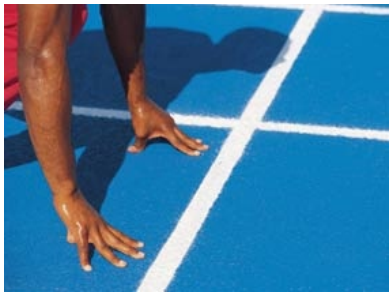
maXbox



maXbox Collab

Start Sharing Code

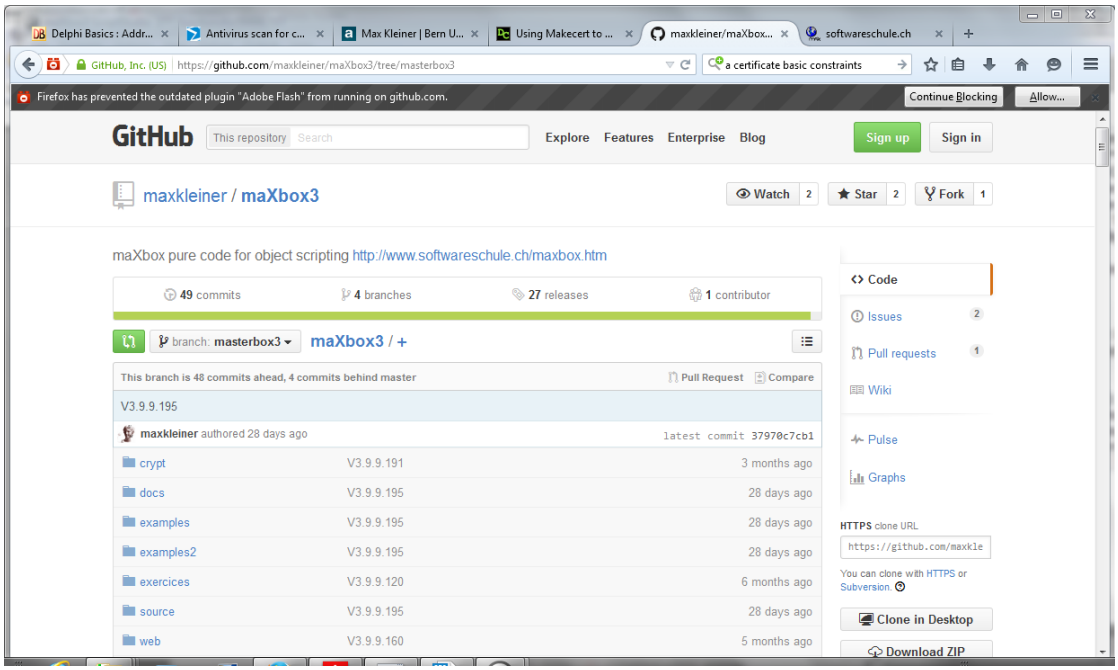
1.1 A Programming Central

Today we go through the sharing of code.

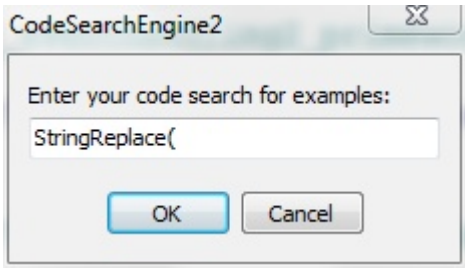
Browse and access the source code for your projects to learn more about the code is a reuse too. When I first needed to use, for ex., certificates to secure a service, I didn't really understand how certificates worked, how to create them, and where they go and above all where to get some code. A lot of the tutorials on the web just give you a certain hint or a code snippet but not running code. So I decided to set all running examples as scripts in my own directory, for examples find with <Ctrl> F3:

```
\\maxbox3\examples\287_eventhandling2_primewordcount.txt
```

You do find more examples on web on GitHub or SourceForge:

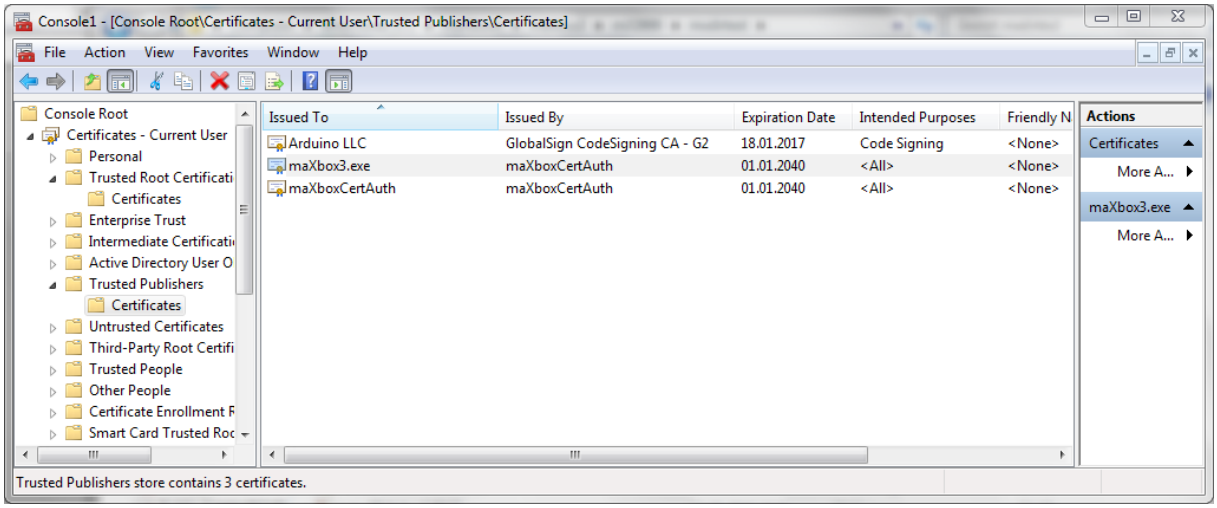


After starting the box you can search for keywords in code with <Ctrl> F3 and a full text retrieval on /examples is running. Helping you find real world examples of functions and running scripts:



Then you enter with OK and you get a list of sections with the full path of the file-name. Open this scripts from the repository[1] to produce new ones is the idea.
http://sourceforge.net/projects/maxbox/files/Examples/

What about scripts that reside on a file share inside the network? On my machine they run just fine but keep in mind, that all scripts must reside in a directory called /examples, sort of convention over configuration.



What about trust in `maXbox3.exe` as the running host machine of the scripts. After test, scan and examine you get the SHA1 of the executable to make sure for example V 3.9.9.195 the SHA1 (20 bytes):

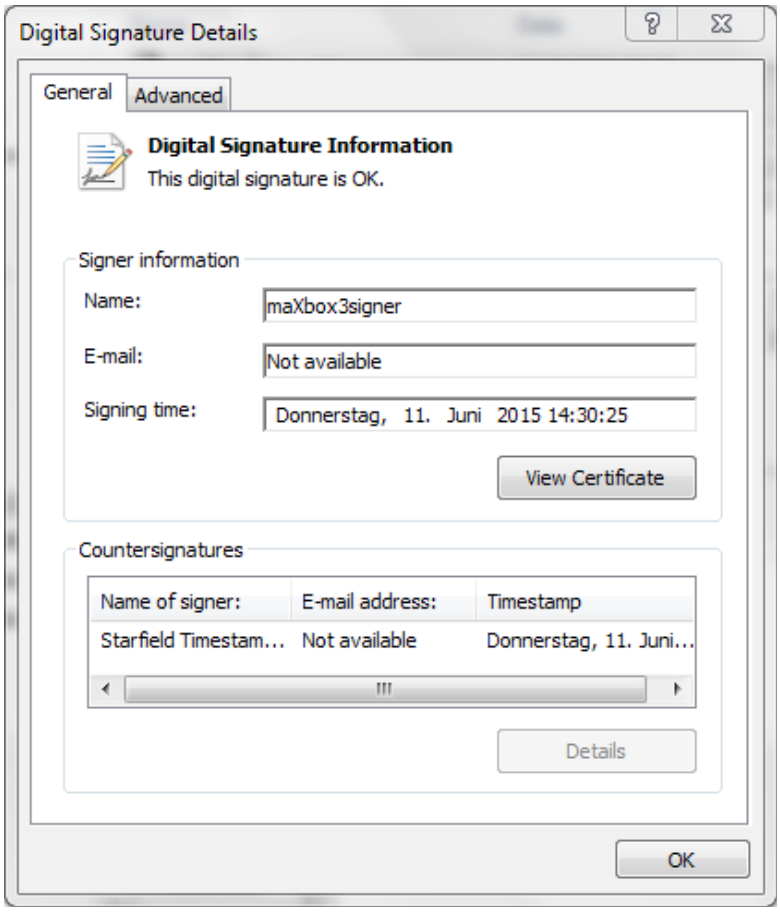
`F0AB7D054111F5CE46BA122D6280397A841C6FAB`

In menu `/Help/About` you can check this with a recompute of the Hash.

Also a certificate for digital signing is possible (see above).
Certificates are a type of identification that try to ensure that you know who you are talking to, and that it is not somebody else just impersonating

[1]By default CodeSearchEngine points to a local drive.

the person you are expecting to be talking to. In more technical terms, a certificate binds together a name or binary (an identity) and a public key. So a public key certificate, usually just called a certificate, is a digitally signed statement that binds the value of a public key to the identity of the person, device, or service that holds the corresponding private key.



The same goes for scripts placing on a network drive. If the function attempts to make a connection and the network provider returns the message ERROR_INVALID_PASSWORD, the system prompts the user to enter a password. The system uses the new password in another attempt to make the net share connection.
I think each code you want to publish or share has the 3 steps produce, share and get some code from others in exchange.

1.2 Share Bytecode

Firstly, a few concepts. Most certificates in common use are based on the X.509 v3 certificate standard and the name bytecode stems from instruction sets which have one-byte opcodes followed by optional parameters. Intermediate representations such as bytecode may be output by programming language implementations to ease interpretation, or it may be used to reduce hardware and operating system dependence by allowing the same code to run on different platforms. So you can share your code as source in a normal text-file (*.txt) or as bytecocde (*.psb).

In some cases, you may want to export or deliver a script with its bytecode to store on removable media or to use on a different computer without the source as a text-file. This is how you can do that:

1. You open a script and compile it before.
2. you go to /Options/Save Bytecode/ and the console writes:

```
-----PS-BYTECODE (PSB) mX4-----13:48:38
-----BYTECODE saved as:
C:\maXbook\maxbox3\mX3999\maxbox3\examples\287_eventhandl
ing2_primewordcount.psb -----
IFPS#
```

3. you load the bytecode by /Options/Load Bytecode...

```
IFPS#
### mX3 byte code executed: 10.06.2015 13:53:20  Runtime:
0:0:1.577  Memoryload: 60% use
ByteCode Success Message of:
287_eventhandling2_primewordcount.psb
```

4. When testing is finished you send the bytecode to your client

But there are some restrictions to this procedure:

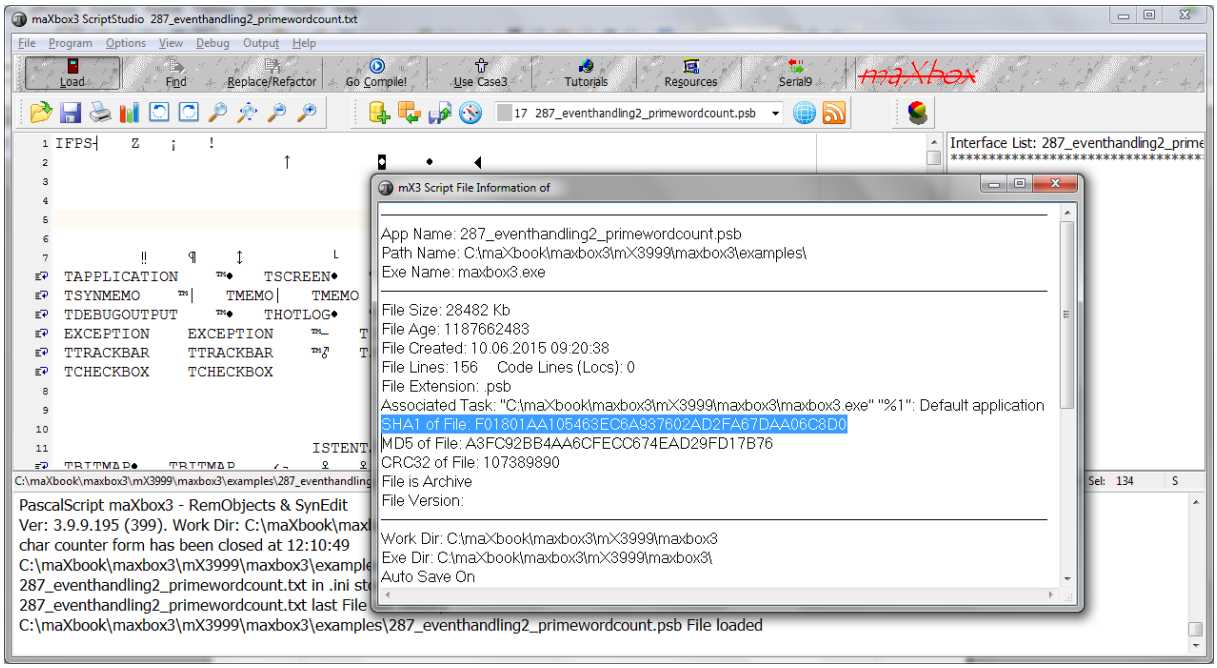
You should avoid self referencing commands like `maxform1.color` or `memo1.text` in your bytecode. Also reflection calls, unsafe type casts or runtime type information can fail.

So during development, we may want to create code for our own purposes and then implicitly share them to test, deploy or reuse. But we don't really want to go to an app- or store authority and get a signed certificate for code signing, because that costs money and makes us dependent.
If you don't want to share the source code for property or security reasons you can deliver the bytecode of the script which they can load and run in maXbox like previously said.
On the other side you want that others can trust to your code so you deliver them simply the SHA1 hash of the file as a signature:

5. You open the bytecode as a normal file in the editor (see below):
6. ..\examples\287_eventhandling2_primewordcount.**psb** File loaded
7. you go to /Program/Information and copy the SHA1 of file: F01801AA105463EC6A937602AD2FA67DAA06C8D0
8. you send this number (fingerprint) to your client
9. Client loads the bytecode and compares the fingerprint to verify!

By meaning signature is just a hash. Real Code Signing uses a certificate associated with key pairs used to sign active content like a script or an application explained above. The storage location is called the certificate store. A certificate store often has numerous certificates, possibly issued from a number of different certification authorities.

But where does this byte code function come from? Byte code is an intermediate step, right before compilation into machine code. Because the last step is left to load time (and often runtime, as is the case with Just-In-Time (JIT) compilation, byte code is architecture independent.



But having a call to a DLL isn't independent from the operating system – this native call cant load on a Linux maXbox. So the same call is valuable from a DLL only on Win:

```
function  MyNETConnect(hw: hwnd; dw: dword): Longint;
    external 'WNetConnectionDialog@Mpr.dll stdcall';
```

You see the library is called `Mpr.dll` as the external DLL.

And this is how we call the external function:

```
if MyNETConnect(GetForegroundWindow, RESOURCETYPE_DISK)
                        = NO_ERROR then  writeln('connect dialog success');
```

And this is how bytecode decompile works:

```
Proc [3] Export: STSPAWNAPPLICATION1COMPLETED -1 @44
 [0] PUSHTYPE 27(U8) // 1
 [5] ASSIGN Base[1], [1]
 [17] CALL 6
 [22] POP // 0
 [23] PUSHTYPE 18(String) // 1
 [28] ASSIGN Base[1], ['Process Done']
 [55] CALL 18
 [60] POP // 0
 [61] RET
```

We do not need to understand this bytecode, but doing so can assist debugging and can improve performance and memory convention.

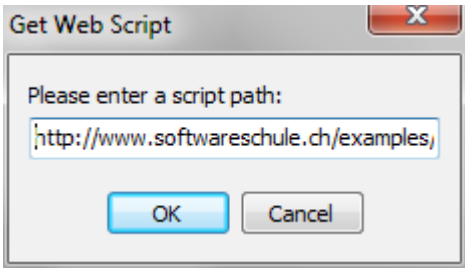
1.3 Get Scripts from the Web

Create an account of your own on code share to collaborate with project owners and other members on existing projects, and create and register projects of your own.
In maXbox you can download, load and run directly scripts from the web. When setting the script execution policy, I generally first see what it is, then set the policy, and then check again to make sure that I have properly set the policy. If you don't want execute scripts from the web you can stop this in the ini-file `maxboxdef.ini` with N. This is seen in the following figure.

```
LINENUMBERS=Y
EXCEPTIONLOG=Y
EXECUTESHELL=N
MEMORYREPORT=Y
BOOTSCRIPT=Y
```

To set the execution policy to require digital signatures if the script comes from a remote location is planned for a next version.

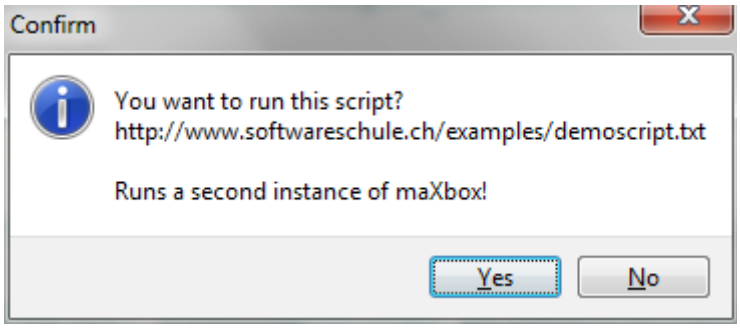
But how do you get a web-script?
Just type in menu ../Help/Get Web Script



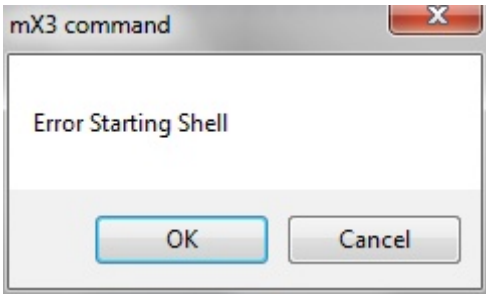
If you only want to run a script from a remote share or http site without modifying the default script execution policy, you can use this simple dialog when launching maXbox shell. You can specify the path to the script by using the file or url-site parameter. This is seen in the figure above.

Then it will ask you to run the script in a second box. Yes it will start in any case a second instance (like a sandbox) so your first instance of maXbox won't touched.
After you have a second box or console that is running for the execution policy and you said yes, you can run the remote script again and again. Remember, the script execution policy in the ini-file is not a security feature to protect you against malware cause no scanning or checking whatsoever of the file is made.

It is a convenience or convention feature. In a car, a seat belt actually protects you, and is therefore a security feature. The beeper (like a dialog) that annoys you until you put the seat belt on is a convenience feature (i.e., it does not actually protect you).
It's up to you to decide yes or no and study the script before you run it.



The error seen in the figure below is because the script was downloaded from the Internet and you set in the ini-file EXECUTESHELL=N . The file itself has now the block property set.

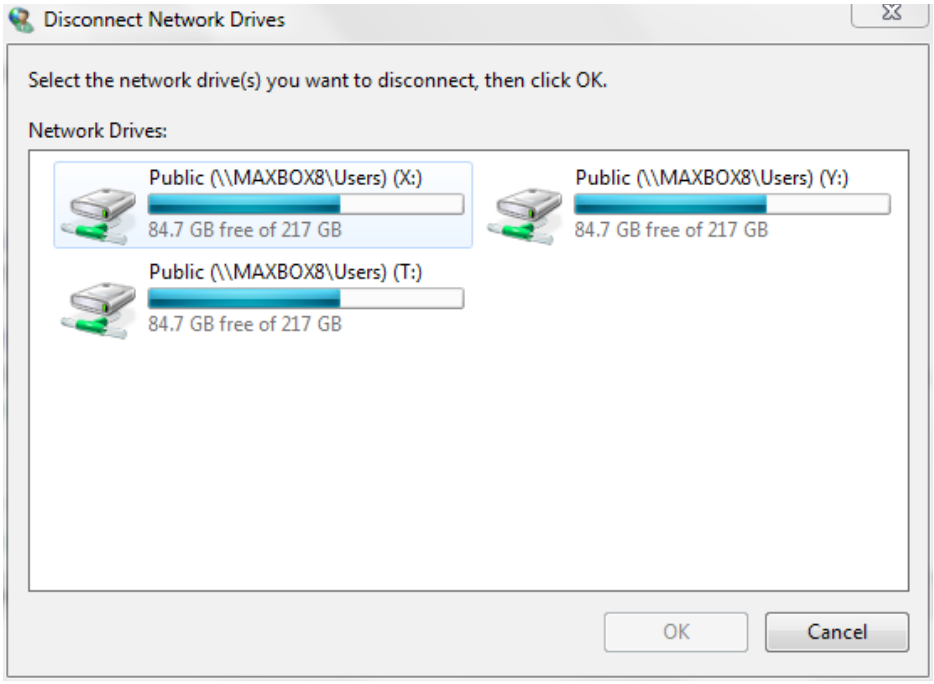


To unblock the script, you have to change the property in the ini-file and restart the box again. After the script is unblocked, it should run without any problems.

☝ Be aware you should first examine a script to make sure that you know what it actually does! In addition, testing scripts in a Virtual Machine is a great practice to enter.
You can also copy just a section or a function from the web-script like this:

```
function TJvNetworkConnect_Execute: Boolean;
var fconnect: boolean;
begin
  //RESOURCETYPE_DISK
  //WNetDisconnectDialog
  fconnect:= true;
  if fconnect then
    Result:= WnetConnectionDialog(GetForegroundWindow,
                                  RESOURCETYPE_DISK) = NO_ERROR
  end;
```

You copy it to the clipboard, and then paste it into your script editor. This makes maXbox think the script was locally generated. Therefore, it will not be blocked. As with any other script, you should also ensure you know what scripts downloaded from the web repository do also.



☝ So the idea of download and opening the bytecode file directly isn't at all successful, hence we shall use a Hex editor to disassemble the bytecode first, which will produce the implementation logic in hexadecimal bytes.

By the way at last:
In maXbox you can also map (connect) or disconnect a network drive without a dialog as a console or shell call!:

```
ConnectDrive('Z:', '\\Servername\C', True, True);
```

```
Const SHARENAME = '\\MAXBOX8\Users\Public';

  if ConnectDrive('Z:','\\MAXBOX8\Users\Public', True,True) = NO_ERROR then
       writeln('Net Share Z:\ Connected');  //see appendix with const

  DisconnectNetDrive('Z:', True, True, True);
```

Hope you did already work with the Starter 28 on DLL Code topics:

http://www.softwareschule.ch/download/maxbox_starter28.pdf

http://www.softwareschule.ch/download/maxbox_starter37.pdf

Feedback @ max@kleiner.com
Literature: Kleiner et al., Patterns konkret, 2003, Software & Support

http://www.softwareschule.ch/download/codesign_2015.pdf

http://superuser.com/questions/462940/digitally-signing-software-self-signing-cerificate

http://www.digitallycreated.net/Blog/38/using-makecert-to-create-certificates-for-development

http://resources.infosecinstitute.com/java-bytecode-reverse-engineering/

https://github.com/maxkleiner/maXbox3/releases

1.4 Appendix Event Log Study

```
// WriteToOSEventLog//
procedure WriteToOSEventLog(const logName, logCaption, logDetails : UnicodeString;
                            const logRawData : String = '');
var
  eventSource : THandle;
  detailsPtr : array [0..1] of PWideChar;
begin
   if logName<>'' then
      eventSource:=RegisterEventSourceW(nil, PWideChar(logName))
   else eventSource:=RegisterEventSourceW(nil,
PWideChar(ChangeFileExt(ExtractFileName(ParamStr(0)), '')));
   if eventSource>0 then begin
      try
         detailsPtr[0]:=PWideChar(logCaption);
         detailsPtr[1]:=PWideChar(logDetails);
         ReportEventW(eventSource, EVENTLOG_INFORMATION_TYPE, 0, 0, nil,
                      2, Length(logRawData),
                      @detailsPtr, Pointer(logRawData));
      finally
         DeregisterEventSource(eventSource);
      end;
   end;
end;
```