project Run Package Tools Window Help
Standard Additional Common Controls Dialogs Data Controls Data Access System Misc LazControls SynEdit RTTI IPro Chart SQLdb Pascal Script
Object Inspector
Components
Form1: TForm1
Properties Events Favorites
Source Editor
unit1
1: 1 INS unit1.pas
Messages

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs;

type
  TForm1 = class(TForm)
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

end.
```

Action	
ActiveControl	
Align	alNone
AllowDropFiles	(False)
AlphaBlend	(False)
AlphaBlendValu	255
Anchors	[akTop,akLeft]
AutoScroll	(False)
AutoSize	(False)
BiDiMode	bdLeftToRight
BorderIcons	[biSystemMenu,I
BorderStyle	bsSizeable
BorderWidth	0
Caption	Form1
ChildSizing	(TControlChildSi

There are many windows, but it gets real easy once you get the hang of it.

At the **top** there are some tools that you would drop into your form. (Tools are formally called “components”. But let’s forget that for now!)

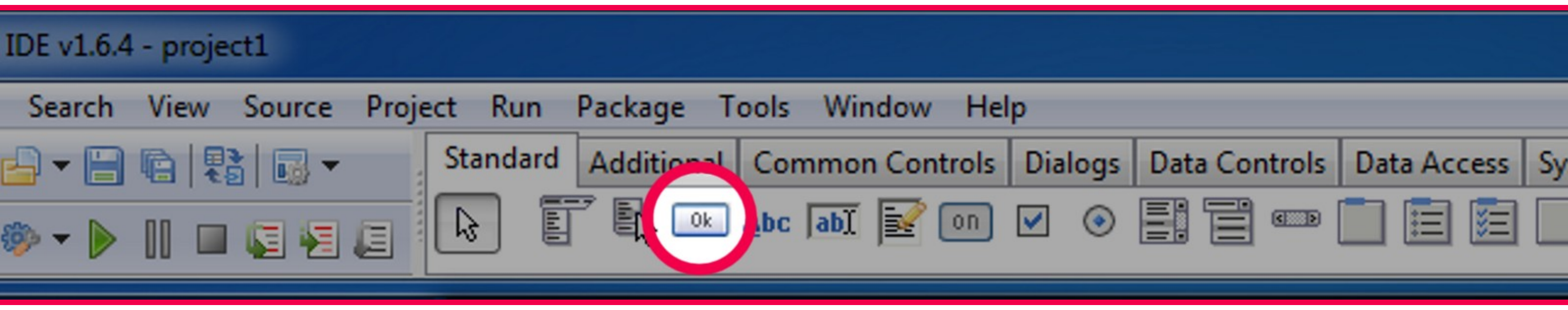
At the **left** you can customize those tools, change appearance etc.

On the **right** you can write code to do something with your program.

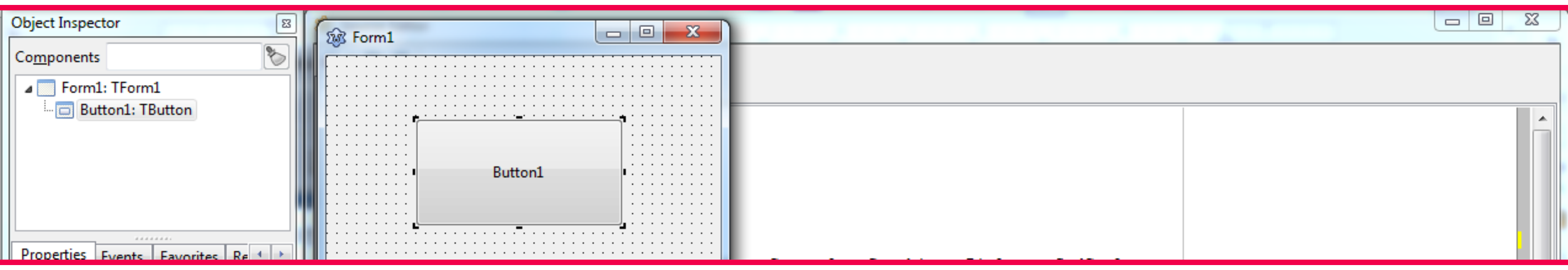
Let’s get into designing. Press **F12**. You should see your form on your screen. It is not currently running. You just design your program here.

Now focus at the top of your screen. You will see endless list of tools.

By default, you should be on the **Standard** tab. Now forget all the other tools, just select the icon that has a little **Ok** written on it.


IDE v1.6.4 - project1
Search View Source Project Run Package Tools Window Help
Standard Additional Common Controls Dialogs Data Controls Data Access Sy
Ok

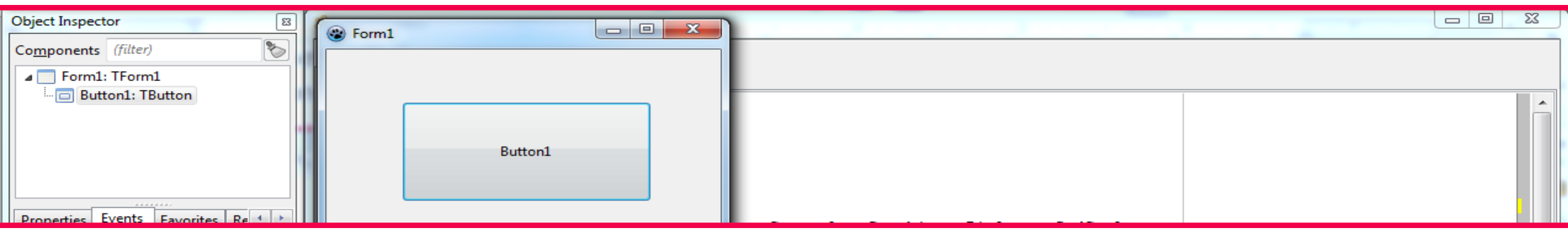
With it selected, click and drag your mouse cursor to draw an imaginary rectangle on your form where you want your button to appear on the form. You will see a button appear on that area.



Now double click on that button. It will create some code and place your cursor in a place. Now, copy-paste the following code where your cursor is placed:

```
Close;
```

Now Run your program pressing **F9** or clicking on **Run – Run** menu. Your program will now run, like it would for a user. Now you should see the exact form you designed earlier onto your screen:



Now you can click on the button and your program will close.

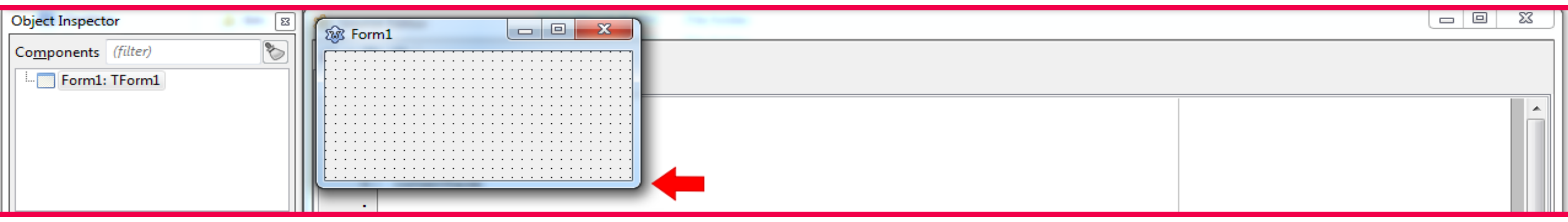
So, now you have created your first app in Lazarus. Sweet!

4: How to Position Stuff on a Form

Positioning stuff on the form lets your users use your software with ease. We can't just stack buttons on top of one another and call it software! So let's move some tools and make it look great...

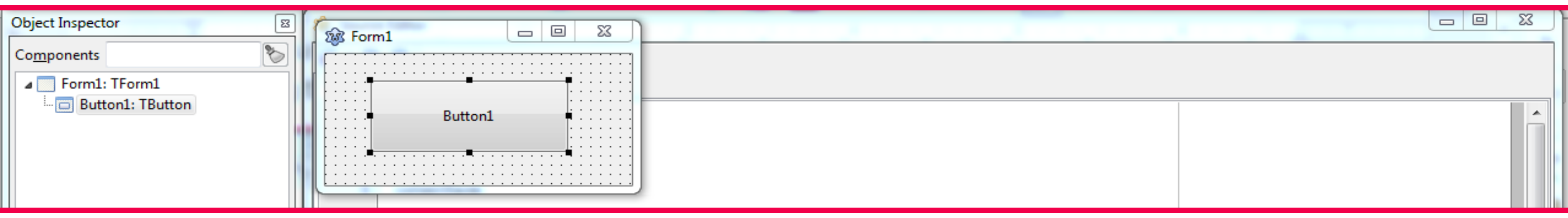
Press **F12** until you bring up the form on screen.

You can use any tools from the top. We would get into it, but first, did you know, you can drag the corner of the form to resize?! Go ahead, try it.

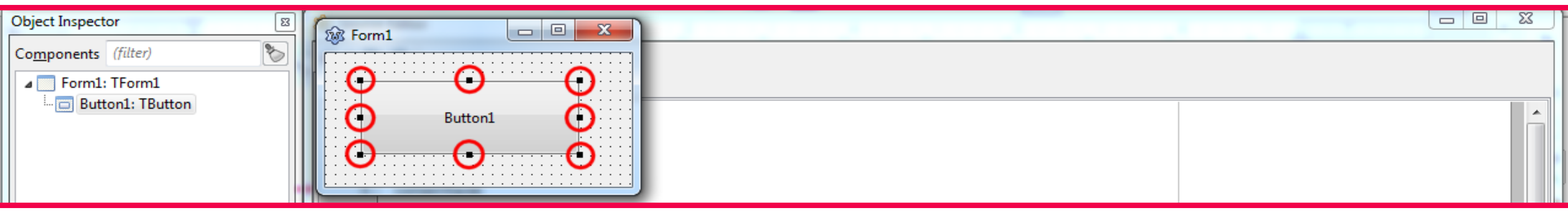


You can click on any tools from the top. Whenever you click on it, it gets selected. Let's click on **TButton**, the one with the **Ok** button.

And then when you draw a rectangle on the form, you will see the tool to be placed on the form. Easy-peasy!



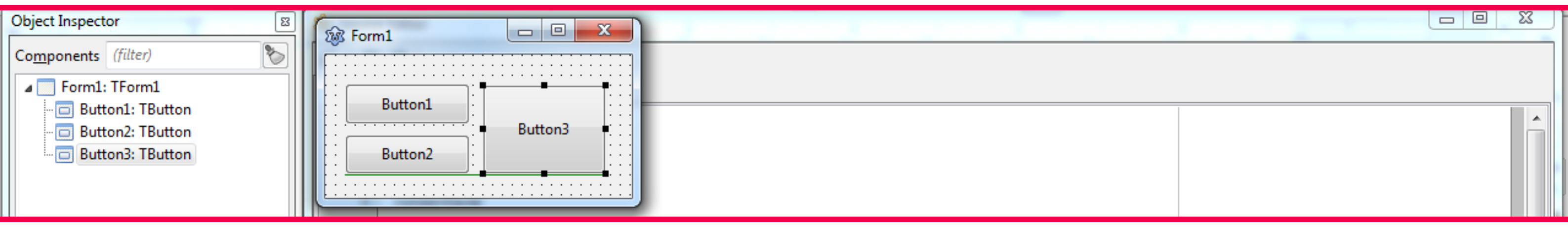
After that you would see a bunch of **black boxes** around the tool you placed:



Drag these **black boxes** – to resize
Drag the **tool itself** – to move

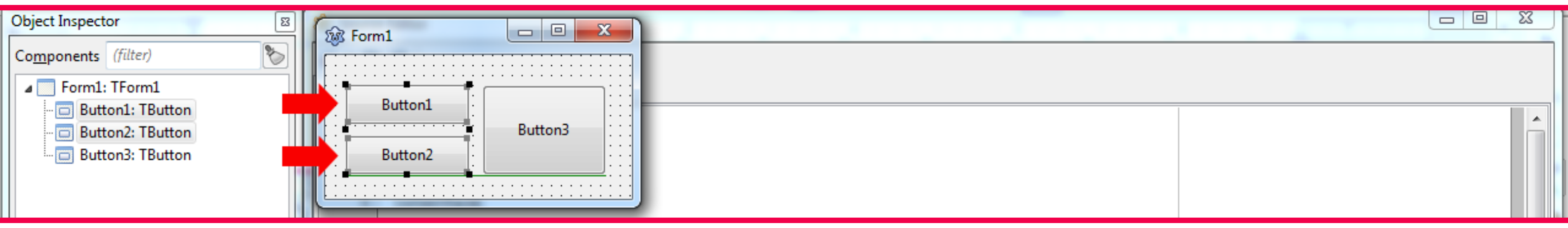
Select **TButton** again from the top and drag another rectangle on your form. Now another tool will be placed and it will be selected. You can do the previous things to it as well – resize or move. Neat!

Go ahead, play all you want. You've earned it! Place some buttons here and there, move them, resize them, do anything with them!



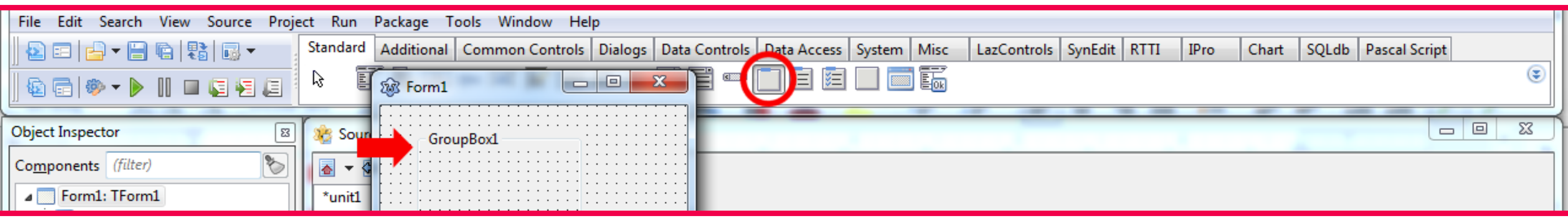
You can **select** any other tool on the form, if you click on it. With it selected you will see the black boxes around it and you can move and resize them anyway you want.

You can also **select multiple items** by holding down **shift and clicking** on them.

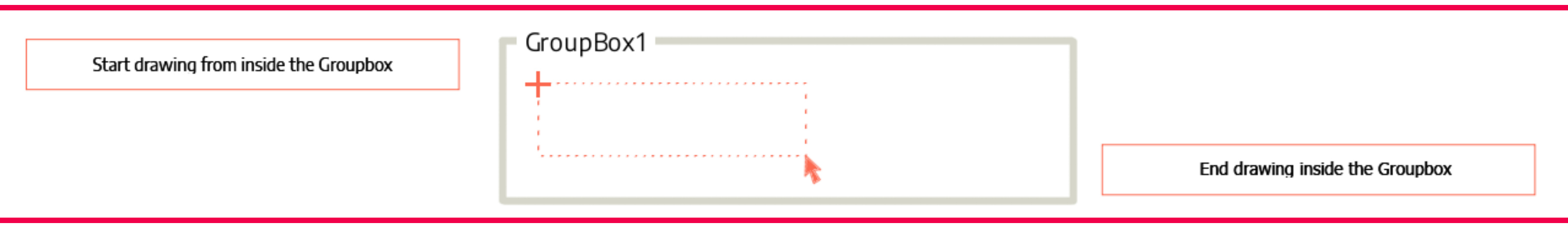


There are some other tools that can hold other tools inside them. For example, TGroupBox, TPanel, TScrollBox etc. (Don't worry, they're just names. You don't have to memorize them. ☺)

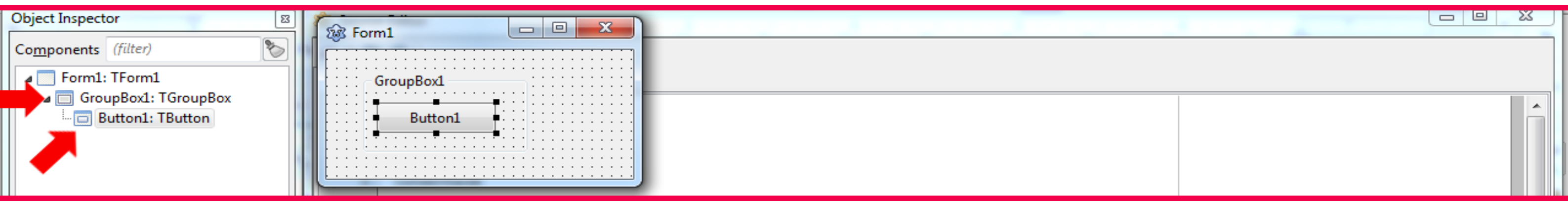
You can draw a **TGroupBox** on the form.


File Edit Search View Source Project Run Package Tools Window Help
Standard Additional Common Controls Dialogs Data Controls Data Access System Misc LazControls SynEdit RTTI IPro Chart SQLdb Pascal Script
Object Inspector
Components (filter)
Form1: TForm1
Form1
GroupBox1
*unit1

To put some tools inside it, when drawing the tool, just start drawing from inside the GroupBox and it will let you keep it inside the GroupBox.


Start drawing from inside the Groupbox
GroupBox1
End drawing inside the Groupbox

When you will do this successfully, the Object Inspector will also show the tool inside the Groupbox.

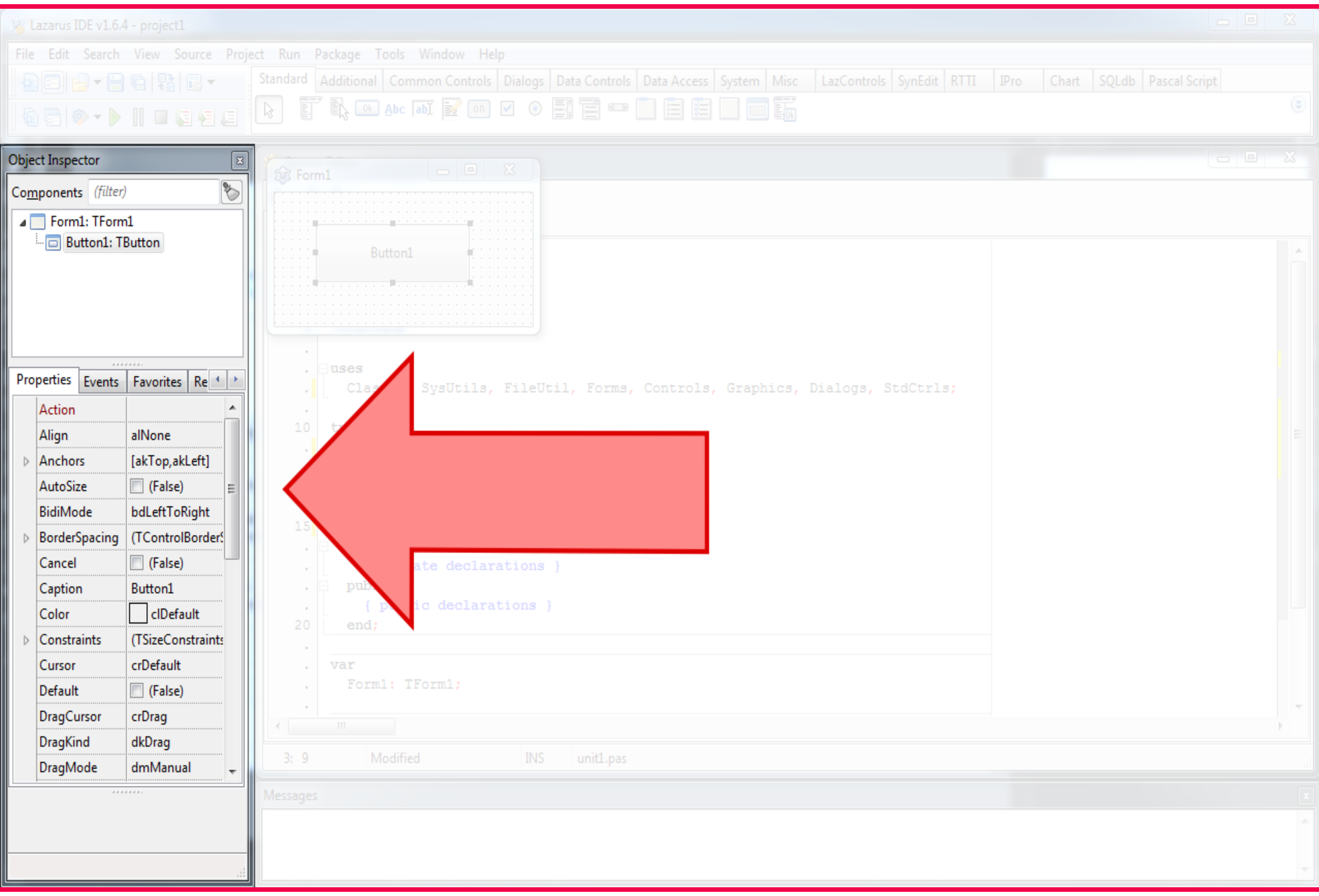


Now, when you move the GroupBox, it will move the tools inside it as well.

5: How to Customize your Tools

There are some default settings on your tools. But you may not always like how it looks or works by default. That's why you need to change some properties once in a while...

On the left you will see a panel with a title **Object Inspector**. This has the **Properties** area. It shows the various aspects you can change for the selected tool.


Lazarus IDE v1.6.4 - project1
File Edit Search View Source Project Run Package Tools Window Help
Standard Additional Common Controls Dialogs Data Controls Data Access System Misc LazControls SynEdit RTTI IPro Chart SQLdb Pascal Script
Object Inspector
Components (filter)
Properties Events Favorites Re
Form1
Button1
Messages

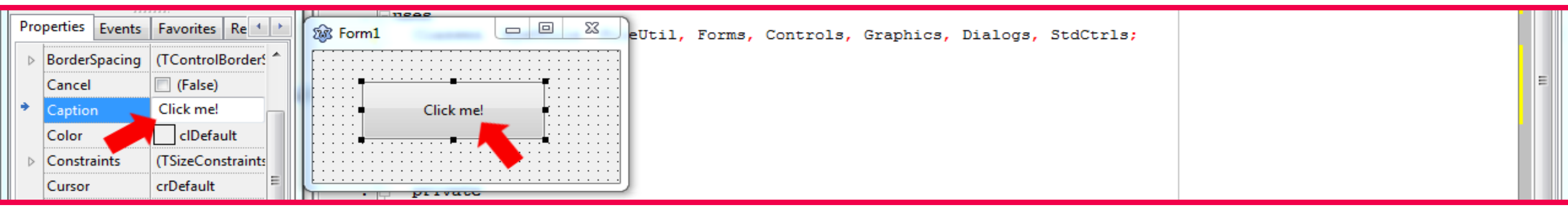
Form1: TForm1
Button1: TButton

Action	
Align	alNone
Anchors	[akTop,akLeft]
AutoSize	(False)
BidiMode	bdLeftToRight
BorderSpacing	(TControlBorderS
Cancel	(False)
Caption	Button1
Color	clDefault
Constraints	(TSizeConstraints
Cursor	crDefault
Default	(False)
DragCursor	crDrag
DragKind	dkDrag
DragMode	dmManual

You can change the Font name, Font size, Caption, Name, Visibility etc. from this pane.

Let's practice... Go on and create a new project: **Project → New Project → Application → OK**. Place a **TButton** on the form. Then click the box to the right of the **Caption** property. Then write a Caption for the button and press enter.

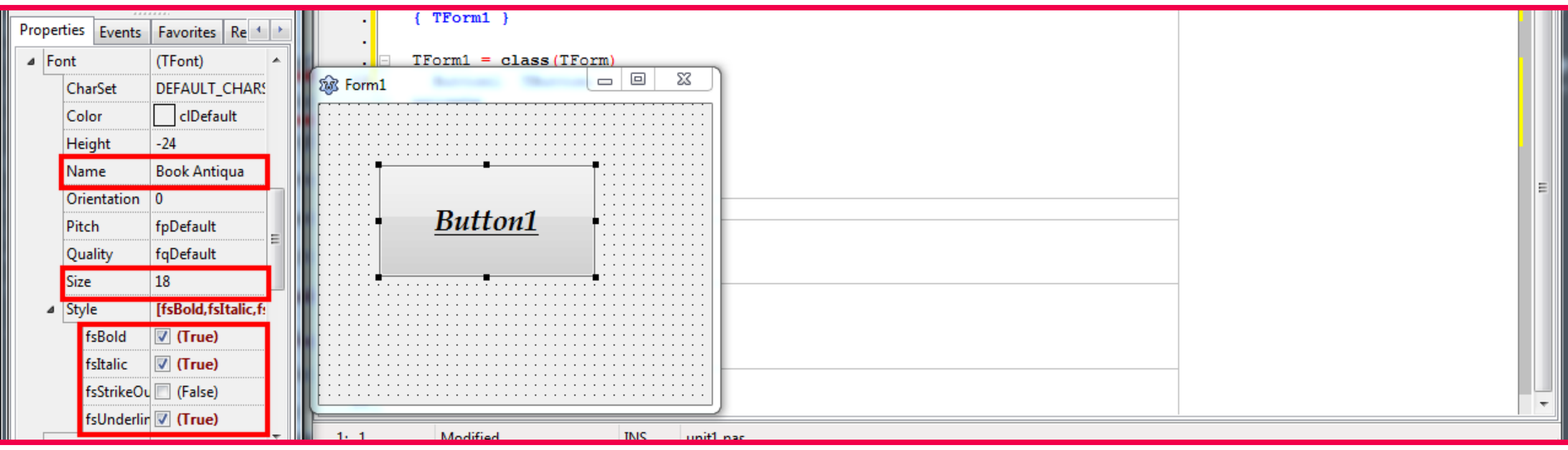
It will change the Caption of the button to your new text:



Properties	
BorderSpacing	(TControlBorder
Cancel	(False)
Caption	Click me!
Color	clDefault
Constraints	(TSizeConstraints
Cursor	crDefault

```
eUtil, Forms, Controls, Graphics, Dialogs, StdCtrls;
```

You can change the button's text size and font. Look into these properties:



Properties	
Font	(TFont)
CharSet	DEFAULT_CHARS
Color	clDefault
Height	-24
Name	Book Antiqua
Orientation	0
Pitch	fpDefault
Quality	fqDefault
Size	18
Style	[fsBold,fsItalic,f
fsBold	✓ (True)
fsItalic	✓ (True)
fsStrikeOu	(False)
fsUnderlir	✓ (True)

```
{ TForm1 }

TForm1 = class(TForm)
```

You can get this if you expand the items under **Font** and **Style** Property. You can see the results on the right.

There are many other properties as well. Go ahead and change them and run your little experiments on them!

Changing Properties with Code

We can also change these from our code if we want. And this is also a great fun to play with. Create a new project and place a button. Double click on `Button1` and enter:

```
Button1.Visible := false;
```

Place another button. This would possibly be named `Button2`. Double click it and enter:

```
Button1.Visible := true;
```

Now Run (F9) and then click on Button1. This would hide Button1. Click on Button2 and it will show again. Just something fun to try on.

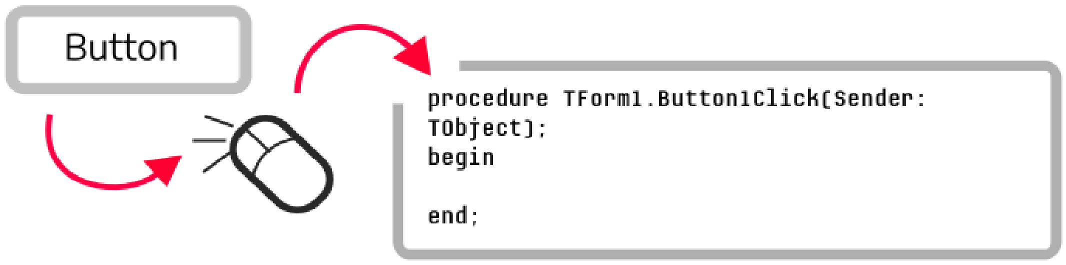
If you don't like calling your buttons Button1, Button2 etc. on your code, you can change their **Name** property to something like btnShow, btnHide etc.

6: How to use Events

An Event runs whenever the user does something. If the user clicks on something an event runs. If he types something an event runs. Whatever he does on your program is an event.

In gist:

- When user clicks, double clicks, moves mouse, types something – it's an **event**.
- We can do something when user does these things – with **code**.

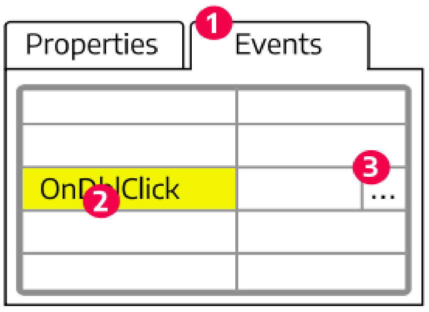


```
procedure TForm1.Button1Click(Sender:
TObject);
begin

end;
```

Whenever you double click a button on the form, Lazarus automatically creates an `OnClick` event and let's you write code. `OnClick` is an event.

We have many other events. When you select a tool on the form, then (1) go to **Object Inspector – Events** tab, we can see these events. (2) Click on an event, (3) then click on **[...]** button beside it. You will see that it creates some code automatically! You can write code where Lazarus placed the cursor, and it will be run when that event occurs.



How to use an Event

1. Go to Events tab
2. Click on an Event
3. Click on the [...] button

Enough talk! Now let's see events in action!

Go on and create a new project: **Project → New Project → Application → OK**. Now Draw a **TButton** on the form (select TButton from the top and draw a rectangle on the form).

Now we will change this button's Caption when user does various things to it. We would show a Caption when user clicks on it. So **double click** on it to create a code for **OnClick** event. And enter:

```
Button1.Caption:='Clicked!';
```

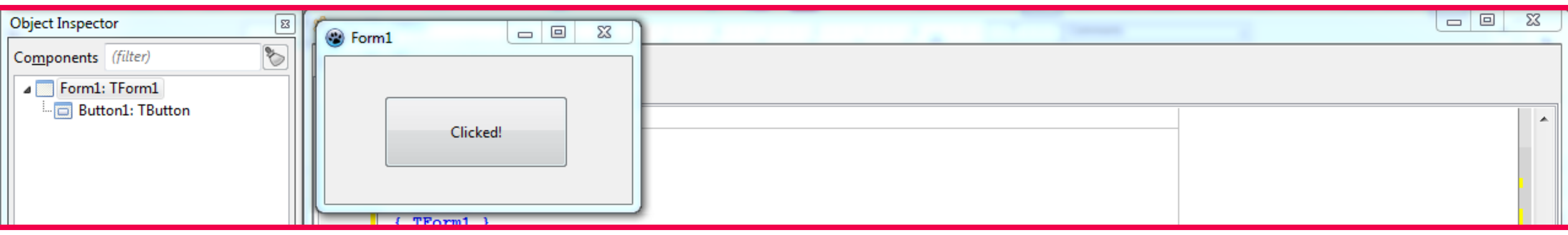
Now, click on the **Events** tab on the left. Click on **OnKeyPress**, click the **[...]** button beside it. Now we would write some code to show when some key is pressed. Enter:

```
Button1.Caption:='Key Pressed!';
```

Now, again, on the **Events** tab click on **OnMouseWheel**, click the **[...]** button beside it. Enter:

```
Button1.Caption:='Mouse wheel scrolled!';
```

Now Run the program by **Run – Run** menu or **F9** key.



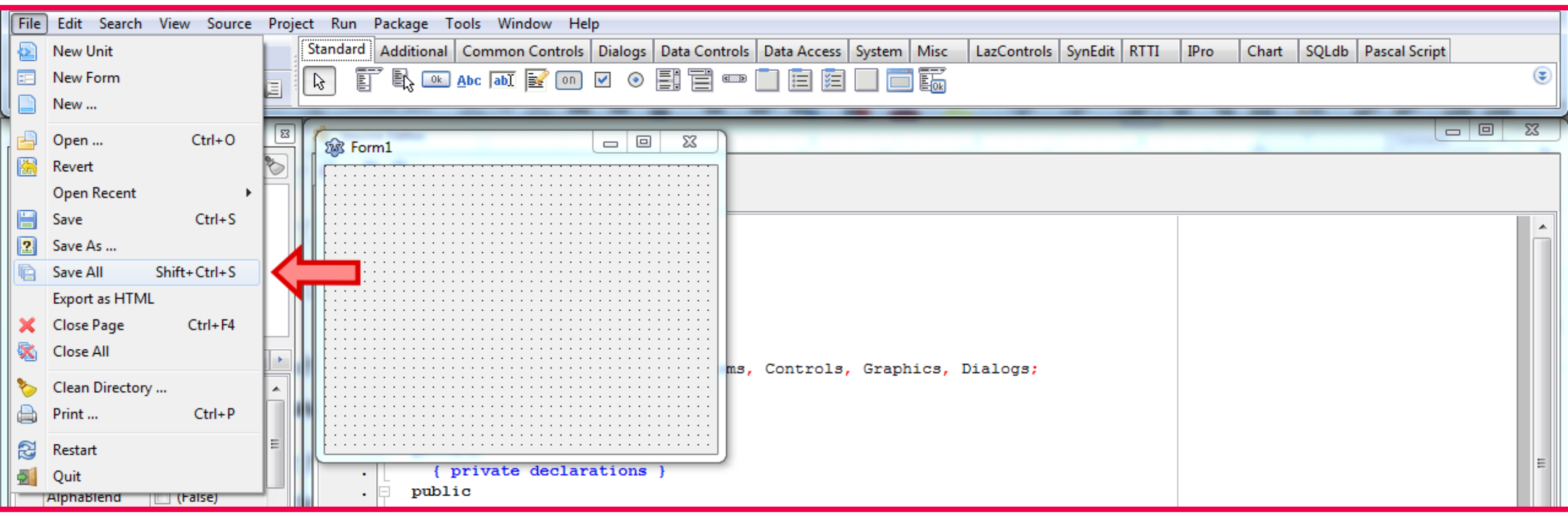
Now click on the button, it will say “Clicked!” Now press some letter keys in the keyboard, it will say “Key Pressed!” Now move your mouse cursor over the button and scroll the mouse wheel (the wheel between your left click and right click buttons), it will show “Mouse Wheel Scrolled!”

Now there you go, a fun project to show how the events work. Have fun!

7: How to Save your Project

There is an art in saving a Lazarus Project! You would need to create a folder, give some appropriate names to your files – it is much more than just saving a file. But at the end it is worth it!

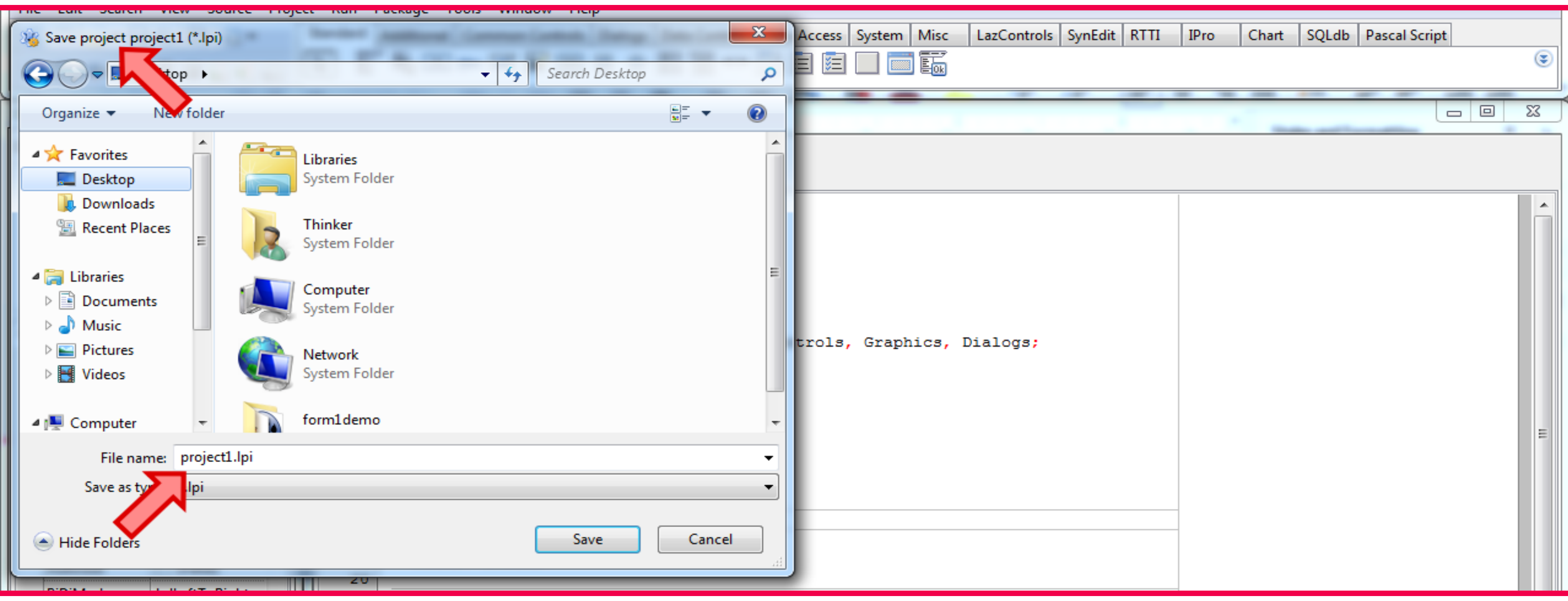
You know what they say - write early, save early! To save a project, you would need to use the **File – Save All** menu.


File Edit Search View Source Project Run Package Tools Window Help
New Unit
New Form
New ...
Open ... Ctrl+O
Revert
Open Recent
Save Ctrl+S
Save As ...
Save All Shift+Ctrl+S
Export as HTML
Close Page Ctrl+F4
Close All
Clean Directory ...
Print ... Ctrl+P
Restart
Quit
Standard Additional Common Controls Dialogs Data Controls Data Access System Misc LazControls SynEdit RTTI IPro Chart SQLdb Pascal Script
Form1

```
ms, Controls, Graphics, Dialogs;
    { private declarations }
  public
```

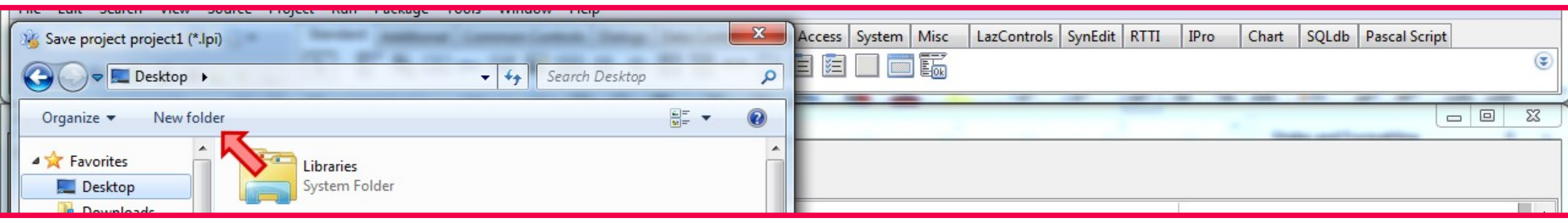
Your project has multiple files for keeping form data, code, and project data. So, be sure to create a folder to accommodate them all.

1. First, when you hit **Save All**, you would be asked for the name of your project file.

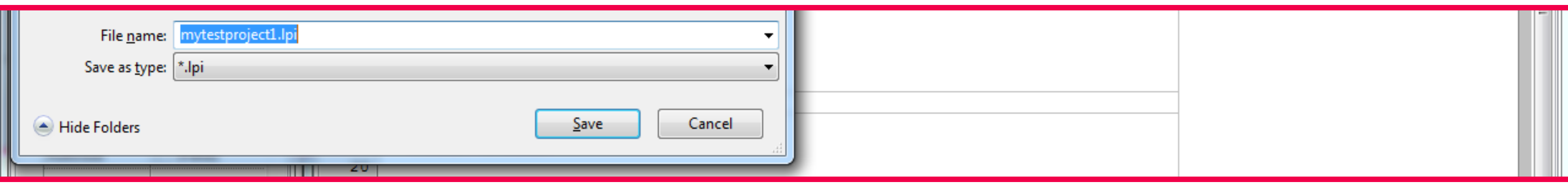


File name: project1.lpi
Save as type: .lpi

2. Don't save yet. Create a folder for it first. (This may vary according to your system, but you know how to create folders, right?! Easy thing to do.)



3. Then give an appropriate file name, like project_1, hello_world, message_demo etc. Remember, Lazarus does not like spaces in these names, and lowercase letters is the norm.

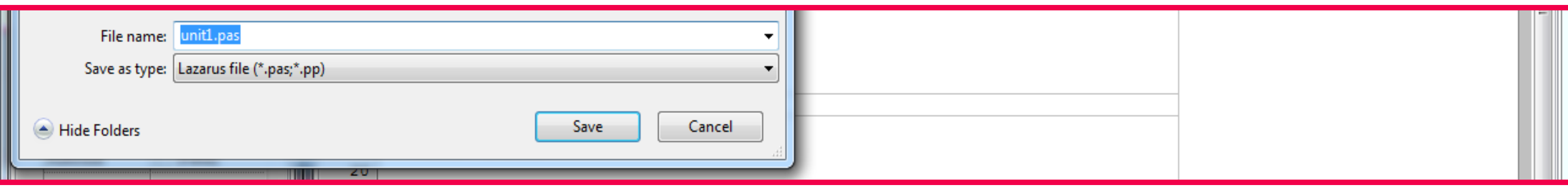


File name: mytestproject1.lpi

Save as type: *.lpi

4. Second, your unit or form files. When saving form data, you can keep the filename as is, or change as you wish.

It can be something like frm1, frmmain, frmmessage etc. But remember, don't name this exactly with your form's name. E.g. naming this `form1` will cause a compile error.



File name: unit1.pas

Save as type: Lazarus file (*.pas;*.pp)

And done!

Now all the project files should be available on the directory. You can double click on the **.lpi** or **.lpr** file to open the project.

8: How to Extend Code with Custom Functions and Variables

We have looked into customizing tools. We can also enjoy convenience by extending code as well. We look into some coding goodies that help ease our writing.

The codes you've written so far, is on a magical language called **Free Pascal**. It has an amazing feature to divide codes into smaller and more manageable pieces – called functions and procedures. Functions and procedures are nearly the same – so some call them just "functions".

- A Function is a collection of code that we group together.
- We give it a name, and later we call it by that name.
- Calling it runs the codes on that function.
- Writing Functions or Procedures is completely optional. We can use it whenever we like, for our convenience.
- We can use a function again and again without rewriting the same code. We can just run the function every time, to run that group of code.

We have some coding involved in this. But don't worry. Just copy-paste and have fun!

Procedure

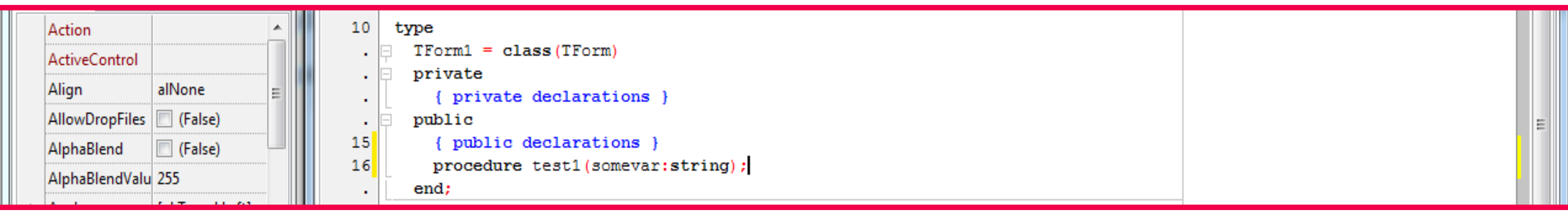
Start Lazarus and create a new Project (**Project – New Project – Application – OK**).

Press **F12** until you see your **Source Editor**. Now find something like `TForm1 = class(TForm)` and some lines below, a line with the text `public`.

Under `public`, there is a line saying `{public declarations}`. This is because we **declare** all the functions and procedures here. (Declare means we just register the name of the functions here. We will elaborate the function later on, on a different place, you'll see.)

Now copy-and-paste the text below after the `public` line:

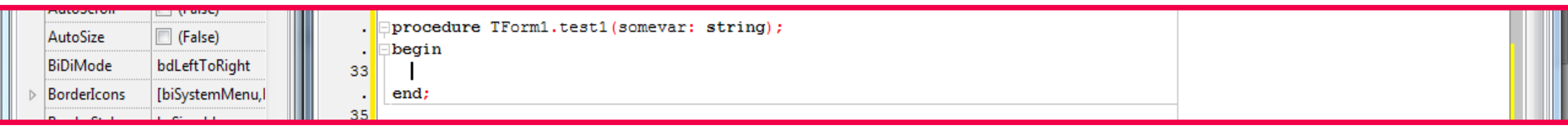
```
procedure test1(somevar:string);
```



Action	
ActiveControl	
Align	alNone
AllowDropFiles	☐ (False)
AlphaBlend	☐ (False)
AlphaBlendValu	255

```
type
  TForm1 = class(TForm)
  private
    { private declarations }
  public
    { public declarations }
    procedure test1(somevar:string);
  end;
```

After you have pasted it, with your cursor still on the line, press **Ctrl+Shift+C**. Lazarus will then automatically create a place for this procedure so that you can write some code on it.



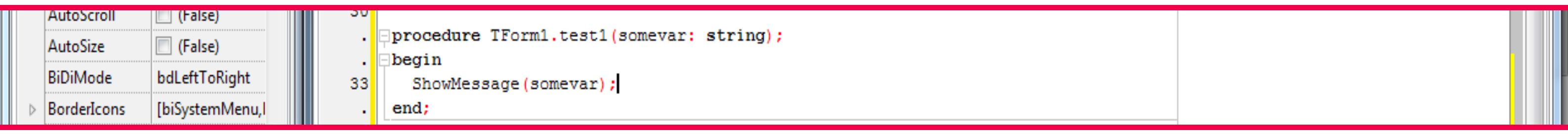
```
procedure TForm1.test1(somevar: string);
begin

end;
```

You can write anything on it, such as:

```
ShowMessage(somevar);
```

This will use the `somevar` variable that it passed to our procedure. But let's forget this for the moment.



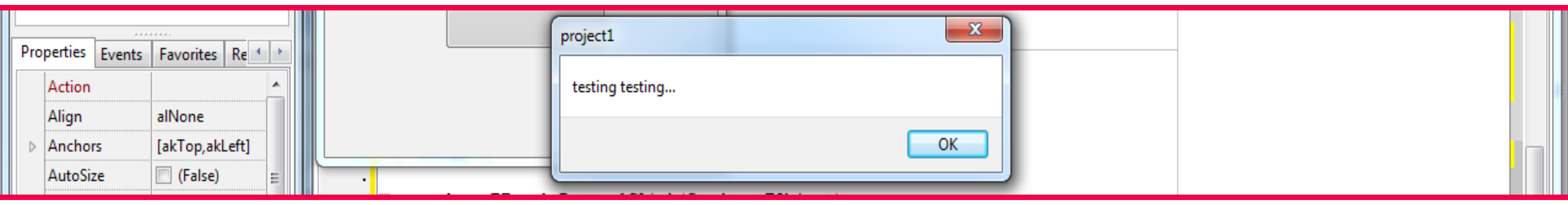
```
procedure TForm1.test1(somevar: string);
begin
  ShowMessage(somevar);
end;
```

Now, press **F12** to show the form. Now create a button on the form. Then double click on the button. A procedure will be created.

Write something like this on the procedure:

```
test1('testing testing...');
```

Then Run the project (**F9** or **Run - Run** menu). Now click the button. It will show `testing testing`... in a messagebox. Cool!

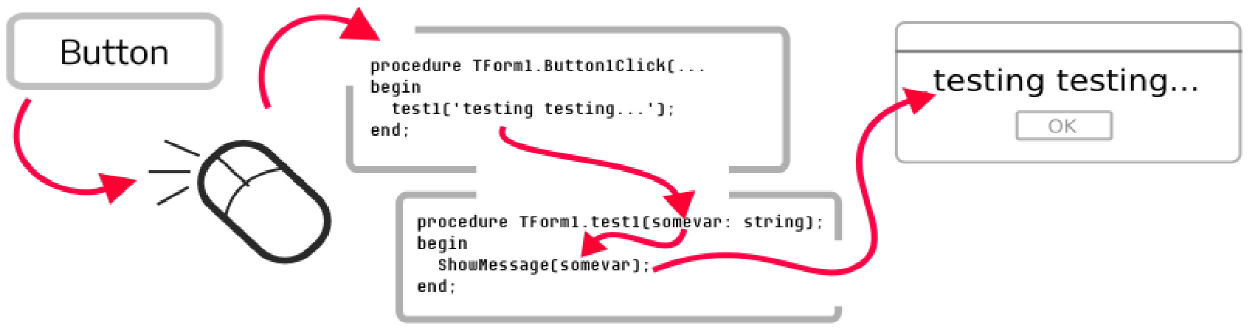


project1

testing testing...

What is happening is:

- when we click the button we are running our `test1` procedure.
- Our `test1` procedure is then running the `ShowMessage` procedure.
- `ShowMessage` procedure is getting the message from our `somevar` variable.



```
procedure TForm1.Button1Click(...
begin
  test1('testing testing...');
end;
```

```
procedure TForm1.test1(somevar: string);
begin
  ShowMessage(somevar);
end;
```

We could've avoided the use of a procedure since the code is small. But we are just using it to learn procedures.

Trust me, it will get easier over time.

Functions

Functions are very similar to procedures. As an extra, it can just give us something after it's finished.

Let's say, we want to write a function to add 2 numbers. It would take 2 numbers, sum them up, and then give us the total to us. Programmers use it all the time to make the computer do stuff and get stuff!

Go to Project – New Project – Application – OK. Then press F12 to see your code editor. Now find `TForm1 = class(TForm)` and then `public` under the line.

Now copy-and-paste the text below after this line:

```
function add(num1:integer; num2:integer):integer;
```

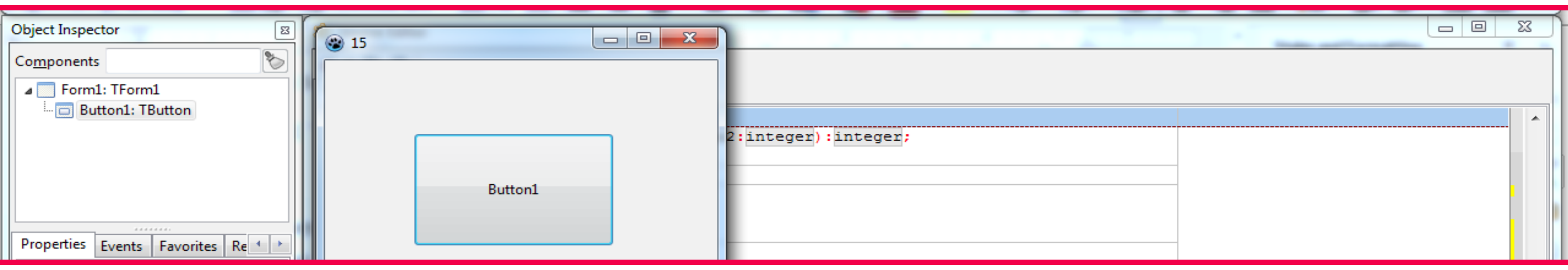
Now with your cursor still on the line, press **Ctrl+Shift+C**. You should get the function code to be automatically generated. Now write:

```
Result := num1 + num2;
```

Now press **F12** to go to form view. Place a button, then double click it. Then write the following code:

```
Caption := inttostr( add(10,5) );
```

Now click **Run – Run** or press **F9** to run. Then click on the button. Tada! It will add 10 and 5 and show 15 as a result in the Titlebar of our form.



For curious ones...

You can skip this part...

You would notice that we've used `inttostr()` in our code. It is also a function, something like this:

```
function IntToStr(Value: Longint): string;
```

It takes a number (which is also called `Integer` or `LongInt`) and converts it into a `string` (text). **Integers** are numbers, and **Strings** are text – this is enough to know right now.

As humans, we see numbers and text as the same, but computer sees numbers in one way and text in a different way. That's why the separate naming!

So, the problem was that our Form Caption was a String and in our function code, we had an interger. So we needed to convert it to string to show it on the Caption.

You will also notice that we have seen Procedures before. Remember when we created events? Whenever we pressed the [...] button beside an event, it creates a procedure for us. The procedure had many extra data passed to us. For example, if a key is pressed, we have a variable to determine which key was pressed:

```
procedure TForm1.Button1KeyPress(Sender: TObject; var Key:
char);
begin
  …
```

The `Key` here is a variable that has the key that has been pressed. Speaking of variables...

Variables

Well, Variables are nothing, they are just names so that you can store some stuff in them!

You can store numbers, letters, sentences in them. You can also change them, work with them, add them, multiply them etc. Fun stuff!

For example, in math:

```
Let x = 10
So, x + 20 = 30
```

Similarly, when coding:

```
procedure examplemath();
var
  x: integer;
begin
  x := 10;
  ShowMessage( inttostr( x + 20 ) );
end;
```

It is the same as we use variables in math. In both cases, we are storing the number `10` in the variable `x`.

- When we need to store **numbers** (1, 2, 3...) – call it **integer**
- ...to store **text** – call it **string**
- ...to store **numbers with decimal places** (1.1, 1.2, 1.3...) – call it **real** or **single** or **double**
- ...to store only **yes and no** or **true and false** – call it **boolean**

I know it's a mouthful. But you will get used to it in no time once you start working on your projects! I know I did.

Constants

Let me mention just one little thing. There is also a kind of variable, the value of which cannot be changed. Once you set it, it's set. It's called a **constant**. You can write it under the `const` clause, something like this:

```
const
  pi = 3.1415;
```

Some Other Tiny, but Important Things...

We take decisions in our life all the time. Our code can do that too, based on **logic**.

For example, "if it is raining outside, take the umbrella, otherwise not." This is perfect logic!

If we code this, it can become something like:

```
var
  raining: boolean; // true = raining, false = not raining
begin
  raining := SomeFunctionToSeeIfRaining(); // this function
will do its thing and tell us if it is raining, true or false

  if (raining = true)
    ShowMessage('Take the umbrella');
  else
    ShowMessage('No need to take the umbrella, for now');
end;
```

So, our code is getting information and taking a decision based on that information.

Another thing is **looping**. Doing the same thing again and again. Loops are very common to **automate** something. For example, an automated robot hand in a car factory does the same thing again and again. It is looping the same actions all day.

We can make the computer say `Hello!` five times, just for fun, with something like this:

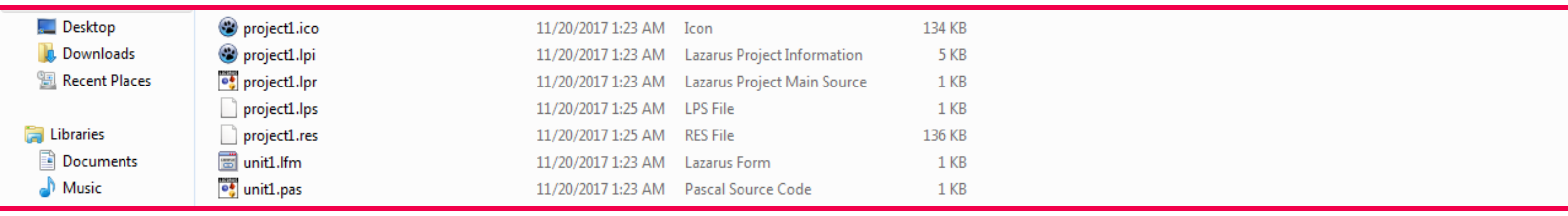
```
var
  index: integer;
begin
  for index := 1 to 5 do
    ShowMessage('Hello!');
```

We make a variable named `index`, and the `for` command gives this variable a value of 1, then runs the code, value of 2, then runs the code, value of 3, then runs the code and so on. You can write `ShowMessage( 'Hello '+inttostr(index) );` and see what happens!

9: Sharing Your Project

Sharing is caring. In order to share your code to the public or someone in private, you can tidy up things for the other person. Such as, cleaning up unnecessary files and including hints in the code etc.

When you Save your project, some files get created. These are just to make the project work.


Desktop
Downloads
Recent Places
Libraries
Documents
Music

project1.ico	11/20/2017 1:23 AM	Icon	134 KB
project1.lpi	11/20/2017 1:23 AM	Lazarus Project Information	5 KB
project1.lpr	11/20/2017 1:23 AM	Lazarus Project Main Source	1 KB
project1.lps	11/20/2017 1:25 AM	LPS File	1 KB
project1.res	11/20/2017 1:25 AM	RES File	136 KB
unit1.lfm	11/20/2017 1:23 AM	Lazarus Form	1 KB
unit1.pas	11/20/2017 1:23 AM	Pascal Source Code	1 KB

But there are some files that are not needed.

- Delete `backup` and `lib` folders from the project folder as a basic cleanup.
- Sometimes you can have a `.dbg` file on the project directory. You can delete that as well.

If you want, you can include the exe with the project files. But some people like to keep the source code separate from the non-source files, like exe files. So it depends.

Note: Windows executables have .exe extension. Mac OS X has .dmg. Linux does not have any extension, only permissions to determine if it's an executable. If it is not executing for some reason, you can run `chmod +x /path/to/executable` on Terminal.

Also, you can include a **Readme.txt** file to let people know what your code does.

Before creating exes, you can **comment** your codes. They say, the most mysterious code is what you, yourself wrote 5 years ago and did not comment it! Without comments it is difficult to understand what some code does. This is especially true if you are creating a large project that does big things.

Comments are like little notes in the code. The compiler does not bother with them, so you can write anything you want – it will not run. Something like:

```
// Some notes
```

or you can use multi-line comments:

```
{ Some long
notes }
```

For curious ones...

You can skip this...

After you have done the cleanup and commenting, you can share your code on the internet. You can share your code on Github.com, BitBucket.com, notabug.org etc. You can upload the code or you can also use tools like `git`.

These sites have separate sections in the repository to keep the exe files, named "Downloads" or "Releases".

To host your code on one of these, you can create a code repository, then clone the repository, put your code in, add the files, then commit. It would look something like this:

```
git clone somegiturl.git
cd some-repo-directory
# add some files, then...
git add .
git commit -m "initial commit"
git push origin master
```

When you make some change, you can:

```
cd some-repo-directory
git add .
git commit -m "some change"
git push origin master
```

Seems like too much, but once you get used to it, it will be so smooth that someone seeing you do all this would get jealous. I know I get some people jealous!

Plus, git lets you see what you changed through your codes for its lifetime. So no matter what you do to your code, it will keep a record of your changes, line by line.

Git is used by professionals to maintain open source projects. Git is incredibly powerful and very useful to share your code with the world. And more importantly, it helps multiple people to work on the same code – isn't that amazing?!

Although it's hard for some people, there are tools like Github for Windows that makes the process easier.

But again, you can just use plain simple email to share your code.

Sharing is fun, once you get the hang of it. But it is completely optional if you want to keep the source to yourself.

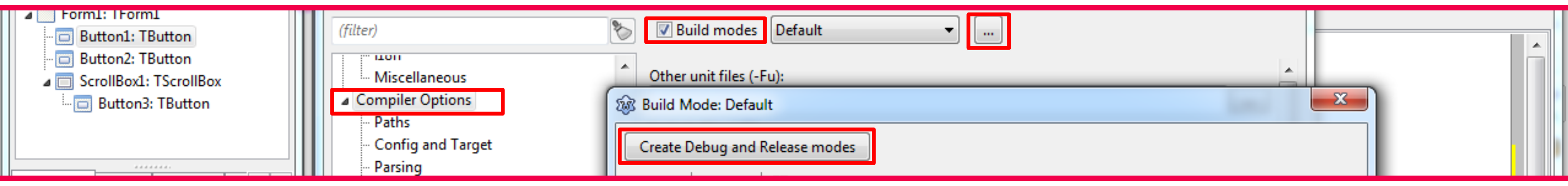
10: Compiling and Sharing Your Executable

You can share your source code or your executables. Whatever you do, you need to read this.

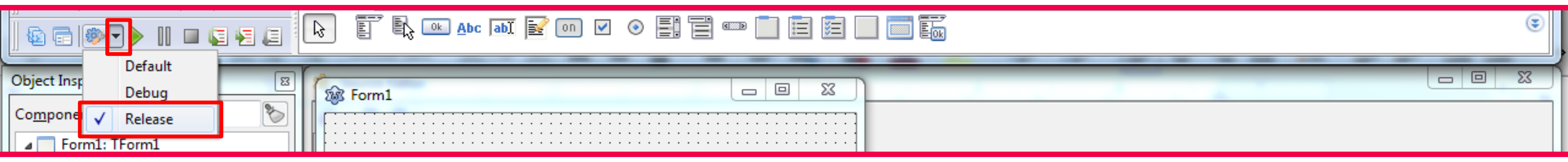
We compile all the time. Whenever we Run our program it is actually being compiled, it creates a .exe file and then executes it. But it has debug data, so the exe is huge in size.

To be able to ditch debug data from exe, you can follow these crazy steps...

Go to **Project – Project Options – Compiler Options**.
Check **Build Modes**.
Then click **[...]**.
Then **Create Debug and Release modes**.
Press **OK**, then **OK** again.



Now click the arrow beside the **Cog icon** on toolbar and select **Release**.



Now when you **Run - Run** the project or select **Run - Build**, it will create an exe file and get rid of all the debug data.

You should get the exe file on your project folder. You can get it and share it with everyone on the internet.

Tip: To change the icon of the exe file, you can use: **Project – Project Options – Icon – Load Icon**. You can use png, gif, ico etc. files for icon. You can find many icons on the internet which are aptly licensed. http://aiconica.net is a good place to start.

Congratulations!

If you have followed along, then you have successfully landed on the Planet of Lazarus Users! You are part of the community!

But is this the end? No. This is just the beginning! This is some stuff to let you know how to do basic things in Lazarus. You've got many other things to build and code!

To continue this wonderful journey, you can check these pages:

- This video more or less summarizes what we have learned in this book: https://www.youtube.com/watch?v=3APFv8Rj2iU
- Other videos to get you started easily. Good for beginners: http://wiki.freepascal.org/Lazarus_videos
- Incredibly detailed Lazarus basics page: http://wiki.lazarus.freepascal.org/Lazarus_Tutorial
- Build cool things. Has little sample codes and projects to tingle your curiosity: http://lazplanet.blogspot.com
- Easy FreePascal Tutorial. Short and Informative: http://wiki.freepascal.org/Object_Pascal_Tutorial
- E-Book for learning Free Pascal. In depth, but good for reference. Teaches through code examples: http://code-sd.com/startprog/StartProgUsingPascal.pdf