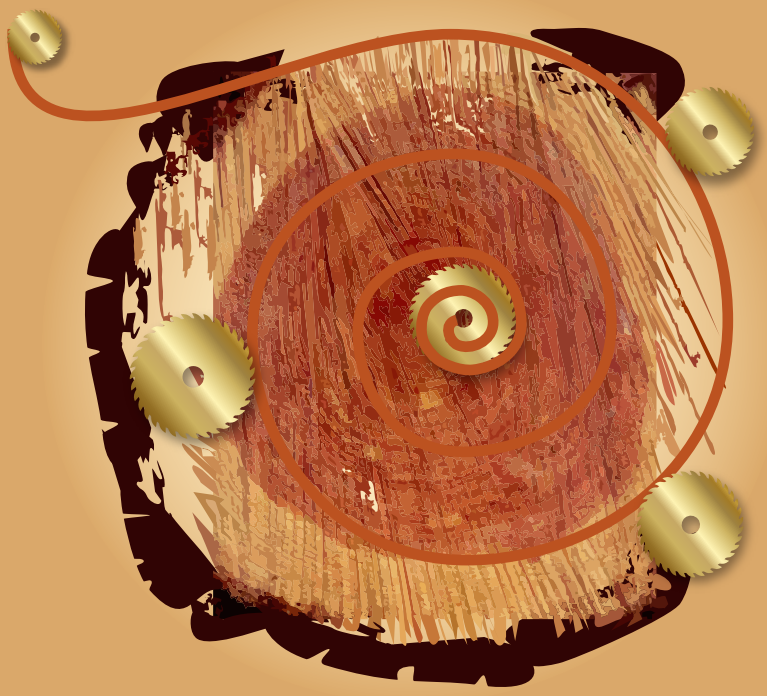
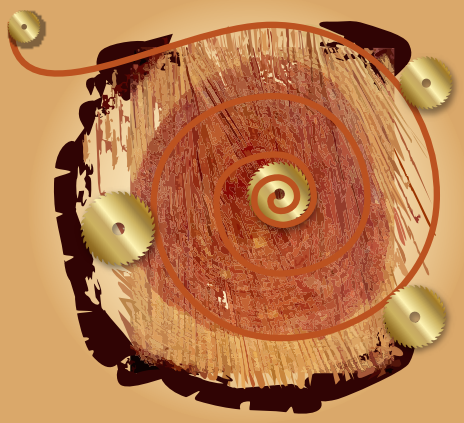
BLAISE PASCAL MAGAZINE 97

Multi platform /Object Pascal / Internet / JavaScript / WebAssembly / Pas2Js / Databases
CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux



CONTENT

ARTICLES



ADVERTISERS

Niklaus Wirth

Pascal is an imperative and procedural programming language, which Niklaus Wirth designed (left below) in 1968–69 and published in 1970, as a small, efficient language intended to encourage good programming practices using structured programming and data structuring. A derivative known as Object Pascal designed for object-oriented programming was developed in 1985. The language name was chosen to honour the Mathematician, Inventor of the first calculator: Blaise Pascal (see top right).

Publisher: PRO PASCAL FOUNDATION in collaboration © Stichting Ondersteuning Programmeertaal Pascal - Netherlands

Stephen Ball
http://delphiaball.co.uk
@DelphiABall

Peter Bijlsma -Editor
peter @ blaisepascal.eu

Dmitry Boyarintsev
dmitry.living @ gmail.com

Michaël Van Canneyt,
michael @ freepascal.org

Marco Cantù
www.marcocantu.com
marco.cantu @ gmail.com

David Dirkse
www.davdata.nl
E-mail: David @ davdata.nl

Benno Evers
b.evers @ everscustomtechnology.nl

Bruno Fierens
www.tmssoftware.com
bruno.fierens @ tmssoftware.com

Holger Flick
holger @ flixments.com

Primož Gabrijelčič
primoz @ gabrijelcic.org

Mattias Gärtner
nc-gaertnma@netcologne.de

Peter Johnson
http://delphidabbler.com
delphidabbler @ gmail.com

Max Kleiner
www.softwareschule.ch
max @ kleiner.com

John Kuiper
john_kuiper @ kpnmail.nl

Wagner R. Landgraf
wagner @ tmssoftware.com

Vsevolod Leonov
vsevolod.leonov@mail.ru

Andrea Magni www.andreamagni.eu
andrea.magni @ gmail.com
www.andreamagni.eu/wp

Paul Nauta PLM Solution Architect
CyberNautics
paul.nauta @ cybernautics.nl

Kim Madsen
www.component4developers.com

Boian Mitov
mitov @ mitov.com

Jeremy North
jeremy.north @ gmail.com

Detlef Overbeek - Editor in Chief
www.blaisepascal.eu
editor @ blaisepascal.eu

Howard Page Clark
hdpc @ talktalk.net

Heiko Rompel
info @ rompelsoft.de

Wim Van Ingen Schenau -Editor
wisone @ xs4all.nl

Peter van der Sman
sman @ prisman.nl

Rik Smit
rik @ blaisepascal.eu

Bob Swart
www.eBob42.com
Bob @ eBob42.com

B.J. Rao
contact @ intricad.com

Daniele Teti
www.danieleteti.it
d.teti @ bittime.it

Anton Vogelaar
ajv @ vogelaar-electronics.com

Danny Wind
dwind @ delphicompany.nl

Siegfried Zuhr
siegfried @ zuhr.nl

Editor - in - chief
Detlef D. Overbeek, Netherlands Tel.: Mobile: +31 (0)6 21.23.62.68
News and Press Releases email only to **editor@blaisepascal.eu**

Editors
Peter Bijlsma, W. (Wim) van Ingen Schenau, Rik Smit

Correctors
Howard Page-Clark, Peter Bijlsma

Trademarks All trademarks used are acknowledged as the property of their respective owners.

Caveat Whilst we endeavour to ensure that what is published in the magazine is correct, we cannot accept responsibility for any errors or omissions. If you notice something which may be incorrect, please contact the Editor and we will publish a correction where relevant.

Subscriptions (2019 prices)

	Internat. excl. VAT	Internat. incl. 9% VAT	Shipment
Printed Issue ±60 pages	€ 155,96	€ 250	€ 80,00
Electronic Download Issue 60 pages	€ 64,20	€ 70	———
Printed Issue inside Holland (Netherlands) 60 pages	———	€ 250,00	€ 70,00

Subscriptions can be taken out online at **www.blaisepascal.eu** or by written order, or by sending an email to **office@blaisepascal.eu**
Subscriptions can start at any date. All issues published in the calendar year of the subscription will be sent as well.
Subscriptions run 365 days. Subscriptions will not be prolonged without notice. Receipt of payment will be sent by email.
Subscriptions can be paid by sending the payment to:
ABN AMRO Bank Account no. 44 19 60 863 or by credit card or Paypal
Name: Pro Pascal Foundation-Foundation for Supporting the Pascal Programming Language (Stichting Ondersteuning Programeertaal Pascal)
IBAN: NL82 ABNA 0441960863 BIC ABNANL2A VAT no.: 81 42 54 147 (Stichting Programmeertaal Pascal)
Subscription department
Edelstenenbaan 21 / 3402 XA IJsselstein, The Netherlands
Mobile: + 31 (0) 6 21.23.62.68 **office@blaisepascal.eu**

Member and donator of WIKIPEDIA
Member of the **Royal Dutch Library**
KB

From your editor

These are hard times.
I lost a friend and the best Corrector I had.
He was a great lover of dogs, and we could do long chats about philosophical issues.
We could disagree and laugh and chat again.
I convinced him to eat black garlic, a fermentation and he became very fond of that.
I miss him.

There are even more sad stories:
Chad Hower (IntraWeb) was falsely accused of kidnapping his son.
That's not all. He got ill: the kidneys.
Now he's dependant on help.
Please help spreading his story so politicians will make it clear to the FBI they need to stop their false accusations, especially because Judges have told them to do so.
He needs to go to hospital and the FBI does not allow that…
So lets make some noise, so he can be treated…
watch the video for the hole story.

This is the end of an other summer: in September the Autumn officially starts.
We still have an other three months before we can publish our 100th issue.
So we plan to create something special about that time: November.

I must be very careful saying we want to have real event,
where we can meet each other again.
Hopefully we can do that.
But we sure will do something like a big online event, together with Barnsten.
I 'd love to make a festival out of that.
With lots of special ideas and plans for the future.
Let's have some fun again.

In this issue Michael van Canneyt wrote an article about Git.
That was very much necessary, because lots of developers don't use any kind of version control.
Because it is necessary for Delphi and Pascal to use it, he will write an other extra article about this.

In September Delphi will be upgraded to a new version.
Lets see what comes. 8th of September it should officially be presented.
I heard a few things that might be very interesting.
I am not allowed to tell you.
So you'll have to be patient.

From our Technical advisor: Cartoons from Jerry King

"Just how powerful of a fan did you put in your computer?"

Peter Bijlsma passed away.

Because of Cancer.
Not by age:
only 75 years old.
He was a very good friend
you could disagree with
which enlivened our discussions.

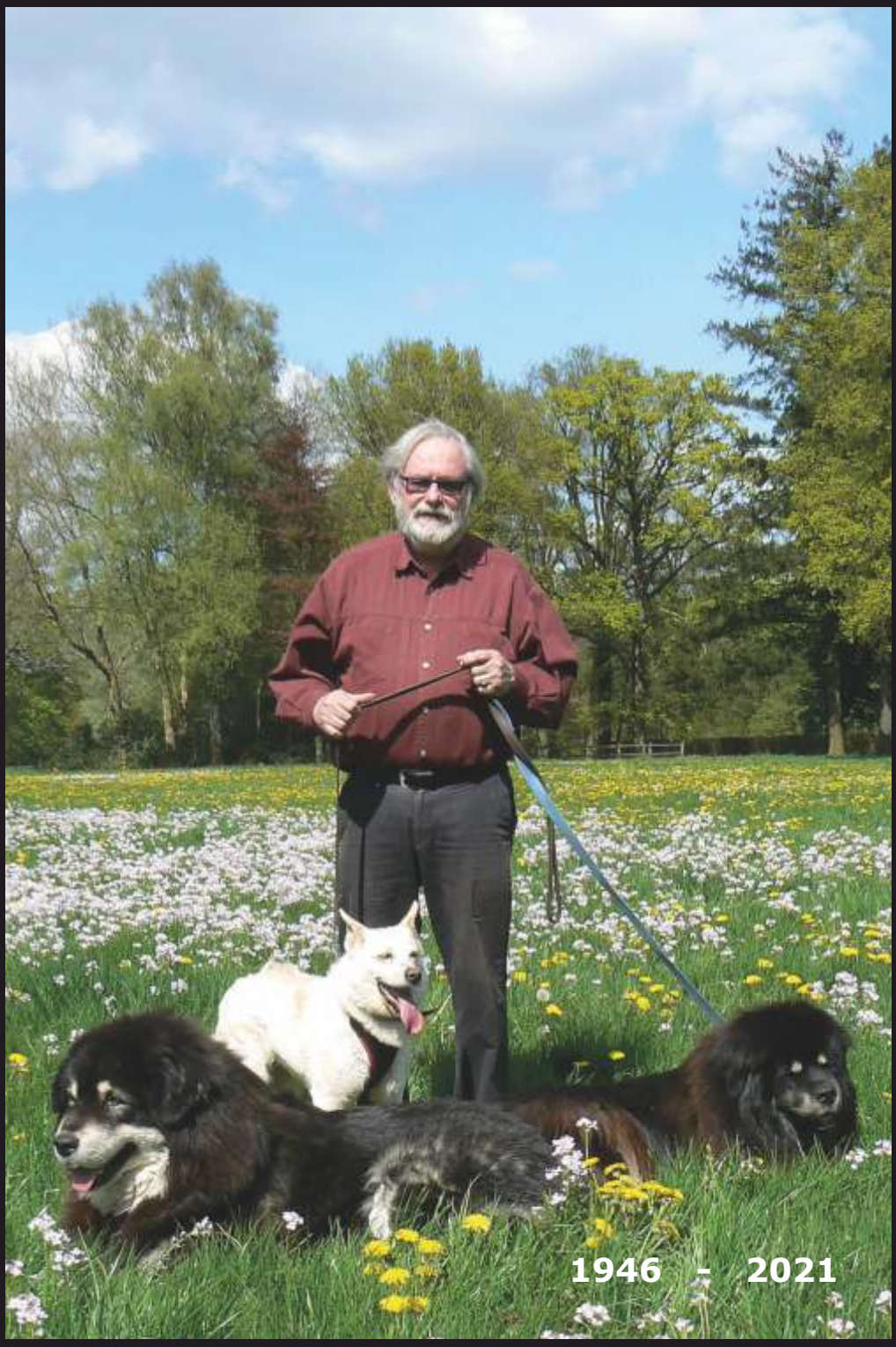
What we never discussed:
he was Corrector for BPM
and I took him very seriously about that.
Painfully precise.

He loved life with his Ria
and his dogs:
Tibetan Mastives.
Incredibly beautiful dogs.

Peter was a social and very helpful person.
He was human.

Peter Bijlsma
was born in the Netherlands in 1946. He studied electronics at university and took a course in Algol. In 1982 he became the proud owner of an Apple II computer, and he programmed in Basic and 6502 assembly language. Learned other languages (Pascal, Perl) by self-study.
He worked as a Technical Communication specialist, writing technical manuals and acting as an instructor for military command & control systems.

Peter Bijlsma †



Be yourself; Everyone else is already taken. — Oscar Wilde.

maXbox Starter86_2

In the last Article we have seen that P4D is a set of free components that wrap up the Python DLL into Delphi and Lazarus (FPC). For the next section I want to show more practical implementations. Let's start with P4D in Delphi:

First create a new Form

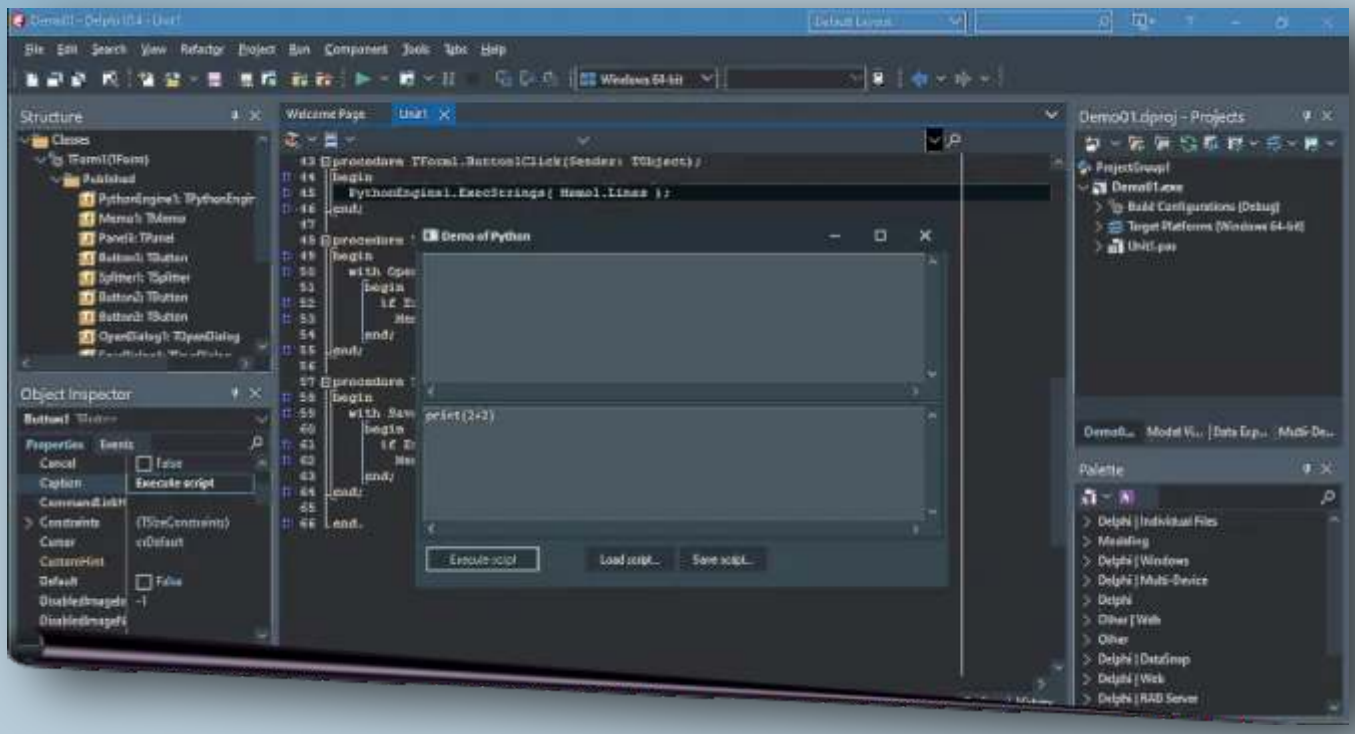
- Drop a `TMemo` (or a TRichEdit)
- Drop a `TPythonGUIInputOutput` for displaying Python's results
- Drop a `Tmemo` for source code
- Drop a `TPythonEngine`

- Connect the attribute IO of the `TPythonEngine` to `TPythonGUIInputOutput`.
- Connect attribute Output of `TPythonGUIInputOutput` to `TRichEdit`.

- Drop a TButton and call it "Execute script"
- Double-click on the button and add: `PythonEngine1.ExecStrings(Memo1.Lines);`

That's almost all! Compile and execute.

Write in the Memo1: print(2+3)
Click on the Execute button
You should see in the Output as Memo2 window: 5



As we can see the memo-control manifests the Python-script as input in memo1 and output in memo2:

```
object Memo1: TMemo
  ...
  Font.Pitch = fpVariable
  Font.Style = []
  Lines.Strings = (
       'print(2+3)')
  ParentFont = False
  ScrollBars = ssBoth
  TabOrder = 1
end

object PythonGUIInputOutput1: TpythonGUIInputOutput

  UnicodeIO = True
  RawOutput = False
  Output = Memo2
  Left = 64
end
```

So in a more complicated script we do have a same memo-control but simply with more lines:

```
Lines.Strings = (
  'import sys'
  'print ("Version:", sys.version)'
  'import spam'
  'print (spam.foo('#39'hello world'#39', 1))'
  'p = spam.CreatePoint( 10, 25 )'
  'print ("Point:", p)'
  'p.x = 58'
  'print (p.x, p)'
  'p.OffsetBy( 5, 5 )'
  'print (p)'
  'print ("Current value of var test is: ", test)'
  'test.Value = "New value set by Python"'
  'print (spam.getdouble())'
  'print (spam.getdouble2())')
ParentFont = False
```

You do also have the evaluation of an expression. But the evaluation of an expression works only for arithmetic expressions and not for instructions ! The use of variables and functions is of course possible but constructs like for, def, catch, class, print, import... are not implemented, you use for this `ExecStrings()` and not `EvalStrings()`.

USING DELPHI METHODS AS PYTHON FUNCTIONS

What would be if we use in a internal Python-script some Delphi-methods like in the above script methods of the import module spam? First we had to initialize the module spam, we just need to add our new methods:

(See next page)

```
procedure TForm1.PythonModule1Initialization(Sender: TObject);
begin
 with Sender as TPythonModule do
  begin
   AddDelphiMethod( 'foo',
            spam_foo,
            'foo' );
   AddDelphiMethod( 'CreatePoint',
            spam_CreatePoint,
            'function CreatePoint'+LF+
            'Args: x, y'+LF+
            'Result: a new Point object' );
   AddDelphiMethod( 'getdouble',
            spam_getdouble,
            'getdouble' );
   AddDelphiMethod( 'getdouble2',
            spam_getdouble2,
            'getdouble2' );
  end;
end;
```

And here's the example of functions defined for the module spam in this context the function spam_foo with forms caption return:

```
function TForm1.spam_foo(pself, args : PPyObject): PPyObject; cdecl;
begin
 with GetPythonEngine do
  begin
   ShowMessage( 'args of foo: '+PyObjectAsString(args) );
   ShowMessage( 'Form''s caption = ' + Caption );
   Result := ReturnNone;
  end;
end;
```

Handshaking with Python arrays or tuples layout does have some complications. Normal Python arrays (as for standard CPython) are normally called “Lists”. A `numpy.array` type (or a mutable list) in Python is a special type that is more memory and layout efficient than a normal Python list of normal Py floating point objects.

If you want to use Delphi and access `Numpy.array` or list, I really suppose that the straightest way to do it would be to implement a way to export some simple straight C functions that access the `Numpy.array` type.
Numpy.array wraps a standard block of memory that is accessed as a native C array type. This in turn, does NOT map cleanly to Delphi array types as created by a Delphi method to Python.

Let me go deeper in that point, converting a Delphi-array or list to for example a list goes in the end with a dll-function from the Python library ('PyList_SetItem'):

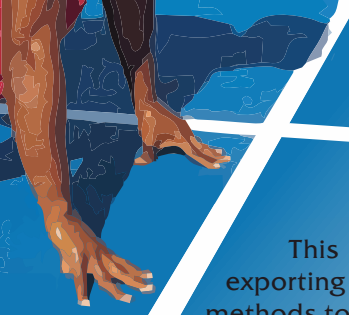
```
function TPythonEngine.ArrayToPyList(const items: array of const) : PPyObject;
var
 i : Integer;
begin
 Result := PyList_New( High(items)+1 );
 if not Assigned(Result) then
  raise EPythonError.Create('Could not create a new list object');
 for i := Low(items) to High(items) do
  PyList_SetItem( Result, i, VarRecAsPyObject( items[i] ) );
end;

PyList_SetItem:function (dp:PPyObject;idx:NativeInt;item:PPyObject):integer;
cdecl;

PyList_SetItem:= Import('PyList_SetItem');
```

The other way round, as I said we can't map cleanly Python lists to Delphi array types, we get the data sort of as the base type strings from **`PyObjectAsString`**:

```
procedure TPythonEngine.PyListToStrings(list: PPyObject;
strings: TStrings );
var
 i : Integer;
begin
 if not PyList_Check(list) then
  raise EPythonError.Create('the python object is not a list');
 strings.Clear;
 for i:= 0 to PyList_Size( list )- 1 do
  strings.Add( PyObjectAsString( PyList_GetItem( list, i ) ) );
end;
```

I think the common base type in Delphi (to export) is the array and the common base type in Python (to import) is the list. So this we can see as a proof of concept code:

```
function PythonToDelphi(obj : PPyObject ) : TPyObject;
begin
 if IsDelphiObject( obj ) then
  Result := TPyObject(PAnsiChar(obj)+Sizeof(PyObject))
 else
  raise EPythonError.CreateFmt( 'Python object "%s" is not a Delphi class',
                  [GetPythonEngine.PyObjectAsString(obj)] );
end;
```



This
exporting of Delphi-
methods to use in Python-scripts
works also with the creation of a dll as Demo09
Making a Python module as a dll explains (I'll show that in the Tutor III).

The Demo for the AddDelphiMethod concept you find at:
https://github.com/maxkleiner/python4delphi/blob/master/Demos/Demo07/test.py
http://py4d.pbworks.com/w/page/9174535/Wrapping%20Delphi%20Objects

More or less some external files as normal Python-scripts is also on your way. For example we call the script test.py and we import explicit the module spam, previously generated in Delphi:

```
import sys
print "Win version:", sys.winver
import spam
print (spam.foo('hello world', 1))
p = spam.CreatePoint( 10, 25 )
print ("Point:", p)
p.x = 58
print (p.x, p)
p.OffsetBy( 5, 5 )
print (p)
print ("Current value of var test is: ", test)
test.Value = "New value set by Python"
print (spam.getdouble())
```

BUILD YOUR ENVIRONMENT:

On Win, the standard Python installer already associates the .py extension with a file type (Python.File) and gives that file type an open command that runs the interpreter (**F:\Program Files\Python\python.exe "%1" %***). This is enough to make scripts executable from the command prompt. We use the python-dll as we use a windows dll. Therefore ***.pyd** files are dll's, but there are a few differences:
So far you have to know 3 different file types:

1. ***.py**:
   The norm input source code that we've written.
2. ***.pyc**:
   The compiled bytecode. If you import a module, py will build a *.pyc file that contains bytecode to make importing it again later easier and faster.
3. ***.pyd**:
   The mentioned Windows dll file from Python.

If you have a DLL named foo.pyd, then it must have a function `PyInit_foo()`. You can then write Python "import foo", and Python will search for foo.pyd (as well as `foo.py`, `foo.pyc`) and if it finds it, will attempt to call `PyInit_foo()` to initialize it. Of course you do not link your .exe with `foo.lib`, as that would cause Windows to require the DLL to be present, we load it dynamically.

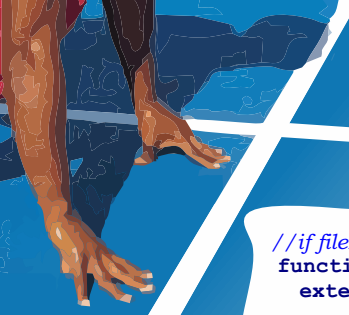
First we check our Python installation. Python provides for all user and current user installations. All user installations place the Py dll in the Windows System directory and write registry info to `HKEY_LOCAL_MACHINE`.
Current user installations place the dll in the install path and the registry info in `HKEY_CURRENT_USER` version `< py 3.5`.
So, for current user installations we need to try and find the install path since it may not be on the system path.

```
{$IFDEF MSWINDOWS}
function IsPythonVersionRegistered(PythonVersion : string;
 out InstallPath: string; out AllUserInstall: Boolean) : Boolean;
 // The above convention was changed in Python 3.5.  Now even for all user
 // installations the dll is located at the InstallPath.
 // Also from vers.3.5 onwards 32 bit version have a suffix -32 e.g. "3.6-32"
 // See also PEP 514

var
 key: string;
 VersionSuffix: string;
 MajorVersion : integer;
 MinorVersion : integer;
begin
 Result := False;
 InstallPath := '';
 AllUserInstall := False;
 MajorVersion := StrToInt(PythonVersion[1]);
 MinorVersion := StrToInt(PythonVersion[3]);
 VersionSuffix := '';
{$IFDEF CPUX86}
 if (MajorVersion > 3) or ((MajorVersion = 3)  and (MinorVersion >= 5)) then
  VersionSuffix := '-32';
{$ENDIF}
 key:= Format('\Software\Python\PythonCore\%s%s\InstallPath',
                        [PythonVersion, VersionSuffix]);
```

```
// First try HKEY_CURRENT_USER as per PEP514
try
 with TRegistry.Create1(KEY_READ and not KEY_NOTIFY) do
  try
   RootKey := HKEY_CURRENT_USER;
   if OpenKey(Key, False) then begin
    InstallPath := ReadString('');
    Result := True;
    Exit;
   end;
  finally
   Free;
  end;
except
 writeln(' HKEY_CURRENT_USER except');
end;

//Then try for an all user installation
try
 with TRegistry.Create1(KEY_READ and not KEY_NOTIFY) do
  try
   RootKey := HKEY_LOCAL_MACHINE;
   if OpenKey(Key, False) then begin
    AllUserInstall := True;
    if (MajorVersion > 3) or ((MajorVersion = 3)
                  and (MinorVersion >= 5)) then
     InstallPath := ReadString('');
    Result := True;
   end;
  finally
   Free;
  end;
 except
  writeln(' HKEY__LOCAL_MACHINE except');
 end;
end;
{$ENDIF}
```

In my case the path is on:
`C:\Users\max\AppData\Local\Programs\Python\Python36\Lib\`

Then we can simple check a first function or load on runtime the PyRun_SimpleString for our next example:



```
//if fileExiststst(PYDLLPATH+ 'python37.dll';
 function getCopyRight: PChar;
   external 'Py_GetCopyright@C:\maXbox\EKON25\python37.dll stdcall';

function pyrun(command : pchar):integer;
   external 'PyRun_SimpleString@C:\maXbox\EKON25\python37.dll cdecl';

procedure pyinit;
   external 'Py_Initialize@C:\maXbox\EKON25\python37.dll cdecl';
procedure pyexit(retval: integer);
   external 'Py_Exit@C:\maXbox\EKON24\python37.dll cdecl';
```

Now we use to invoke a Python script as an embedding const and use the dll functionality of Import('PyRun_SimpleString'); To run python code direct in a maXbox, Free Pascal or whatever script you need to import just the 3 dll functions, above all PyRun_SimpleStringFlags or without flags:

```
Const PYDLLPATH = 'C:\maXbox\EKON25\';
   PYDLLNAME = 'python37.dll';
   PSCRIPTNAME = 'initpy.py';
```

This is a simplified interface to PyRun_**SimpleString** leaving the **PyCompilerFlags*** argument set to NULL. Normally the Python interpreter is initialized by **Py_Initialize()** so we use the same interpreter as from a shell or terminal:

```
int PyRun_SimpleString(const char *command)
 //function pyrun(command :pChar) :integer;
 //writeln('pyinitback: '+itoa
pyinit();
//retp:= 'print("hello low")'
retp:= 'print()';
//PyRun_SimpleString:   function( str: PAnsiChar): Integer; cdecl;
//writeln(itoa(pyrun(retp)));
writeln(itoa(pyrun('print("this is box")')));
writeln(itoa(pyrun('import sys')));
writeln(itoa(pyrun('f=open(r"C:\maXbox\maxbox4\pytest.txt","w")')));
writeln(itoa(pyrun('f.write("Hello PyWorld_, \n")')));
writeln(itoa(pyrun('f.write("Data will be written on the file.")')));
writeln(itoa(pyrun('f.close()')));
```

You do also have helper functions in the unit PythonEngine.pas as Global Subroutines to test the environment:

- GetPythonEngine (Returns the global TPythonEngine)
- PythonOK
- PythonToDelphi
- IsDelphiObject
- PyObjectDestructor
- FreeSubtypeInst
- PyType_HasFeature

```
function GetPythonEngine : TPythonEngine;
function PythonOK : Boolean;
function PythonToDelphi( obj : PPyObject ) : TPyObject;
function IsDelphiObject( obj : PPyObject ) : Boolean;
procedure PyObjectDestructor( pSelf : PPyObject); cdecl;
procedure FreeSubtypeInst(ob:PPyObject); cdecl;
procedure Register;
function PyType_HasFeature(AType : PPyTypeObject; AFlag : Integer): Boolean;
function SysVersionFromDLLName(const DLLFileName : string): string;
procedure PythonVersionFromDLLName(LibName: string; out MajorVersion,
                                   MinorVersion: integer);
```

For example the PythonOK:

```
function PythonOK : Boolean;
begin
 Result := Assigned( gPythonEngine ) and
      (gPythonEngine.Initialized or gPythonEngine.Finalizing);
end;
```

To run python code integrated in a maXbox, Free Pascal, GNU Pascal or whatever script you need to import just the 3 dll functions, above all PyRun_SimpleStringFlags or without flags:

```
Const PYDLLPATH = 'C:\maXbox\EKON25\decimals';
   PYDLLNAME = 'python37.dll';
   PSCRIPTNAME = 'initpy.py';
```

This is a simplified interface to `PyRun_SimpleString` leaving the `PyCompilerFlags*` argument set to NULL. Normally the Python inter-preter is initialized by `Py_Initialize()` so we use the samefrom a shell, command or terminal.

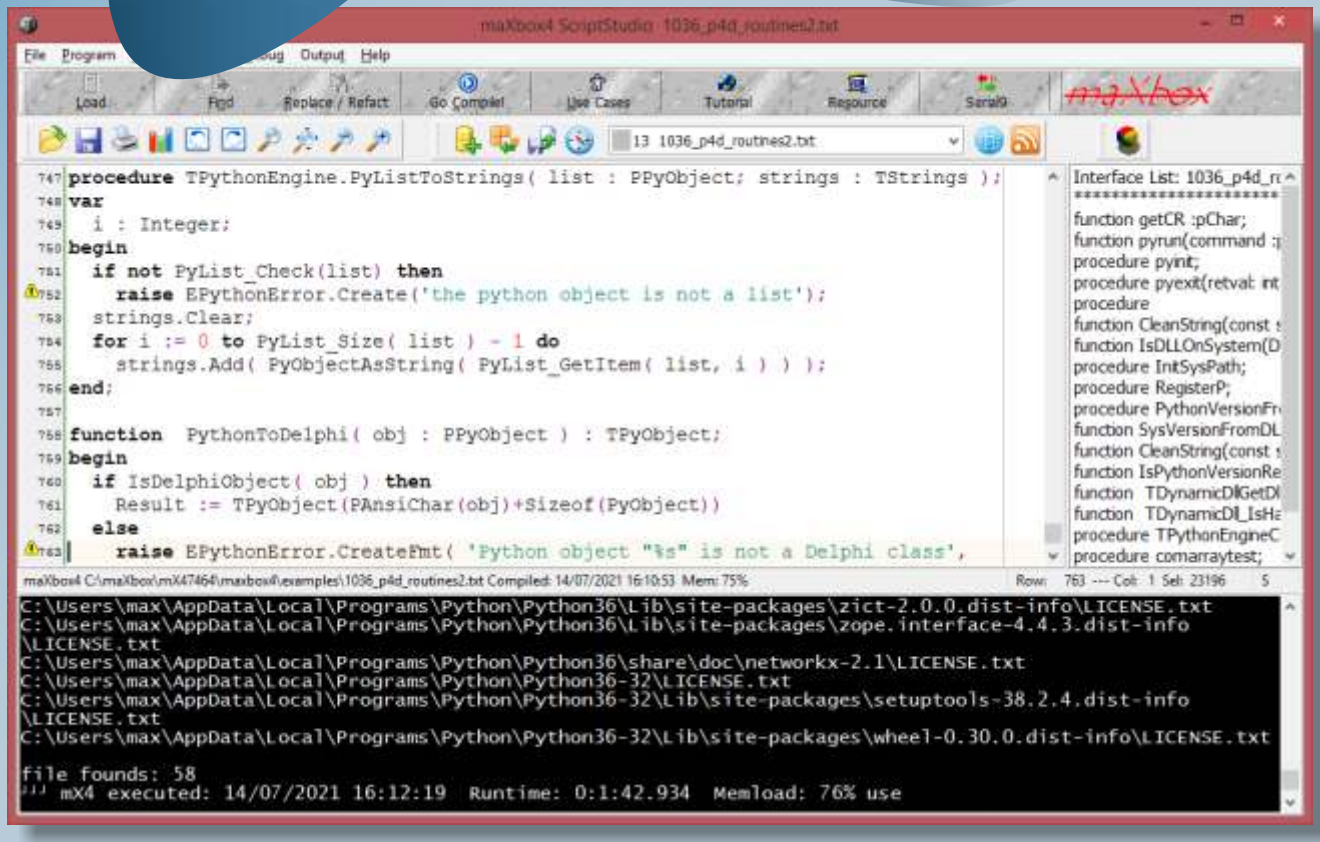
In P4D you do invoke the mentioned memo with ExeStrings:

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
  PythonEngine1.ExecStrings( Memo1.Lines );
end;
```

This explains best the code behind,
to evaluate, run or execute an internal Python expression.
This is also possible in maXbox, So eval expects an expression, import is a statement.
That said, what you can trying is the following combination:

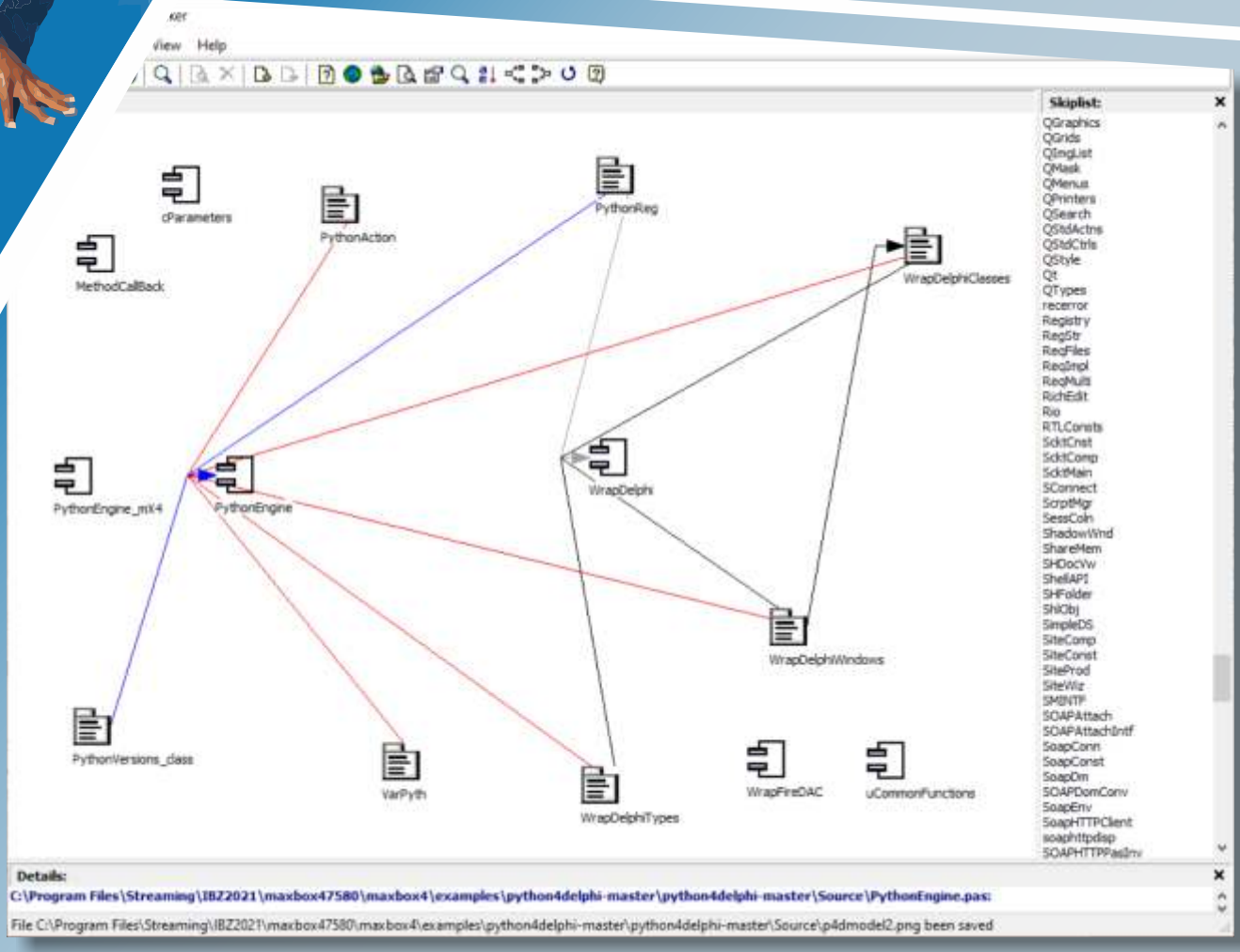
```
Println('exec as eval: '+eng.EvalStr('exec("import os as o")'));
Println('exec: '+eng.EvalStr('o.getcwd()'));
>>> exec as eval: None
>>> exec: C:\maXbox\mX47580\maxbox4
writeln('uuid: '+eng.evalstr('exec("import uuid") or
                              str(uuid.uuid4())'));
>>> uuid: 3b2e10f9-0e31-4961-9246-00852fd508bd
```

See the demo:
http://www.softwareschule.ch/examples/pydemo.txt



The unit PythonEngine.pas is the main core-unit of the framework.
Most of the Python/C API is presented as published/public member functions of the engine unit and a clever Dll loader/mapper.

```
  ...
  Py_BuildValue           := Import('Py_BuildValue');
  Py_Initialize           := Import('Py_Initialize');
  PyRun_String            := Import('PyRun_String');
  PyRun_SimpleString      := Import('PyRun_SimpleString');
  PyDict_GetItemString    := Import('PyDict_GetItemString');
  PySys_SetArgv           := Import('PySys_SetArgv');
  Py_Exit                 := Import('Py_Exit');
  ...
```


View Help
cParameters
PythonAction
PythonReg
MethodCallBack
WrapDelphiClasses
PythonEngine_mX4
PythonEngine
WrapDelphi
WrapDelphiWindows
PythonVersions_class
VarPyth
WrapDelphiTypes
WrapFireDAC
uCommonFunctions
Skiplist:
QGraphics
QGrids
QImgList
QMask
QMenus
QPrinters
QSearch
QStdActns
QStdCtrls
QStyle
Qt
QTypes
recerror
Registry
RegStr
RegFiles
RegImpl
RegMulti
RichEdit
Rio
RTLConsts
SckltCnst
SckltComp
SckltMain
SConnect
ScrptMgr
SessColn
ShadowWnd
ShareMem
SHDocVw
ShellAPI
SHFolder
ShlObj
SimpleDS
SiteComp
SiteConst
SiteProd
SiteWiz
SMBNTP
SOAPAttach
SOAPAttachIntf
SoapConn
SoapConst
SoapDm
SOAPDomConv
SoapEnv
SoapHTTPClient
soaphttpdisp
SOAPHTTPPasInv
Details:
C:\Program Files\Streaming\IBZ2021\maxbox47580\maxbox4\examples\python4delphi-master\python4delphi-master\Source\PythonEngine.pas:
File C:\Program Files\Streaming\IBZ2021\maxbox47580\maxbox4\examples\python4delphi-master\python4delphi-master\Source\p4dmodel2.png been saved


SESSION
Python4Delphi
Max Kleiner | kleiner kommunikation
EKON 25
Die Konferenz für Delphi & More | 8. – 10. November 2021 | Düsseldorf

WIKI & EKON P4D TOPICS

- https://entwickler-konferenz.de/delphi-innovations-fundamentals/python4delphi/
- http://www.softwareschule.ch/examples/weatherbox.txt
- https://learndelphi.org/python-native-windows-gui-with-delphi-vcl/

LEARN ABOUT PYTHON FOR DELPHI

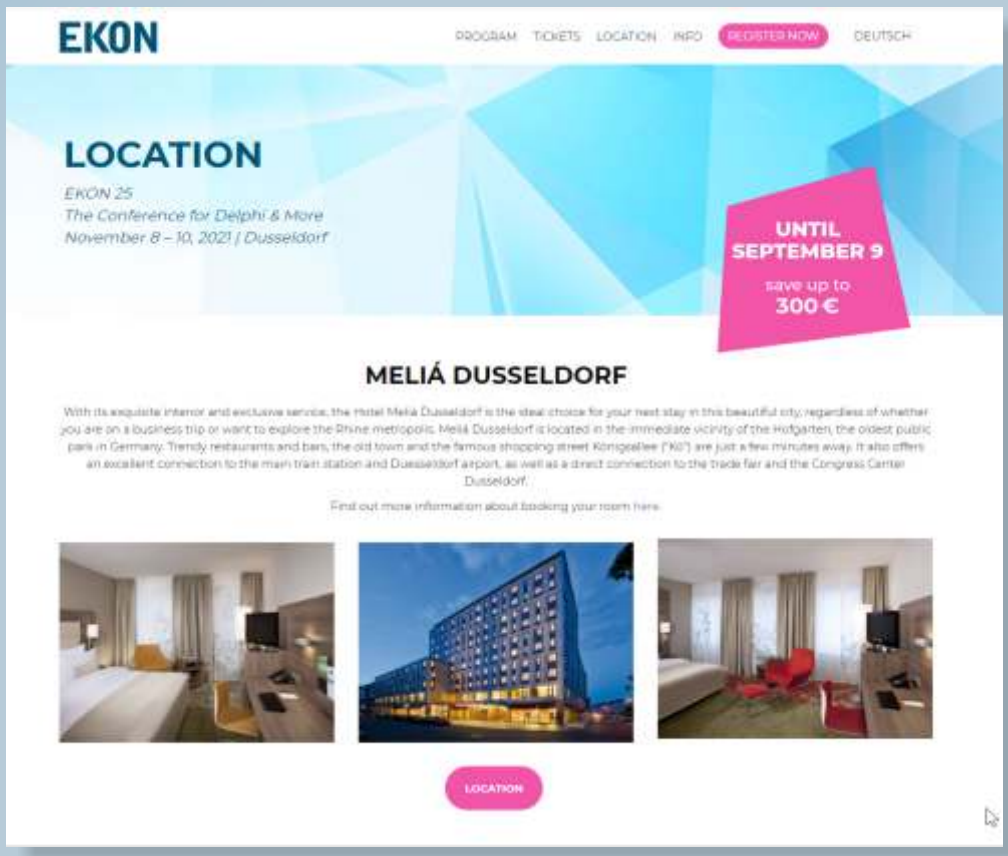
- Tutorials
- Demos https://github.com/maxkleiner/python4delphi

Note: You will need to adjust the demos from github accordingly, to successfully load the Python distribution that you have installed on your computer.

Docs:

https://maxbox4.wordpress.com/blog/
http://www.softwareschule.ch/download/maxbox_starter86.pdf
http://www.softwareschule.ch/download/maxbox_starter86_1.pdf
http://www.softwareschule.ch/download/maxbox_starter86_2.pdf

https://entwickler-konferenz.de/location-en/



Advertisement

Cats Eye Demonstrator

By David Dirkse

Figure 1: The Cats Eye

INTRODUCTION:
A cat's eye indicator also called magic eye, is a fluorescent cathode ray indicator used for radio receiver tuning.
A voltage controls the area of illumination.
This Delphi project paints a cat's eye indicator to be used for progress reporting.

The project is available in **DELPHI** as well **LAZARUS** and all necessary COMPONENTS ARE ALSO MADE AVAILABLE FOR LAZARUS

Figure 2: The Cats Eye indicator

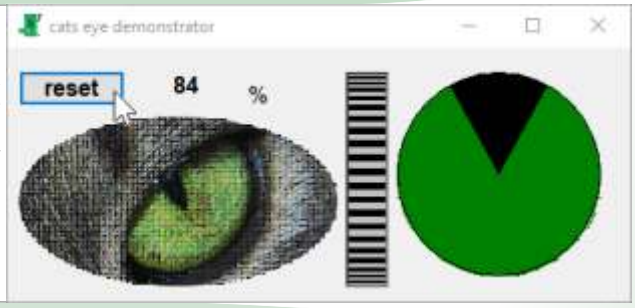
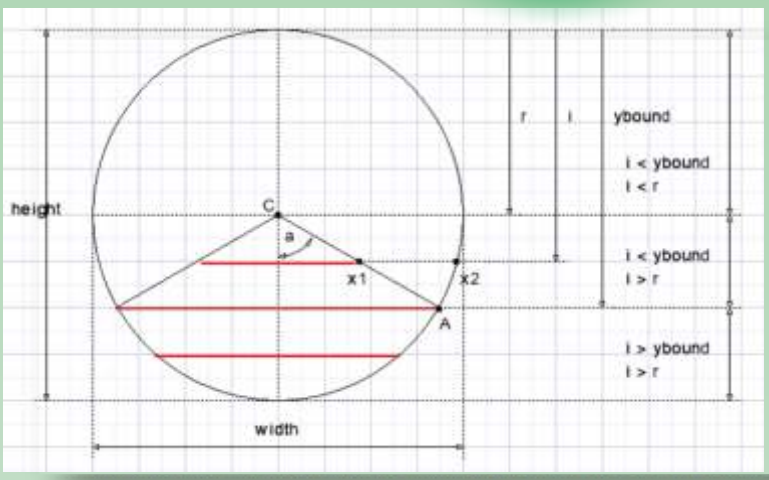


Figure 3: The redesigned result of the application in **DELPHI Sydney 4.2**

Painting is done on a `TImage` component.
A rotation button (home brew component) supplies a number 0..100 as percentage to the cat's eye paint procedure.
The reset button causes the `TImage` to set it's canvas to the form color.
Drawing of the cat's eye does not alter pixels outside the circle.

Figure 4: The application was originally created in **Delphi 7**, now converted to **Delphi Sidney** and **Lazarus**



Variable i steps from height-1 downto 0.
Painting is done by drawing horizontal lines for each i.
Center C has coordinates (0,0) during calculations.

Radius r := height div 2.
Angle a (radians) depends on the percentage v.

```
a  := 0.01*v*pi  {pi = 180 degrees}
ta := tan(a)     the slope of line CA.
```

Figure 5: The painting of the Cats Eye indicator

Vertical coordinate of A is

```
ybound := round(r*(1+cos(a)))
```

x1,x2 are calculated for each i.

```
x2 := round(sqrt(r2 - sqr(i-r))) on circle
x1 := round((i-r)*ta)            on line
```

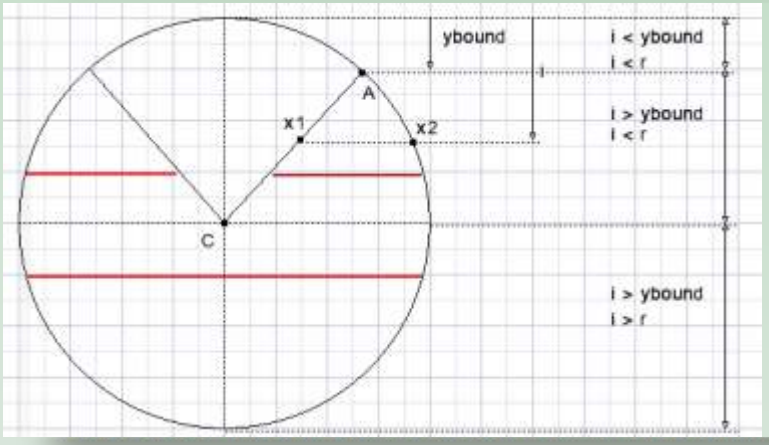


Figure 6: The coordination of the Cats Eye

To distinguish between these cases 2 flags (Boolean variables) are used:

```
BF := i > ybound
RF := i > r
```

Colors `c1,c2` are set to red or white according to `BF, RF`
Then procedure line(`xstart,xend,color`) is called to draw the lines at `y = i`
Also the line procedure adds `r` to the `x` coordinates.

The following cases for line drawing are observed:

```
1.  red : -x2 to x2
2.  white:   -x2 to -x1, red: -x1 to x1, white: x1 to x2
3.  red: -x2 to -x1, white -x1 to x1,red: x1 to x2
4.  white: -x2 to x2
```

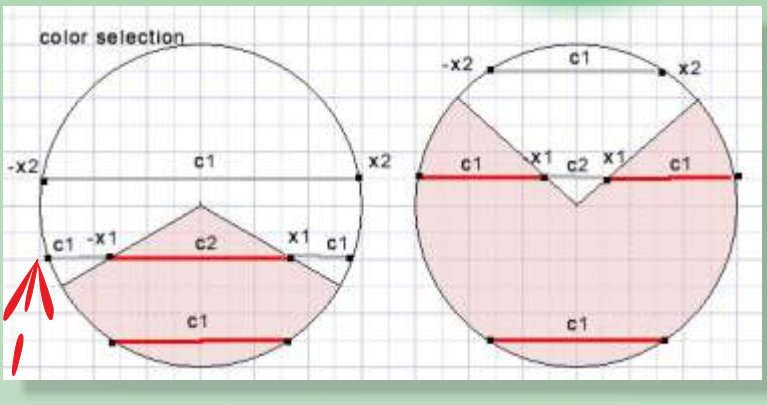


Figure: 7 The color selection overview

Finally the black circle is drawn. Three lines are required if (BF xor RF) is true In the other cases one line is sufficient. Please refer to the source code for more details.

IMAGE DIMENSIONS

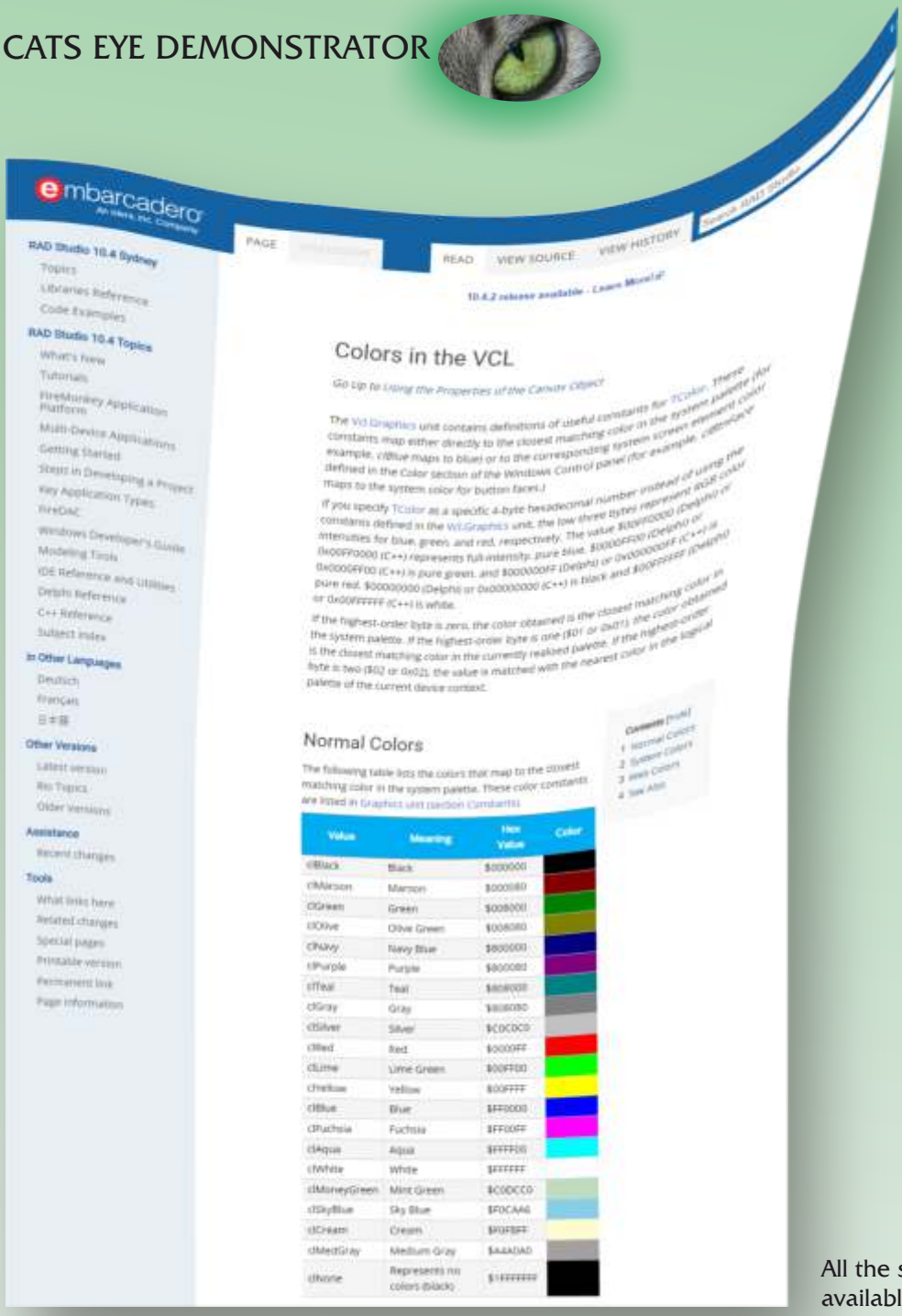
Different image dimensions may be selected. Make sure that `width = height`. Best is to make width and height odd, so center C is really the center of the circle. The cat's eye paint procedure:

```
procedure paintCatsEye(v : byte); //v is percentage 0..100
var i,r,r2,x1,x2,ybound : smallInt;  a,ta : single;  BF,RF : boolean;   c1,c2 : dword;  //colors

  procedure line(a,b: smallInt; col : dword);
  begin
   with form1.Image1.Canvas do begin
     pen.Color := col;
     moveto(a+r,i);
     lineto(b+r,i);
   end;
  end;

begin
  with form1.Image1 do begin
   r := width shr 1;
   r2 := r*r;
   a := 0.01*v*pi;  //angle in radians
   ta := tan(a);
   ybound := round(r*(1+cos(a)));
   for i := height-1 downto 0 do begin
     BF := i > ybound;  RF := i > r;
     x2 := round(sqrt(r2 - sqr(i-r))); x1 := round((i-r)*ta);
   if BF then begin
         c1 := $ff; c2 := $ffffff;
       end
   else begin
      c1 := $ffffff; c2 := $ff;
     end;
   if RF xor BF then  begin
    line(-x2,-x1,c1); line(-x1,x1,c2); line(x1,x2,c1);
   end
   else line(-x2,x2,c1);
   end;//for i
  with canvas do  begin
    brush.style := bsClear;
    pen.color := 0;
    ellipse(0,0,width,height);
   end;
  end; //with
end;
```

to change colours

The procedure to initialize the Image:

```
procedure InitCatsEye;
begin
   with form1.Image1 do with canvas do
   begin
     brush.style := bsSolid;
     brush.Color := form1.Canvas.Brush.color;
     fillrect(rect(0,0,width,height));
   end;
end;
```



All the special components are available for subscribers

The web page for colors:
http://docwiki.embarcadero.com/RADStudio/Sydney/en/Colors_in_the_VCL

Summersale:

1 year PDF Subscription + the new LibStick

(on USB Card - 95 Issues)

€ 70

ex vat / no shipment

URGENT

FBI vs Chad Hower/Alex - (Father & Son)

We seek help for Chad Hower and his Son.
He is being **wrongfully accused of kidnapping his son** by the FBI
and is in big trouble now.

The kidnapping accusation goes back to 2006 and
all charges have already been dropped,
except from the FBI.

Chad is now trapped on a small Caribbean Island
He is unable to travel due to health conditions.
He has life critical kidney problems.
He needs an air ambulance to get proper treatment.
The FBI refuses to help on that matter,
as he is also on **the Interpol RED** list,
despite 2 extraditions to US being denied.

Chad already used almost all his financial resources on this case.
We are seeking help from all media to create more awareness of this case and others
in the hope this kind of false accusations do not happen to anyone in the future.

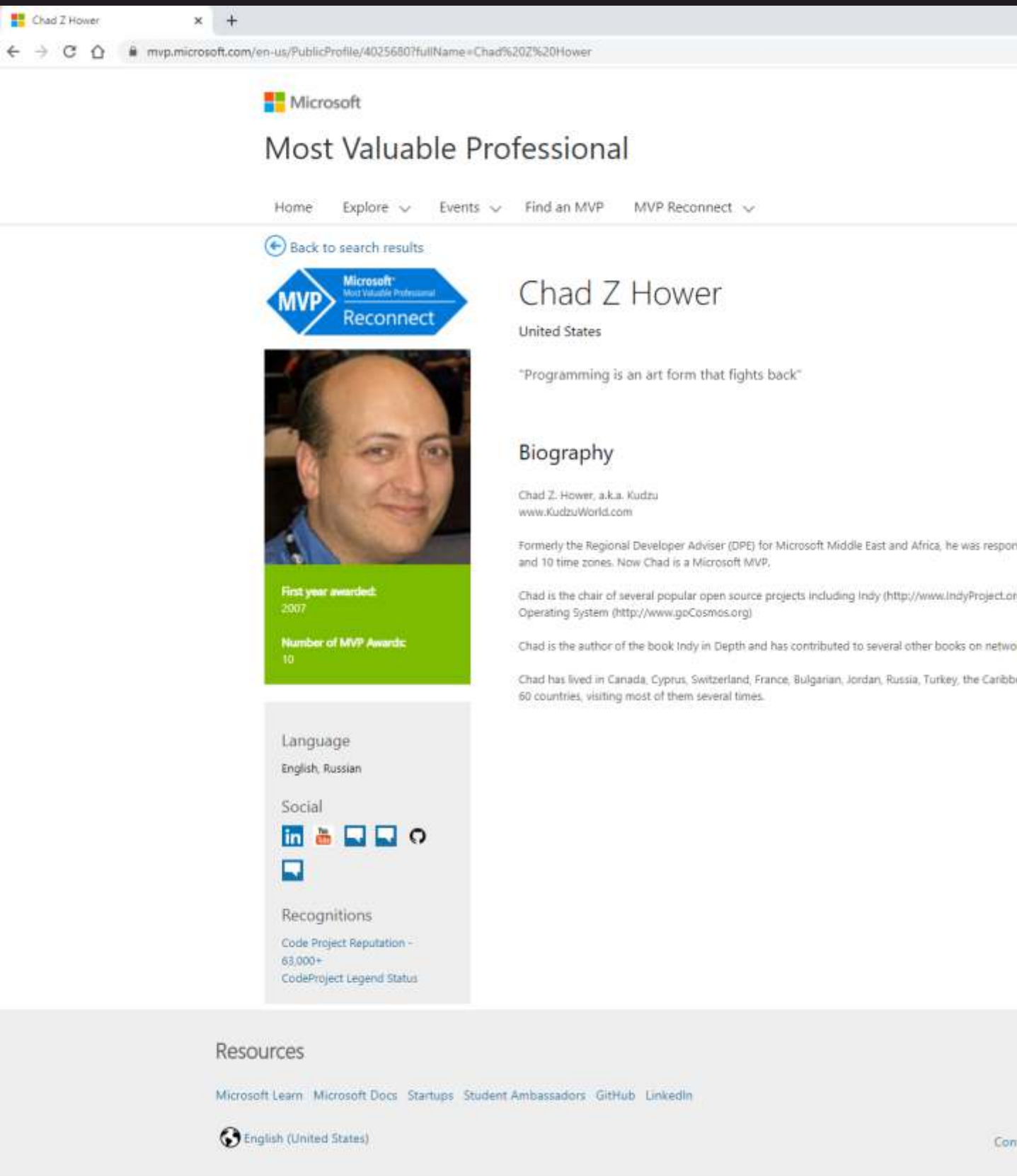
Please enter on the page below,
watch the video that was aired on a TV channel recently.
The video has a good summary of whole case:
https://www.alexisnotmissing.com

If you can, please help in covering this case and exposing it to the public.
If you want to learn more about this case, please check the websites

https://youtu.be/mzIDu0oESn4

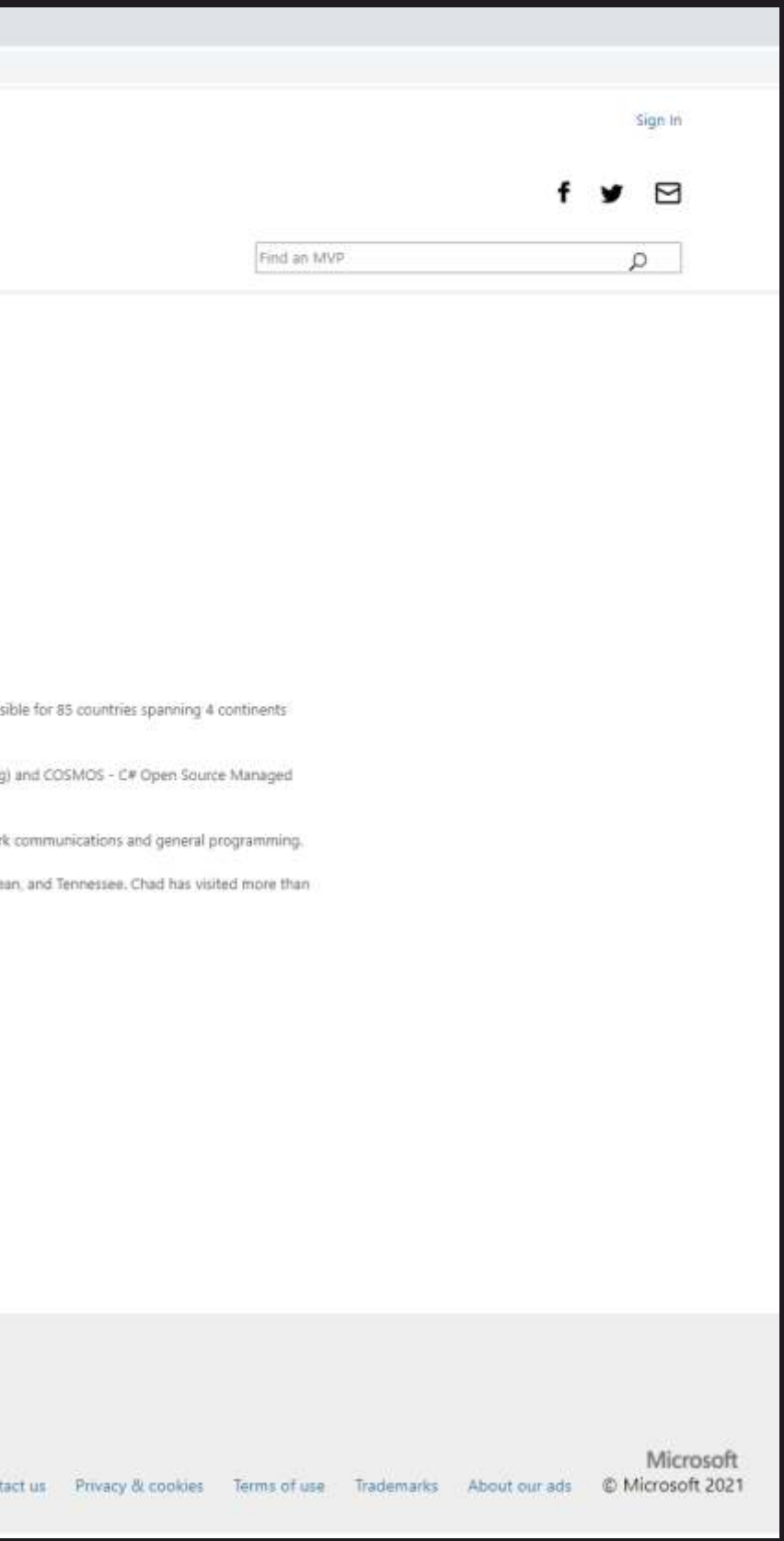
wrongfully accused of kidnapping his son

ABOUT CHAD HOWER



https://mvp.microsoft.com/en-us/PublicProfile/4025680?fullName=Chad%20Z%20Hower

ABOUT CHAD HOWER



KUDZU WORLD

`https://www.kudzuworld.com/`

WHO?

I am a professional software developer, a former Microsoft Regional DPE for The Middle East & Africa, and a former Microsoft Regional Director.

I began developing software in 1980 and am proficient in dozens of programming languages. I have lived in almost a dozen countries and visited nearly 70 countries. Read more at my bio.

CONTENT

ARTICLES

A collection of some of the articles that I have written.

BLOGS

My blogs about tech, software development, web development, and a little bit about the kudzu plant as well.

THE KUDZU PLANT

Learn about this strangely unique plant.

OPEN SOURCE

I have been continually active in open source projects including several that I founded since 1995.

PET PROJECTS

Miscelaneous technical but non software development projects of mine.

Wanted by Interpol and the FBI – Yes, seriously.
Not a joke, I am wanted by Interpol and the FBI.

THE DELPHI COMPANY

-est 1998-

OS X Android iOs Windows

Four Platforms
One develop environment
One Expertise

Vier platforms
Eén ontwikkelomgeving
Eén expertise

DELPHI

www.delphicompany.nl
info@delphicompany.nl

barnsten

The RAD Studio 11 release date is fast approaching, and with its support for high DPI screens, a vastly improved user experience and new features for rapid app development, it's going to change the game for Delphi and C++ developers.

Are you and your team ready for the new release?
Do you need to prepare at all?
What can you do to make the most of this important new release?
As Object Pascal and RAD Studio evolve, so should your business.
To save you time we've put together the most useful suggestions for preparing for the upcoming release of RAD Studio 11. Here they are:

1. Download Marco Cantu's Free "Object Pascal Handbook"

Launch your preparations for RAD Studio 11 by downloading Marco Cantu's "Object Pascal Handbook," one of the world's most popular, thorough and up-to-date books on building apps with Delphi, C++Builder and RAD Studio. This must-have book is also free to download.
Object Pascal Handbook Get this definitive resource on Object Pascal here.

2. Get Your Free Ticket For The Upcoming "Desktop First UX Summit 2021"

The annual digital event "Desktop First UX Summit" sponsored by Embarcadero Technologies' RAD Studio, will be held from Monday August 30th, to Friday September 3rd 2021.
The Desktop First UX Summit will bring together the brightest minds in UX for panel discussions and webinars around desktop UX. Attendees will have access to content around their favorite development tools and general content on effective User Interface Design and good User Experience. Helping developers prepare their applications for the upcoming release of Windows 11 will also be an important theme, and there will be an early preview of RAD Studio 11. Click here or on the image to register free and save your seat.

3. Sign Up For The "Sneak Peek For RAD Studio" Webinar

RAD Studio, Delphi, and C++Builder 11 are just around the corner. Join Product Management and Developer Relations for this sneak peek at this huge leap forward in your favorite development tools. In this webinar learn about how your programs will look even better on Windows 10 and 11 with new High-DPI IDE with VCL Style preview on the designer. See how you will write more powerful code with the new extensions to the Delphi language, improve speeds thanks to the math performance improvements. See the new support for Apple's custom silicon with macOS 64-bit ARM – bringing ARM to the desktop!

4. Take Advantage of Our Promo Offer

We have a very special early-bird offer created uniquely for the RAD Studio 11 release. It's a great opportunity to 1) get the 10.4.2 edition of Delphi, C++Builder or RAD Studio at a discount and 2) prepare your systems for the free upgrade to RAD Studio 11. This means you will not only get the full-featured latest version of any of these IDEs today for a discount, you will also get a two-month extension on your license, which means you will automatically get the new edition, RAD Studio 11, when it is released. The 10.4.2 version is already compatible with the M1 Mac Mini and WIndows 11, so you're all set from the get-go for the most advanced tech coming out today.

We've done everything we can to make it easy for you to upgrade your software to the latest editions and benefit from their newest features.

What else is there to do?

https://www.barnsten.com/how-to-prepare-for-rad-studio-11/

ABSTRACT

Recently, the **Free Pascal** and **Lazarus** teams switched from using **Subversion** to using **Git** as a source control system: the sources of the projects are now hosted on **Gitlab.** Time for a gentle introduction to git.

INTRODUCTION

People who want to contribute to an open source project are sooner or later confronted with a source control management system:
In the early linux / unix days **RCS** (**Revision Control system**, *a purely local solution*), later **CVS** (**Concurrent Version Control**, *which already offered client-server features*), **Subversion** – similar to **CVS,** and featuring a central file repository. All contributors connect to a central repository to get the sources, and submit changes to this central repository.

To be able to manage the huge community to develop the **Linux** kernel, **Linus Torvalds** (**next page bottom*) created **Git**: Instead of a central repository (*which would be prohibitively difficult to manage*) it is a distributed version control system. What sets it apart from solutions such as **Subversion** is the lack of a central repository to which all contributors must connect.

Instead, there can be many repositories, all sharing the same source code. Changes can be migrated from one repository to another (*a so-called pull request*) and *(usually)* end up in the original repository.

The **Git** versioning system took the programming world by storm: the appearance of **Github, Bitbucket** and **Gitlab** source code project collaboration sites and derivatives such as **Gitea,** have created an enormous ecosystem of tools around **Git.** You can even use **Git** to interact with a subversion repository, and **Github** allows (limited) access to a **Git repository** using **subversion.**

These projects build on top of git to provide easy to use merge-requests, issue tracking, CD/CI management, wikis and project management tools: all tools to facilitate cooperation on a software project.

❷ HOSTING A GIT REPOSITORY

When working with a version system, the central repository must be hosted somewhere so all collaborators can reach it. With git (*as with Subversion*), you can host this yourself: gitlab has an installable version of their project (including an open source version), gitea is a small-scale open source server project with functionality that is a subset of what github or gitlab offer – but installing it is as easy as copying a single binary.
However, hosting it yourself means the maintainance burden also lies with you. It makes sense to use a **SaaS** offering and host it on **gitlab, github** or **bitbucket:** in that case the platform maintainers will do all the maintenance. For small personal projects, it's not even necessary to purchase a license. For larger projects which require a lot of management, purchasing a license is probably the better idea.
For these reasons, the Free Pascal and Lazarus teams opted to host their sources on **Gitlab.**

❸ CONNECTING TO A GIT REPOSITORY

To connect to a local or remote repository, you need a **Git** client. There are many git clients available.
On **Linux** the command-line git client is available from the package manager of your **Linux** distribution. For **MacOS,** the command-line **Git** client is installed with relatively recent versions of **XCode.** For both operating systems many **GUI** clients are available, some of them cross-platform.
On **Windows,** there are several available clients, many of them are cross-platform.

- **Git** for **Windows:** a command-line client with a bash shell, so you can copy and paste commands you find on internet:
  `https://gitforwindows.org/`
- **TortoiseGit** integrates with the **Windows** explorer, much like **TortoiseSVN** does, and as such it is the only **GUI** client in this list that works only on windows:
  `https://tortoisegit.org/`
  If you already have **TortoiseSVN** installed, both can work side-by-side.
  If you install this client, you must also install the **Git** For windows client, as **TortoiseGit** uses that to do all the work behind the scenes.
- **Github** desktop is a **GUI** client for **Github,** made by the **Github** developers, but it can be used to connect to any **Git** repository:
  `https://desktop.github.com/` It works on all major platforms.
- Similarly, **GitAhead** is a git client that works on all major platforms:
  `https://gitahead.github.io/gitahead.com/`
- Sourcetree is a free client for **Windows** and **macOS:**
  `https://www.sourcetreeapp.com/` It is made by the developers of BitBucket.
- **SmartGit** runs on **Windows, Linux** and **macOS:**
  `https://www.syntevo.com/smartgit/`

** Linus Benedict Torvalds: Finland Swedish: born 28 December 1969) is a Finnish-American software engineer who is the creator and, historically, the main developer of the Linux kernel, used by Linux distributions and other operating systems such as Android. He also created the distributed version control system Git and the scuba dive logging and planning software Subsurface.*

Last but not least, the popular **VSCode** and **Atom** editors have git support built-in: for most common operations, it is sufficient. Similarly, **Delphi** offers support for **Git** right in the **IDE** as well.
Delphi it self has a very small version which allows only a very few commands.

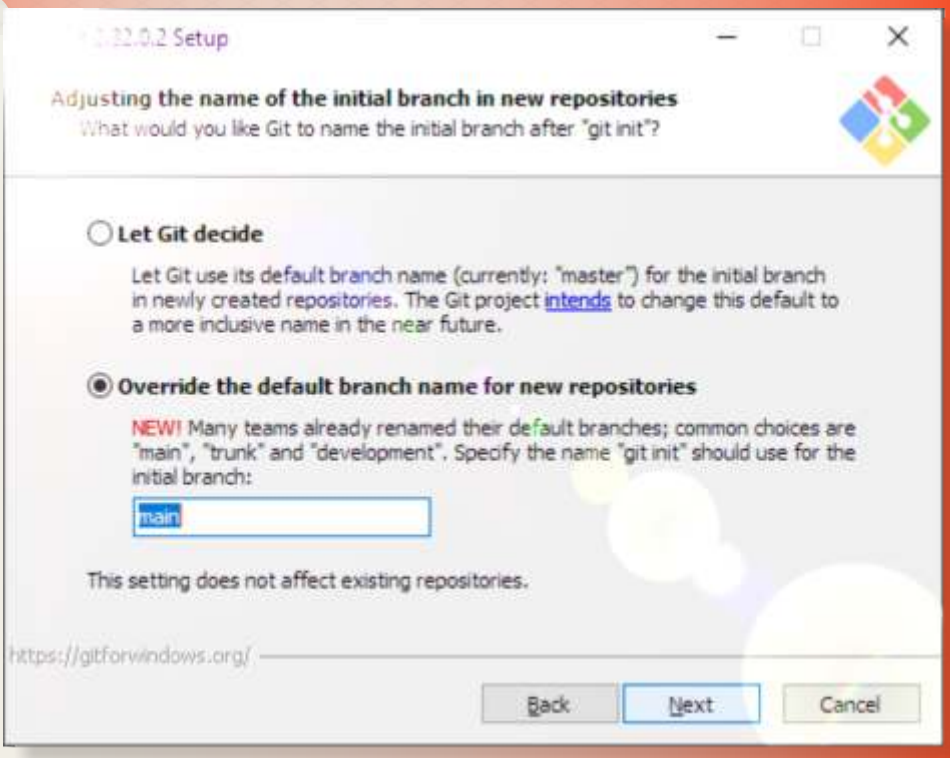


Figure 1: Override the default branche name

❹ GIT FOR WINDOWS INSTALLATION

We'll discuss the installation of the **Git** for windows and **TortoiseGit** client here, because they require some configuration during setup, which can be confusing. The other mentioned tools can simply be installed without too much questions beyond the installation directory;
Their configuration happens usually using some dialogs in the program itself.

Once you've downloaded the **Git** for **Windows** installer, it's a regular **Windows** installer like any other, but it asks some questions during the installation process. For a new git user, these questions can be confusing, so we'll go over them.

The first git-related question is what the initial branch of a new repository should be called, see figure 1 on page 3. Traditionally, the default initial branch created by **Git** was called master. This choice raised some concerns about political inclusiveness of the software, so now the installer asks what the default name must be.

The second git-related question is what should be done with the `Windows PATH` environment variable, (*see figure 2 on page 4*). **Git** for windows is made to run in a simulation of the unix shell (bash). But it can also be used outside this unix shell. This question asks you how you will want to use **Git:** exclusively in the bash shell, do you want it to be usable by other tools as well, or should all provided **Unix-like** tools also be made available in **Windows** terminal environment. If you want to use git for **TortoiseGit,** then the default choice (*from the command-line and from 3rd-party software*) is the correct one.

When connecting to a repository over **HTTPS,** the **Git** client can use the **OpenSSL** library or it can use the **Windows API**, (*see figure 3 on page 4*). Using the **Windows API** makes sense in corporate settings, where private server certificates should be accessible from e.g. an active directory. For most common setups, the use of **OpenSSL** is sufficient.

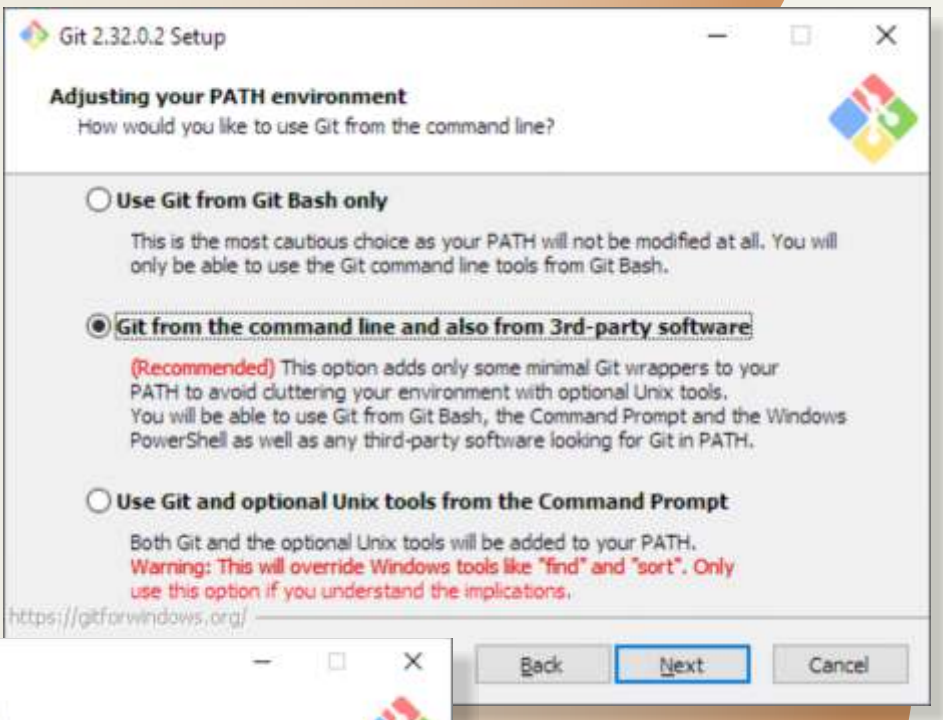


Figure 2: Git from the command line

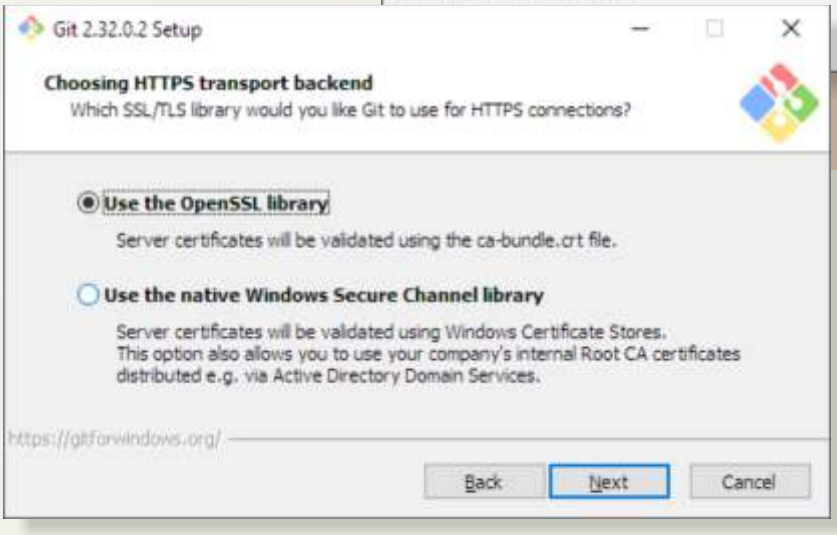


Figure 3: Use open SSL library

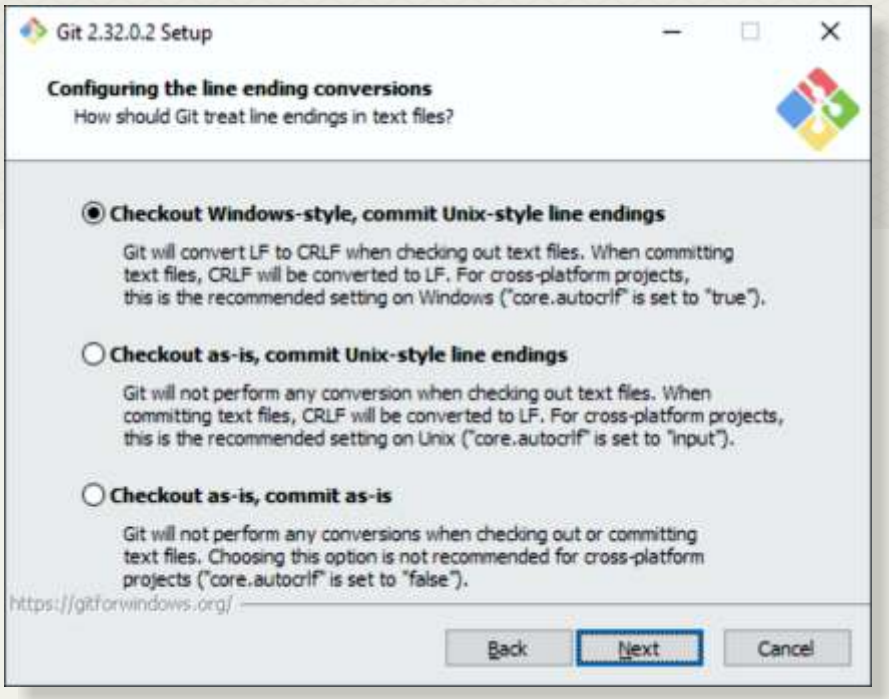


Figure 4: Check out windows style

When working together with people that do not work on windows, it is important to use the same settings for the line-ending character. On **Windows** this is traditionally **CRLF,** on **Linux** and **macOS,** this is **LF. Git** can help you keeping consistent linefeeds in the source code and adjust to your platform's setting. This question asks you what you want git to do: The first choice is usually the best one. (*See figure 4 on page 5*).

Git for **Windows** is designed to work in a unix-like shell: `bash`. This runs in a terminal window. A special terminal window program (**MinTTY**) is provided with **Git** for **Windows,** which offers more functionality than the standard windows terminal. This question asks you which terminal window you want to use when you activate the **Git for windows** program. (*See figure 5 on page 6*).

With the next question the installer wants to know what the behaviour of **Git** should be when you get changes from the remote repository server, (*see figure 14 on page 11*). When you merge branches in git, or pull changes from the server, there are several ways in which git can do this. The default is fast forward: it just replays all the diffs on top of what you already have. But if that is impossible (*because you have local changes that interfere with this*), it will create a `merge commit`: **Git** will create a new commit that marks the incorporation of the changes in the current branch.There is an alternative, which is called 'rebase': **Git** internally stores diffs, not files. What this option does is change the diffs which you committed locally so they look like they were applied on the last version retrieved from the server. The **Git** `pull` operation gets updates from the server and incorporates them in your local checkout. Here there is also an option to let **Git** fail if it cannot do a fast-forward, i.e. when

When you want to push updates to a remote server (*or pull from a protected server*), you must supply credentials: an **SSH** key when connecting through **SSH,** or a username / password when connecting using **HTTPS.** Unless you save the credentials, git will ask you for them every time it needs credentials. This question (*figure 7 on page 7*) asks you which credentials manager you want to use. You don't need to choose any one of the proposed managers, git has a default manager which stores the credentials in a text file, but you must configure that one separately.
The last question asked by the installer is whether it should cache some information from the file system, in order to speed up configuration. This has no influence on the working of **Git.**
All of these questions result in the creation of a default system-wide configuration: your choices can be changed or reconfigured later on using the **TortoiseGit** client or on the **Git** command-line.

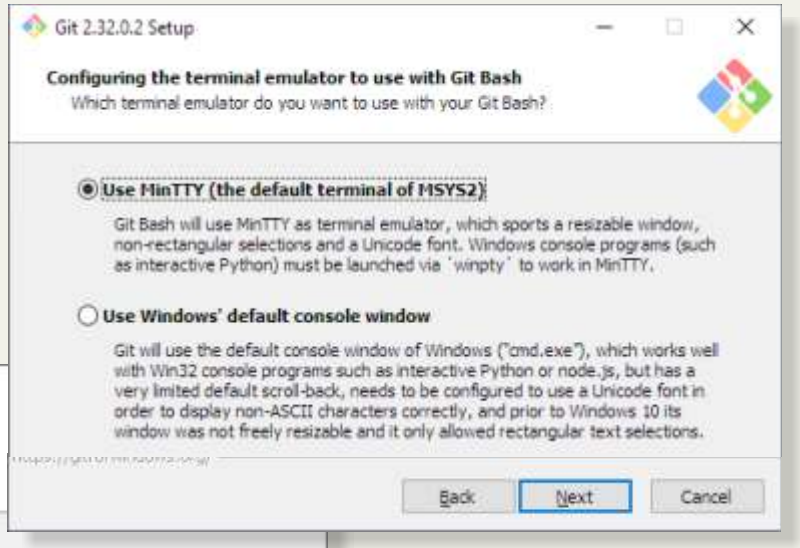


Figure 5: Use Min TTY

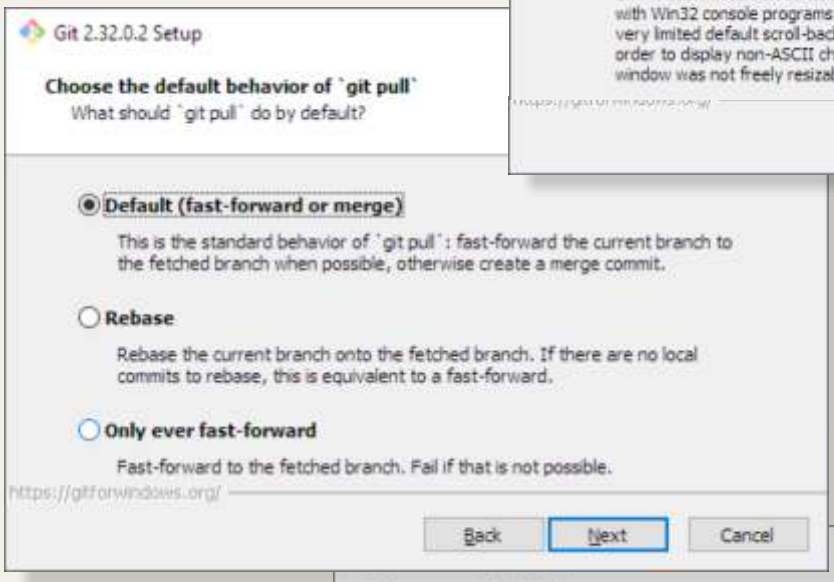


Figure 6: Default

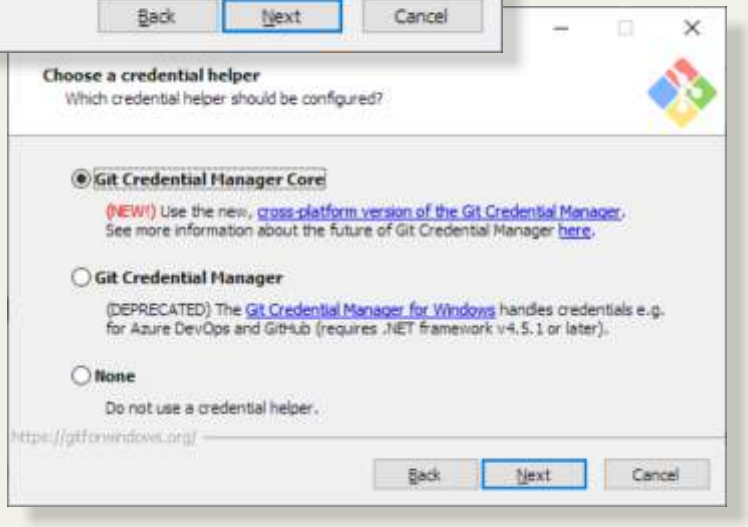


Figure 7:
Git Credential Manger Core

❺ TORTOISEGIT INSTALLATION

The **TortoiseGit** client installer asks far less questions during installation. In fact, the only important one is whether it should start the first-start wizard. (*See figure 8 on page 7*). The first-start wizard will do the actual minimal configuration of **TortoiseGit.** After asking which language you want to use in the **TortoiseGit** interface, it will ask you where the git binaries are located. If you used the default settings for the `PATH` in the **Git** for **Windows** installer, then the **TortoiseGit** wizard should have detected the correct location and proposes it. (*See figure 9 on page 8*).

When you commit source code, **Git** will add your name and email address to the commit.
It will attempt to guess it from your operating system username, which is more often than not a bad choice.
Systems such as **Bitbucket, Gitlab** or **Github** use the email address stored with the commit to identify the registered user that made a commit: The email address must therefore be an email address that these systems know. They have functionality so you can associate multiple email addresses with your account, but still your **Git** client (**tortoisegit**) must associate one of these email addresses with each commit.
In this step of the wizard (*figure 10 on page 8*), the wizard asks you which username/email address you want to use. Note that this is not the identity information used to authenticate to a remote server (*see also next question*).
Last but not least, the **TortoiseGit** wizard asks how it should store or retrieve the authentication info used by git (*figure 11 on page 8*). In essence, this is the same question asked by the **Git for Windows** installer for **HTTPS** requests;
Additionally, **TortoiseGit** allows you to configure **SSH** access to the server: it can generate a public/private key pair, and the public key can then be uploaded to **Gitea, Github, Bitbucket** or **Gitlab.**

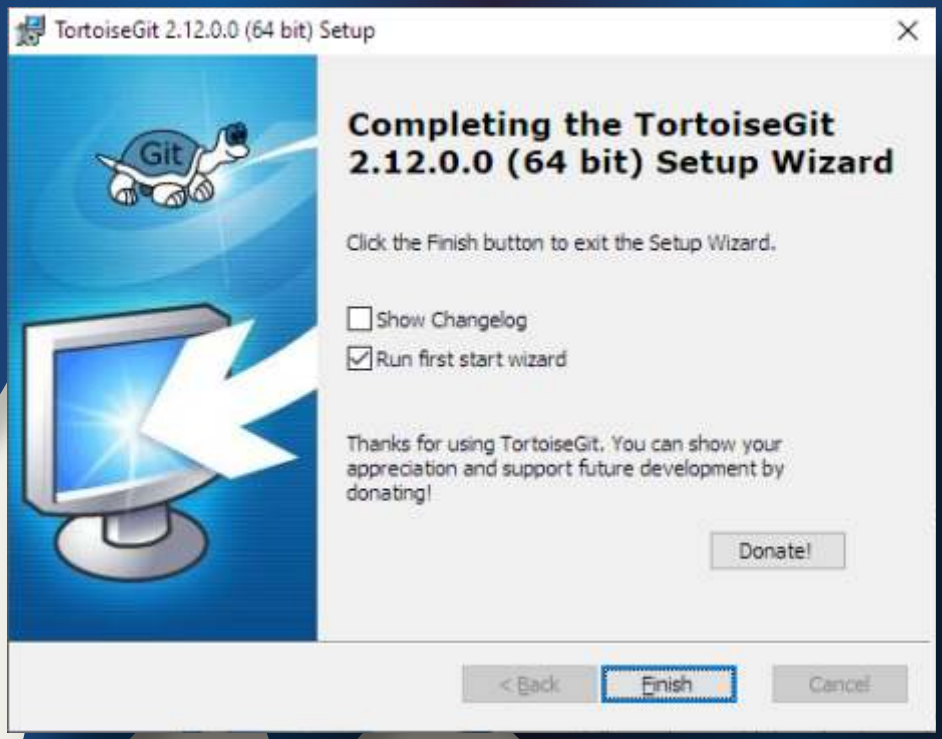


Figure 8: Tortoisegit installer

❻ GETTING CODE FROM A REMOTE REPOSITORY

To get a copy of the code in a remote repository on your local harddisk, git uses the clone command. If you want to check out the **Free Pascal** sources in a directory `D:\FPC\Source`, you would type the following commands in the bash window that the git installer has given you:

```
mkdir /d/FPC
cd /d/FPC
git clone https://gitlab.com/freepascal.org/fpc/source.git Source
```

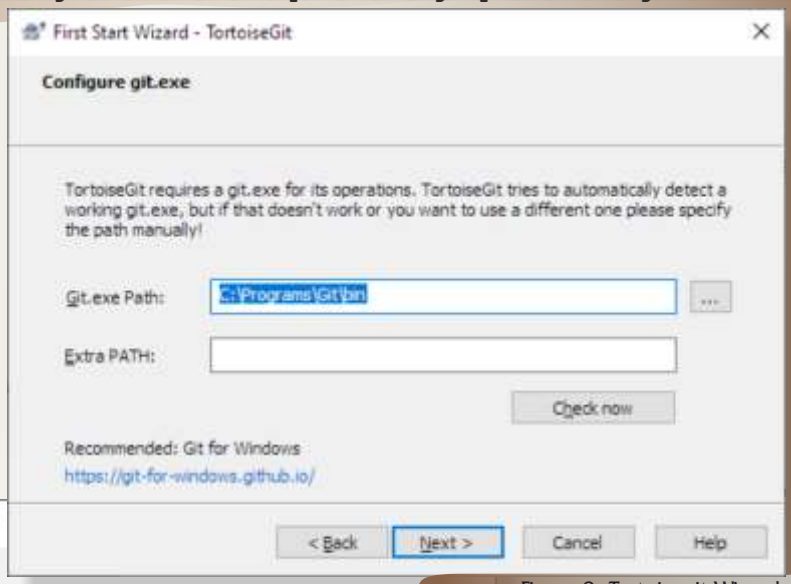


Figure 9: Tortoisegit Wizard – git location

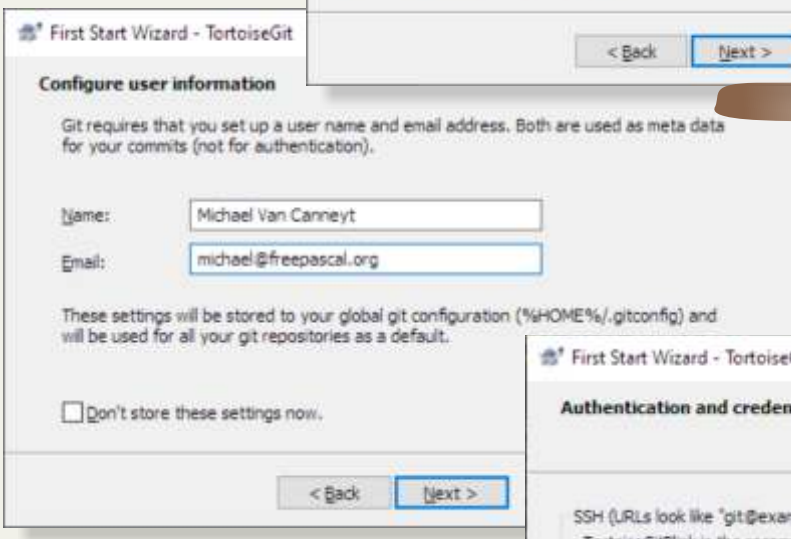


Figure 10: Tortoisegit Wizard – user identity

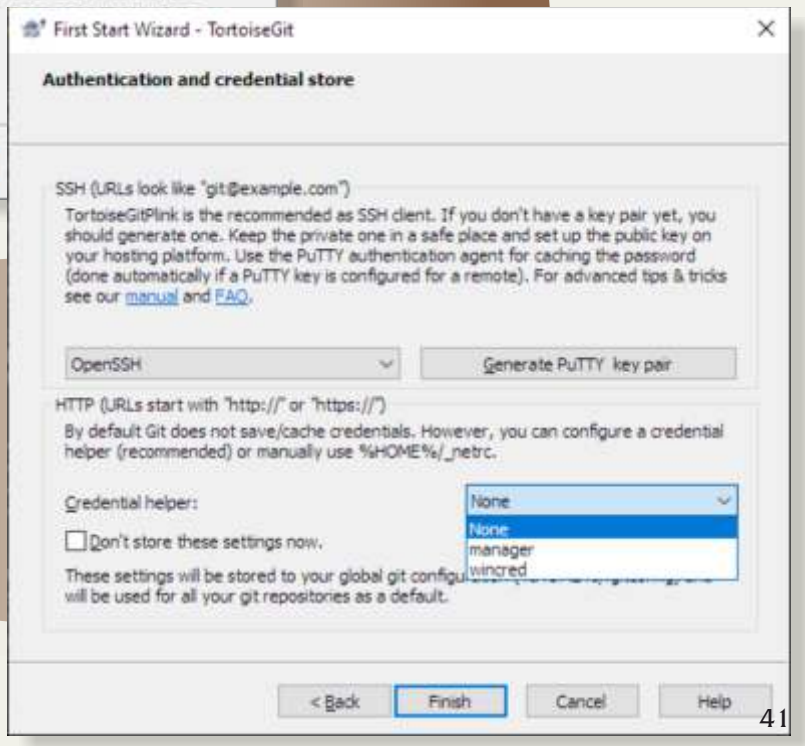
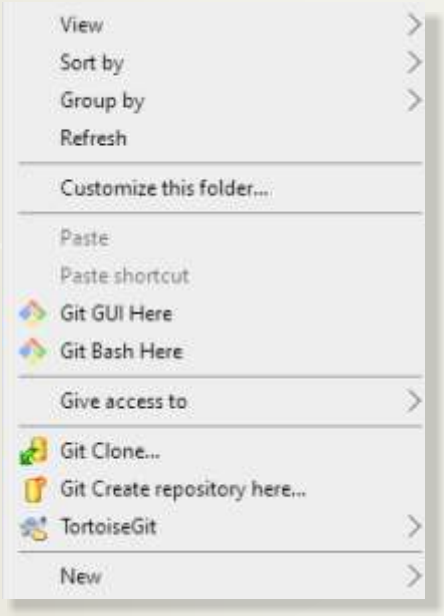


Figure 11: Tortoisegit Wizard – Credentials

The first command is not necessary if you already have a **FPC** directory. Note that the **bash unix** environment uses **forward slashes in path names**. This command does 2 things:

1. It fetches the contents of the remote repository, and puts them in an administrative directory (*called* `.git`).
2. From the local repository, it checks out the default branch.
   This step can be avoided with the `-no-checkout` or `-n command-line` option.

To do the same in the **Tortoisegit** shell extension, click right in the `D:\FPC` directory, and select **Git clone** from the context menu:



This will pop up the dialog in figure 12 on page 9. You can enter the **URL** as shown above, and the directory in which to check out the sources.
There are many options you can specify, but for most purposes, the **URL** and directory are sufficient.
Note that you can also use **Git** as a client for remote **Subversion** repositories: you can also specify a **Subversion** server **URL** as remote repository.

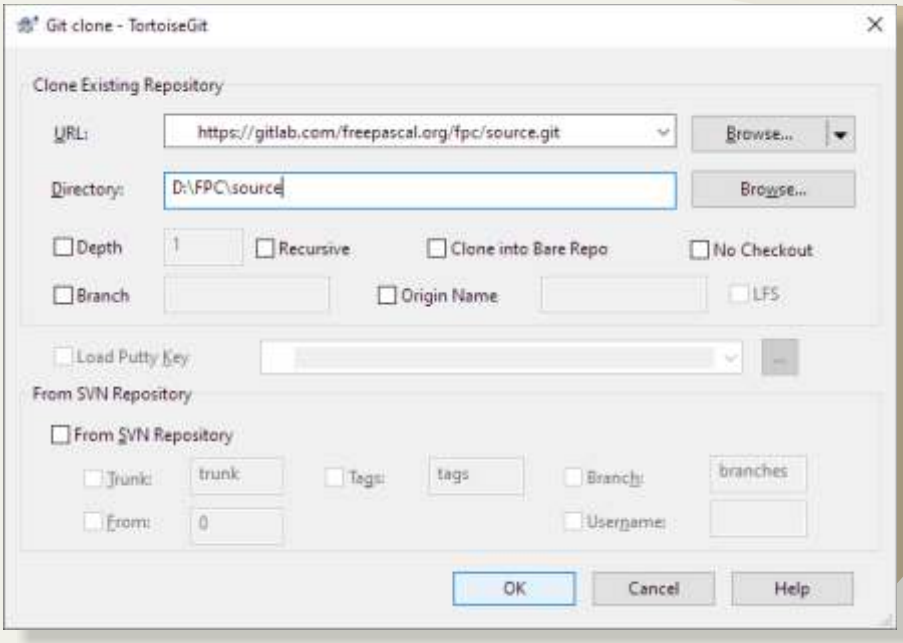


Figure 12: TortoiseGit clone dialog

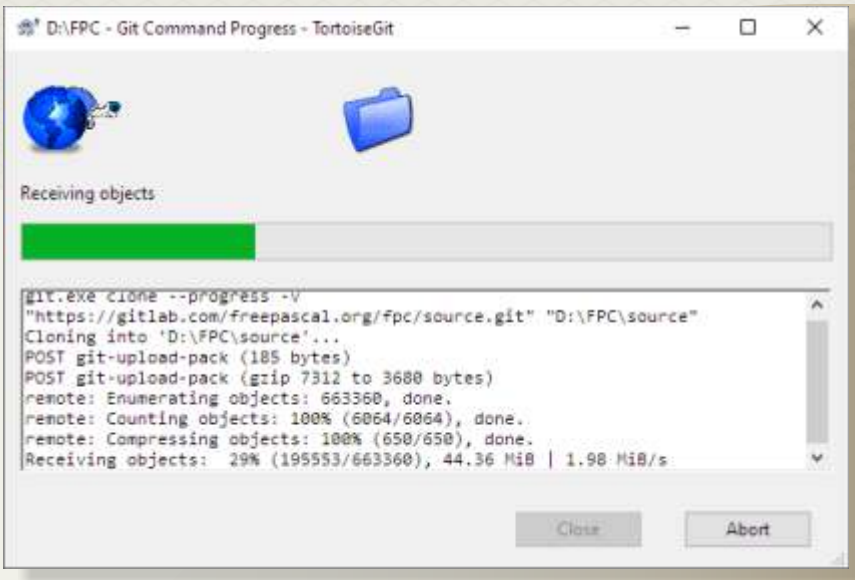
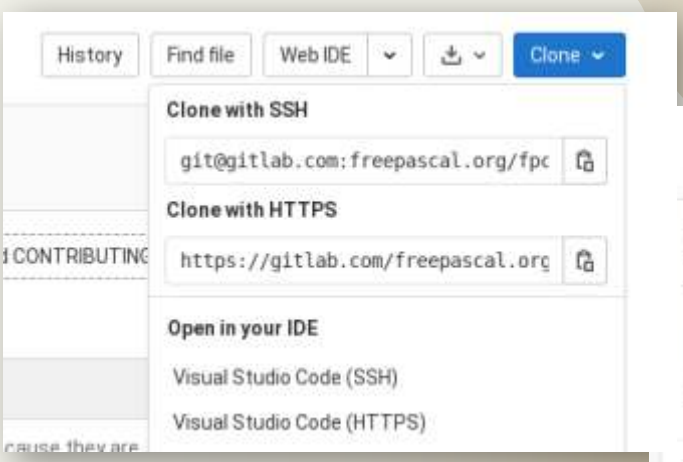
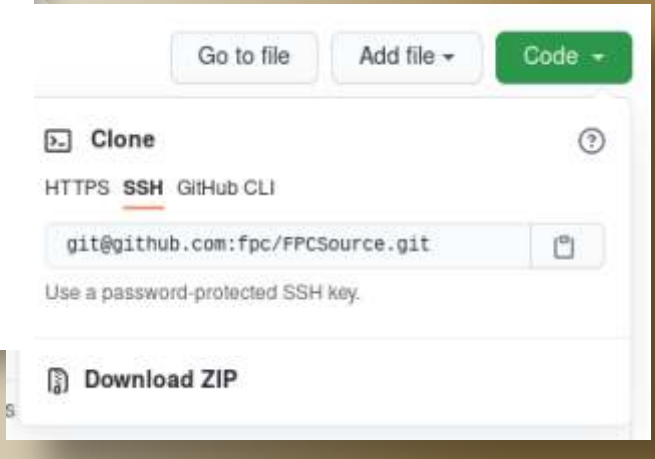


Figure 13: TortoiseGit clone progress

When you press the OK button, git will fetch the complete repository from the server, and will check out the main branch (*or any other branch you named*). This can be a lengthy operation, and **TortoiseGit** will show a progress dialog as in figure 13 on page 10. It shows the same output as the command-line version of **Git.**
How to get the remote repository **URL** you need from systems like **Github, Gitea** or **Gitlab?** All these systems in their **GUI** show a button that, when clicked, allows you to see and copy the **URL** to clipboard. For example, for **Gitlab,** the button looks as follows:



And for github, it looks like this:



❼ UPDATING YOUR COPY OF THE SOURCES

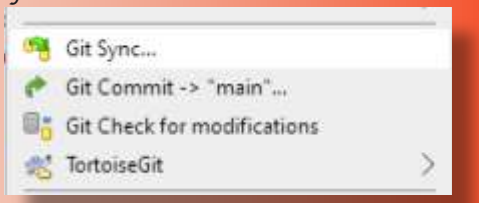
When you've cloned the repository locally, after some time, your copy will most likely be outdated: Project developers will have published new changes in the remote repository, and your copy does not yet contain these changes. To bring your local copy up to date with the remote repository, you must issue a **Git pull** command. In subversion, this was the update command:

```
cd /d/FPC/source
git pull
```

As for **Git** clone, the effect of the pull command is twofold:

- It fetches all changes from the remote repository and stores them in your local copy of repository. This part is called fetching – **Git** has actually a separate fetch command.
- If in the fetching operation changes were fetched for the checked out branch, it applies these changes to your checked out branch.

In **TortoiseGit,** the pull operation is available under the **Git** sync menu item:



This brings up the synchronisation dialog, in which the Pull button can be used to do the actual pull operation. *(The result of the operation is shown in figure 14 on page 11).*

❽ SWITCHING BRANCHES

In **Git,** traditionally a lot of branches (*parallel lines of development*) are used: branching is a very cheap operation in **Git.** Often, a new feature is developed in a branch and merged to the release branch when it is deemed ready. Using **Git,** you can check out the branch and the sources before the feature is released. You can get a list of available branches by simply entering the git branch command: `cd /d/FPC/source`

```
git branch -r
```

The -r switch tells git it should show only remote branches. You can also specify -a, in which case all branches are shown.

In **Tortoisegit,** the same list can be obtained by choosing the `Browse references` menu item from the **'TortoiseGit'** context menu. It will show the same information in a dialog (*as shown in figure 15 on page 12*):

To actually switch to an existing branch, the **Git** `switch` command can be used (*in older git versions, this operation is called* `checkout`).

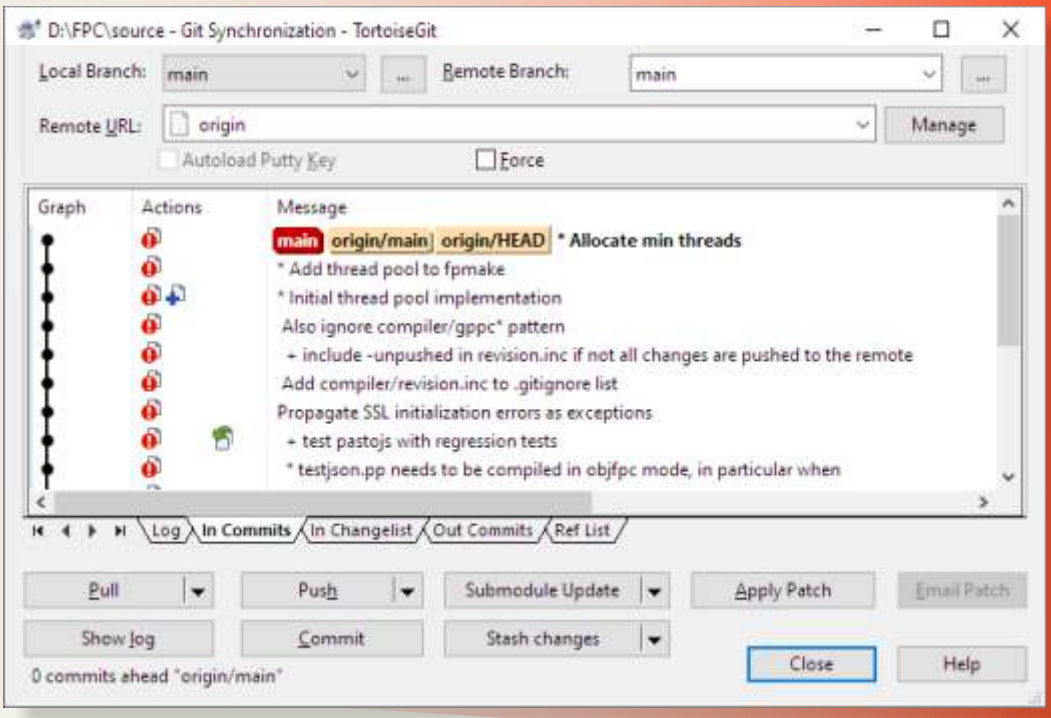


Figure 14: Tortoisegit synchronisation - Pull result

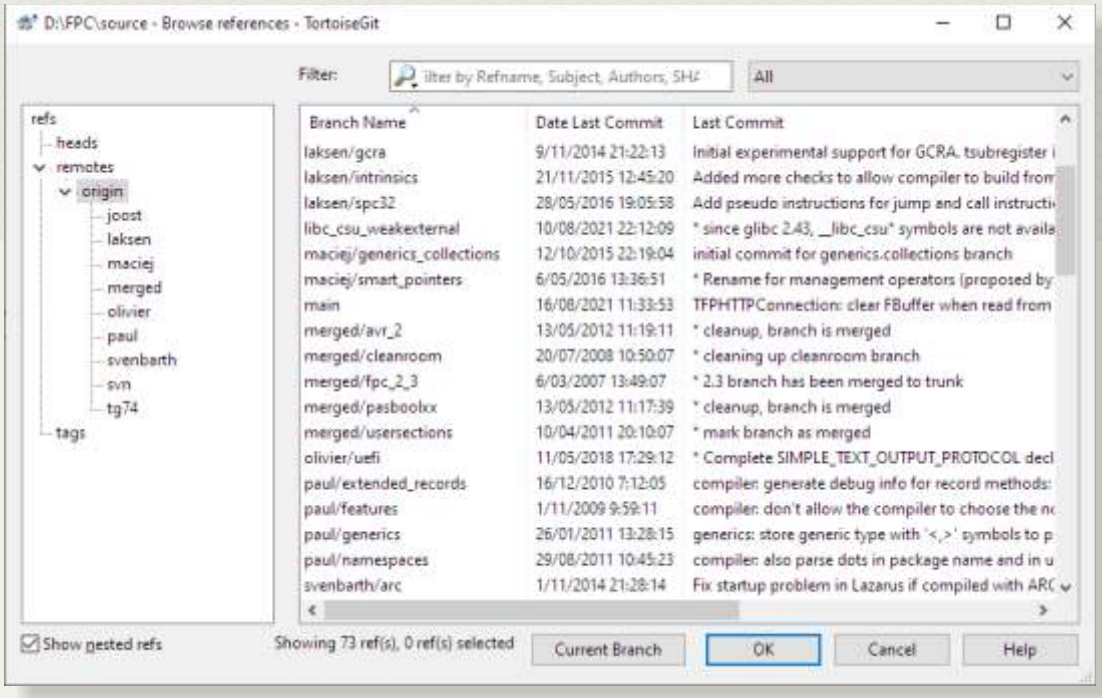


Figure 15: TortoiseGit browse references: branches

```
cd d/FPC/source
git switch libc_csu_weakexternal
```

When you do this, **Git** will check if the branch exists locally and switch to it if it exists. If it does not exist, but a remote branch exists with the same name, **Git** will create a local branch with the same name as the remote one, and point the local one to the remote one: whenever the remote branch is updated, a **Git** pull will update your local branch as well. Then it will switch to the new branch.
To create a new branch and switch to it, again the git switch command can be used:

```
cd /d/FPC/source
git switch -c issue_40100
```

You can also specify a different start point:

```
cd /d/FPC/source
git switch -c issue_40100 main
```

This will create a new branch issue_40100 starting from the main branch. When using an older **Git** version, you use the checkout command with the -b option:

```
cd /d/FPC/source
  git checkout -b issue_40100 main
```

You can also simply create a branch with the branch command, without switching to it:

```
cd /d/FPC/source
git branch libc_csu_weakexternal
```

In **TortoiseGit,** the **Switch/Checkout...** menu item must be used to switch to an existing branch or to create a new one. A dialog is shown (*figure 16 on page 13*) where the necessary parameters are given: You can select an existing branch or create a new one.

❾ TEMPORARY STORING CHANGES

Switching branches (*or pulling from a remote repository*) can fail if there are uncommitted changes in your working copy. If you don't want to commit these changes yet, for example because they're not yet done or not yet properly tested, this can be solved by using the stash command. This command sets aside the uncommitted changes in a diff, and restores the working copy to the state it was in before you created the changes.

```
cd /d/FPC/source
git stash -m '* Temp work for feature X'
```

You can leave the message empty, in that case git will create a message with a reference to the last commit.
To do this in **TortoiseGit,** you must select the 'Stash changes' command from the **'Tortoise-Git'** context menu. In the dialog that appears then (*figure 17 on page 13*) you can enter the stash message.

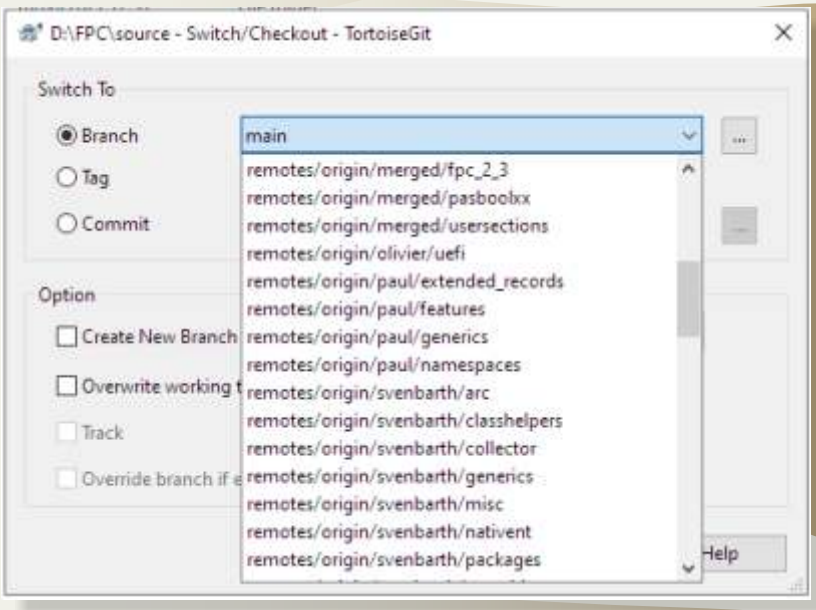


Figure 16: TortoiseGit switch dialog

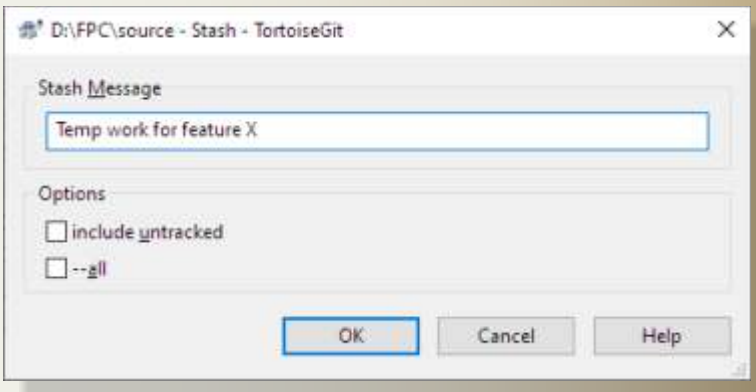


Figure 17: TortoiseGit stash changes

⑩ DISCARDING CHANGES

Instead of setting aside changes, you may prefer to discard any changes you've made to the working copy. You can do this to resolve potential conflicts when pulling changes from a remote, or simply because you've done some changes you do not like and wish to undo them. In subversion, this operation was called revert. In **Git** this operation is called `restore`.
On the command-line, the following will undo all changes to all files:

```
cd /d/FPC/source
git restore .
```

You can also specify one or more filenames instead of a directory. In older versions of **Git,** this had to be done with the checkout command:

```
cd /d/FPC/source
git checkout .
```

But this was confusing because the checkout command was used for a lot of other things, and the **Git** developers made a special command for it. The checkout command also still works in newer versions of git.

In **TortoiseGit,** you can achieve the same with the 'Revert' menu item below the 'Tortoise- Git' popup menu. You will get a dialog with a list of changed files and can select the files which must be reverted, see figure 18 on page 14.

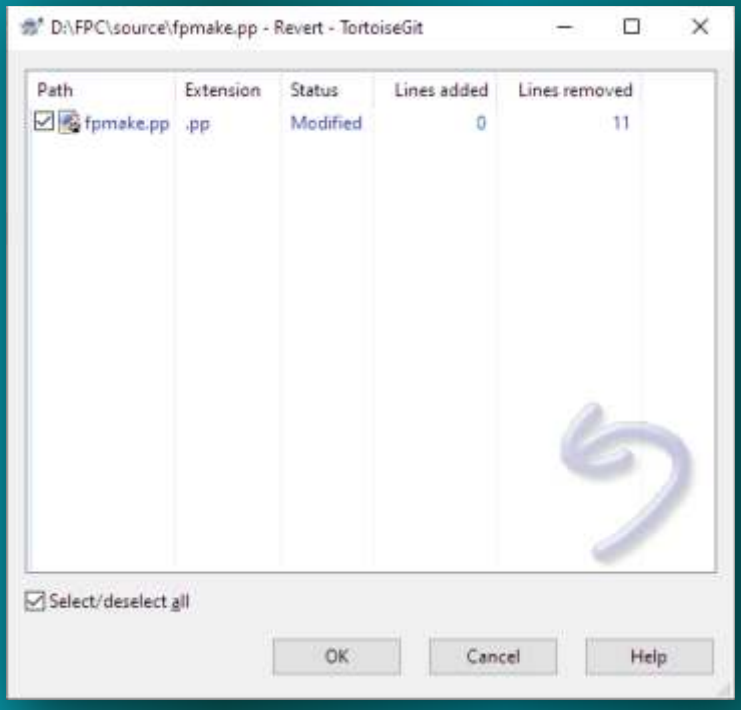
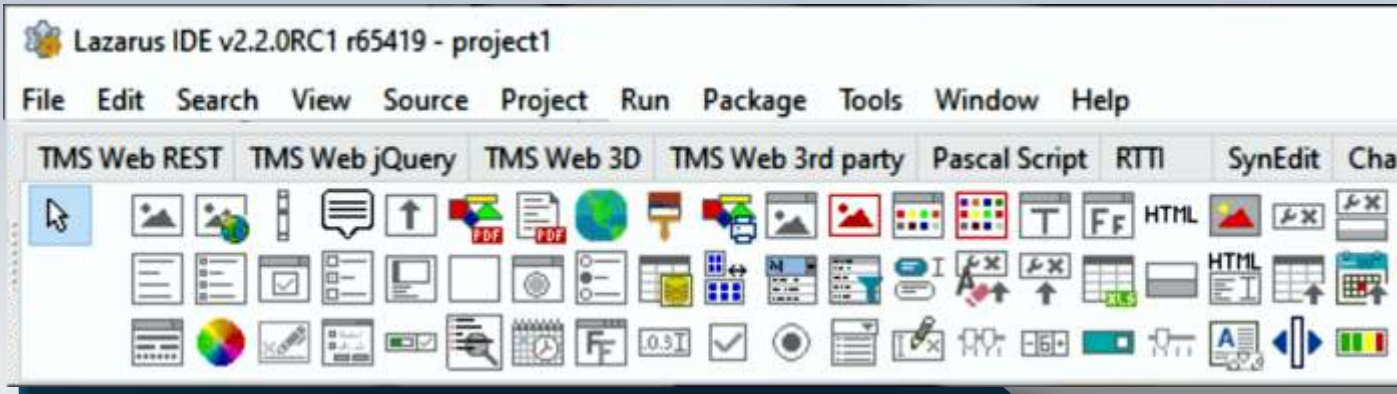


Figure 18: TortoiseGit revert

⑪ CONCLUSION

In this article we've shown how to use git to get sources from remote repositories, and how to update your local copy. We've also shown what you can do when you changed files and wish to undo these changes, with the option to redo them later on. In a next article we'll show in more detail how to handle your own changes to sources, and how you can contribute them back to the project. This will require a deeper dive in the eco-systems built around git.



ABSTRACT:

As shown in several other articles (Nr….) I showed how to create in very simple ways to create a small Rich Editor. The editor I found is been shown under Lazarus and Delphi. However I found that if you have to create Rich Editor from scratch it is a tedious task. It can be done. But the commercial possibilities are great and not very expensive. TMS Software as so often offers a great (big as well) suite: the FNC Component Group. (*see Figure 1 at het top of the page*)
Why this one? Simply because you can use this suite under **Delphi** as well as **Lazarus.** And that makes it extra attractive.

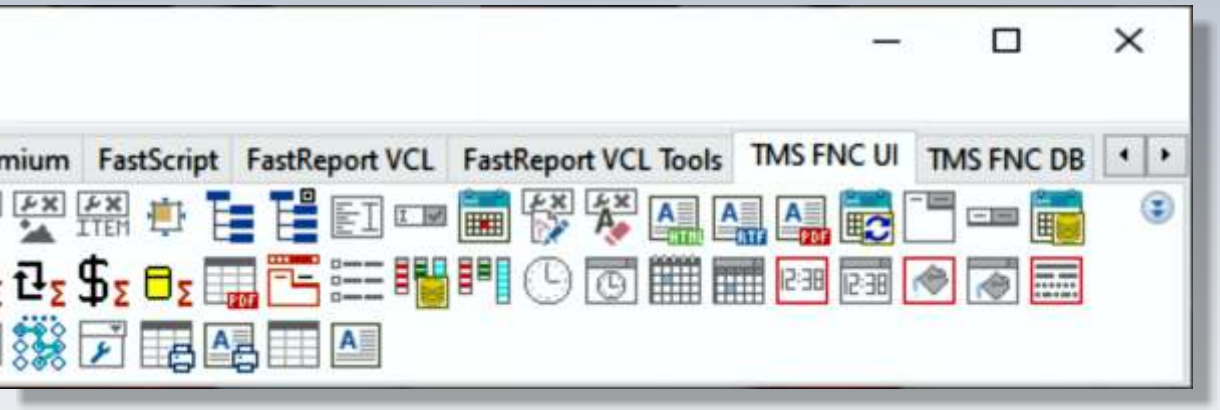
INTRODUCTION:

I will show you how to find the components to install for Lazarus. When you do so, and have Delphi installed it will create packages for Lazarus as well. The configuration is a bit confusing so I need to explain where they are located and how to find them. The actual creation of the Rich editor is very simple and I wrote about that before. There is a PDF file which gives you all the details. There is also a version you can use for the web.

START WITH THE INSTALLATION OF THE PACKAGE:

Of course you better install the latest version of TMS FNC Components:
(*Framework Neutral Components for use with Delphi & C++Builder VCL framework, Delphi & C++Builder FMX framework and* ***Lazarus LCL*** *framework and for cross-platform application development targeting* ***Windows, macOS, iOS, Android, Linux and Web***)
`https://www.tmssoftware.com/site/products.asp?t=fnc`
So this was the easy step: there is a price example and when you look at it you'll have to agree this is very very cheap. (*See Figure 2*)
Especially when you realize that this is for all platforms. But the best way of getting convinced: just try it(*see Figure 3 on the next page*).
After downloading the normal formalities have to be done. Install and read carefully.
The installation is done in a way that for **Delphi** you do not need to do anything special. Its installed for you as long as yo will tell your correct **Delphi Version.** (*On September 9, 2021 the version 11 will be added: Olympus*). Here is the path where under windows 10 everything is installed. I have searched for the pdf File that handles the Rich Editor and the **`.lpk`** files (Lazarus package files)

Figure 2: The price example



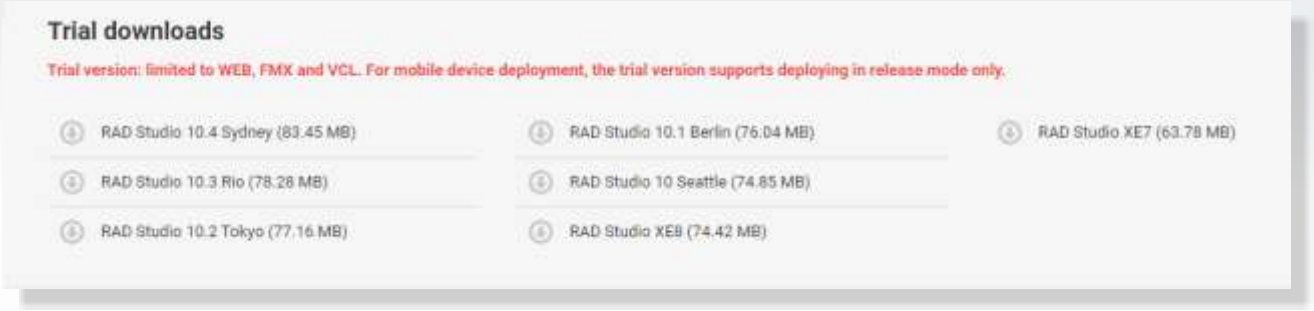


Figure 3: Trial downloads

Here is the path where under windows 10 everything is installed.
I have searched for the pdf File that handles the Rich Editor and the `.lpk` files (Lazarus package files)
It is normally place over here:

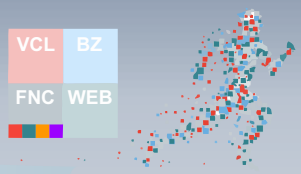
```
c:\Users\YourName\AppData\Local\tmssoftware\registered\TMS FNC Core\
c:\Users\YourName\AppData\Local\tmssoftware\registered\TMS FNC UI Pack\
```

let us inspect where everything is located.

FIND YOUR LAZARUS COMPONENTS

The Dir contains an install file **`install.txt`**. The last lines are about the installation of the Lazarus packages:

```
For Lazarus 1.44 or higher
-------------------------
Uninstall previously installed FNC packages and rebuild the Lazarus IDE
Remove LIB folders generated in the source directory.
In the IDE, select File, Open and browse for LCLTMSFNCCorePkg.lpk
and select "Open Package" when prompted.
From the package window select Compile
In the IDE, select File, Open and browse for LCLTMSFNCCorePkgDE.lpk
and select "Open Package" when prompted
From the package window select Compile.
```



FOR LAZARUS 1.44 OR HIGHER:

1. Uninstall previously installed FNC packages and rebuild the Lazarus IDE
2. Remove LIB folders generated in the source directory.
3. In the IDE, select File, (See Fig)
4. Open and browse for **LCLTMSFNCCorePkg.lpk**
5. and select "Open Package" when prompted
6. From the package window select **Compile**

Now repeat most of the steps for this

In the IDE, select File,
Open and browse for **LCLTMSFNCCorePkgDE.lpk**
and select "Open Package" when prompted
From the package window select **Compile**.

There is an other lpk file that has no meaning for us: **tmswebcorefnccorepkg.lpk**
This is only for the "Core" package.

After you have done all this, you will have to do it for the other package: **The "UI" Pack.**
Its principally the same way of working. You need to download this package and install it normally.
Then after installing you can see the dir:

```
c:\Users\YourName\AppData\Local\tmssoftware\registered\TMS FNC UI
 Pack\
```

in this directory you will find numerous information:

```
Android Support\
Delphi104Sydney\
Demos\Demos:FMX\
       LCL\
       Shared Code\
       VCL\
       WEB\
Doc\
```

in the `c:\Users\edito\AppData\Local\tmssoftware\registered\TMS FNC UI Pack\Doc\` you will find the documentation of all the components.

```
TMSFNCCalendarDevGuide.pdf
TMSFNCGridDevGuide.pdf
TMSFNCKanbanBoardDevGuide.pdf
TMSFNCObjectInspectorDevGuide.pdf
TMSFNCPlannerDevGuide.pdf
TMSFNCResponsiveListDevGuide.pdf
TMSFNCRibbonDevGuide.pdf
TMSFNCRichEditorDevGuide.pdf
TMSFNCSearchListDevGuide.pdf
TMSFNCSpreadGridDevGuide.pdf
TMSFNCTableViewDevGuide.pdf
TMSFNCTreeViewDevGuide.pdf
TMSFNCUIPackDevGuide.pdf

LCLTMSFNCUIPackPkg.lpk
LCLTMSFNCUIPackPkgDE.lpk
```

The `install.txt` for the UI pack is slightly different

INSTALL THEM IN TO THE LAZARUS IDE
In the IDE, select File, (*Figure 4*)
Open and browse for `LCLTMSFNCUIPackPkg.lpk`
and select "Open Package" when prompted
From the package window select **Compile** (*Figure 5*)
In the IDE, select File,
Open and browse for `LCLTMSFNCUIPackPkgDE.lpk`
and select "Open Package" when prompted
From the package window **select Use, Install** (*Figure 5*)
And than the miracle happens:
you will have a cloud of components to work with. Under **Lazarus:** which means **Windows/Linux/macOS...** and for **Delphi**

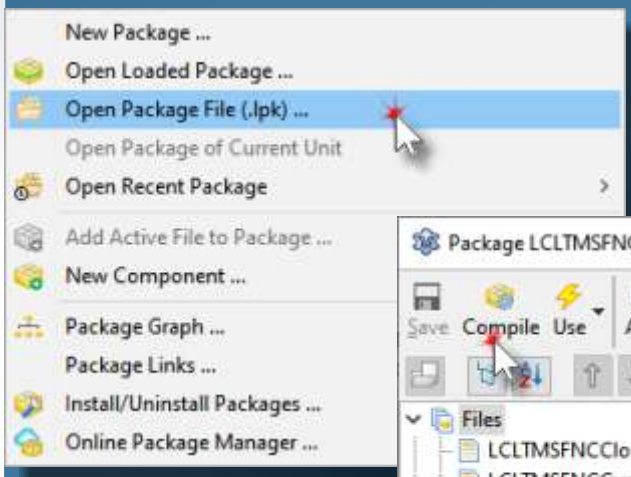


Figure 4: Open and browse

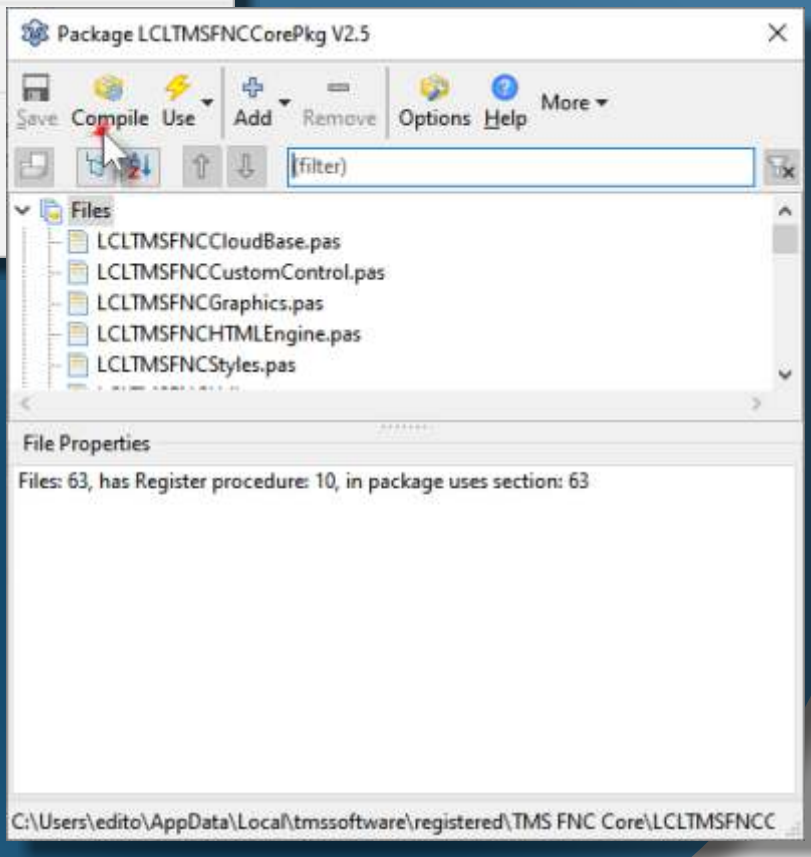


Figure 5: Compile

CREATE A SMALL PROJECT AND TEST IT.
Before I show how to create the project I want to explain something special which might be needed by the vast number of components for FNC. As you can see (in Figure 1: on page 1 and2 of the article) it is so much it is even confusing.

But there is a way to reorganize all to your wishes.
First of all: Lazarus has a special command for compiling without debugging.
Since it is much quicker than some other compilation commands it might be interesting to you:
Shift-Contrl-F9 is the direct command, but if you want to create a button for it like I did you can go to **Tools** and change some settings. Go to the **IDE CoolBar**. At the bottom there is a button Configure.
After clicking that button there is a list: search for Run menu commands (or anything else you like to do) Choose Run without debugging. In Figure… you'll see a new command icon.

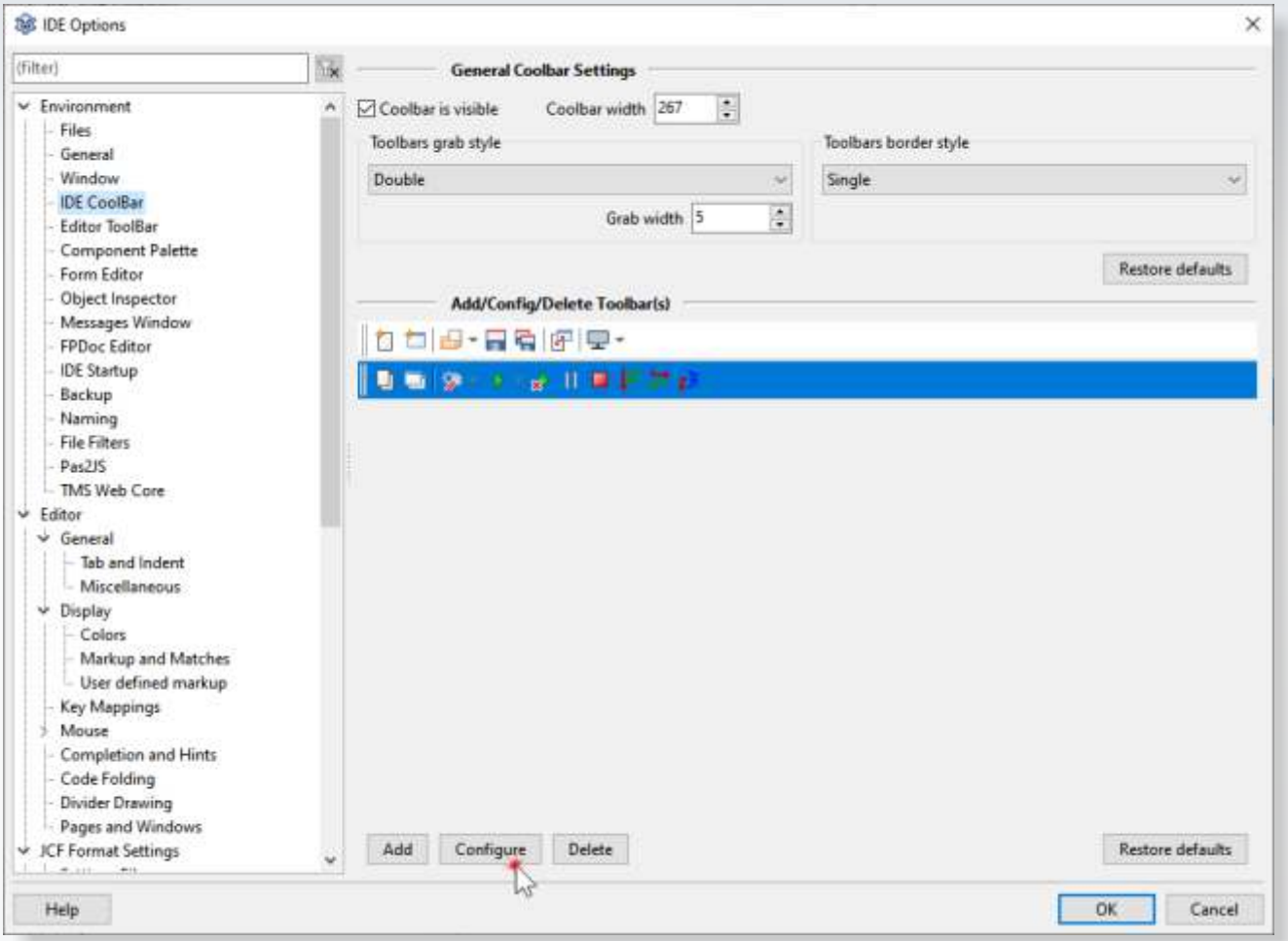


Figure 6: Add new commands

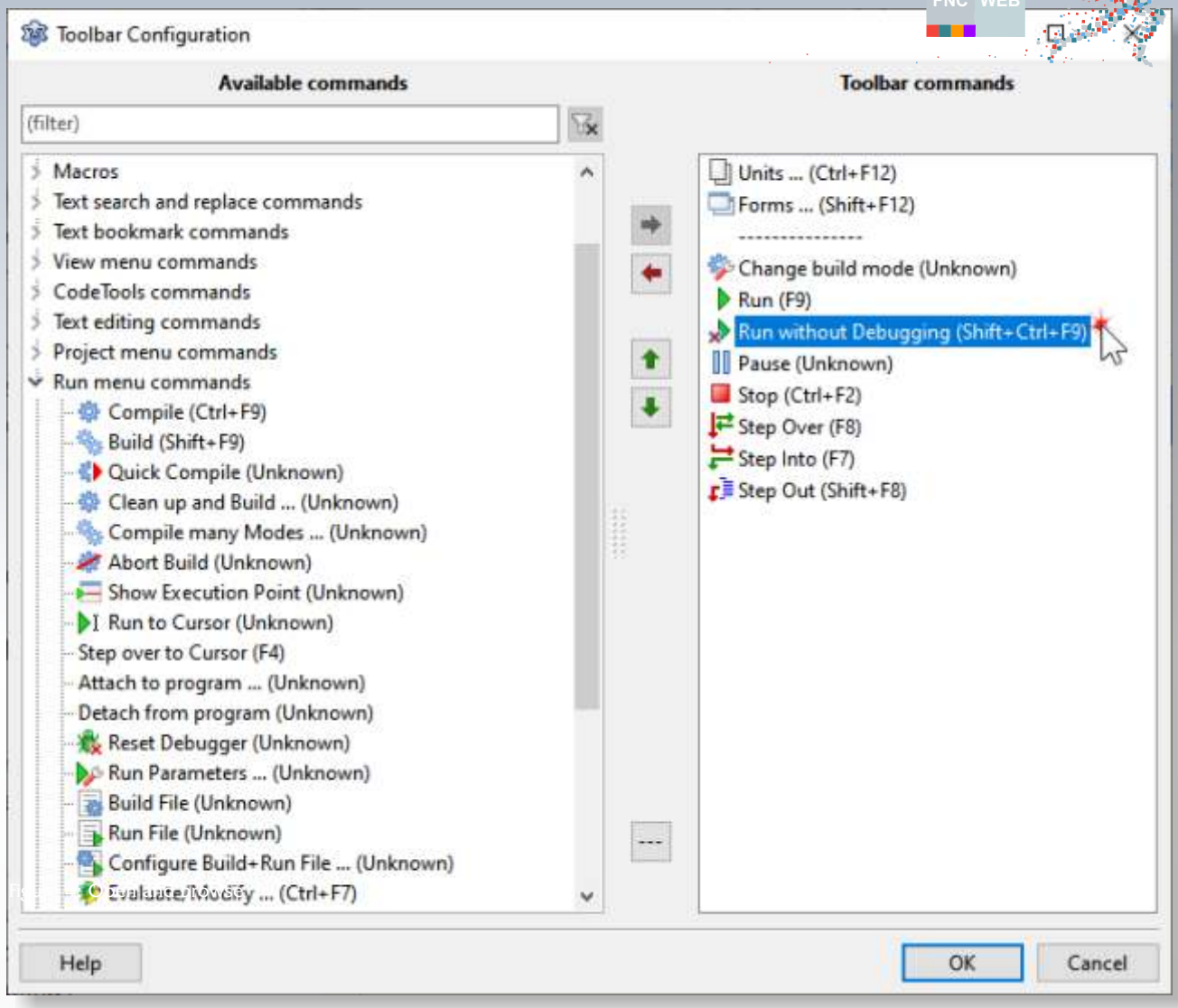


Figure 7: Adding the new command

Run without Debugging:
select it and use the arrows for switching to the other column.
You can also change order of components.

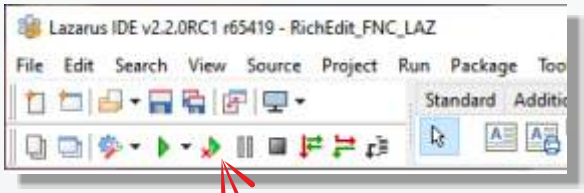


Figure 8: The new command

So now lets go for the real thing:
I would like to have a better overview of the components from **TMSFNC** group.

Again go to **Tools** and now choose **Component Palette...**
To make it easy to understand I'll simply first create a new Page (Tab) (*See Figure 9 and 10*) where my group of components will be added. **"TMSLazFncDetlef".** A fantasy name.

A window pops up where you can enter the name.
So now we have created the page, we can gather the components we want on that page (*as you can see in figure 11 on the next page*). You also can change the order of these pages. After finding the component you wanted to add, you simply drag and drop it in your page.
Fantastic!

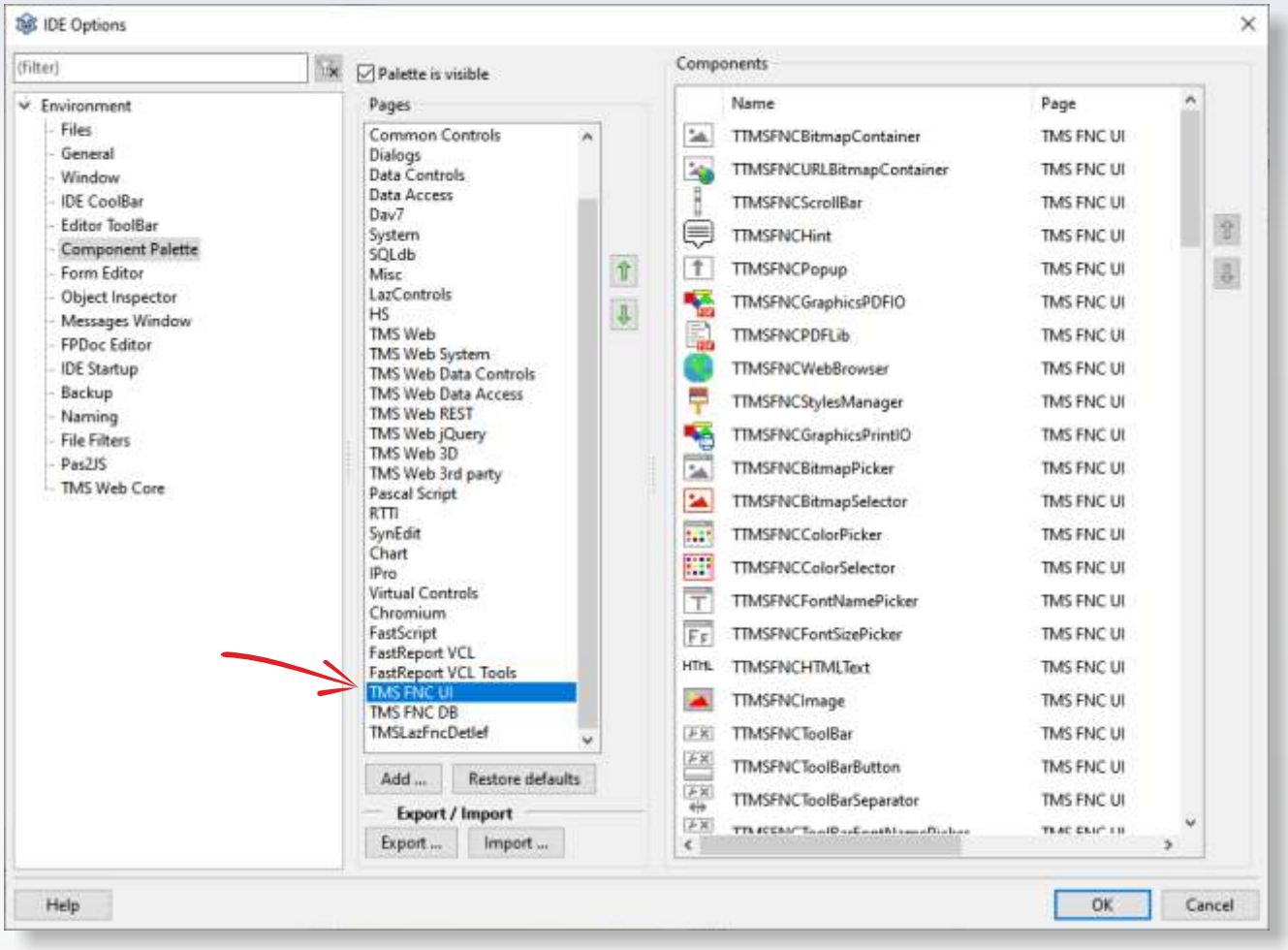


Figure 9: The Page to extract the components from

Figure 10: Enter the new page name

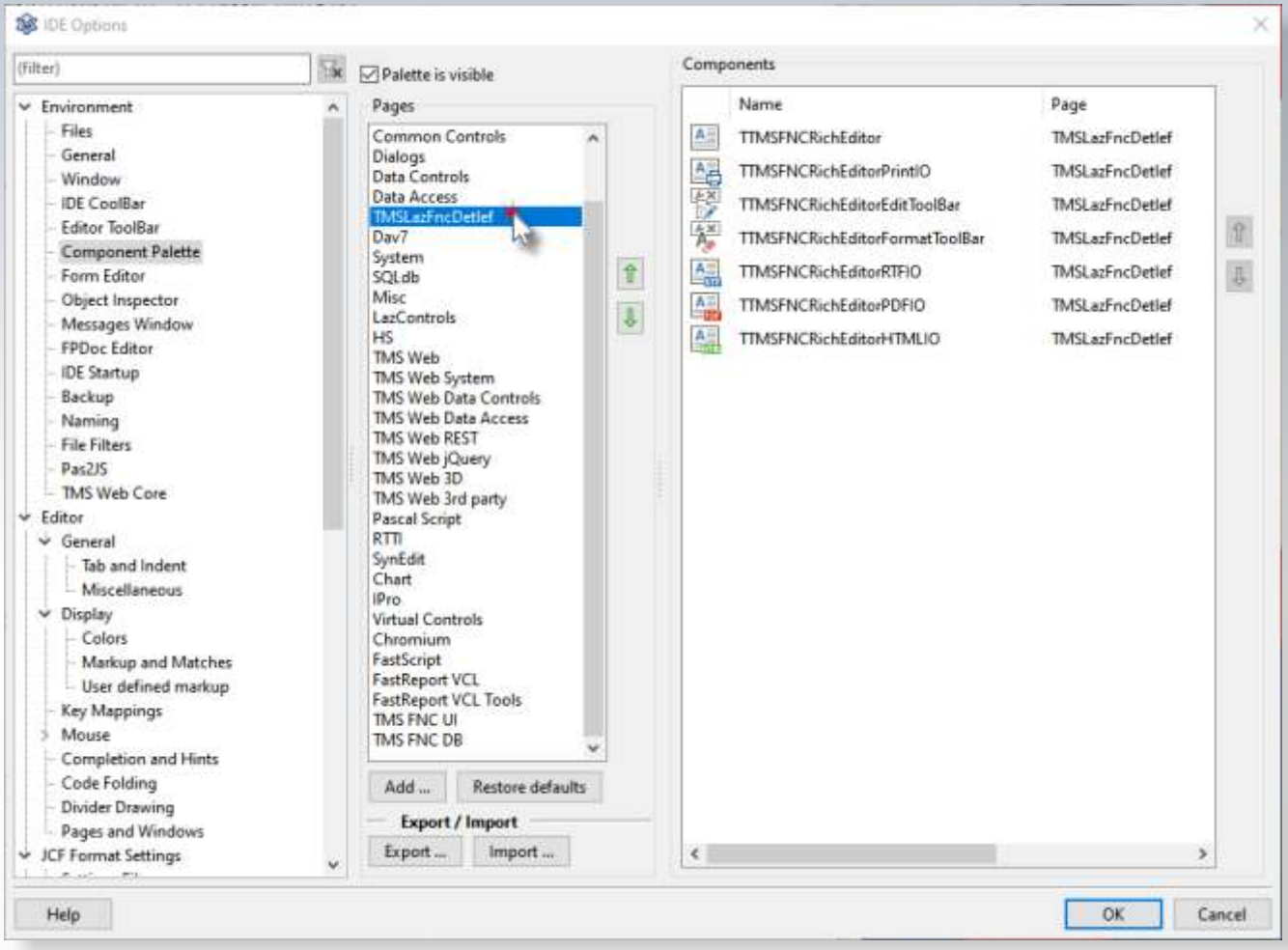


Figure 11: The new page and its content

Figure 12: The new page and its and its location that is now more close to the beginning of the page register

So we have now our wanted group of components and can easily find them.
Let's use the components in a new application.

NOW THE PROJECT:

We need only three components to create a fully working Rich Edit program. That said: it is not a sort of Word Application from Microsoft or alike Libre Office!

Create a new application and put on the form: `TTMSFNCRichEditor`
`TTMSFNCRichEditorEditToolBar`
`TTMSFNCRichEditorFormatToolBar`

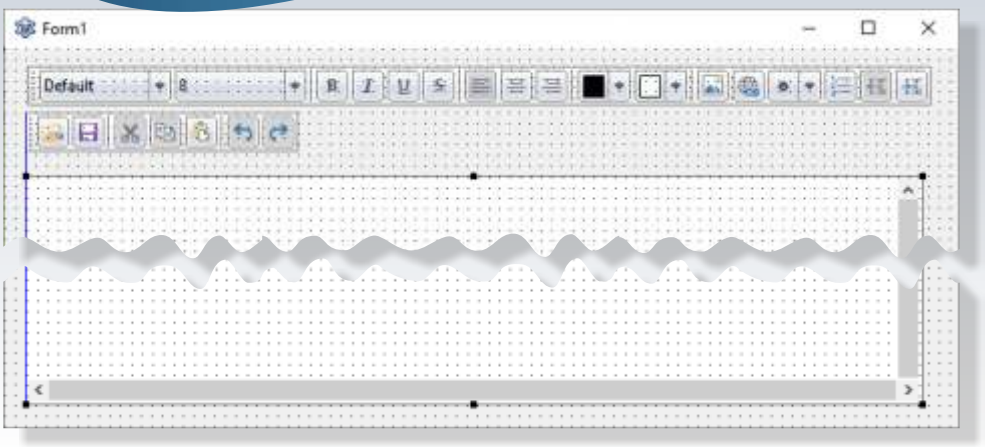


Figure 13:
The form with components

This is what the form now looks like. I have not used the Helpfile PDF that is availabel at:

`c:\Users\edito\AppData\Local\`
`tmssoftware\registered\TMS FNC UI Pack\`
`Doc\TMSFNCRichEditorDevGuide.pdf`

which contains the following items:

```
Description  Organization
Getting Started
Properties & Events
Methods
Programmatic access to the document
Using merge fields
Using accompanying toolbars
Importing & exporting in rich text
Importing & exporting in HTML format
Exporting in PDF format
```

There is a saving and an opening button and all the extra possibilities to edit your text.
I'd say what more do you want?
Go and play with it...

Figure 14:
The running Rich Edit App

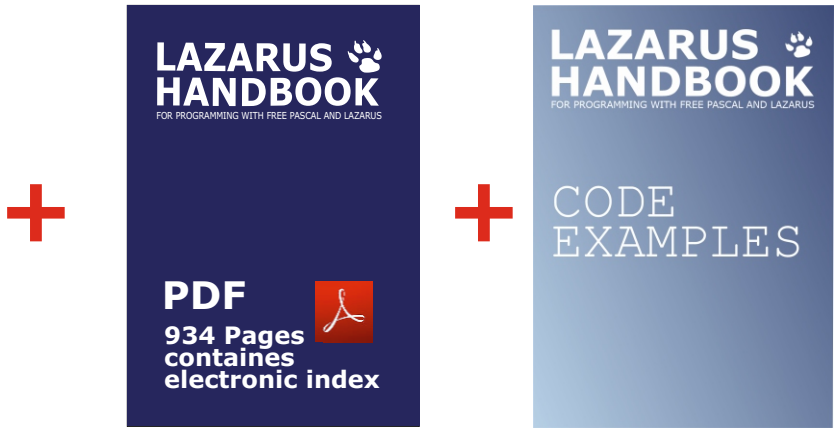
SUMMERSALE

The books

Sewn POCKET ,
almost thousand pages

written by the makers of FPC and Lazarus

934 Pages in two books 40 € (euro)

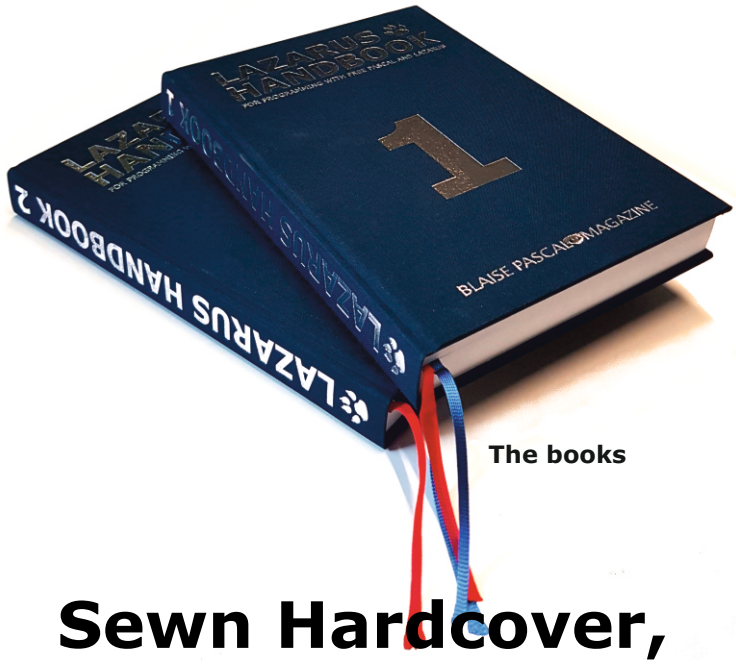


Including the PDF and Code Examples

`https://www.blaisepascalmagazine.eu/product/lazarus-handbook-pocket/`

We have a new service at our website, for shipping you can make three choices:
1.The shipping cost depending on which part of the world you want the book to be shipped:
Europe or the other countries: the World

Summersale

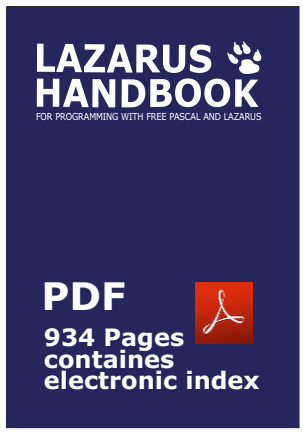


The books

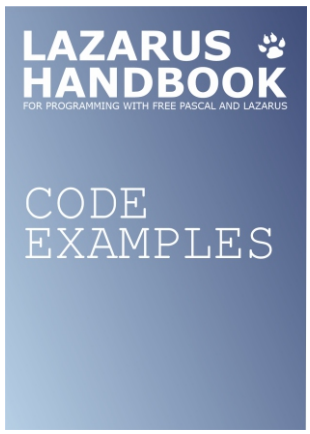
Sewn Hardcover, almost thousand pages

written by the makers of FPC and Lazarus

934 Pages in two books **65 €** (euro)



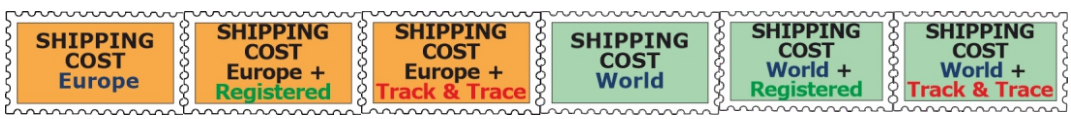
+



Including the PDF and Code Examples

https://www.blaisepascalmagazine.eu/product/lazarus-handbook-hardcover/

We have a new service at our website, for shipping you can make three choices:
1.The shipping cost depending on which part of the world you want the book to be shipped:
Europe or the other countries: the World



ADDING PALETTE ICONS TO YOUR COMPONENTS UNDER LAZARUS

starter expert

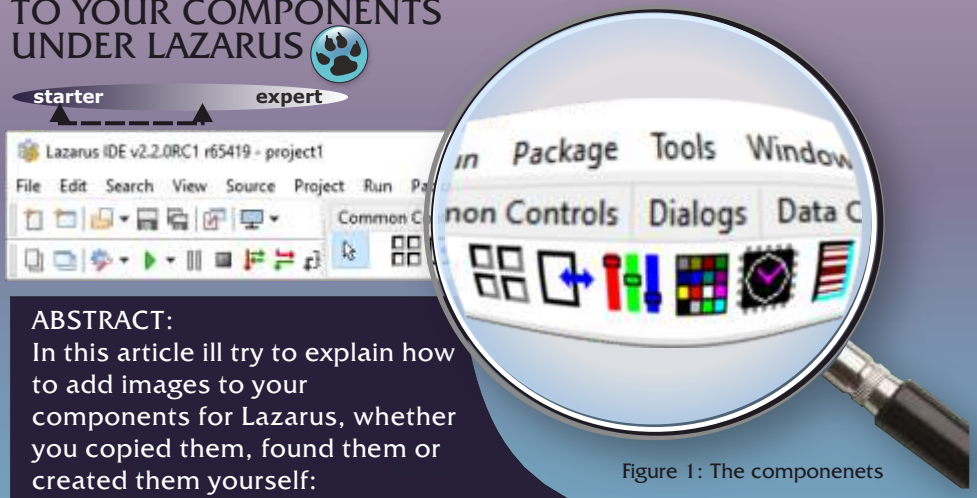


Figure 1: The componenets

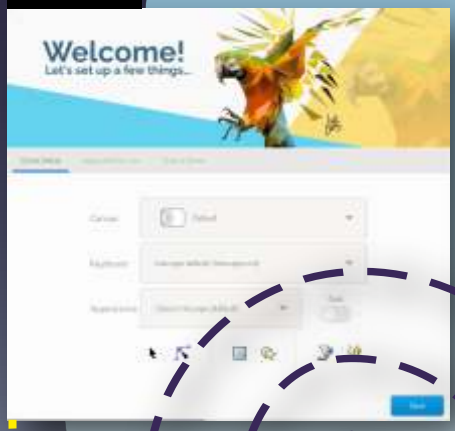
ABSTRACT:
In this article ill try to explain how to add images to your components for Lazarus, whether you copied them, found them or created them yourself:
Ill start with creating them myself.

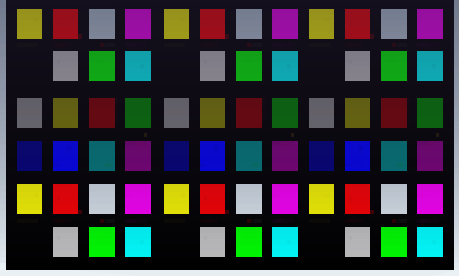
INTRODUCTION:

If you create icons for your program its not such an easy task Youll have to get prepared for that. There are free programs e can use and try.
We need any program which allows you to create .icon files and PNG files at pixel level.
That is because we have to create 3 different sizes for each.

Figure 2: InkScape

I suggest “inkscape”
https://inkscape.org/
It is available for all platforms.
The program is free and has very good possibilities to do image and icon editing and has all export features you would want. In the next issue I’ll write more explicitly about it.
You can create your icons or even start using them from the icon library I published.
For Subscribers it here:
Issue89_Blaise_Pascal_Magazine_FreeIcons.zip





In this case I'll add the specially designed png files. You can start with them of course.
If you try the **"inkscape"** program be sure to choose the blocked version:

This is because you could start with an icon and it otherwise would be blurred or vague...
So to start with: Load the `TestIcon.icon - 24x24 pixels`. Its all within the project path.

Figure 3:Choose your GUI color

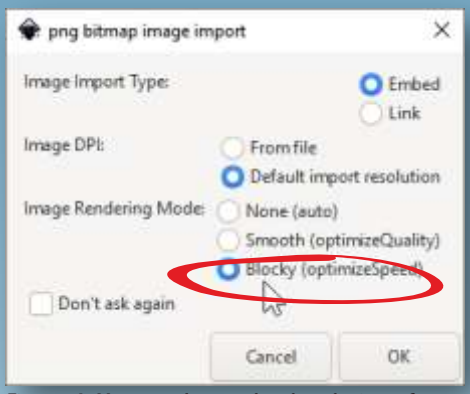


Figure 4: You need to make the choice of **Blocky(optimizeSpeed)**

Figure 5; close all applications

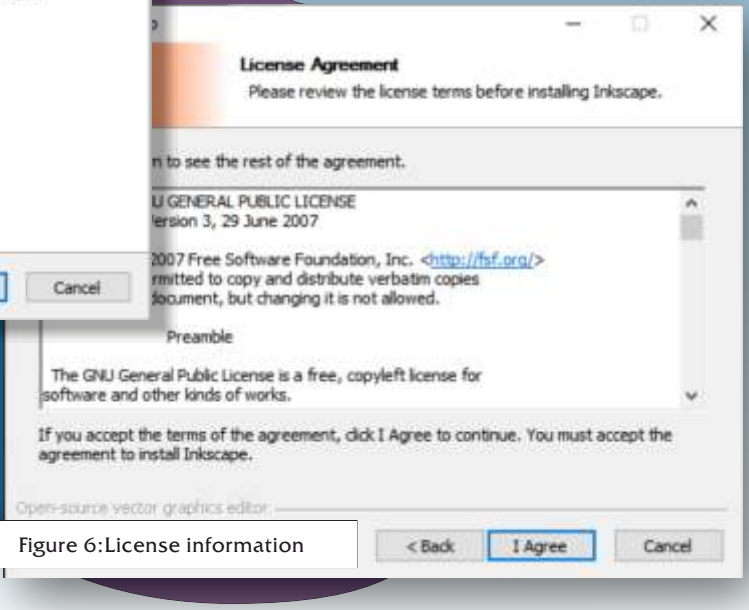


Figure 6:License information

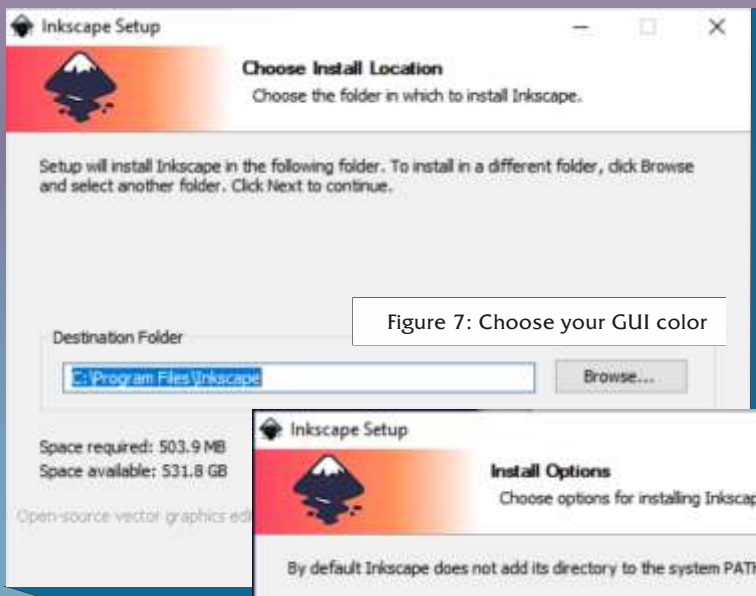


Figure 7: Choose your GUI color

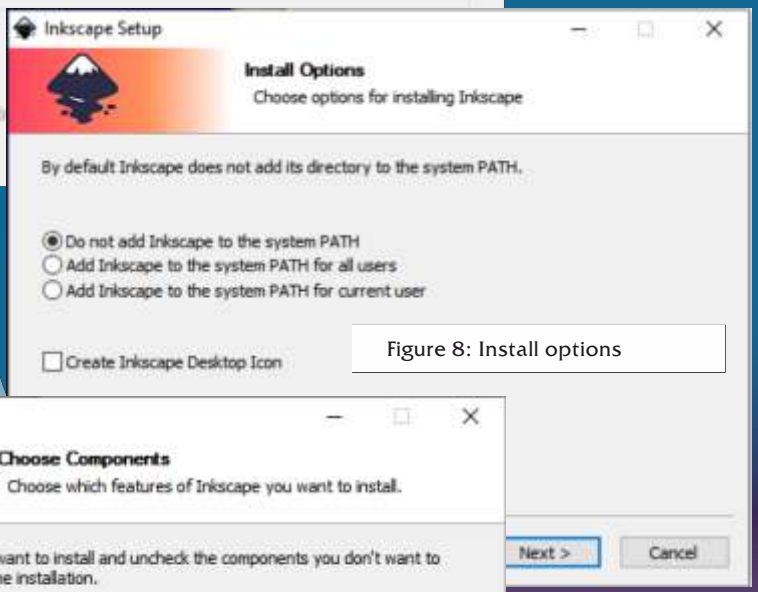


Figure 8: Install options

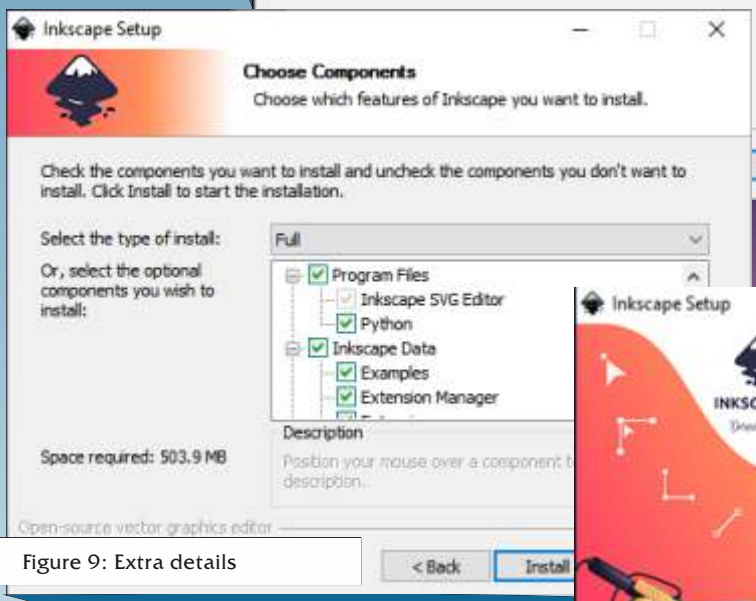


Figure 9: Extra details

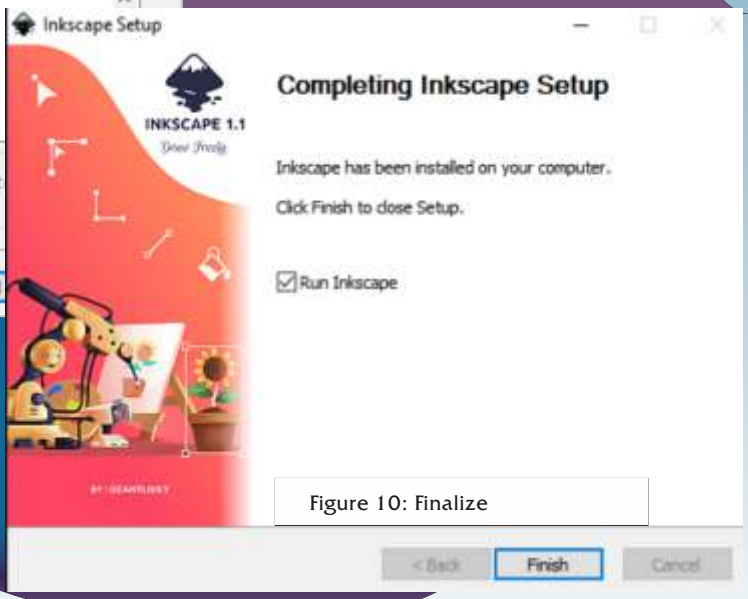


Figure 10: Finalize

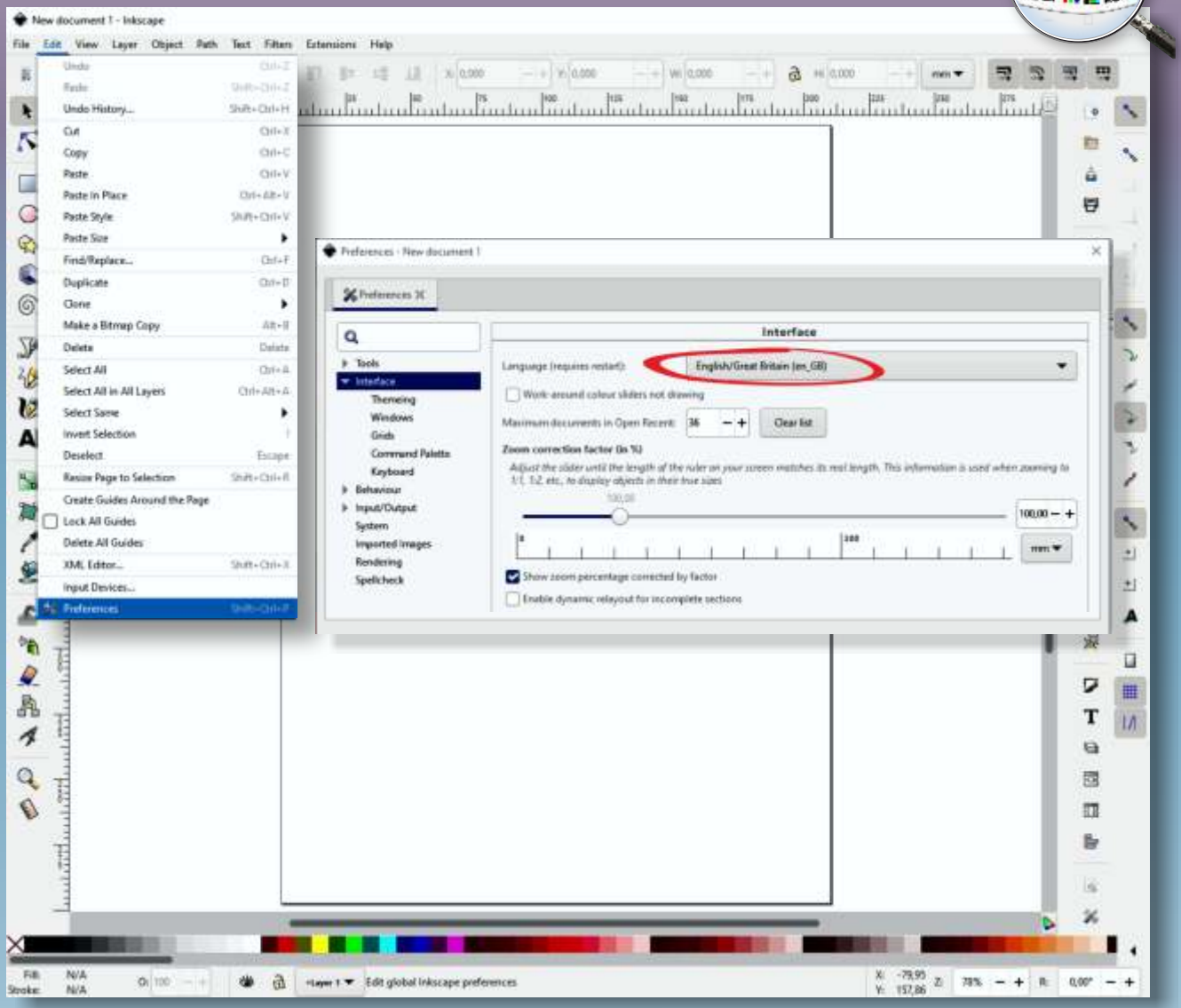


Figure 11: Preferences for Language

ADDING A PALETTE ICON

- In order to set up a dedicated palette icon for the new component we must create a bitmap image at a size of 24x24 pixels. The **PNG** format is preferred because of its nice alpha-channel transparency, but BMP or XPM are valid formats too.

- You must ensure that you name the icon file identically to the name of the component class (except for the image-file extension of course).

- On monitors of higher resolution than 96 ppi the palette icons will be scaled to a larger size automatically. In Lazarus versions before 1.8, this caused the images to become very pixelated. In newer Lazarus versions, you can provide additional larger images for better scaling.

Some years ago I bought a small program called Microangelo Studio.
`http://microangelo.us/free-icon-editor-download.asp`.
It is very easy to use and works as I want it: quick.

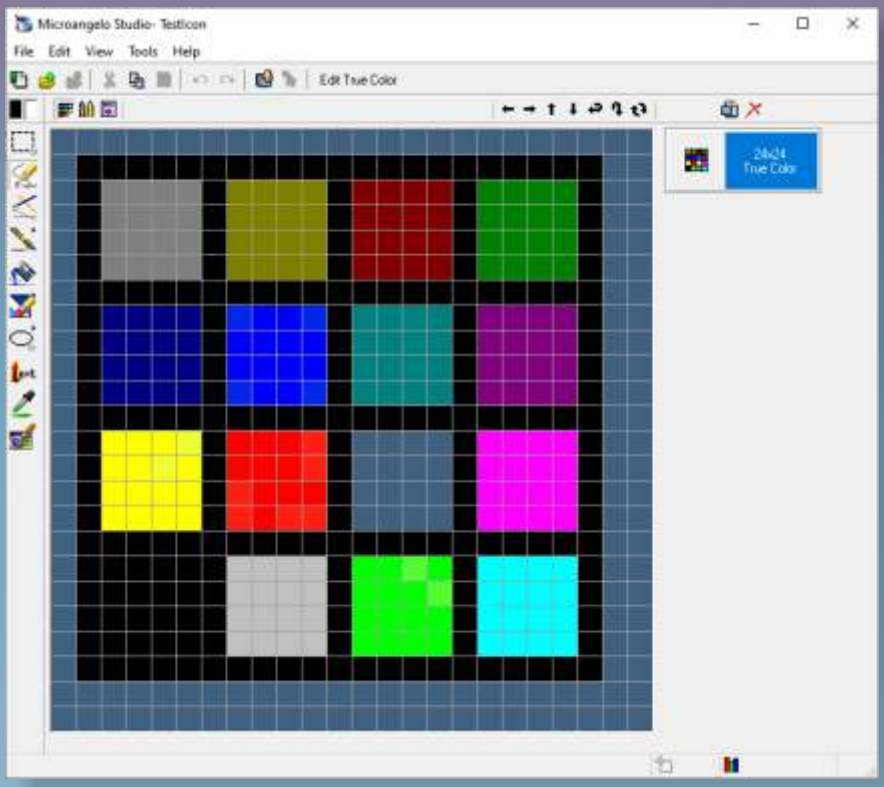


Figure 12: Michael Angelo is a very good editor at pixel level. Just try it!

To be specific, you should also design icons at 36x36 and 48x48 pixels for monitors at 144 ppi (150%) and 196 ppi (200%), respectively. For the IDE to recognize the presence of these additional images, the percentage resolution (with a spacing underline) _150 or _200, must be appended to the filename. This means that the required filenames of PNG images for the TDAV component must be:

• `TDAV7COLORMIXER.png`	`TDAV7COLORMIXER_150.png`	`TDAV7COLORMIXER_200.png`
• `TDAV7COLORPICKER.png`	`TDAV7COLORPICKER_150.png`	`TDAV7COLORPICKER_200.png`
• `TDAV7ELBOX.png`	`TDAV7ELBOX_150.png`	`TDAV7ELBOX_200.png`
• `TDAV7TIMER.png`	`TDAV7TIMER_150.png`	`TDAV7TIMER_200.png`
• `TDAVARRAYBTN.png`	`TDAVARRAYBTN_150.png`	`TDAVARRAYBTN_200.png`
• `TDAV7ROTATIONBTN.png`	`TDAV7ROTATIONBTN_150.png`	`TDAV7ROTATIONBTN_200.png`

To simplify the artwork it is recommended to draw the images as scalable vector graphics (for instance using the open source Inkscape program, https://inkscape.org/) from which you can easily export image files of different resolutions.

There is also the free XN Resource Editor available.

This is the standard work we use for creating the images.
The next task is to assemble the palette icons into a resource file. While Lazarus resources in the LRS format were used in older versions, nowadays the standard Windows Delphi-compatible RES format is more popular. This can also be done by XN Resource Editor.
Both kinds of resource files can be created with the Lazarus resource compiler named lazres which is available in the tools folder of any Lazarus installation:
`c:\lazarus\tools\lazres.exe`

The tool is provided as cross-platform source, not as a binary. So you have to compile this program for your own platform to create the executable before you can use it.
On Windows you write a batch file with the following content in order to build the resource file.
`c:\lazarus\tools\lazres.exe davidpkg.res @images.txt`

this is the text containing the images.txt:

```
TDAV7COLORMIXER.png
TDAV7COLORMIXER_150.png
TDAV7COLORMIXER_200.png
TDAV7COLORPICKER.png
TDAV7COLORPICKER_150.png
TDAV7COLORPICKER_200.png
TDAV7ELBOX.png
TDAV7ELBOX_150.png
TDAV7ELBOX_200.png
TDAV7TIMER.png
TDAV7TIMER_150.png
TDAV7TIMER_200.png
TDAVARRAYBTN.png
TDAVARRAYBTN_150.png
TDAVARRAYBTN_200.png
TDAV7ROTATIONBTN.png
TDAV7ROTATIONBTN_150.png
TDAV7ROTATIONBTN_200.pn
```

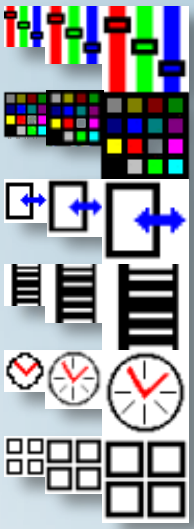


Figure 13: The created icons in .png format

it needs to be noted in the same way!
On Linux you can use the same file content as a shell script if you precede it with the usual `shebang line: #!/bin/sh`

The first parameter following the executable name is the name of the resource file to be generated. You can name it whatever you like, but its extension determines whether a resource file in the RES or in the older LRS format is to be built.
The names of the image files follow as additional parameters.
If you have to include many images in the resource file you can also specify, after an @character, the name of a file containing the image file names, one per line:
lazres palette_images.res @images.txt
Note that in these examples all files are expected to be in the same directory, otherwise you must add path specifications.

The resource file is created when you execute the batch file/shell script.
If you have specified the RES format, you would add the directive
{$R palette_images.res}
to the unit, usually at the beginning of the implementation section.

If you specified the LRS format, you must add an initialization section at the end of the component unit so that the resource is added as an include file:

```
initialization
{$I palette_images.lrs}
```

The compiler looks for the resource file in the package file's folder. If the resource file is elsewhere, you must specify the correct path, remembering that paths should be relative to the package file. When you now do a clean recompilation of the package, the correct palette icon (*rather than the generic default*) will appear on the component palette.

THE LIST OF STEPS TO DO:

❶ Go to : **`your path..\Components\`** here are all the files including the images for the components

❷ Create a bat file **`make_res.bat`** containing **`lazres palette_images.res @images.txt`**
If you open the file you can see what it exactly contains. To be able to start that file you need to be in the right dir and open a Command prompt or a terminal (Linux or macOS)

❸ Create an images.txt file with the help of Notepad (or alike). Containing:

```
TDAV7COLORMIXER.png
TDAV7COLORMIXER_150.png
TDAV7COLORMIXER_200.png
TDAV7COLORPICKER.png
TDAV7COLORPICKER_150.png
TDAV7COLORPICKER_200.png
TDAV7ELBOX.png
TDAV7ELBOX_150.png
TDAV7ELBOX_200.png
TDAV7TIMER.png
TDAV7TIMER_150.png
TDAV7TIMER_200.png
TDAVARRAYBTN.png
TDAVARRAYBTN_150.png
TDAVARRAYBTN_200.png
TDAV7ROTATIONBTN.png
TDAV7ROTATIONBTN_150.png
TDAV7ROTATIONBTN_200.png
```

❹ execute the following file from the dir where the file is located

```
c:\lazarus\tools\lazres.exe davidpkg.res @images.txt
```

This is are pieces of code you could find in the project of the component:

```
procedure Register;
begin
 RegisterComponents('Dav7',[TdavArrayBtn]);
 RegisterComponents('Dav7',[TDav7ELbox]);
 RegisterComponents('Dav7',[TDav7ColorMixer]);
 RegisterComponents('Dav7',[TDav7ColorPicker]);
 RegisterComponents('Dav7',[TDav7Timer]);

end;

procedure Register;
begin
 RegisterComponents('Dav7',[Tdav7RotationBtn]);

end;
```

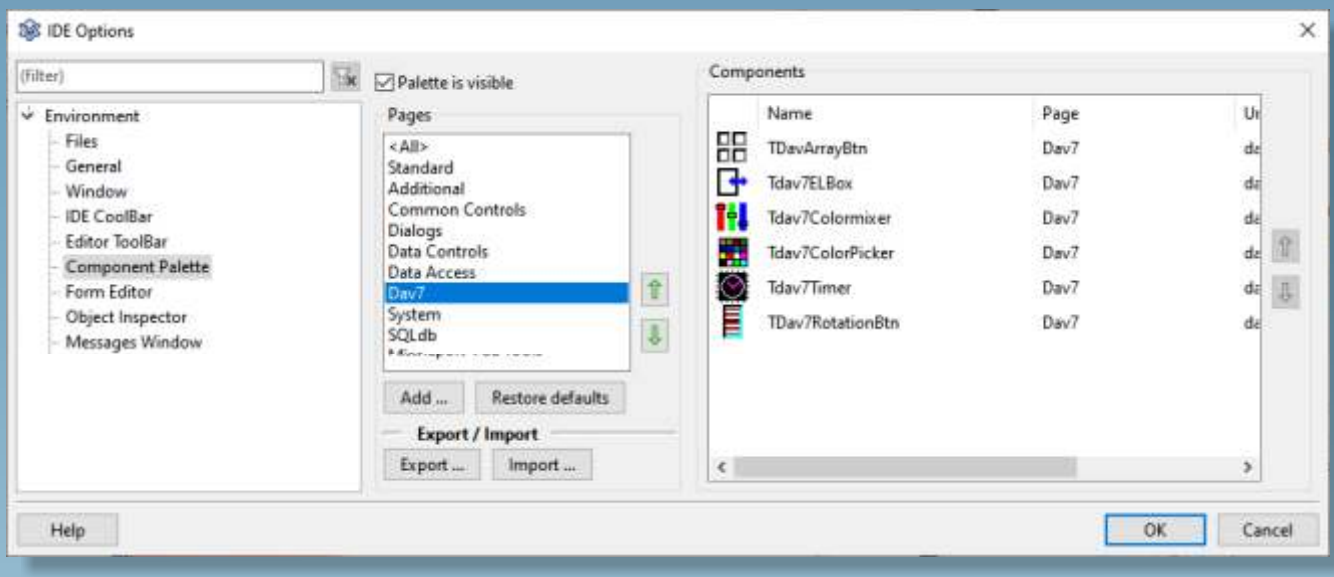


Figure 14: The component palette

If you right click in the **Component Palette** or go : **Tools | Component Palette** this window pops up, which gives you an overview of the new items.

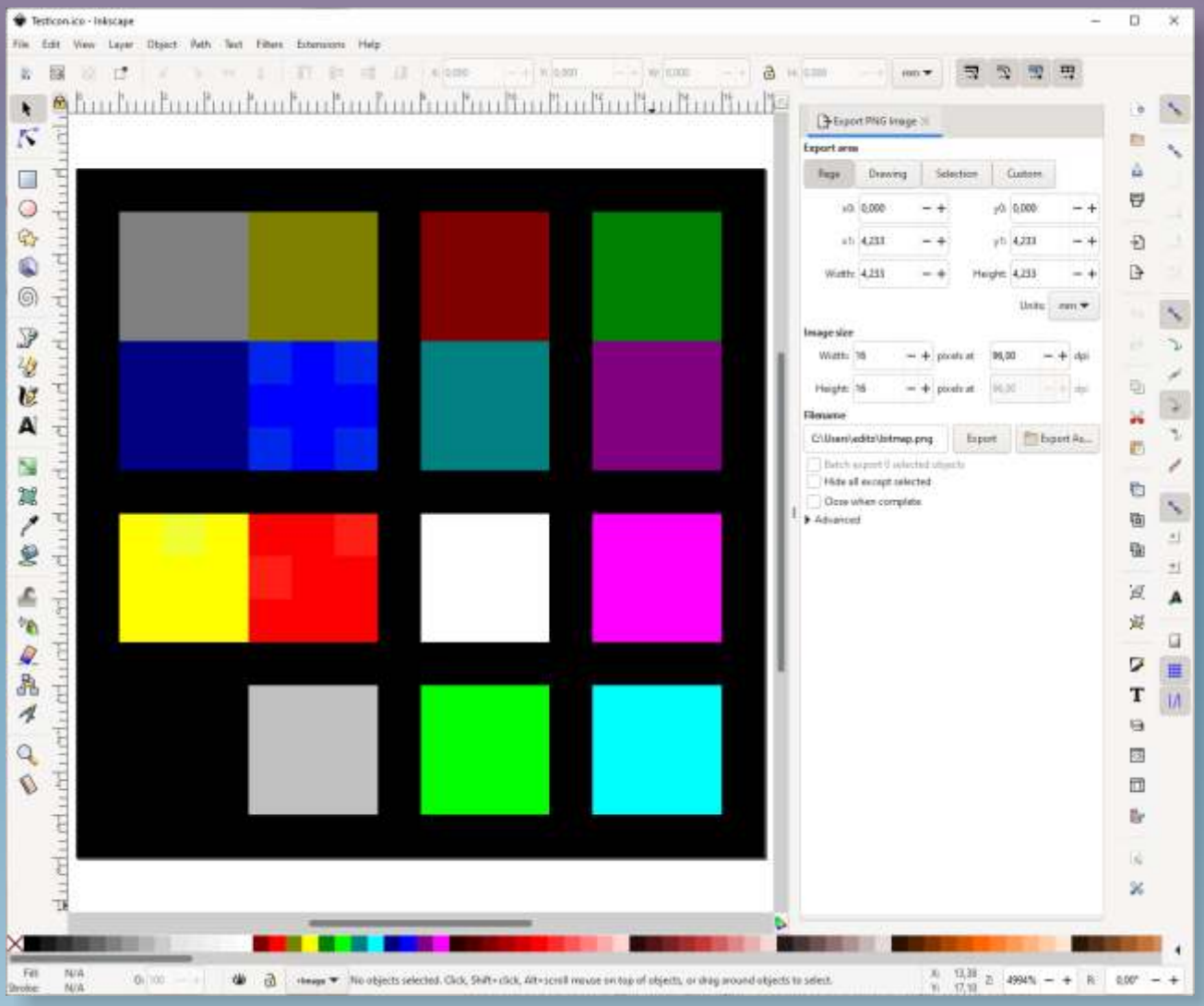


Figure 15: The “incsape” program can do it, but has a larger learning curve.

If you really want a good program for evereything I advise you to have alook at Affinity. The Affinity Desginer cost only € 54,99 : `https://affinity.serif.com/en-gb/`

kbmMW Community Edition v. 5.16.00 released for Delphi 10.4.2 Sydney!!

THIS IS A MAJOR NEW RELEASE WITH THE INTRODUCTION OF OUR NEW CONTEXT SENSITIVE I18N INTERNATIONALIZATION FRAMEWORK.

Components4Developers has decided to let the world have the opportunity to play with the hundreds of thousands of lines of code, containing thousands of functions, procedures, methods, classes and types, covering the vast number of features that makes kbmMW the leader of n-tier development environments.

We have now released our **second Community Edition** which contains almost all features from kbmMW Enterprise Edition, but limited in some areas, partly due to technical limitations, partly due to a few artificial limitations.
It further contains kbmMemTable Community Edition which is a direct match of kbmMemTable Standard Edition.

Community Edition is FREE to use as long as the applications it is used in provides a direct or indirect revenue not reaching US$5000 and the company distributing/producing the applications have a yearly turnover of less than US$50000.

Read the `license.txt` file for all details.

Community Edition can only be used for producing 32 bit Windows applications, but you have access to SmartServices, REST, WIB, kbmMemTable, SQL, ORM, SmartEvent, SmartBinding, Scheduler, Logging, Configuration, Ciphers, transports, XML, JSON, YAML, BSON, MessagePack, i18n and the countless of other features you have read about here on the blog!

Community Edition can thus also be used as a 60 day trial for kbmMW Enterprise Edition.
Community Edition does not contain source code, and will only work together with the version of Delphi it was released for.

Current release match latest Delphi Sydney 10.4.2, including Community Edition.

If you want go further, then purchase kbmMW Enterprise Edition which really is the best value for money and while being the oldest existing, mostly backwards compatible, n-tier framework for Delphi, it is also the only one bringing you to the bleeding edge of current technology.

Sign up at **`https://portal.components4developers.com and download immediately.`**

—-
kbmMW is the premiere **n-tier product** for Delphi, C++Builder and FPC on Win32, Win64, Linux, Java, PHP, Android, IOS, .Net, embedded devices, websites, mainframes and more. Please visit
`http://www.components4developers.com for more information about kbmMW.`

Components4Developers is a company established in 1999 with the purpose of providing high quality development tools for developers and enterprises. The primary focus is on SOA, EAI and systems integration via our flagship product kbmMW.

kbmMW is currently used as the backbone in hundreds of central systems, in hospitals, courts, private, industries, offshore industry, finance, telecom, governments, schools, laboratories, rentals, culture institutions, FDA approved medical devices, military and more.

COMPONENTS 4 DEVELOPERS

DX

https://thingsbiological.wordpress.com/2011/08/24/continued-bumblebee-decline/

KBMMW PROFESSIONAL AND ENTERPRISE EDITION V. 5.16.00 RELEASED!

- **RAD Studio XE5 to 10.4.2 Sydney supported**
- Win32, Win64, Linux64, Android, IOS 32, IOS 64 and OSX client and server support
- Native high performance 100% developer defined application server
- Full support for centralized and distributed load balancing and failover
- Advanced ORM/OPF support including support of existing databases
- Advanced logging support
- Advanced configuration framework
- Advanced scheduling support for easy access to multithread programming
- Advanced smart service and clients for very easy publication of functionality
- High quality random functions.
- High quality pronouncable password generators.
- High performance LZ4 and Jpeg compression
- Complete object notation framework including full support for YAML, BSON, Messagepack, JSON and XML
- Advanced object and value marshalling to and from YAML, BSON, Messagepack, JSON and XML
- High performance native TCP transport support
- High performance HTTPSys transport for Windows.
- CORS support in REST/HTML services.
- Native PHP, Java, OCX, ANSI C, C#, Apache Flex client support!

kbmMemTable is the fastest and most feature rich in memory table for Embarcadero products.

- **Easily supports large datasets with millions of records**
- **Easy data streaming support**
- **Optional to use native SQL engine**
- **Supports nested transactions and undo**
- **Native and fast build in M/D, aggregation/grouping, range selection features**
- **Advanced indexing features for extreme performance**

THIS IS A MAJOR NEW RELEASE WITH THE INTRODUCTION OF OUR NEW CONTEXT SENSITIVE I18N INTERNATIONALIZATION FRAMEWORK.

- New I18N context sensitive internationalization framework to make your applications multilingual.
- New ORM LINQ support for Delete and Update.
- Comments support in YAML.
- New StreamSec TLS v4 support (by StreamSec)
- Many other feature improvements and fixes.

Please visit
http://www.components4developers.com
for more information about kbmMW

- High speed, unified database access (35+ supported database APIs) with connection pooling, metadata and data caching on all tiers
- Multi head access to the application server, via REST/AJAX, native binary, Publish/Subscribe, SOAP, XML, RTMP from web browsers, embedded devices, linked application servers, PCs, mobile devices, Java systems and many more clients
- Complete support for hosting FastCGI based applications (PHP/Ruby/Perl/Python typically)
- Native complete AMQP 0.91 support (Advanced Message Queuing Protocol)
- Complete end 2 end secure brandable Remote Desktop with near realtime HD video, 8 monitor support, texture detection, compression and clipboard sharing.
- Bundling kbmMemTable Professional which is the fastest and most feature rich in memory table for Embarcadero products.

COMPONENTS 4 DEVELOPERS

EESB, SOA,MoM, EAI TOOLS FOR INTELLIGENT SOLUTIONS. kbmMW IS THE PREMIERE N-TIER PRODUCT FOR DELPHI / C++BUILDER