//

Multicode

maXbox Starter 119 – Get a different step Solution.

"Code behaves like recording – recoding from brain to tool.

Source: 393_QRCode5_64TXT_211_7_tutor119.TXT

Sometimes life is like playing with us: You write some code that solves a problem, and then someone comes along and makes the problem harder. Here's how to continuously integrate new solutions without having to rewrite your old solutions (as much).
Let's start with a simple problem: You've written and testing a class that returns a QR-Code from a text line:

```
  aQRCode:= TDelphiZXingQRCode.Create;
  QRCodBmp:= TBitmap.Create;
  form1:= getform2(700,500,123,'QR Draw PaintPerformPlatform PPP5');
  try
    aQRCode.Data:= aTextline;
    aQRCode.Encoding:= qrcAuto; //TQRCodeEncoding(cmbEncoding.ItemIndex);
```

That class is currently used by a single application in a scripting environment. Inside the class you might have Delphi.VCL or ADO.NET code or LINQ/Entity Framework code, but either way, you're accessing an **internal** component on your operating system on your desktops hard-disk. Then you realize that, because your component data/algorithm doesn't change very often, as long as you don't update or rebuild your component, so your are not up to date with newer features like encoding or error correcting level.
This leads to a second solution of an **external** service call like Google Chart Tools. Using the Google Chart Tools / Image Charts (aka Chart API) you can easily generate QR-codes, this kind of images are a special type of two-dimensional barcodes. They are also known as hard-links or physical world hyperlinks.
The Google Chart Tools also let you generate QR-code images using an HTTP POST or short messages. All do you need to generate a Qr-Code is make a get request to this URI:

http://chart.apis.google.com/chart?chs=200x200&cht=qr&chld=M&chl=Text

So we parametrize this URI[1] for a get request call:

```
Const
  URLGoogleQRCODE='https://chart.apis.google.com/chart?chs=%dx%d&cht=qr&chld=
%s&chl=%s';
```

The API requires 3 simple fields be posted to it:
cht=qr this tells Google to create a QR code;
chld=M the error correction level of the QR code (more later);
chs=wxh the width and height of the image to return (e.g. chs=250x250);
chl=text the URL encoded text to be inserted into the qr-code.

1 Uniform Resource Identifier

As the URL is https with a certificate, an application can identify himself and authenticate himself to any organization trusting the third party.
The second thing to consider is that (I assume) if the web service object is working right we can compare the resulting picture with the first solution. So lets make the call with WinInet Win-API:

```
procedure GetQrCodeInet(Width,Height:Word; C_Level,apath:string;
                                                    const Data:string);
var encodURL: string;
    pngStream: TMemoryStream;
begin
  encodURL:= Format(URLGoogleQRCODE,[Width,Height, C_Level, HTTPEncode(Data)]);
  pngStream:= TMemoryStream.create;
  HttpGet(encodURL, pngStream);   //WinInet
  try
    pngStream.Position:= 0;
    pngStream.savetofile(apath);
    sleep(500);
    OpenDoc(apath);
  finally
    //Dispose;
    pngStream.Free;
  end;
end;
```

And the call with wininet:

```
  GetQrCodeInet(150,150,'Q',ExePath+'examples\'+AFILENAME,QData);
  sleep(500);
  writeln('SHA1 '+sha1(ExePath+AFILENAME));     //}
  //SHA1 FE526D46BA48DFD820276872C969473A7B7DE91C
```

You should be able to see the content of the file with OpenDoc(apath);
and we pass AFILENAME= 'mX5QRCode5.png';

So what's the meaning of the SHA1 hash? For this we compare with a third solution of internet-call with the Indy10 framework. In comparison with Wininet as the internal WinAPI library, Indy is an external library also based on OpenSSL and we compare the result to get the same hash:

```
procedure GetQrCodeIndy(Width,Height: Word; C_Level,apath: string;
                                                    const Data: string);
var encodURL: string;
    idhttp: TIdHttp;// THTTPSend;
    pngStream: TMemoryStream;
begin
  encodURL:= Format(URLGoogleQRCODE,[Width,Height,C_Level, HTTPEncode(Data)]);
  idHTTP:= TIdHTTP.Create(NIL)
  pngStream:= TMemoryStream.create;
  idHTTP.Get1(encodURL, pngStream)
  //Exception: if not Dll-Could not load SSL library. at 827.447
```

```
    try
     pngStream.Position:= 0;
     writeln(itoa(pngStream.size));
     pngStream.savetofile(apath);
     sleep(500);
     OpenDoc(apath);
   finally
     idHTTP.Free
     idHTTP:= Nil;
     pngStream.Free;
   end;
end;
```

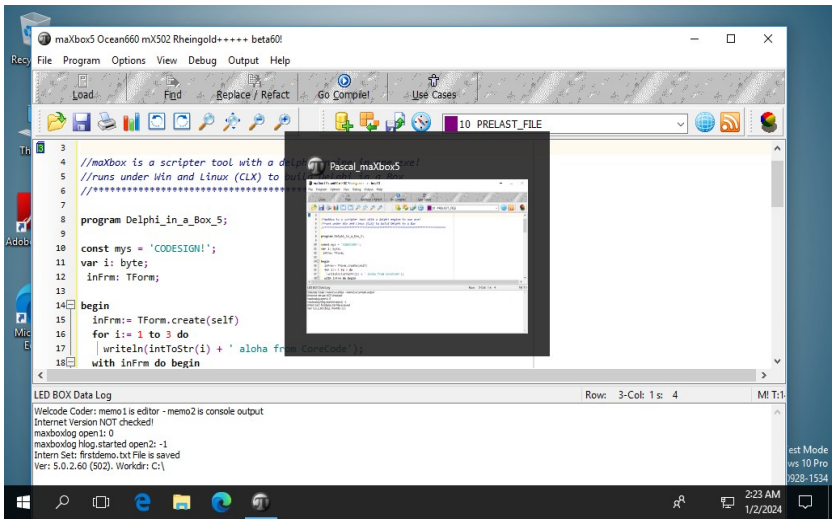
As I said we need two DLLs to support the OpenSSL lib; provided OpenSSL is installed in your system. The call arguments are the same so we get the same hash back:

```
  GetQrCodeIndy(150,150,'Q',ExePath+'examples\'+AFILENAME,QData);
  sleep(500);
  writeln('SHA1 '+sha1(ExePath+AFILENAME));      //}
```

intern: FE526D46BA48DFD820276872C969473A7B7DE91C
SHA1 FE526D46BA48DFD820276872C969473A7B7DE91C

openssl pkcs12 -info -in filename.p12

In cryptography, PKCS #12 defines an archive file format for storing many cryptography objects as a single file. It is commonly used to bundle a private key with its X.509 certificate or to bundle all the members of a chain of trust.



pic1: tutor119_signobjectscreen_6.png

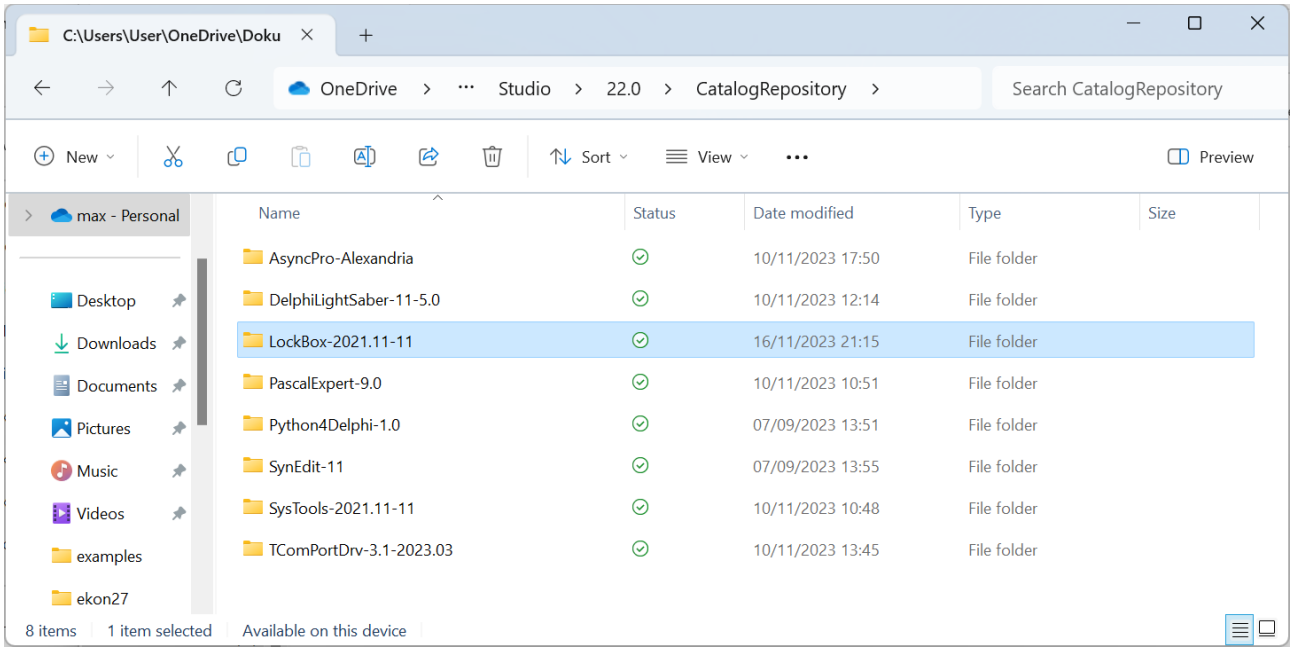
You can either sign files out of a working directory, or you can place them in your Windows SDK\bin folder.

Source Organisation for Multicode

This separation of now three solutions is reflected in a number of ways. The most important distinction is that the code schema for developers in the script division has now 3 folding sections and can be different from a current configuration design:

1. Solution 1 with the internal class needs no https and component.
2. Solution 2 is dependent from external Google API and based on internal OS WinInet.
3. Solution 3 is also dependent on Google but has its own internet suite as Indy 10 but dependent on OpenSSL.

This we should consider and document in our source code repository:



pic2: tutor119_catrepo.png

The interesting point is to know where the code is running and how it is stored in the executable or script itself. Embedding Wininet as one function is nice: HttpGet(encodURL, pngStream); *//WinInet*, but you don't have the flexibility to change for example request- or response-headers of the external web service you consume, so we test a forth solution in detail also to debug with more verbosity:

```
/4. Internal Class mXLib5 -GoogleAPI
procedure GetQrCodeWininetClass(Wid,Hei:Word; C_Level,apath:string;
                                                    const Data:string);
var httpR: THttpConnectionWinInet;
    ms: TMemoryStream;
    heads: TStrings; iht:IHttpConnection;
begin
  httpr:= THttpConnectionWinInet.Create(true);
  ms:= TMemoryStream.create;
  try
    //iht:= httpRq.setHeaders(heads);
    httpR.Get(Format(URLGoogleQRCODE,[Wid,Hei,C_Level,HTTPEncode(Data)]),ms);
    //httpRq.setRequestheader('x-key',aAPIkey);
    if httpr.getresponsecode=200 Then begin
      ms.Position:= 0;
      ms.savetofile(apath);
      sleep(500);
      OpenDoc(apath)
    end Else writeln('Failed responsecode:'+
                                  itoa(HttpR.getresponsecode));
  except
    //writeln('Error: '+HttpRq.GetResponseHeader(''));
```

```
    writeln('EHTTP: '+ExceptiontoString(exceptiontype, exceptionparam));
  finally
    httpr:= Nil;
    ms.Free;
  end;
end;
```

This is the good old Wininet API but this time as an object oriented class with methods and attributes, for example to check the response code of the get request. Also in this mapped import library we get the same hash: intern: FE526D46BA48DFD820276872C969473A7B7DE91C

My goal in refactoring or recoding is to supply "enough engineering" to support the current problem without over-engineering a solution to some later problem that might never exist. Said it another way: A solution is never more complicated than the problem it's solving. But you know it can be the other way round for example in cryptography: Simple problem but complex solution.
The fifth solution is tricky and only in an interpreted script possible, we call the second solution pre-compiled as one function.

5 Solutions Overview

- Internal QR-Component Class mXLib5 TDelphiZXingQRCode
- External call of script with procedure WinInet & Google API
- External call of script with Indy class & Google API
- Internal call of THttpConnectionWinInet class of external API
- Internal call direct in script (precompiled):
  `GetQrCode5(150,150,'Q',QDATA, ExePath+AFILENAME);`

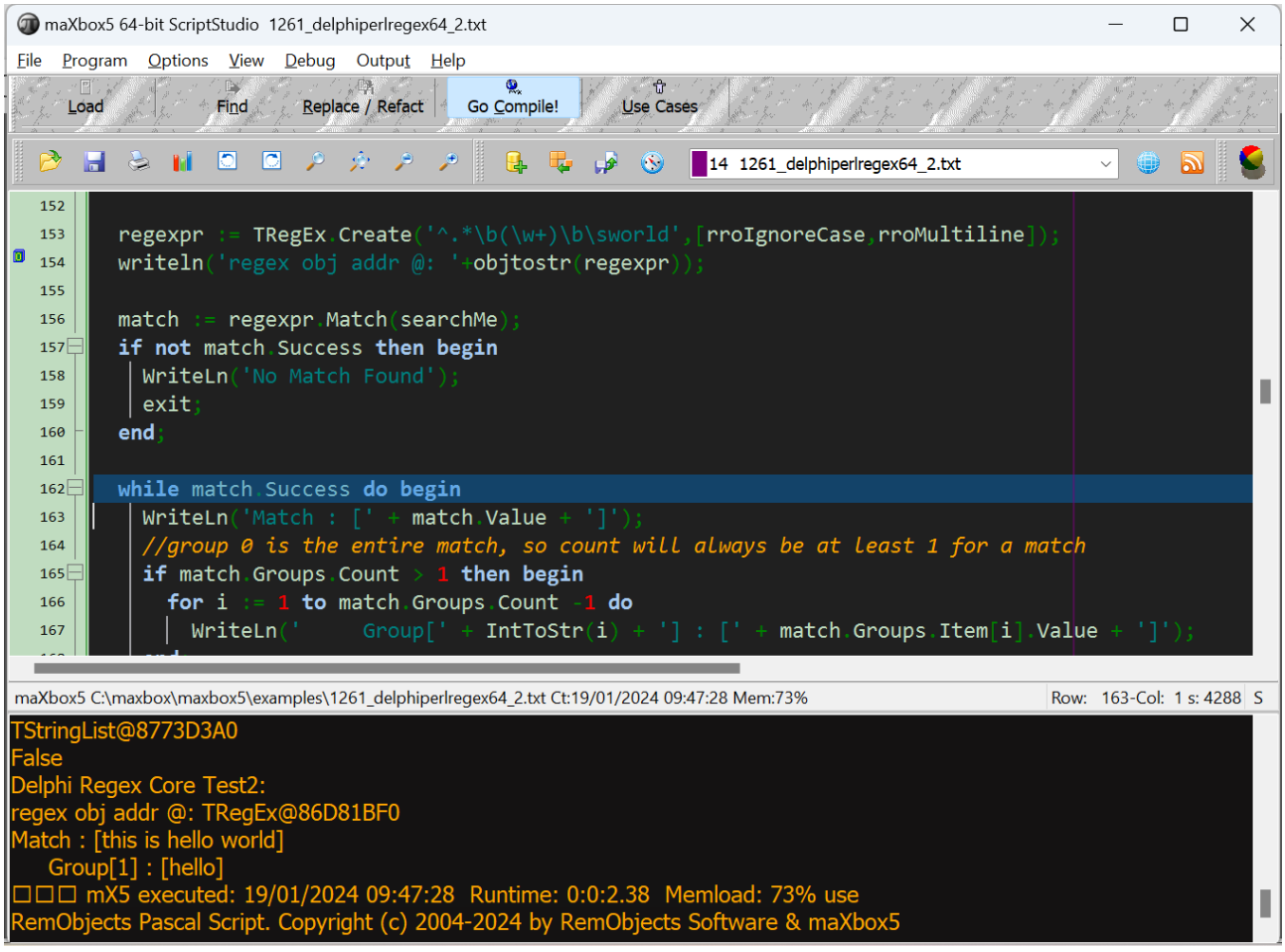
The Mystery of solution 6 and 7

To be really independent from internal calls and run just on runtime a late binding solution can be considered.
Early (or static) binding refers to compile time binding as before and late (or dynamic) binding refers to runtime binding (for example when you use reflection or retyping).
Late binding uses CreateObject to create and instance of the application object, which you can then control. For example, to create a new instance of WinHttp.WinHttpRequest.5.1 using late binding in our sixth solution:

```
function QRCcodeOle(Wid,Hei:Word; C_Level,apath:string;
                                                  const Data:string): string;
var httpReq,hr: Olevariant;  instream: IStream;
    jo: TJSON; strm :TMemoryStream;
begin
  httpReq:= CreateOleObject('WinHttp.WinHttpRequest.5.1');
  //jo:= TJSON.Create();
  hr:= httpReq.Open('GET', format(URLGoogleQRCODE,
                                  [Wid,Hei,C_Level,HTTPEncode(Data)]))
  httpReq.setRequestheader('content-type','application/octet-stream');
  //httpReq.setRequestheader('Authorization','Bearer '+ CHATGPT_APIKEY2);
  if hr= S_OK then
        HttpReq.Send();
  strm:= TMemoryStream.create;
```

```
    If HttpReq.Status = 200 Then begin
      try
        //
        strm:= getmemStreamfromIStream2(HttpReq.responsestream);
        //getmemStreamfromIStream2file(hrstream, apath);
        writeln('responsestream size: '+itoa(strm.size));
        strm.savetoFile(apath)
        openFile(apath);
      except
        writeln('EHTTPex: '+ExceptiontoString(exceptiontype, exceptionparam));
      finally
        strm.free;
        httpreq:= unassigned;
      end;
    end;
end;
```

This solution is the load of an IStream from an OLE response stream as unknown variant type to a well known TMemoryStream in order to save the response stream to a file (in our example the binary QR-code image file as a *.png graphic).



Pic3: tutor119_regex_multicod.png

The crux is the getmemStreamfromIStream2 function. I was probably not aware of TOleStream at the time I wrote this answer. Looking at TOleStream now, I notice that it does not support 64-bit streams. This code does. Other than that, this code is almost identical to the code

that TOleStream uses, with one only exception being that this code's implementation of the Size property getter is more optimized than TOleStream's implementation is, and this code implements the size property setter whereas TOleStream does not. So we can combine the invoke call from HttpReq.responsestream to get a file in one function:

```
function getmemStreamfromIStream2File(avariant: variant;
                                      apath: string): Tmemorystream;
 var instream: IStream; ostream: TStream;
 begin
    instream:= IUnknown(avariant) as IStream;
    ostream:= TOleStream.Create(instream);
    result:=  Tmemorystream.Create;
    try
      result.CopyFrom(OStream, OStream.Size);
      result.SaveToFile(apath)
    finally
      OStream.Free;
    end;
 end;
```

And the last one catches everything from external even the language, its a Python for Delphi Solution:

```
procedure PYdigitQRCode;
begin
  with TPythonEngine.Create(Nil) do begin
    pythonhome:= 'C:\Users\user\AppData\Local\Programs\Python\Python312\';
    UseWindowsConsole:= false;
    OnPathInitialization:= @TPathInitialization;
    try
      loadDLL;
      //opendll(PYDLL64)
      execstr('import sys, os, json, qrcode');
      println(EvalStr('sys.version'));
      println(EvalStr('sys.executable'));
      execstr('qr =
             qrcode.QRCode(error_correction=qrcode.constants.ERROR_CORRECT_Q)');
      execstr('qrcode.make("'+QDATA+'").save(".\examples\'+AFILENAME+'")');
           ///}
    except
      raiseError;
    finally
      unloaddll;
      free;
    end;
  end;
end;
```

3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)]
As a standard install uses pypng to generate PNG files and can also render QR codes directly to the console. A standard install is just:
pip install qrcode;

Conclusion

When it comes to problem-solving, there are often multiple solutions that can be used to solve the same problem. The choice of solution depends on various factors such as performance, storage, correctness, implement-

ation, simplicity, and also scaleability and security.
The Google Chart Tools (Chart API) also let you generate QR-code images using an HTTP POST call.
A Quick Response code is a two-dimensional pictographic code used for its fast readability and comparatively large storage capacity.
Early binding refers to assignment of values to variables during design time whereas late binding refers to assignment of values to variables during run time as a concept of multicode programming.
Implemented often using [special] dynamic types, introspection /reflection, flags and compiler options, or through virtual methods by borrowing and extending dynamic dispatching.

Script: softwareschule.ch/examples/qrcode8.txt
softwareschule.ch/examples/qrcode8.htm

References:

Compiled Project:

https://github.com/maxkleiner/maXbox4/releases/download/V4.2.4.80/maxbox5.zip

Free Automated Malware Analysis Service - powered by Falcon Sandbox (hybrid-analysis.com)

Topic:

https://stackoverflow.com/questions/4938601/getting-an-istream-from-an-olevariant

Preparation:

The Mystery of IStream – Code Blog

Doc and Tool: https://maxbox4.wordpress.com

Max Kleiner 23/02/2024

maXbox4exe

Digitally signed by maXbox4exe
Date: 2024.01.23 21:50:57 +01'00'