# FPC/Lazarus Notes

## *Preamble*

About 7 years ago, when I had thoughts to turn Pascal coding from a favorite hobby into a means of earning money, I began to think about the license purity of my programs.

Since I am not a professional programmer, the thought of buying a Delphi had to be dismissed immediately. In addition, the lack of cross-platform functionality and the insane marketing policy of the companies that owned Delphi did not inspire optimism at all and discouraged any desire to invest money there.

IMHO, just by this time FPC / Lazarus from a raw open source project of a handful of enthusiasts began to turn into a quite serious tool. Of course, at some points Lazarus is still very far from Delphi (for example, the latter's debugger is an order of magnitude more convenient and functional). But the Lazarus code editor with its highlighting and other goodies leaves Delphi far behind. Plus, Lazarus is ABSOLUTELY FREE. And cross-platform. And bugs in it are corrected, if not hourly, then definitely daily.

But, like any open source, FPT/Lazarus requires manual assembly and tuning. And there is very little intelligible literature on it. Therefore, I decided to post my experience of using them in the form of articles.

I would be glad if this material helps anyone.

Zoltanleo, aka Док (2021г.)

## Table of contents

# Building FPC/Lazarus.

## Building via batch files

Coming soon ...

## Building via fpcupdeluxe

Coming soon ...

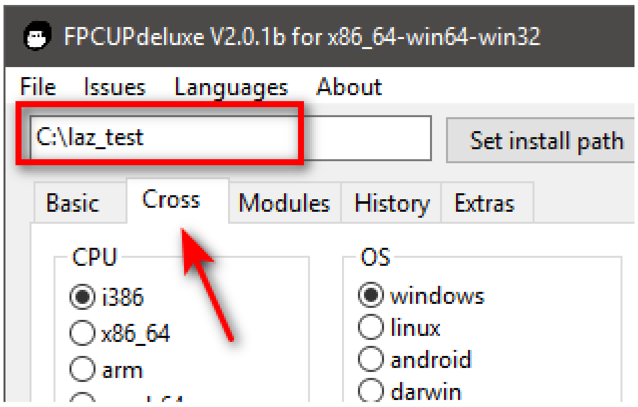
# Cross compilation to FPC

[1]:

## Installing the crosscompiler

We will assume that the compiler and lazarus are already installed using fpcupdeluxe and the "set install path" field contains the folder with installed fpc / lazarus

Go to the "cross" tab

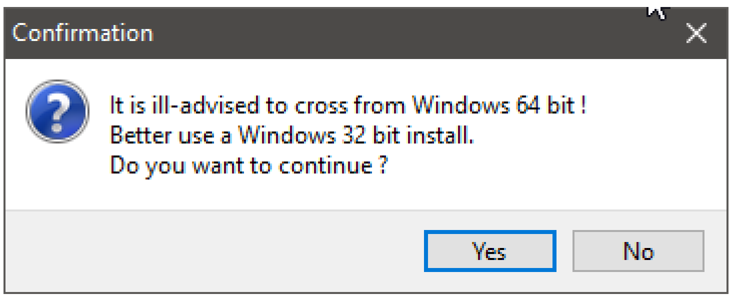


Select the bitness of the processor and the OS.

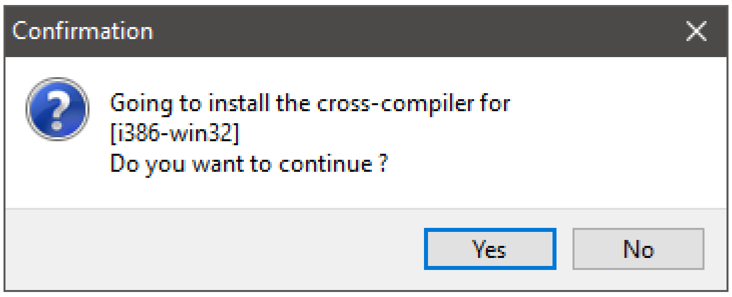
*Note*: you can build a crosscompiler for a processor of a different bit capacity on the current platform. The main thing to remember is what bit depth you have running fpcupdeluxe (ofthe sware shown on the screenshot collects a 64-bit fpc for Windows, which means that the x32/i386 compiler assembly will be the "cross-platform" for Windows, etc.)

### Windows

Select `CPU = i386` and `OS = Windows` respectively, and click "Install compiler". Say "Yes" to both dialogues

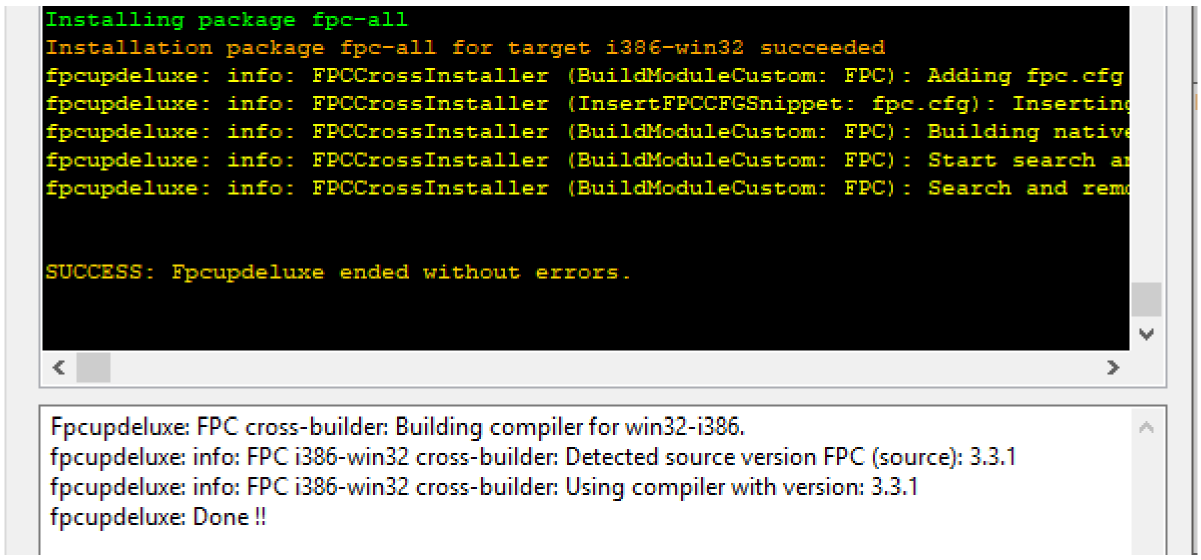

Confirmation

It is ill-advised to cross from Windows 64 bit !
Better use a Windows 32 bit install.
Do you want to continue ?


Confirmation

Going to install the cross-compiler for
[i386-win32]
Do you want to continue ?

The x32 compiler should build easily without any additional hassle.


```
Installing package fpc-all
Installation package fpc-all for target i386-win32 succeeded
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Adding fpc.cfg
fpcupdeluxe: info: FPCCrossInstaller (InsertFPCCFGSnippet: fpc.cfg): Insertin
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Building nativ
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Start search a
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Search and rem


SUCCESS: Fpcupdeluxe ended without errors.
```

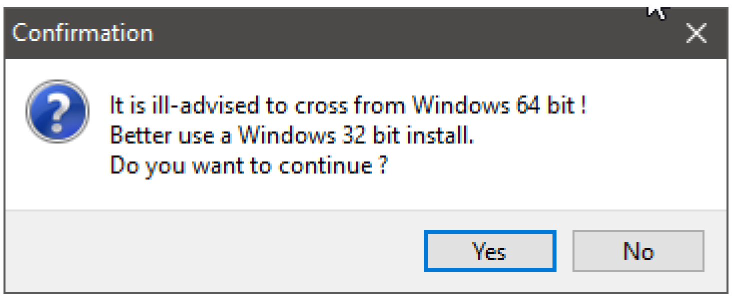
```
Fpcupdeluxe: FPC cross-builder: Building compiler for win32-i386.
fpcupdeluxe: info: FPC i386-win32 cross-builder: Detected source version FPC (source): 3.3.1
fpcupdeluxe: info: FPC i386-win32 cross-builder: Using compiler with version: 3.3.1
fpcupdeluxe: Done !!
```

*Note*: how to make the build mode of the executable file under x32, see next chapter.

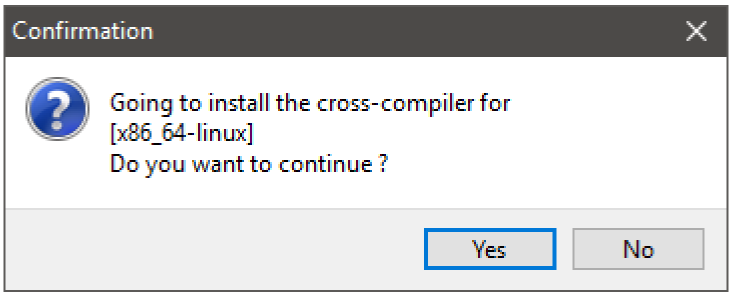
## Linux

Since the assemblies of the crosscompiler for i386 and amd64 are identical, I will describe the process for the last listed one.

Select `CPU=x86_64` and `OS=linux` respectively, and click "Install compiler". Say "Yes" to both dialogues

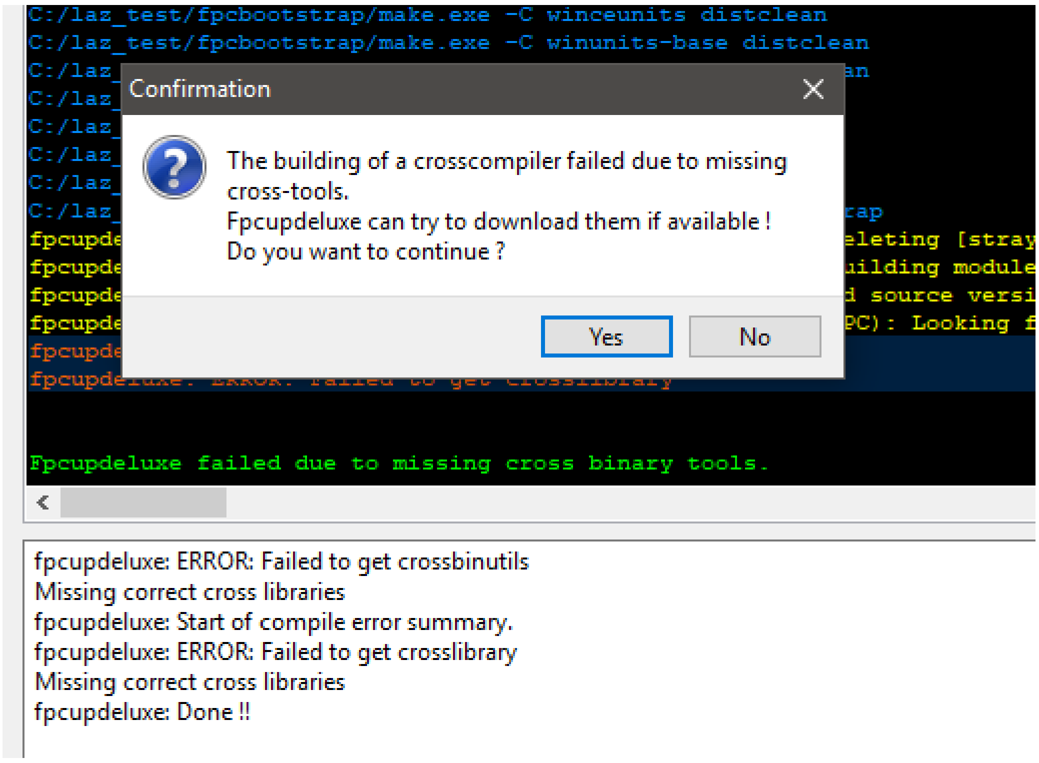


```
Confirmation
It is ill-advised to cross from Windows 64 bit !
Better use a Windows 32 bit install.
Do you want to continue ?
```



```
Confirmation
Going to install the cross-compiler for
[x86_64-linux]
Do you want to continue ?
```

and wait until the utility informs us that it does not currently have a tool for building the crosscompiler. Then she will offer to download them

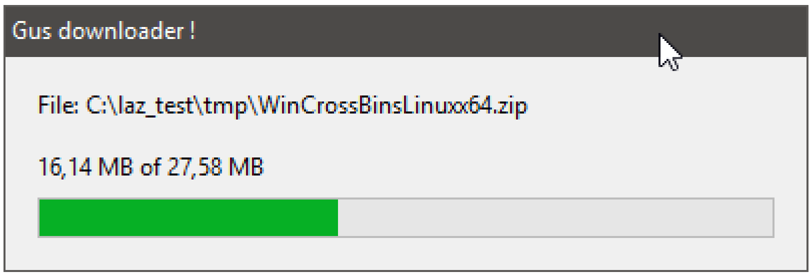


```
C:/laz_test/fpcbootstrap/make.exe -C winceunits distclean
C:/laz_test/fpcbootstrap/make.exe -C winunits-base distclean

Fpcupdeluxe failed due to missing cross binary tools.
```

```
Confirmation
The building of a crosscompiler failed due to missing
cross-tools.
Fpcupdeluxe can try to download them if available !
Do you want to continue ?
```

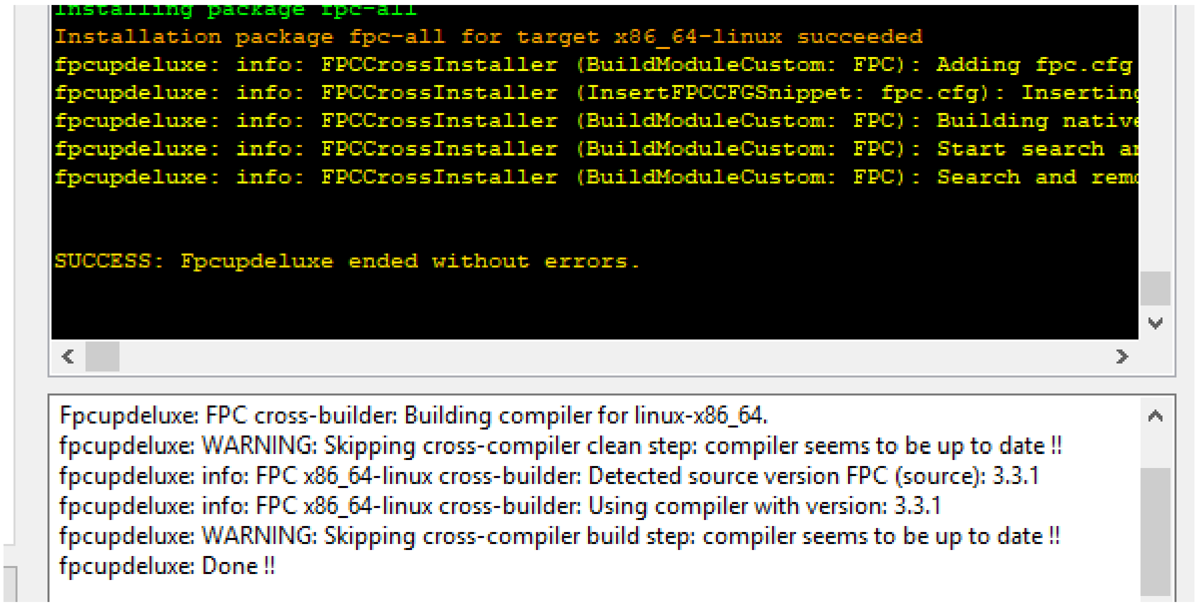
```
fpcupdeluxe: ERROR: Failed to get crossbinutils
Missing correct cross libraries
fpcupdeluxe: Start of compile error summary.
fpcupdeluxe: ERROR: Failed to get crosslibrary
Missing correct cross libraries
fpcupdeluxe: Done !!
```

If we agree, we will be able to observe the process of downloading and unpacking



```
Gus downloader !
File: C:\laz_test\tmp\WinCrossBinsLinuxx64.zip
16,14 MB of 27,58 MB
```

Then we patiently wait for building end.



```
Installing package fpc-all
Installation package fpc-all for target x86_64-linux succeeded
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Adding fpc.cfg
fpcupdeluxe: info: FPCCrossInstaller (InsertFPCCFGSnippet: fpc.cfg): Inserting
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Building native
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Start search ar
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Search and remo


SUCCESS: Fpcupdeluxe ended without errors.
```

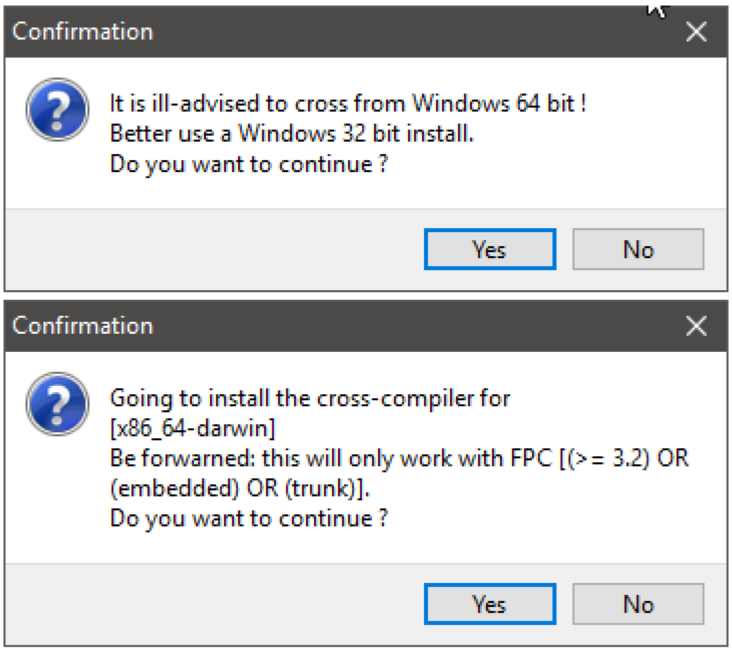
```
Fpcupdeluxe: FPC cross-builder: Building compiler for linux-x86_64.
fpcupdeluxe: WARNING: Skipping cross-compiler clean step: compiler seems to be up to date !!
fpcupdeluxe: info: FPC x86_64-linux cross-builder: Detected source version FPC (source): 3.3.1
fpcupdeluxe: info: FPC x86_64-linux cross-builder: Using compiler with version: 3.3.1
fpcupdeluxe: WARNING: Skipping cross-compiler build step: compiler seems to be up to date !!
fpcupdeluxe: Done !!
```

*Note*: how to make the build mode of the executable file under i386/amd64, see next chapter.

## Darwin

Since for testing my projects under MacOS I use its hackintosh assemblies in a virtual machine that use an x86_64 processor, here I will describe the process of building a crosscompiler of this architecture.

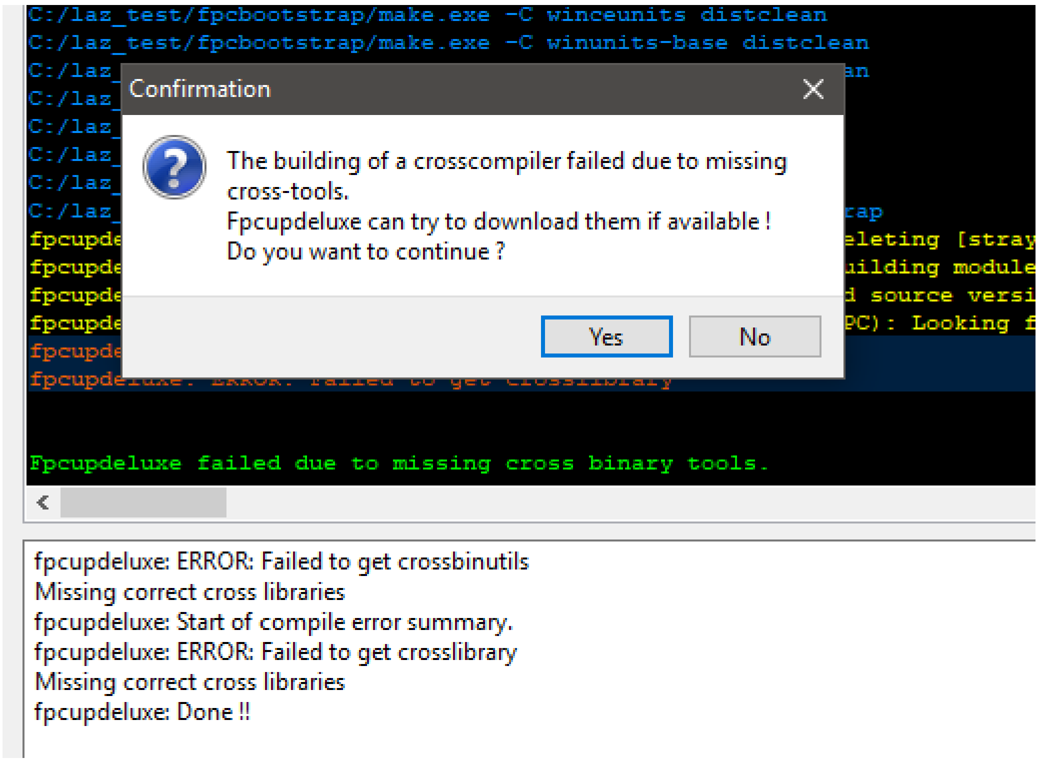
Select `CPU=x86_64` and `OS=Darwin` respectively, and click "Install compiler". Say "Yes" to both dialogues



It is ill-advised to cross from Windows 64 bit !
Better use a Windows 32 bit install.
Do you want to continue ?

Going to install the cross-compiler for
[x86_64-darwin]
Be forwarned: this will only work with FPC [(>= 3.2) OR (embedded) OR (trunk)].
Do you want to continue ?

and wait until the utility informs us that it does not currently have a tool for building the crosscompiler. Then she will offer to download them



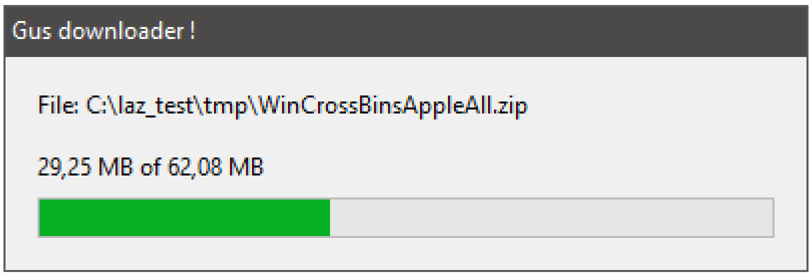
```
C:/laz_test/fpcbootstrap/make.exe -C winceunits distclean
C:/laz_test/fpcbootstrap/make.exe -C winunits-base distclean
C:/laz                                                   an
C:/laz
C:/laz
C:/laz
C:/laz
C:/laz                                                    rap
fpcupde                                                   eleting [stray
fpcupde                                                   uilding module
fpcupde                                                   d source versi
fpcupde                                                   PC): Looking f
fpcupde
fpcupdeluxe: ERROR: Failed to get crosslibrary


Fpcupdeluxe failed due to missing cross binary tools.
```

Confirmation

The building of a crosscompiler failed due to missing cross-tools.
Fpcupdeluxe can try to download them if available !
Do you want to continue ?

```
fpcupdeluxe: ERROR: Failed to get crossbinutils
Missing correct cross libraries
fpcupdeluxe: Start of compile error summary.
fpcupdeluxe: ERROR: Failed to get crosslibrary
Missing correct cross libraries
fpcupdeluxe: Done !!
```

If we agree, we will be able to observe the process of downloading and unpacking

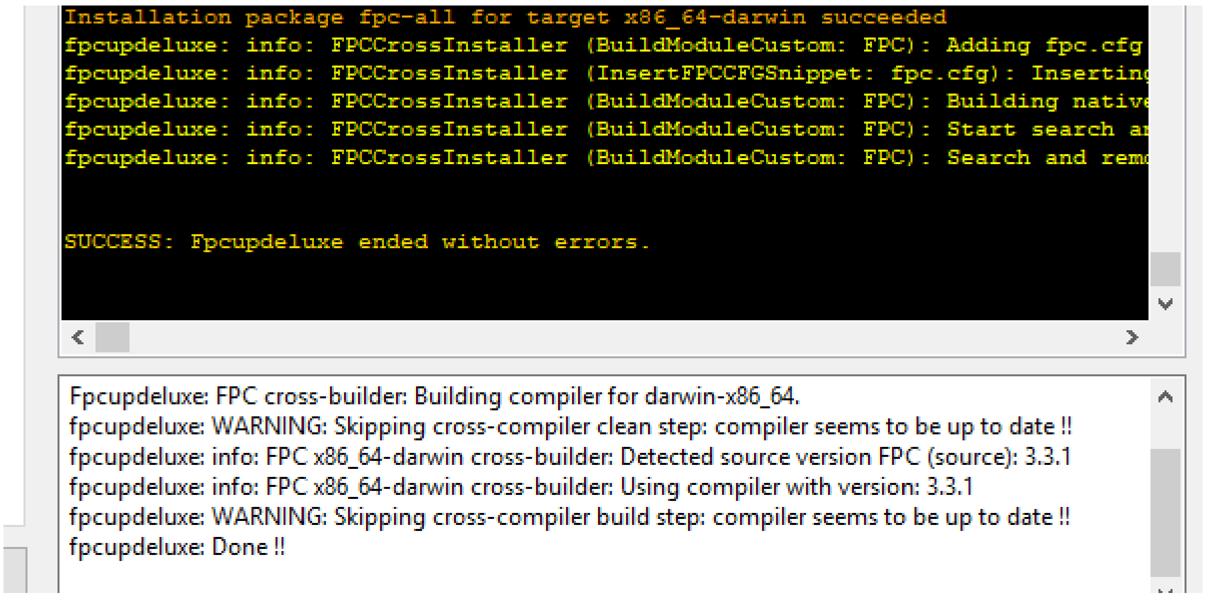


Gus downloader !

File: C:\laz_test\tmp\WinCrossBinsAppleAll.zip

29,25 MB of 62,08 MB

Then we patiently wait for building end.



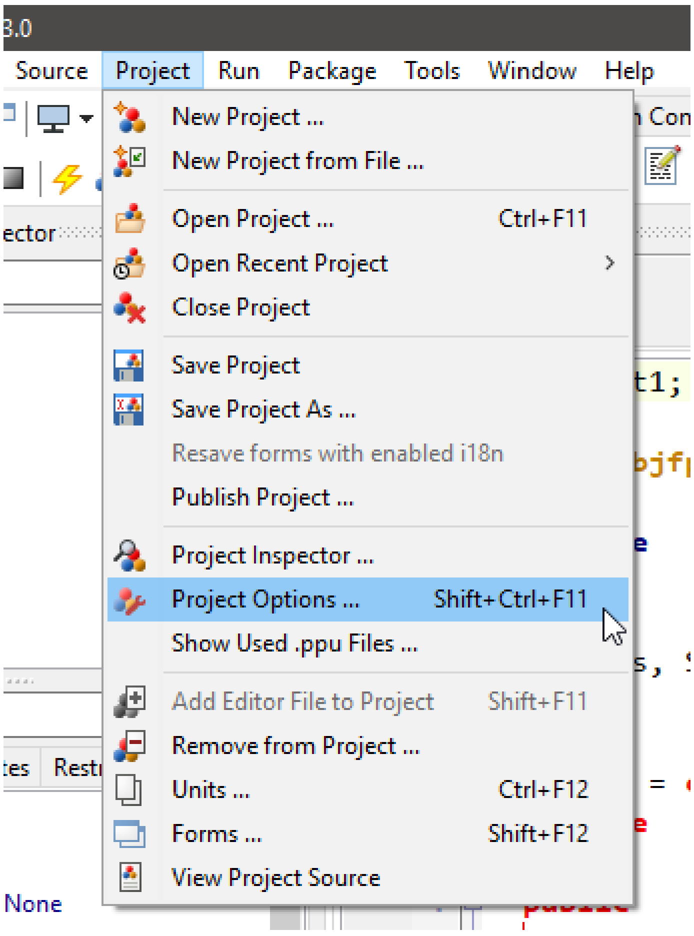
```
Installation package fpc-all for target x86_64-darwin succeeded
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Adding fpc.cfg
fpcupdeluxe: info: FPCCrossInstaller (InsertFPCCFGSnippet: fpc.cfg): Insertin
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Building nativ
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Start search a
fpcupdeluxe: info: FPCCrossInstaller (BuildModuleCustom: FPC): Search and rem


SUCCESS: Fpcupdeluxe ended without errors.
```

```
Fpcupdeluxe: FPC cross-builder: Building compiler for darwin-x86_64.
fpcupdeluxe: WARNING: Skipping cross-compiler clean step: compiler seems to be up to date !!
fpcupdeluxe: info: FPC x86_64-darwin cross-builder: Detected source version FPC (source): 3.3.1
fpcupdeluxe: info: FPC x86_64-darwin cross-builder: Using compiler with version: 3.3.1
fpcupdeluxe: WARNING: Skipping cross-compiler build step: compiler seems to be up to date !!
fpcupdeluxe: Done !!
```

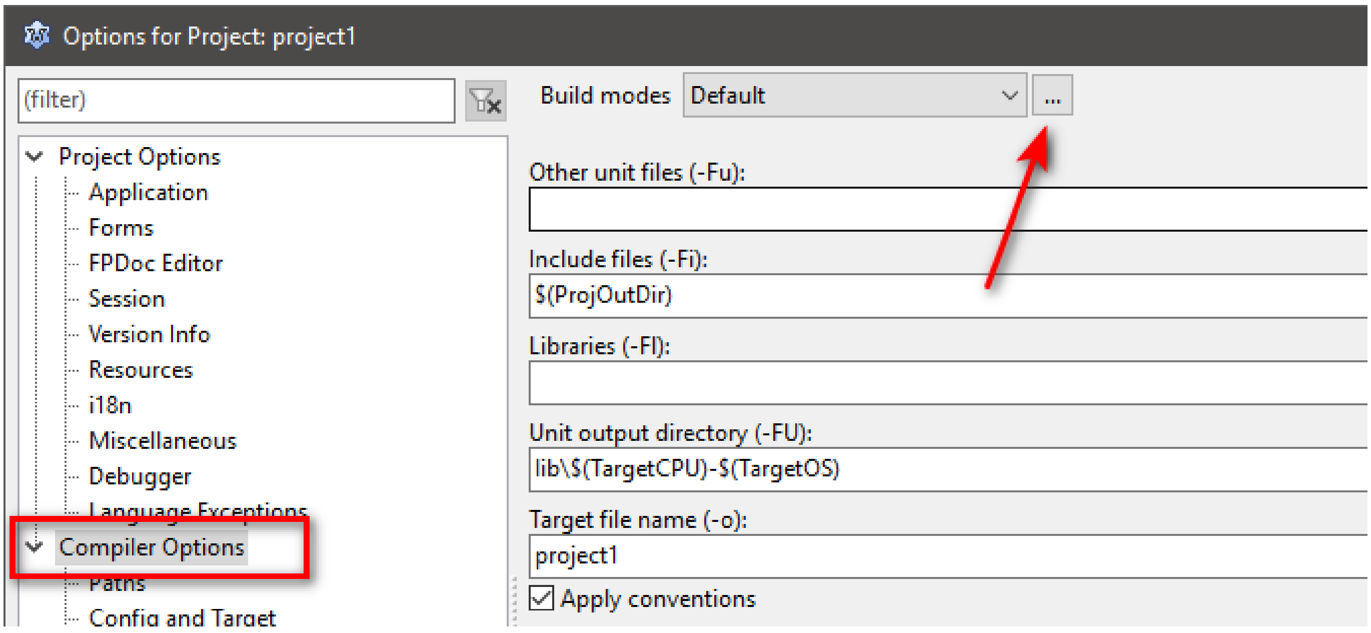
*Note*: how to make the build mode of the executable file under x86-64, see next chapter.

## Build modes for cross compiler

To build executable files for different platforms using the cross compiler(s), you need to configure the appropriate build modes in your project. To do this, open your project by Lazarus and open the project properties



In the tree that opens, find "Compiler options" item and proceed to setting the "Build modes" by clicking on the button with the ellipsis


Options for Project: project1

Build modes: Default

Other unit files (-Fu):

Include files (-Fi): $(ProjOutDir)

Libraries (-Fl):

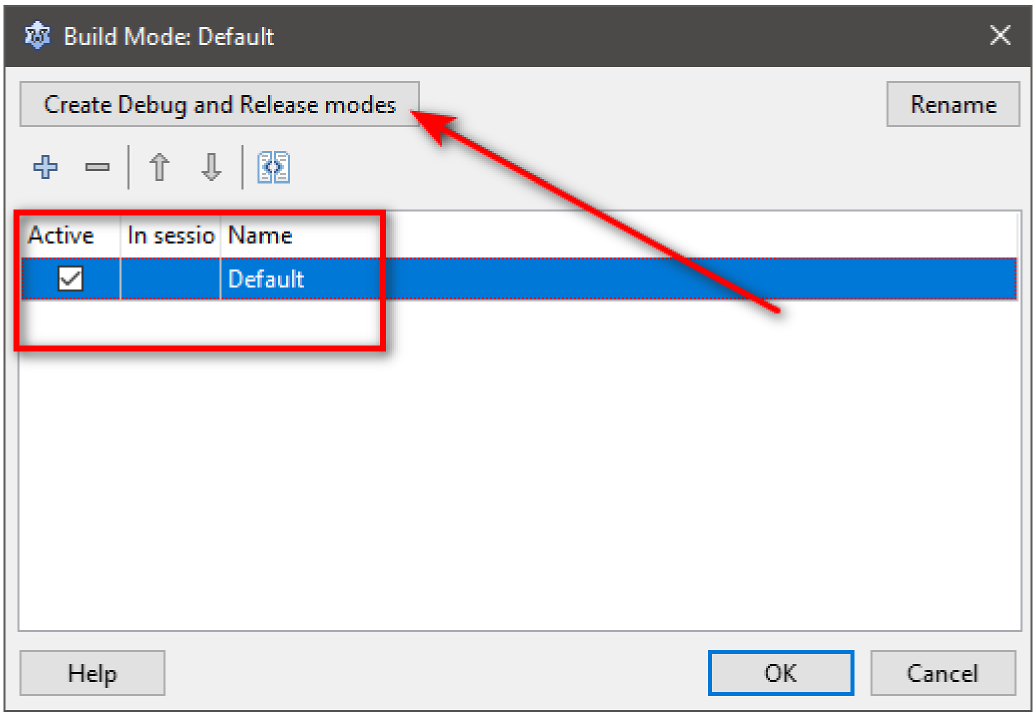
Unit output directory (-FU): lib\$(TargetCPU)-$(TargetOS)

Target file name (-o): project1

☑ Apply conventions

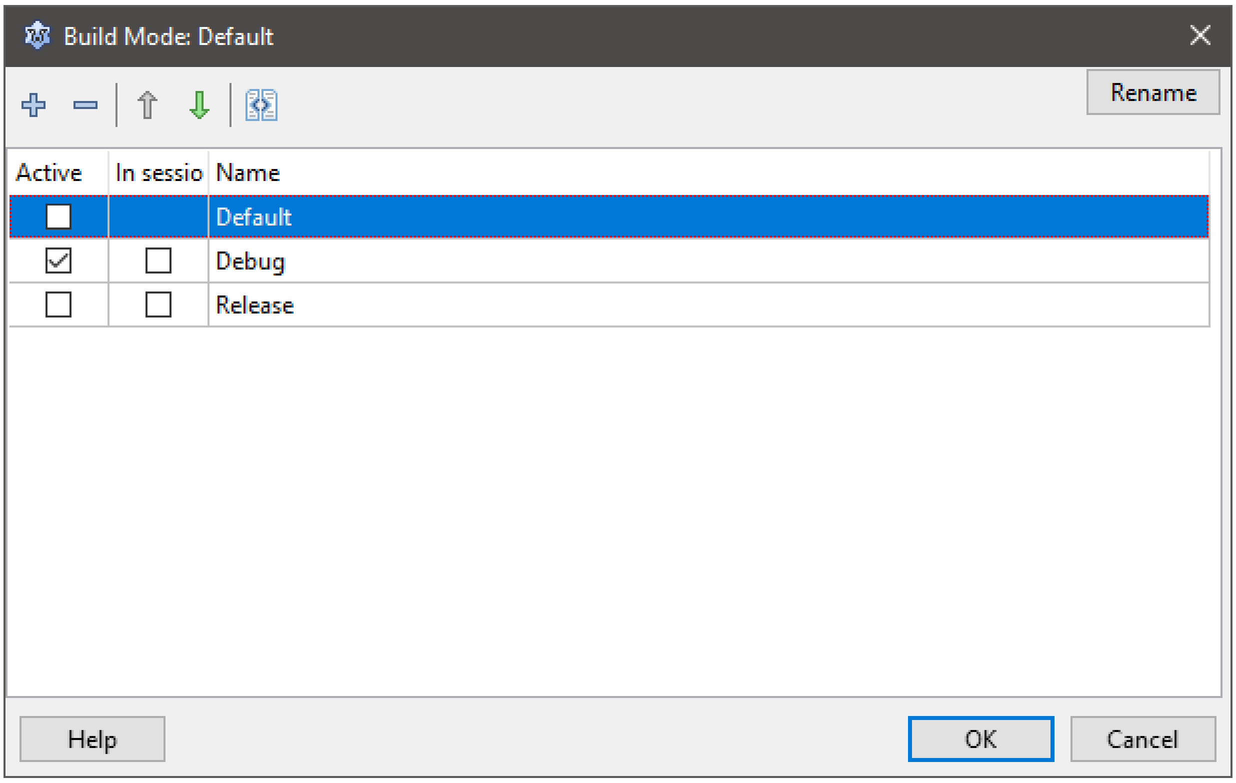
By default, Lazarus has one build mode ("Default"), which builds an executable file with debug information.

*In this case, the executable file in the "Default" mode will correspond to the bitness of the compiler (as we remember from the previous chapter, the bitness of the compiler will correspond to the bitness of fpcupdeluxe, with which it was builded).*


Build Mode: Default

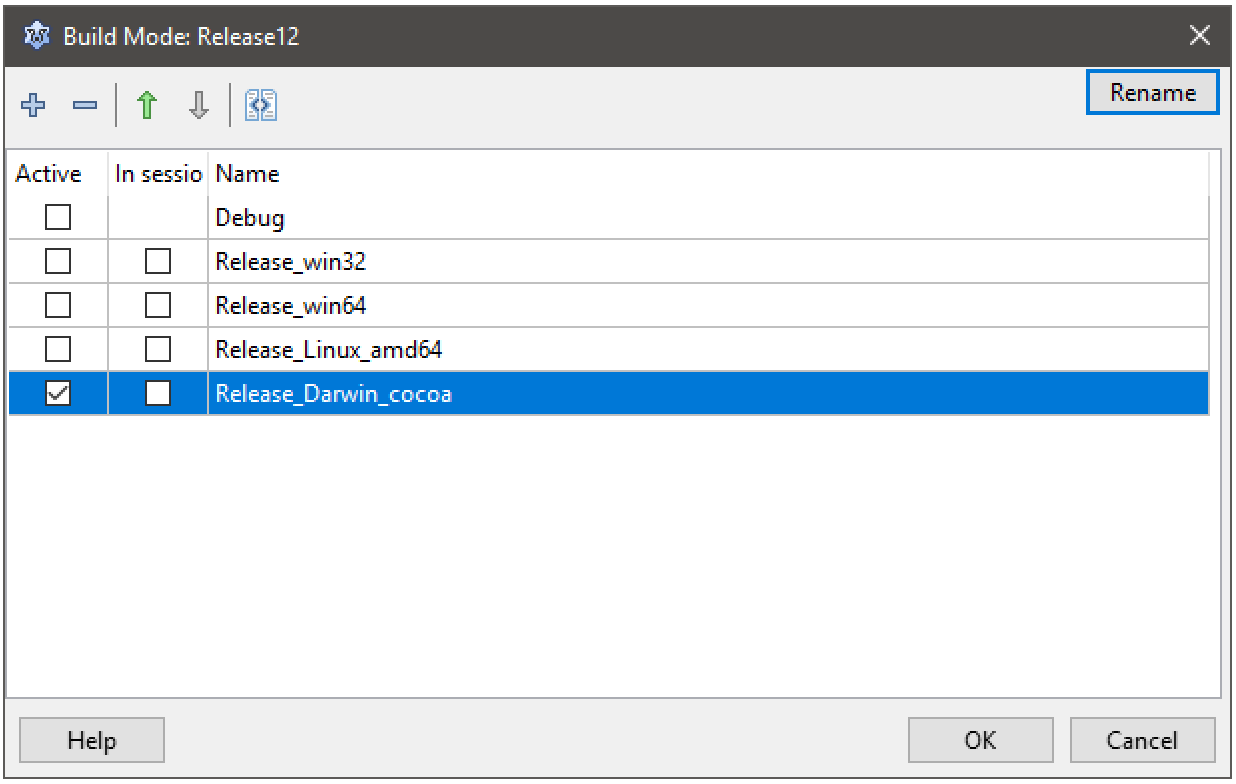
| Active | In sessio | Name |
|---|---|---|
| ✓ | | Default |

Click on the "Create Debug and Release modes" button to create separate debug and release modes.


Build Mode: Default

| Active | In sessio | Name |
|---|---|---|
| ☐ | | Default |
| ☑ | ☐ | Debug |
| ☐ | ☐ | Release |

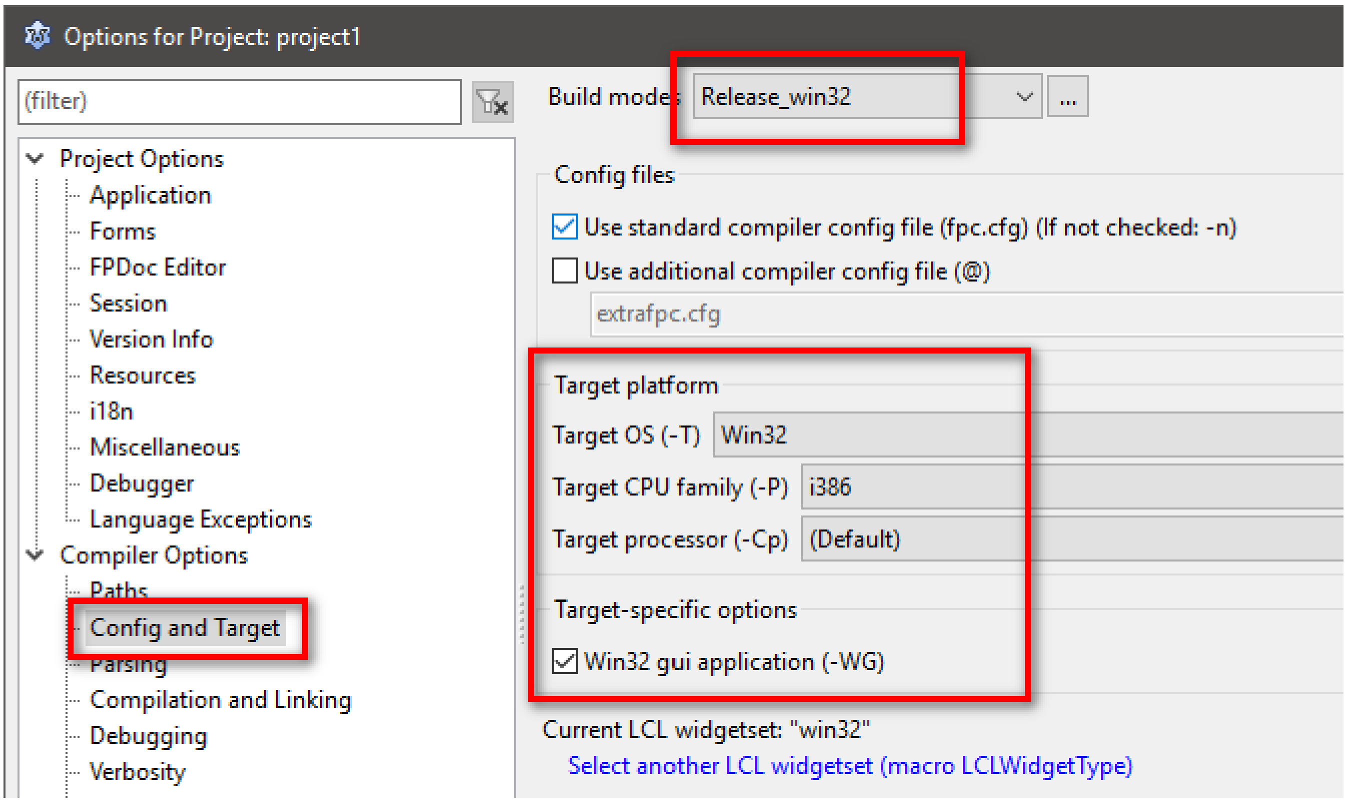
Now you can delete the "Default" mode and add a couple of more release modes for our needs, remembering to rename them accordingly


Build Mode: Release12

| Active | In sessio | Name |
|---|---|---|
| ☐ | | Debug |
| ☐ | ☐ | Release_win32 |
| ☐ | ☐ | Release_win64 |
| ☐ | ☐ | Release_Linux_amd64 |
| ☑ | ☐ | Release_Darwin_cocoa |

Now close the window and set up your modes.

Go to the settings tree to the "Config and Target" item


Options for Project: project1

Build modes: Release_win32

Config files

☑ Use standard compiler config file (fpc.cfg) (If not checked: -n)

☐ Use additional compiler config file (@)

extrafpc.cfg

Target platform

Target OS (-T): Win32

Target CPU family (-P): i386

Target processor (-Cp): (Default)

Target-specific options

☑ Win32 gui application (-WG)

Current LCL widgetset: "win32"

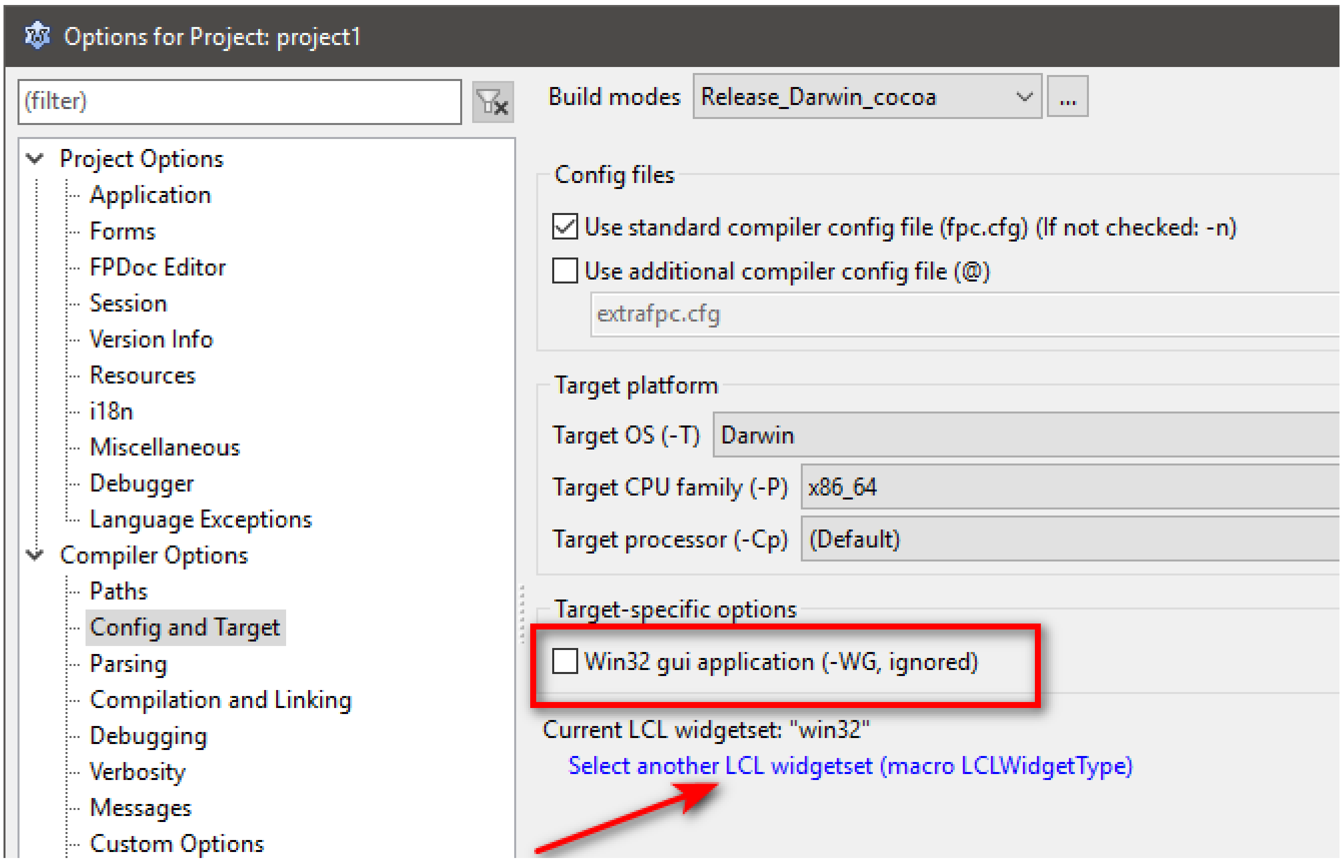
Select another LCL widgetset (macro LCLWidgetType)

Sequentially selecting the specified build modes in the "Build modes" drop-down list, set the appropriate settings for each of them:

| Mode | OS | CPU | Note |
|---|---|---|---|
| Debug | | | do not change the default settings! (if only you are sure of your actions) |
| Release_win32 | win32 | i386 | Don't uncheck the "Win32 GUI application" checkbox! |
| Release_win64 | win64 | x86_64 | Don't uncheck the "Win32 GUI application" checkbox! |
| Release_linux_i386 | Linux | i386 | Don't uncheck the "Win32 GUI application" checkbox! |
| Release_linux_amd64 | Linux | x86_64 | Don't uncheck the "Win32 GUI application" checkbox! |
| Release_Darwin_Cocoa | Darwin | x86_64 | Be sure to uncheck "Win32 GUI application"! See note |

*Note*: since each OS uses different widgetsets to display controls on the form(s), then for "my" MacOS (I wrote that I use Darwin with Cocoa widgets in the previous chapter) additional settings.

It is necessary to uncheck "Win32 GUI application"


Options for Project: project1

(filter)

Build modes: Release_Darwin_cocoa

- Project Options
  - Application
  - Forms
  - FPDoc Editor
  - Session
  - Version Info
  - Resources
  - i18n
  - Miscellaneous
  - Debugger
  - Language Exceptions
- Compiler Options
  - Paths
  - Config and Target
  - Parsing
  - Compilation and Linking
  - Debugging
  - Verbosity
  - Messages
  - Custom Options

Config files

[✓] Use standard compiler config file (fpc.cfg) (If not checked: -n)

[ ] Use additional compiler config file (@)

extrafpc.cfg

Target platform

| | |
|---|---|
| Target OS (-T) | Darwin |
| Target CPU family (-P) | x86_64 |
| Target processor (-Cp) | (Default) |

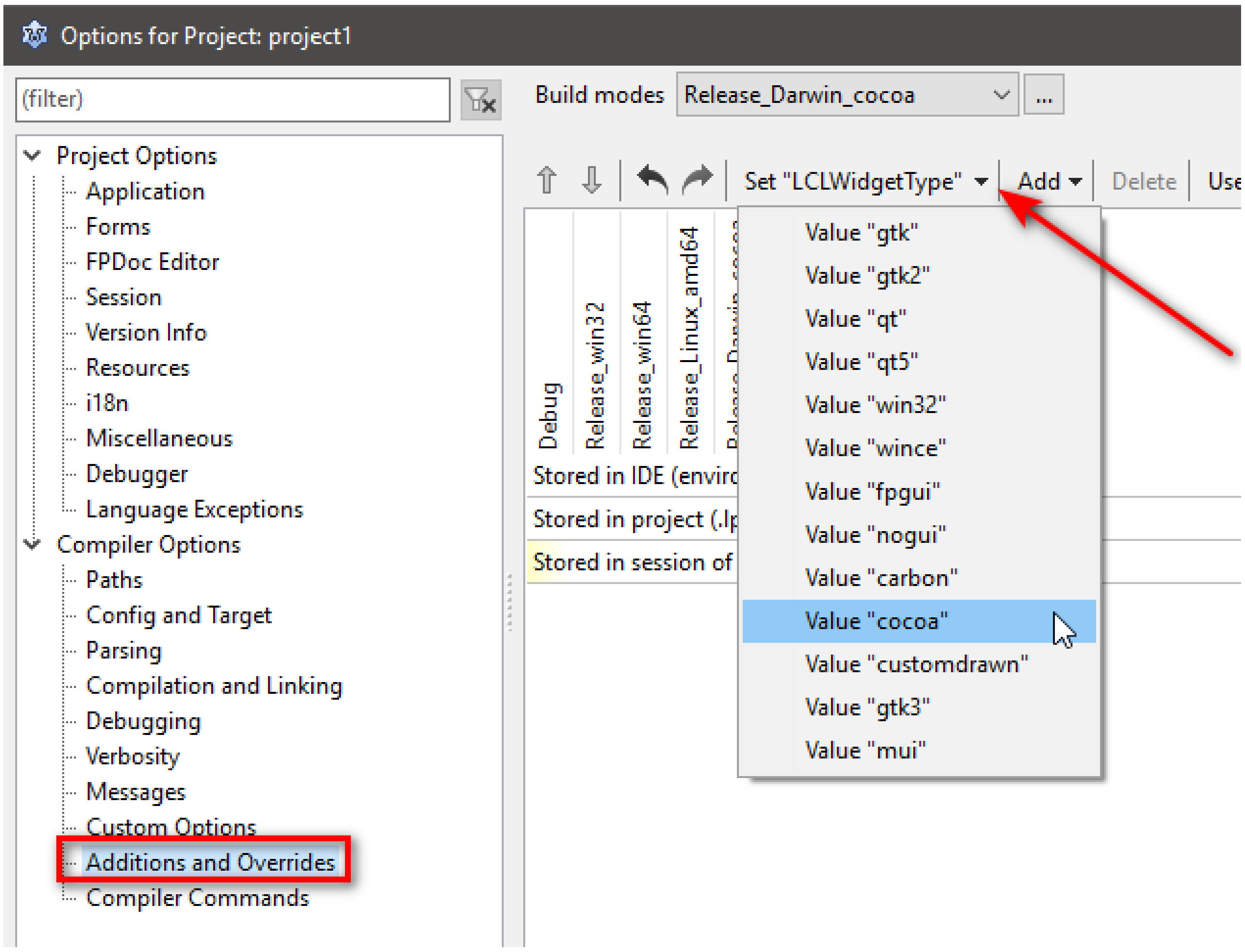
Target-specific options

[ ] Win32 gui application (-WG, ignored)

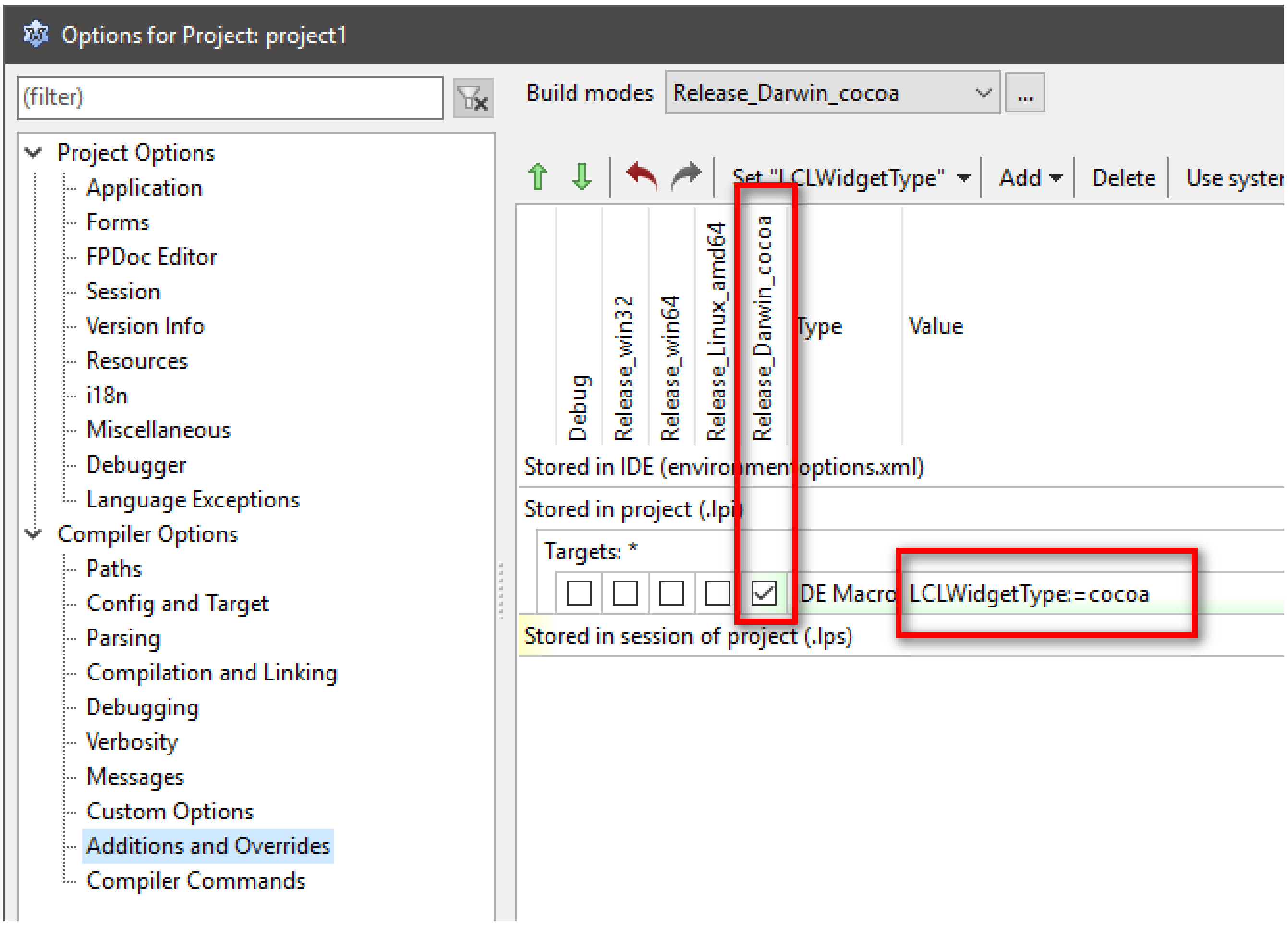
Current LCL widgetset: "win32"

Select another LCL widgetset (macro LCLWidgetType)

and click to link "Select another widgetset" (or go to "Additions and Overrides" item in the project tree)



Select the "Value "Cocoa" item in the "SetLCLWidgetType" drop-down list, and make sure that there is a check mark next to this build mode.



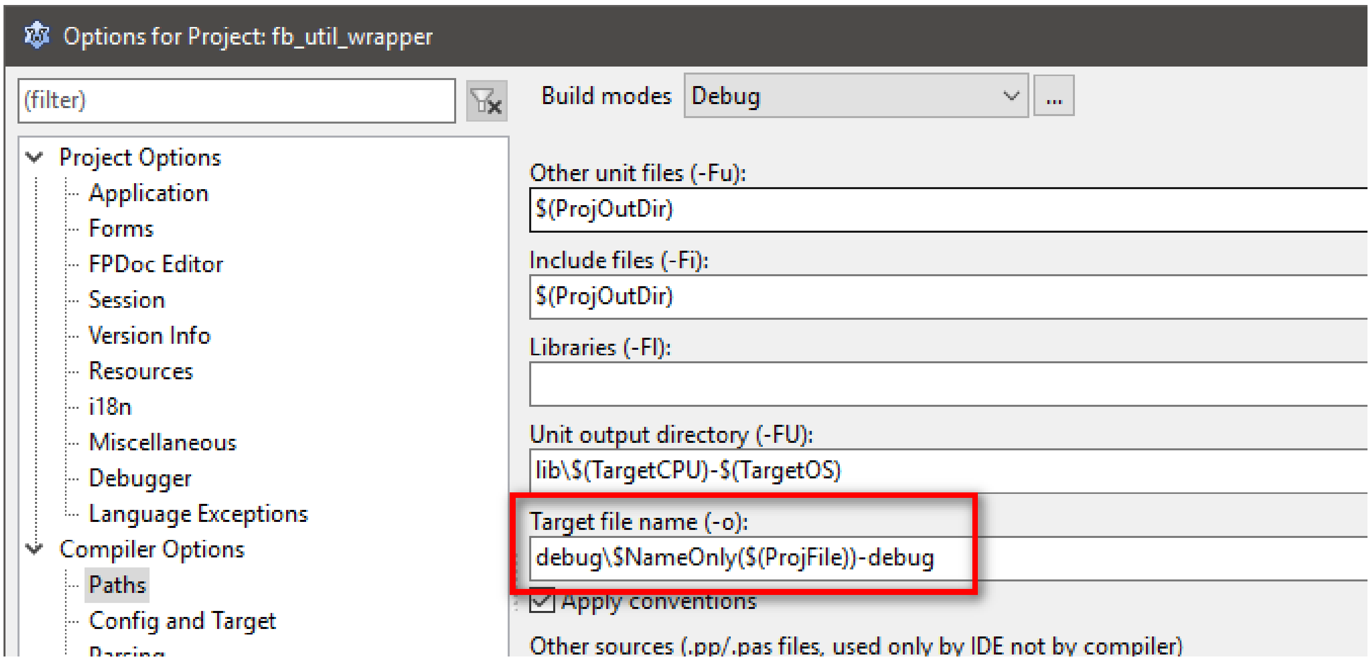
Build modes: Release_Darwin_cocoa

| Debug | Release_win32 | Release_win64 | Release_Linux_amd64 | Release_Darwin_cocoa | Type | Value |
|---|---|---|---|---|---|---|
| Stored in IDE (environmentoptions.xml) | | | | | | |
| Stored in project (.lpi) | | | | | | |
| Targets: * | | | | | | |
| ☐ | ☐ | ☐ | ☐ | ☑ | IDE Macro | LCLWidgetType:=cocoa |
| Stored in session of project (.lps) | | | | | | |

and close the dialog.

If you start building a project in different modes, then all executable files will be compiled under the same name and in the same folder and sometimes will differ only in extension. To avoid this, I make a few more changes to the project properties using IDE macros.

For debug mode, I specify the value for the output file (the "Target file name" field): `debug\$NameOnly($(ProjFile))-debug`


Options for Project: fb_util_wrapper

Build modes: Debug

Other unit files (-Fu): $(ProjOutDir)

Include files (-Fi): $(ProjOutDir)

Libraries (-Fl):

Unit output directory (-FU): lib\$(TargetCPU)-$(TargetOS)

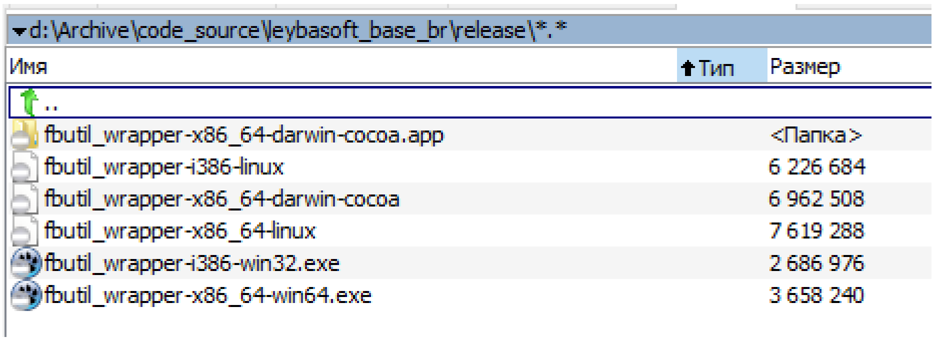
Target file name (-o): debug\$NameOnly($(ProjFile))-debug

☑ Apply conventions

Other sources (.pp/.pas files, used only by IDE not by compiler)

and for "release" assemblies, I set the value:
`release\$NameOnly($(ProjFile))-$(TargetCPU)-$(TargetOS)`

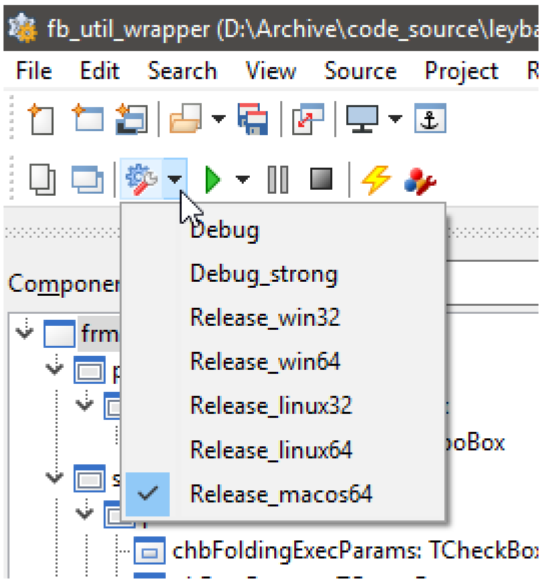
Let's say my project file is named "fbutil_wrapper.lpi". Then the finished files get the following names

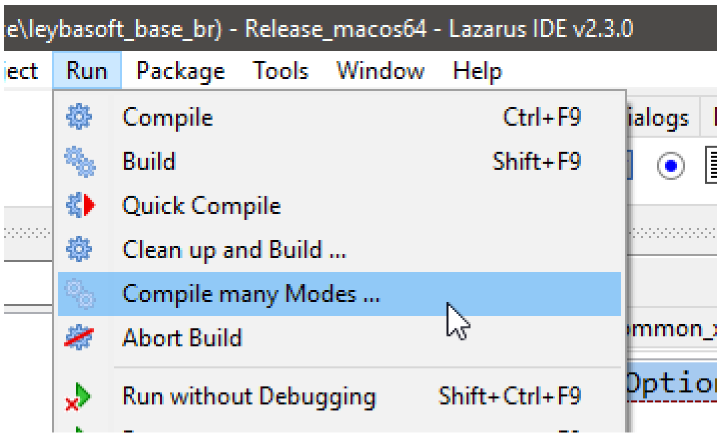

d:\Archive\code_source\leybasoft_base_br\release\*.*

| Имя | Тип | Размер |
|---|---|---|
| .. | | |
| fbutil_wrapper-x86_64-darwin-cocoa.app | | <Папка> |
| fbutil_wrapper-i386-linux | | 6 226 684 |
| fbutil_wrapper-x86_64-darwin-cocoa | | 6 962 508 |
| fbutil_wrapper-x86_64-linux | | 7 619 288 |
| fbutil_wrapper-i386-win32.exe | | 2 686 976 |
| fbutil_wrapper-x86_64-win64.exe | | 3 658 240 |

## Building process of applications via a crosscompiler

To quickly select the build mode, you can use the toolbar or the corresponding menu item. You can build files individually or build all in one pass


fb_util_wrapper (D:\Archive\code_source\leyba
File Edit Search View Source Project R
Debug
Debug_strong
Release_win32
Release_win64
Release_linux32
Release_linux64
✓ Release_macos64
chbFoldingExecParams: TCheckBo


e\leybasoft_base_br) - Release_macos64 - Lazarus IDE v2.3.0
ect Run Package Tools Window Help
Compile Ctrl+F9
Build Shift+F9
Quick Compile
Clean up and Build ...
Compile many Modes ...
Abort Build
Run without Debugging Shift+Ctrl+F9

**An important point(!)**: if the building process is for another platform, then you cannot build/compile the project using `<F9>` - you will get an error about the impossibility to run the executable file. It is necessary to build/compile an executable file using (if by default) `<Shift>+<F9>` / `<Ctrl>+<F9>`.