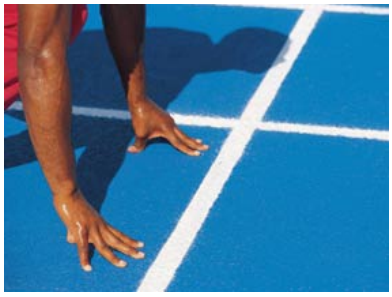
maXbox



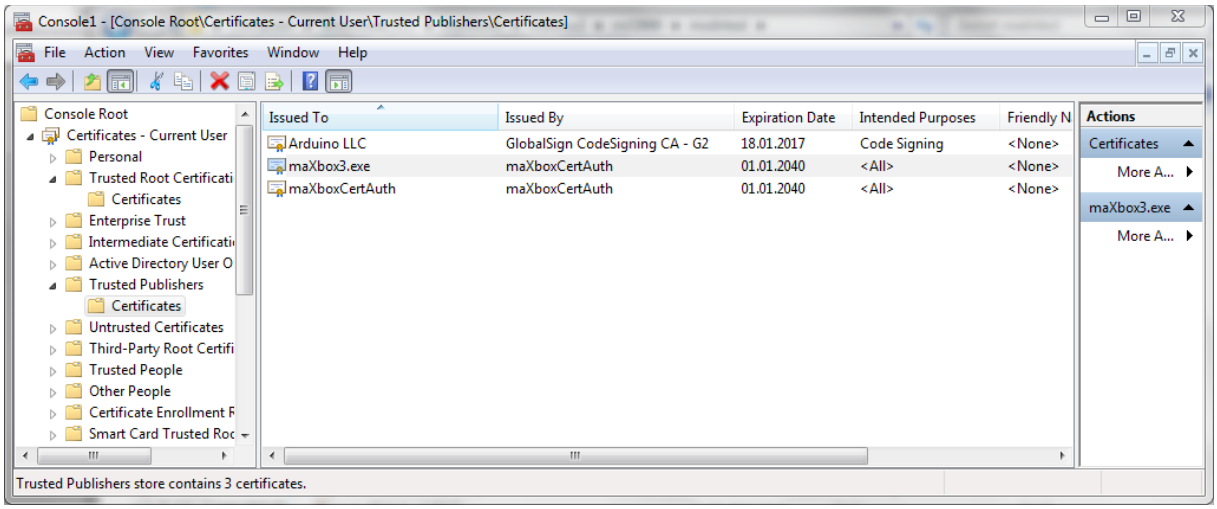
maXbox Signer

Start Signing a File

1.1 A Security Central

Today we go through the signing of code.

What about scripts or executables that reside on a file share inside the network or a cloud? On my machine they run just fine but keep in mind, that all scripts must reside in a certain directory you trust, sort of convention over configuration.
So during development, we may want to create certificates for our own purposes and then implicitly trust them.



What about trust in `maXbox4.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 4.0.2.80 the SHA1 (20 bytes):

```
CDC0D39FE16CE883EA98FF65C7E31C874FE1520B
```

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

Also a certificate for digital signing is possible (see above) what we are gong to do now.
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 the person you are expecting to be talking to. In more technical terms, a certificate binds together a name or binary (an identity) with a public key.

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. How can you sign and share code with self signed certificates and digital signatures[1] is now our topic.
So first we have to build our root certificate:
We do this with a tool from MS called `makecert`. Most certificates in common use are based on the X.509 v3 certificate standard.
First I open the shell with the MS SDK:

C:\maXbox\EKON_BASTA\EKON19\Windows Kits\10\bin\x64>

```
C:\Program Files\Microsoft SDKs\Windows\v7.1\Bin>
         makecert -n "CN=maXboxCertAuth"
 -cy authority -a sha1 -sv "maXboxPrivateKey3.pvk" -r
"maXboxCertAuth3.cer"
Succeeded
```

In the example above I changed the path to my SDK so its different to yours maybe. During the process you need a strong password to protect your private key.

```
maXboxPrivateKey3.pvk
```

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.
Next we create the second certificate with the purpose to sign all files we want. In our case, you remember, we want to sign an executable:

```
C:\Program Files\Microsoft SDKs\Windows\v7.1\Bin>makecert -n
"CN=maXbox3signer"
-ic maxboxcertauth3.cer -iv maXboxprivatekey3.pvk -a sha1 -sky
exchange -pe -sv maxbox3signerprivatekey.pvk maxboxsigner.cer
Succeeded

C:\Program Files\Microsoft SDKs\Windows\v7.1\Bin>
```

So you need (generate) another private key which belongs to the signer certificate. The output should be the second certificate:

```
maxboxsigner.cer
```

[1]Cipher a file with a private key and verify it with the public key

Windows servers and others use .pfx files that contain the public key file (for SSL Certificate file for example) and the associated private key file. Most external cert authorities provides your SSL Certificate file (public key file). You use your server to generate the associated private key file as part of the CSR.
You need both the public and private keys for an official SSL Certificate to function. So, if you need to transfer your SSL Certificates from one server to another, you need to export is as a .pfx file.

Now we generate that as well with the shell:

```
C:\Program Files\Microsoft SDKs\Windows\v7.1\Bin>pvk2pfx -pvk
"maXboxPrivateKey3
.pvk" -spc maXboxCertAuth3.cer -pfx maXboxCertAuth3.pfx -pi
password
```

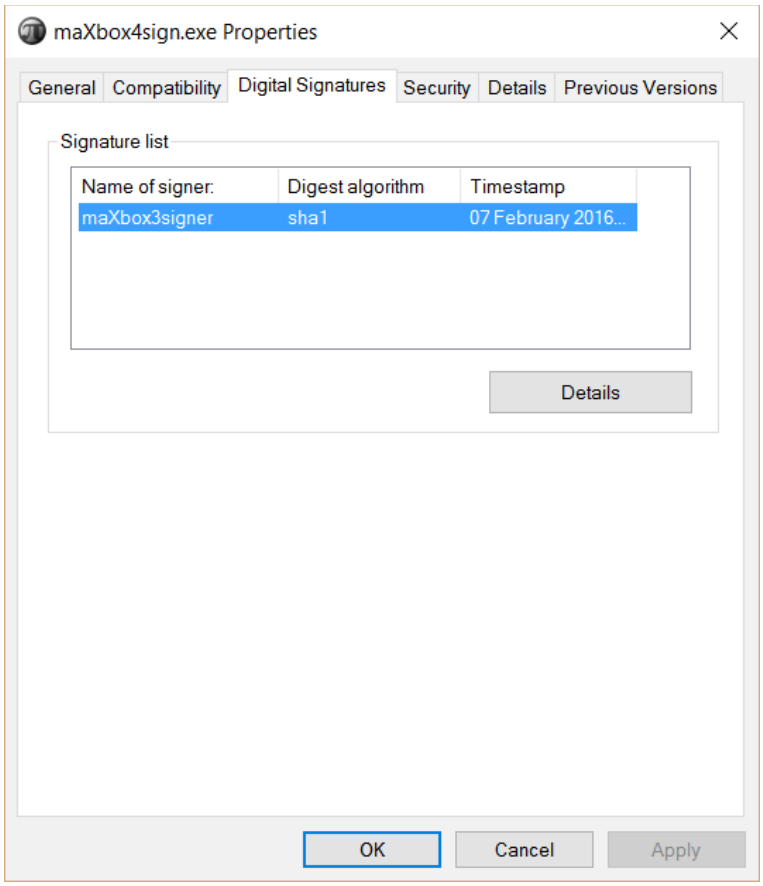
So it's time to make the last step namely to sign our executable with another shell tool called signtool:

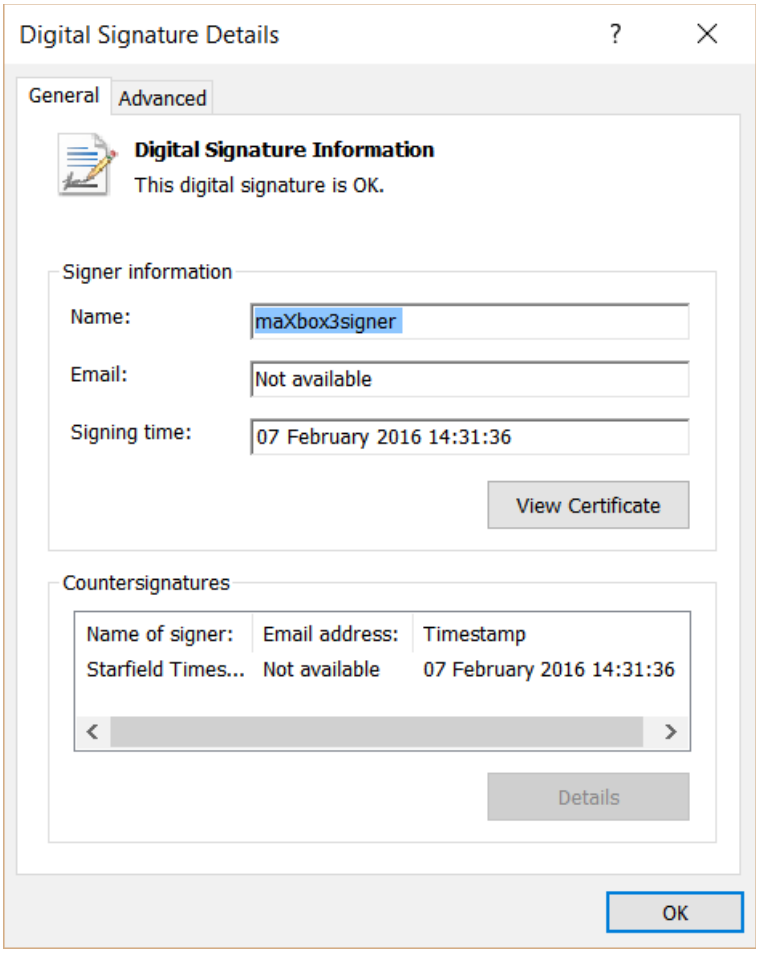
```
C:\maXbox\EKON_BASTA\EKON19\Windows Kits\10\bin\x64>signtool
sign /f "maxboxsigner.pfx" /p "password" /tr
http://tsa.starfieldtech.com /td SHA256
C:\maxbox\maxbox3\work2015\maxbox3digisign_certificates\maxbox
44.exe

Done Adding Additional Store
Successfully signed:
C:\maxbox\maxbox3\work2015\maxbox3digisign_certificates\maX
box44.exe
```

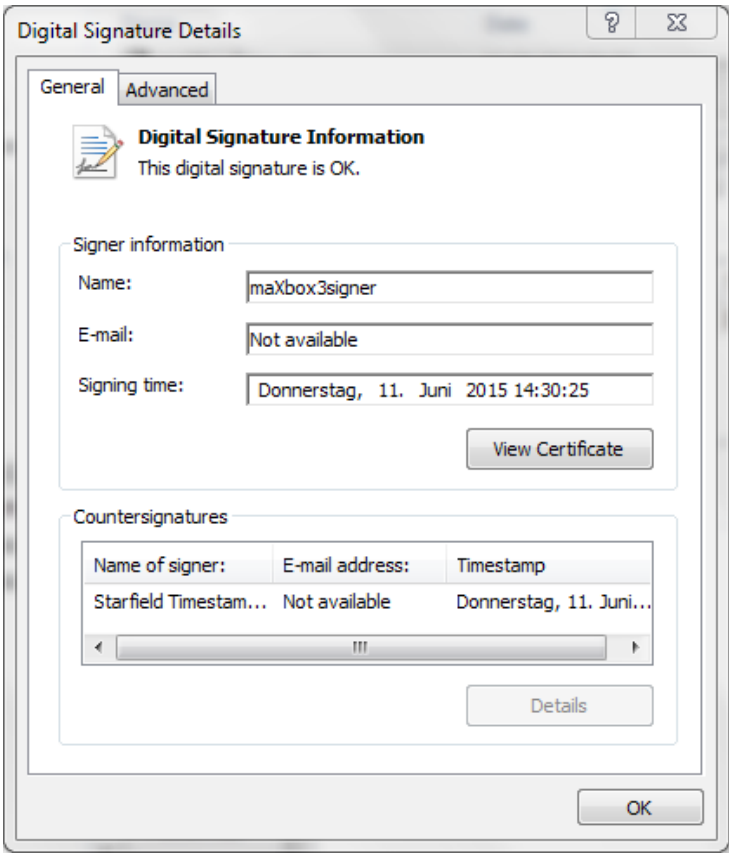
I renamed the exe to `maxbox44.exe` just to control the signing because the new exe is larger filled with cert informations and therefore has another hash code:

D5225CE8645193AD341937142AD858F734DB9EC1

Now your exe is self-signed but not trusted by an authority. 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. Instead, what we can do is create our own certificate authority and then issue certificates to ourselves to use.
We place this fake certificate authority's certificate in our computer's trusted certificate authorities store thereby causing our computer to implicitly trust all the certificates that we issue from that authority.
How can you sign and share code with self signed certificates and digital signatures is the motivation of this report.
In the end you can see extended properties which allows you to inspect the executable and also test it with `signtool verify` see below.
I'm signing an EXE program with a certificate issued by a trusted CA or self-signed. I'm using `signtool.exe` from the Windows SDK v7.1a or v10.


maXbox4sign.exe Properties
General
Compatibility
Digital Signatures
Security
Details
Previous Versions
Signature list
Name of signer:
Digest algorithm
Timestamp
maXbox3signer
sha1
07 February 2016...
Details
OK
Cancel
Apply


Digital Signature Details
General
Advanced
Digital Signature Information
This digital signature is OK.
Signer information
Name:
maXbox3signer
Email:
Not available
Signing time:
07 February 2016 14:31:36
View Certificate
Countersignatures
Name of signer:
Email address:
Timestamp
Starfield Times...
Not available
07 February 2016 14:31:36
Details
OK



As a test you can verify your signature:

```
C:\maXbox\EKON_BASTA\EKON19\Windows Kits\10\bin\x64>signtool
verify /v /pa
C:\maXbox\maxbox3\work2015\maXbox3digisign_certificates\maxbox
4sign.exe
```

Verifying: ..\work2015\maXbox3digisign_certificates\maXbox4sign.exe

Signature Index: 0 (Primary Signature)
Hash of file (sha1): 2E1157EFF8F9EB2F69C7809C1EDCBD1727692B1F

Signing Certificate Chain:
    Issued to: maXboxCertAuth
    Issued by: maXboxCertAuth
    Expires: Sun Jan 01 00:59:59 2040
    SHA1 hash: 6F83207B500DCC0E32A719599CBC6BD7E6B2A04D

        Issued to: maXbox3signer
        Issued by: maXboxCertAuth
        Expires: Sun Jan 01 00:59:59 2040
        SHA1 hash: 80F45386E921A1DD620D59BE83D495E5352A9358

The signature is timestamped: Wed Feb 03 18:32:56 2016
Timestamp Verified by:

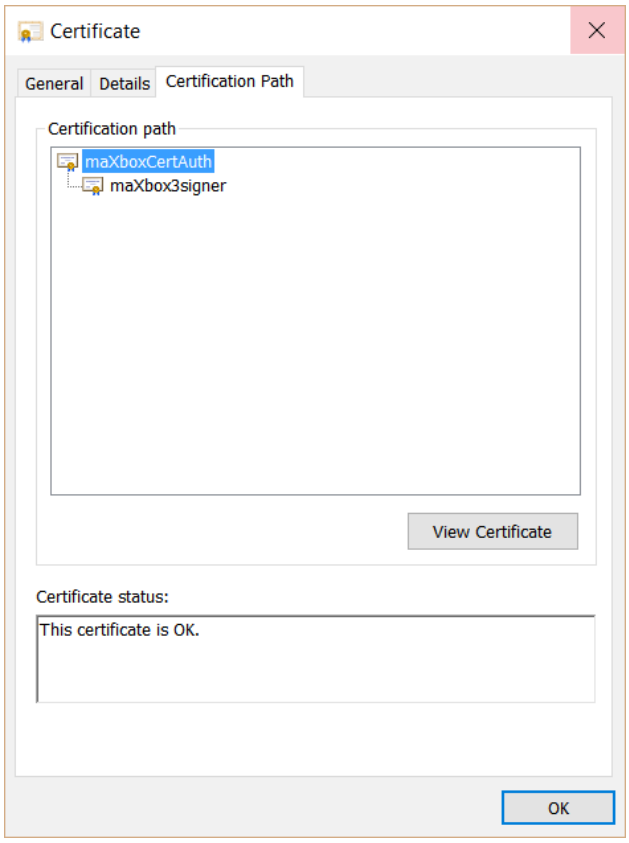
Issued to: Starfield Root Certificate Authority - G2
Issued by: Starfield Root Certificate Authority - G2
Expires: Fri Jan 01 00:59:59 2038
SHA1 hash: B51C067CEE2B0C3DF855AB2D92F4FE39D4E70F0E

Issued to: Starfield Timestamp Authority - G2
Issued by: Starfield Root Certificate Authority - G2
Expires: Mon Mar 16 08:00:00 2020
SHA1 hash: 113B5604A3689AEB636361771C5CA8DEBC97C02C

```
Successfully verified:
C:\maXbox\maxbox3\work2015\maXbox3digisign_certificates\maXbox
4sign.exe
```

Number of files successfully Verified: 1
Number of warnings: 0
Number of errors: 0

Next I want to stress the chain of certificate (block chain is one of the next big thing). A certificate authority themselves have a certificate with which they digitally sign all the certificates they issue. My machine (and pretty much everyone's) has a store of the certificates (see first picture page 1) of these different certificate authorities. The computer then knows that if its sees any certificate that has been signed by one of these trusted certificate authorities' certificate, then the pc should trust that certificate. This concept is called "Chain Trust". The "chain" part refers to the "chain" of certificates-signing-certificates and it looks like:



1.2 Share Trusted Bytecode

Firstly, a few concepts. Most certificates in common use are based on the X.509 v3 certificate standard and the name byte-code stems from instruction sets which have one-byte op codes followed by optional parameters. Intermediate representations such as byte-code 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 byte-code 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 byte-code. Also reflection calls, unsafe type casts or runtime type information can fail.

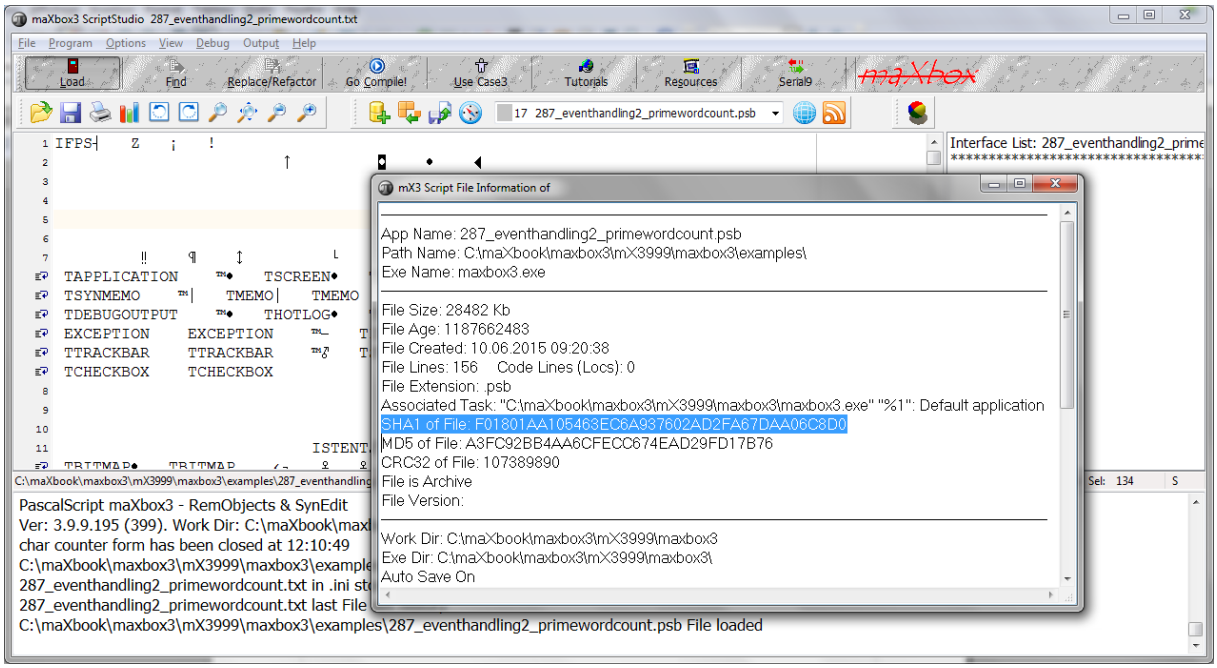
So during development, we may want to create code for our own purposes and then implicitly share them to test, deploy or reuse. But we don't really want to go to an app- or store authority and get a signed certificate for code signing, because that costs money and makes us dependent.
If you don't want to share the source code for property or security reasons you can deliver the byte-code 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 byte-code 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 byte-code 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
```

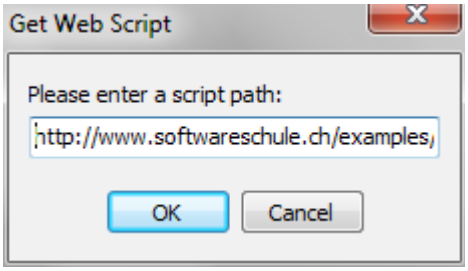
We do not need to understand this byte-code, but doing so can assist debugging and can improve performance and memory convention.

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.

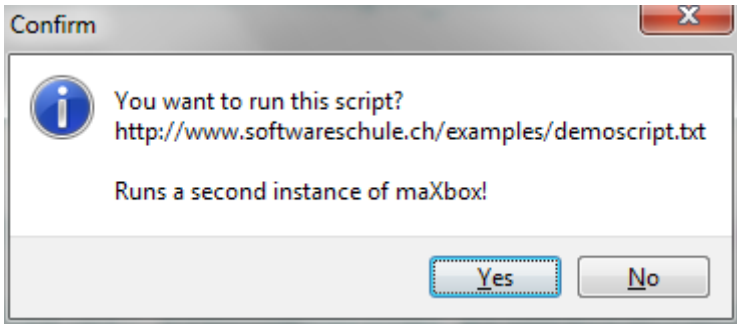
```
LINENUMBERS=Y
EXCEPTIONLOG=Y
EXECUTESHELL=N
MEMORYREPORT=Y
BOOTSCRIPT=Y
```

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

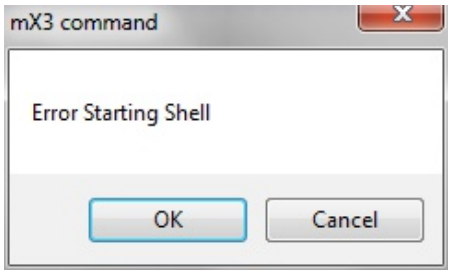


If you only want to run a script from a remote share or http site without modifying the default script execution policy, you can use this simple dialog when launching maXbox shell. You can specify the path to the script by using the file or url-site parameter. This is seen in the figure above.

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.



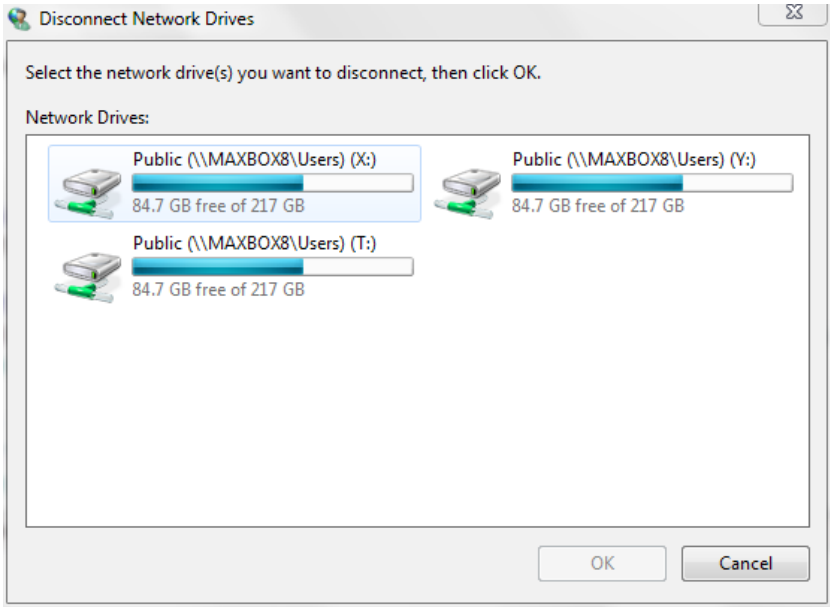
To unblock the script, you have to change the property in the ini-file and restart the box again. After the script is unblocked, it should run without any problems.

☝ 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 byte-code file directly isn't at all successful, hence we shall use a Hex editor to disassemble the byte-code 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);
```

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://sourceforge.net/projects/maxbox/files/Examples/maXbox4digisign_certificates.zip/download

http://sourceforge.net/projects/maxbox/files/Examples/12_Security.zip/download

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;
```