



maXbox Starter 25

Start with maXbox Configuration V3

1.1 From Boot loader to CLI

If a maXbox script or app is programmed to assume the host standard, it is always started relative to the path where the maXbox3.exe as the host itself is:

```
playMP3(ExePath+'examples\maxbox.mp3');
```

So for example you want to play a song or refer to other external resources in a script, your external file will be found relative to ExePath():

```
E:\Program Files\maxbox\maxbox3\examples\maxbox.mp3'
```

In this case ExePath is E:\Program Files\maxbox\maxbox3.

ExePath is a useful function where you always get the path of maXbox.

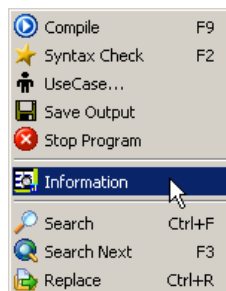
If someone tries to start (install) the script or the app to or from a different drive for space or organizational reasons, it may fail to (install) or to run the script after installation¹.

A solution might be an absolute path:

```
myMemo.lines.saveToFile('D:\data\examples\mymemo_tester.txt');
```

This problem might not be identified in the testing process, since the average user installs to the default drive of the archive and directory and testing might not include the option of changing the installation directory.

You find all info concerning run environment of the app and script in menu /Program/Information/...



¹ You don't have to install maXbox or a script anyway, just unzip or copy the file

Another solution to prevent hard coded literals are a constant or the call of a user dialog. An indirect reference, such as a variable inside the program called 'FileName', could be expanded by accessing a "select browse for file" dialog window, and the program code would not have to be changed if the file moved.

```

procedure GetMediaData(self: TObject);
begin
  if PromptForFileName(selectFile, 'Media files (*.mp3)|*.mp3|*.mpg|*.mpg', '',
    'Select your mX3 media file Directory',
    'D:\kleiner2005\download', False)
  then begin
    // Display this full file/path value

```

However it is advisable for programmers and developers not to fix the installation path of a program or hard code some resources, since the default installation path is different in different natural languages, and different computers may be configured differently. It is a common assumption that all computers running Win have the primary hard disk labelled as drive C:, but this is not the case.

As you will see the configuration of maXbox is possible with a boot loader script and a simple ini-file too. Extensions are possible with the Open Tools API and a small CLI (Command Line Interface). Hope you did already read Starters 1 till 24 at:

<http://sourceforge.net/apps/mediawiki/maxbox/>

First we start with the boot loader and his functionality.

maXbox as the script loader system has so called Special Folders which organize files logically on the hard disk. So you can also copy all 6 folders and root files to a folder on your USB-stick that stores the same content in your maXbox installation folder and it will start from the stick!

23.05.2013	19:54	<DIR>	docs
23.05.2013	19:54	<DIR>	examples
09.05.2013	14:58	<DIR>	exercices
09.05.2012	21:50	<DIR>	crypt
12.05.2013	22:04	<DIR>	source
17.03.2013	13:14	<DIR>	web
29.03.2013	23:59	97'370	bds_delphi.dci
11.12.2007	21:04	254'464	dbxint30.dll
11.11.2012	19:13	580'096	dmath.dll
09.04.2013	13:43	5'426	firstdemo3.txt
28.11.2010	00:39	3'866	firstdemo3.uc
27.10.2005	22:54	103'424	income.dll
07.11.2010	18:53	138	maildef.ini
07.02.2013	00:23	10'544	maxbootscript_.txt
21.10.2011	18:13	59'060	maxbox.mp3
02.01.2009	02:05	71'807	maxbox.png
11.05.2013	23:49	11'887'616	maxbox3.exe
11.05.2013	21:38	5'133'220	maxbox3clx
12.05.2013	23:06	994	maxboxdef.ini
03.12.2012	00:33	12'503	maxboxerrorlog.txt
12.05.2013	15:45	42'773	maxboxnews.htm
12.05.2013	00:46	2'309'571	maxbox_functions_all.pdf
21.04.2012	09:48	9'533	maxdefine.inc
14.11.2005	12:00	383'488	midas.dll
10.12.2012	09:37	17'202	pas_includebox.inc
12.05.2013	00:47	36'854	readmefirst_maxbox3.txt
11.10.2010	22:49	135'168	TIFFRead.dll

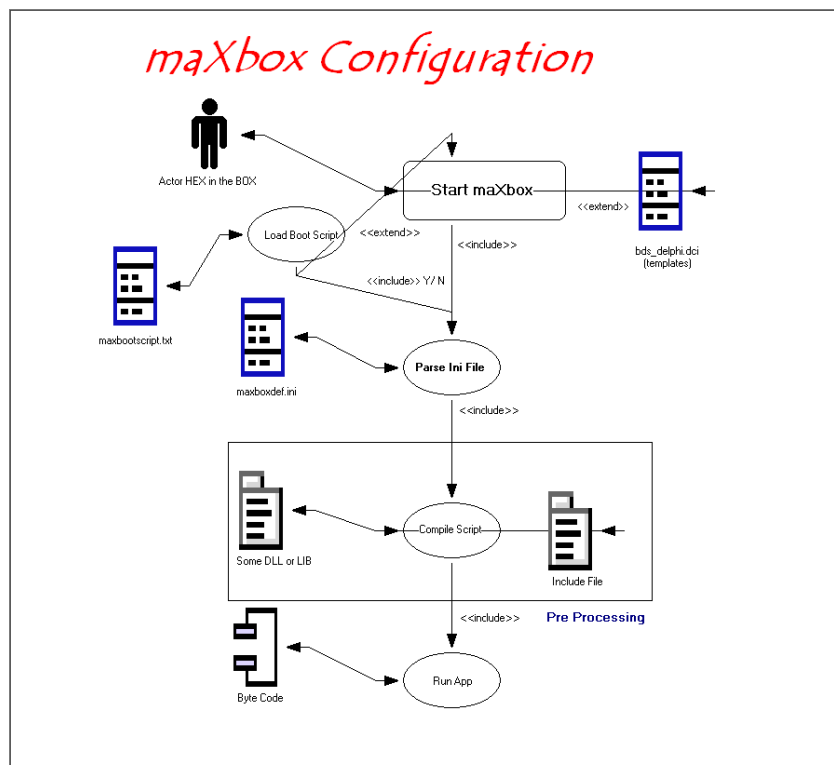
When you start the box a possible boot script is loaded. Within the boot script to the IDE, you can perform common source control tasks, such as file check in, check out, and of course change IDE settings and synchronization of your current version. This is a script where you can put all the global settings and styles of the IDE of maXbox, for example:

```
with maxForm1 do begin
    caption:= caption + 'Boot Loader Script maxbootscript.txt';
    color:= cteal;
    IntfNavigator1Click(self);
    tbtnCompile.caption:= 'Compile!';
    tbtnUsecase.caption:= 'UML UC';
    maxform1.ShellStyle1Click(self);
    memo2.font.size:= 16;
    Infol1Click(self);
end;
Writeln('BOOTSCRIPT ' +BOOTSCRIPT+ ' loaded')
```

When you want to see a complete copy of your file, look at:

07.02.2013 00:23 10'544 maxbootscript_.txt

When you delete the underscore in the filename to maxbootscript.txt the system performs next time when you load maXbox and presents you with a different view. This boot script results in the picture 2 below for example. The trick of renaming the file has a simple explanation. The ini-file default to load the boot script is YES so it can be easier to rename the file instead of change the ini-file to set to YES, cause of missing permissions, testing or so: BOOTSCRIPT=Y Maybe you want to change the colour or the caption of a button or a frame, you can do this by accessing the Open Tools API of the object maxForm1.



Configuration Step by Step

In this lesson we deal with multiple instances of maXbox and his creation. You can create multiple instances of the same app to execute parallel code, just type <F4>. For example, you can launch a new instance within a script of the box in response to some user action, allowing each script to perform the expected response.

```
ExecuteShell(ExePath+'maxbox3.exe', '''+ExePath+'examples\'+ascript+'');  
S_ShellExecute(ExePath+'maxbox3.exe', ExePath+'examples\'+ascript, secmdopen)  
;
```

So creating multiple instances results in launching multiple instances of the application and the scripts too. There's no good way to launch one application (maXbox) with multiple scripts in it. Maybe with OLE Automation in that sense you open Office programs (word, excel) or other external shell objects. `CreateOleObject` creates a single uninitialized object of the class specified by the `ClassName` parameter. `ClassName` specifies the string representation of the Class ID (CLSID). `CreateOleObject` is used to create an object of a specified type when the CLSID is known and when the object is on a local or in-proc server. Only the objects that are not part of an aggregate are created using `CreateOleObject`.



Try the example of OLE Objects 318_excel_export3.TXT and the tutorial 19.

An external script or a least a second one could also be a test case to compare the behaviour. Each test case and test project is reusable and rerunnable, and can be automated through the use of shell scripts or console commands.



An advantage of using `S_ShellExecute` or `ShellExecute3` is no wait condition, means less CPU power and consumption and loose coupling.

In this tutorial we show 4 steps to build a configuration:

1. First building block is the use of a boot script `maxbootscript.txt`.
2. Second we jump to the template file `bds_delphi.dci`
3. Third the Ini-file `maxboxdef.ini` environment and code settings.
4. Forth we set an include file in our script, like `pas_includebox.inc` to call external functions of a unit, it's very powerful, but tough to built.

Ok we have been finished the boot script what about the template file. The template file `bds_delphi.dci` stands for code completion templates.

In the Code Editor, type an object, class, structure or a pattern name followed by <Ctrl J> to display the object. But its not a full code completion where you get after the dot (.) a list of types, properties, methods, and events, if you are using the Delphi or C# languages.

It's more a template copy of your building blocks. In the menu /Debug/Code Completion List you get the defaults, here is an extract:

```
[cases | case statement | Borland.EditOptions.Pascal]  
case | of  
  : ;  
  : ;  
end;  
  
[trye | try except | Borland.EditOptions.Pascal]  
try  
  |  
except  
end;
```

My favour is: `myForm<Ctrl J>` which represents or copies a form builder in your editor.

Useless to say that you can add your own templates to the file. Many of the Code Editor features are also available when editing HTML and CSS files. Code Completion (CTRL+J) and syntax highlighting are available for HTML, XML and CSS files.

Most of the Open Tools API declarations reside also in that file `bds_delphi.dci`. (at the bootom) Note: Call all methods with `maxForm1.`, e.g.:

```
maxForm1.ShellStyle1Click(self);
```

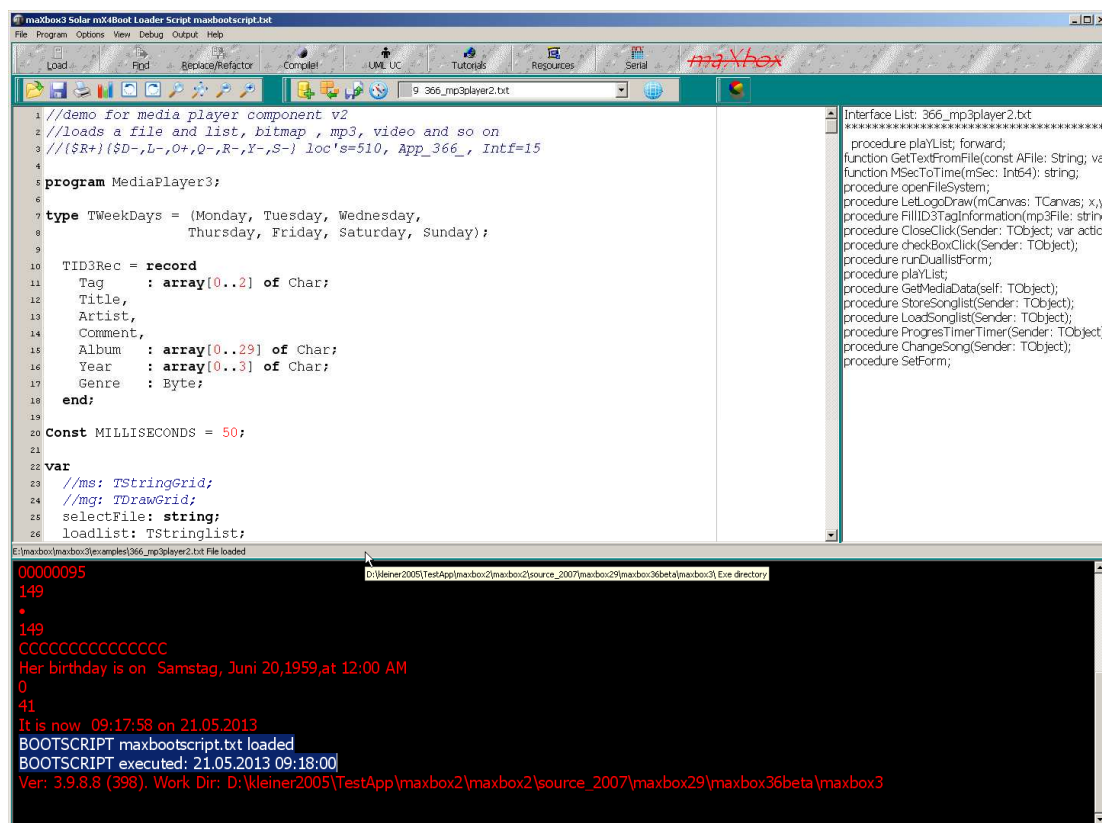
To stop for example a while loop, click on `Options/Show Include` (boolean switch)!

Then in your script you write:

```
//Control a loop in a script with the form event in menu Options:
```

```
IncludeON; //control the while loop
while maxform1.ShowInclude1.checked do begin
```

When I start a loop through another one small UI like a button a service or script has started successfully and it is running in the machine. It is running even I restart another instance. But then you can stop the loop with a click on `Options/Show Include` cause the event is a condition on the while statement. Normally you would handle this in a form but for scripts without a form or a timer it's a useful way of control a loop flow.



2: Boot Script Loaded

Also a standard approach to break a running loop in a script or configuration is the well known `KeyPress` or `IsKeyPressed` function you can check:

```
procedure LoopTest;
begin
  Randomize;
  REPEAT
    Writeln(intToStr(Random(256*256)));
```

```


UNTIL isKeyPressed; //on memo2 output
if isKeyPressed then writeln(Key has been pressed!');
end;

```

As you know the memo2 is the output window as the shell, so the `keypress` is related to memo2; by the way memo1 is still the editor!

With another function `KeyPressed(VK: Integer): Boolean`; returns True, if key VK has been pressed.

Let's jump to the Ini-file. Many applications use ini files to store configuration information. Using ini files has the advantage that they can be used in cross-platform applications and they are easy to read and edit.


 The ini file format is still popular, many configuration files (such as Desktop or Persistence settings file) are in this format. This format is especially useful in cross-platform applications, where you can't always count on a system Registry for storing configuration information. I never was a friend of the Registry so you can also start maXbox from a stick. In maXbox code, `TIniFile` is the game of advantage. When you instantiate the `TIniFile` or `TMemIniFile` object, you pass the name of the ini file as a parameter to the constructor. If the file does not exist, it is automatically created. You are then free to read values using the various read methods, such as `ReadString`, `ReadDate`, `ReadInteger`, Or `ReadBool`. This is how we read the ini file of maXbox : `maxboxdef.ini`

```

procedure getMaxBoxIniShort;
begin
  with TIniFile.Create(ExePath+'maxboxdef.ini') do
    try
      except_conf:= ReadString('Form', 'EXCEPTIONLOG', '');
      execute_conf:= ReadString('Form', 'EXECUTESHELL', '');
      boot_conf:= ReadString('Form', 'BOOTSCRIPT', '');
      ip_port:= ReadInteger('Web', 'IPPORT', 0);
    finally
      writeln('inifile sysdata1: '+except_conf+':'+execute_conf);
      writeln('inifile sysdata2: '+boot_conf+':'+intToStr(ip_port));
      Free;
    end;
end;

```

This process is handled directly, through an object so each time it changes timestamp of the file also and not on demand.


 In other words `TIniFile` works directly with the ini file on disk while `TMemIniFile` buffers all changes in memory and does not write them to disk until you call the `UpdateFile` method.

Alternatively, if you want to read an entire section of the ini file, you can use the `ReadSection` method. Similarly, you can write values using methods such as `WriteBool`, `WriteInteger`, `WriteDate`, Or `WriteString`.

Each of the Read routines takes three parameters. The first parameter (Form in our example) identifies the section of the ini file. The second parameter identifies the value you want to read, and the third is a default value in case the section or value doesn't exist in the ini file.


1.2 The Ini File

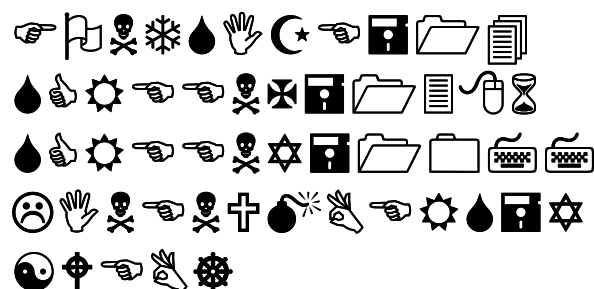
As you already know the object we now step through the meaning of the ini file. On subsequent execution of maXbox, the ini values are read in when the form is created and written back out in the OnClose and other “in between” events.


 In maXbox you can also start with read only mode (Options/Save before Compile), so nothing will be write on the disk.

```
/**/ Definitions for maXbox mX3 **/  
[FORM]  
LAST_FILE=E:\maxbox\maxbox3\examples\140_drive_typedemo.txt //history up to 10 files  
FONTSIZE=14  
EXTENSION=txt  
SCREENX=1386 //window size  
SCREENY=1077  
MEMHEIGHT=350  
PRINTFONT=Courier New //GUI Settings  
LINENUMBERS=Y  
EXCEPTIONLOG=Y //store exceptions in log file – menu Debug/Show Last Exceptions  
EXECUTESHELL=Y //prevents execution of ExecuteShell() or ExecuteCommand()  
BOOTSCRIPT=Y //enabling load a boot script  
MACRO=Y //put macros in your source header file  
NAVIGATOR=Y //set the nav listbox at the right side of editor  
MEMORYREPORT=Y //shows memory report on closing maXbox  
[WEB]  
IPPORT=8080 //for internal webserver – menu /Options/Add Ons/WebServer2  
IHOST=192.168.1.53  
ROOTCERT='filepathY' //for use of HTTPS and certificates...  
SCERT='filepathY'  
RSAKEY='filepathY'  
VERSIONCHECK=Y //checks over web the version
```

//V 3.9.8.9 also expand macros (see right below) in code e.g. #path or #file: and will cover below.


 Try to find out which line of the ini file shows the corresponding windings:?



 Now let's take a look at the code of the memory report in the project file:

```
Application.CreateForm(TMaxForm1, MaxForm1);  
if maxform1.STATMemoryReport = true then  
    ReportMemoryLeaksOnShutdown:= true;
```

We name it, means the ini-file sets the STATMemoryReport true or false.

 This example requires two objects from the classes: TMaxForm1 and TMemoryManager of mX4 so the second one is from the well known VCL Lib.

This re includes a new memory manager that significantly improves start-up time, runtime speed, and hyper threading performance.

If the ini-file doesn't exist, renamed or damaged, maXbox produces a new one with the default values. Test it by copy the maXbox3.exe in an empty directory; just says that a template file is missing but it starts and will run!

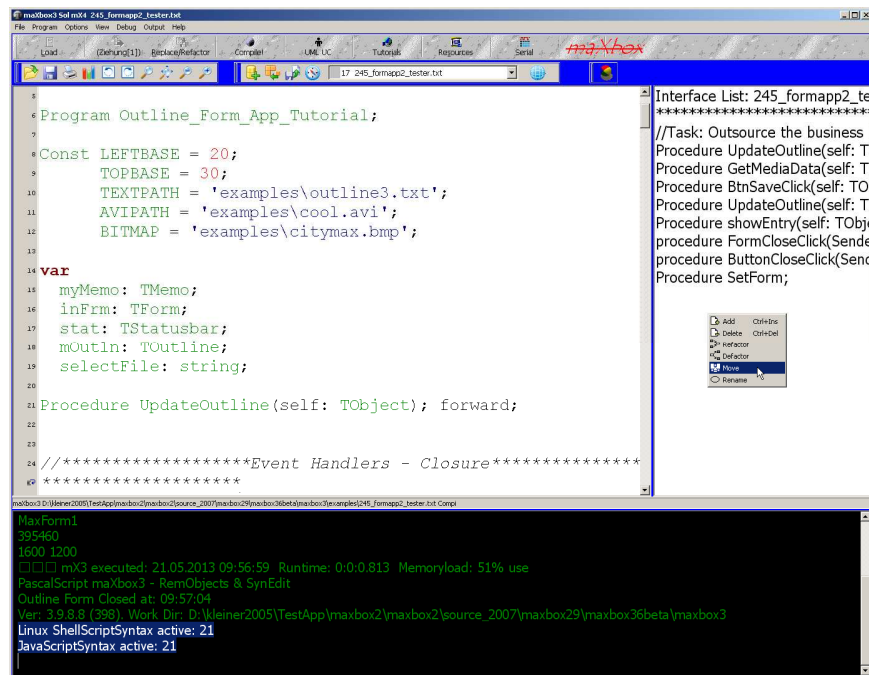
If you only want to "install" a new maXbox with file or directory names, be sure the ini-file will not be overwritten by unpacking the zip (so let's make a copy before).

Maybe you just know that by starting the maXbox it checks on the internet the last version if the ini-file allows this VERSIONCHECK=Y.

Another idea will be to compare with FTP the whole standard directory with your directory and check if something on the configuration is missing or damaged. But I think that will going to far. Something to keep in mind - there are literally hundreds of platform-specific directory listing formats still being used by FTP servers on the Internet today.

The LIST command outlined in the original FTP specification, RFC 959, did not define any kind of formatting to be used for listings, so systems were free to use whatever they wanted to use, and they did do exactly that over the years. Windows and Unix formats are common, but they are not required.


A formal listing format was not defined until RFC 3659 in the MLSD extension to FTP, which replaces the old LIST command (TIdFTP.List() does use MLSD if the server supports it).



3: Another ini file Setting runs


Let's do the last step with an include file. Under Include Files, list the files you want to include in a script. A script can include a script from a file or from another unit. To include script from a file, use the following code statement:

```
305_indy_elizahttpserver.TXT
{$I ..\maxbox3\examples\305_eliza_engine.INC}
```

 If its not find a valid file it says:

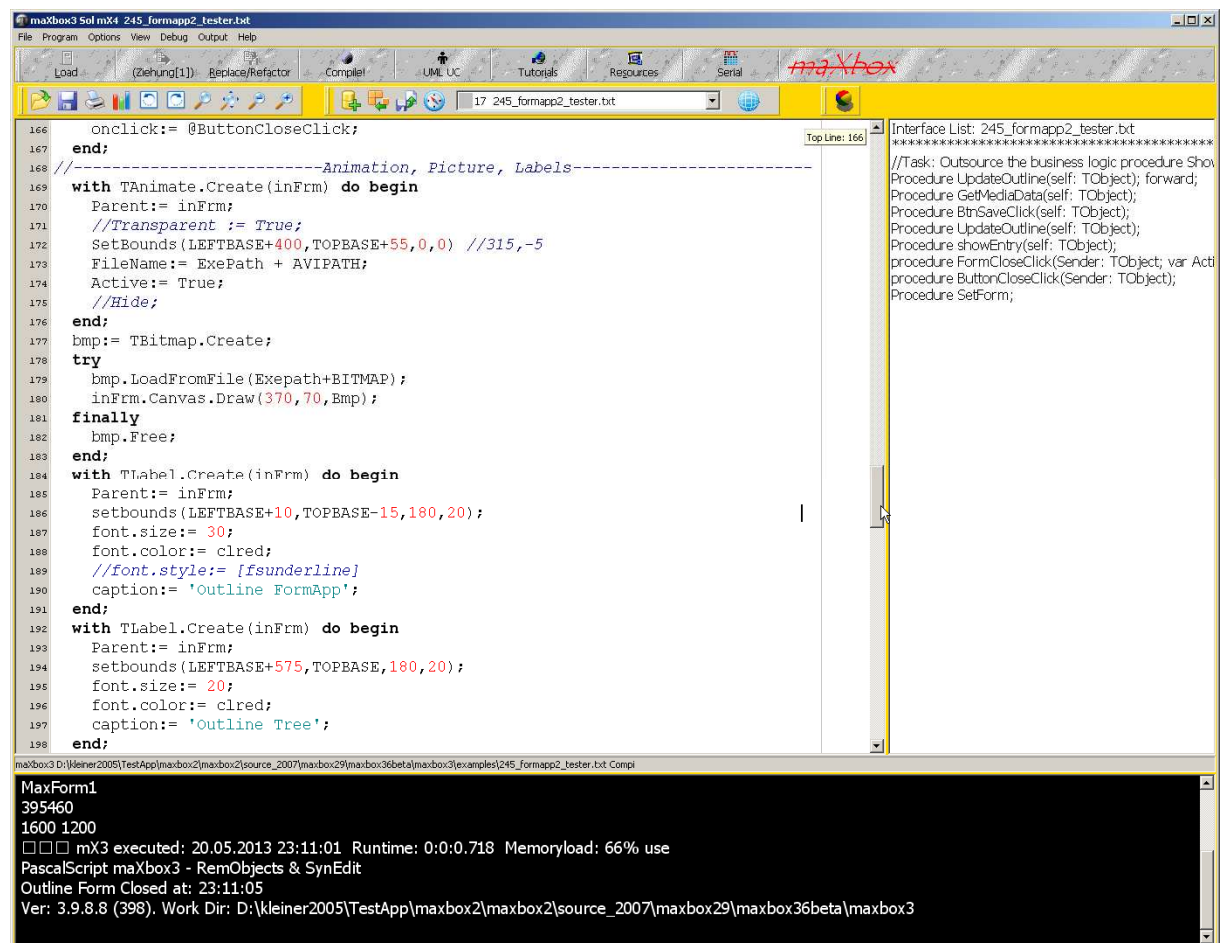

```
>>> Fault : Unable to find file '..\maxbox3\examples\305_eliza_engined.INC'
used from 'E:\maxbox\maxbox3\maxbootscript_.txt'.
```

If you need to specify additional compiler options, you can invoke the compiler from the command line with the Command Line Interface (CLI).

 As you know, there's a simple test to run the CLI out of the box with a `ShellExecute()` or a similar `RunFile()` Command.

```
ShellExecute3(ExePath+'maxbox3.exe',ExePath+'examples\'+'ascript',secmdopen);
ShellExecute3(ExePath+'maxbox3.exe',
              ExePath+'examples\003_pas_motion.txt',secmdopen);
```

A simple CLI is more relevant today than ever for scripting, and modern Shell implementations such as `maXbox` or `PowerShell` have a lot to bring to the table.




4: More of Script Configuration

At pre last is to say you can use DLL's too. Selecting this type of application sets up your project as a DLL dependency, with the exported methods expected by the Library, e.g.:

```
43: procedure SetErrCode(ErrCode: Integer);external 'SetErrCode@dmath.dll';
{ Sets the error code }
```

```
function MyMessageBeep(para: integer): byte;
10:   external 'MessageBeep@user32.dll stdcall';
```

 How can we show the run dialog?

```

procedure TForm1_FormCreateShowRunDialog;
var ShellApplication: Variant;
begin
    ShellApplication:= CreateOleObject('Shell.Application');
    ShellApplication.FileRun;
end;

```

Conclusion: There are two ways to install and configure your box into a directory you want. The first way is to use the unzip command-line tool or IDE, which is discussed above. That means no installation needed. Another way is to copy all the files to navigate to a folder you like, and then simply drag and drop another scripts into the /examples directory

The only thing you need to backup is the ini file with your history or another root files that have changed, otherwise the unzip IDE overwrites it.

Check your system environment with GetEnvironmentString:

```

SaveString(ExePath+'\Examples\envinfo.txt',GetEnvironmentString);
OpenFile(ExePath+'\Examples\envinfo.txt');

```

1.3 The Macro

The only thing you need to know is to set the macros like #host : in your header or elsewhere in a line, but not two or more on the same line when it expands with content:

Let's have a look at the demo 369_macro_demo.txt

```

{ *****
* Project   : Macro Demo
* App Name: #file:369_macro_demo.txt
* Purpose   : Demonstrates the functions of macros in header
* Date      : 21/09/2010 - 14:56 - #date:01.06.2013 16:38:20
* #path     E:\maxbox\maxbox3\examples\
* #file     369_macro_demo.txt
* #perf-50:0:4.484
* History   : translate/implement to maXbox June 2013, #name@max
*           : system demo for mX3, enhanced with macros, #locs:149
***** }

```

All macros are marked with red. One of my favour is #locs means lines of code and you get always the certainty if something has changed by the numbers of line.

So the editor has a programmatic macro system which allows the pre compiler to be extended by user code I would say user tags.

Below an internal extract from the help file All Functions List maxbox_functions_all.pdf:

```

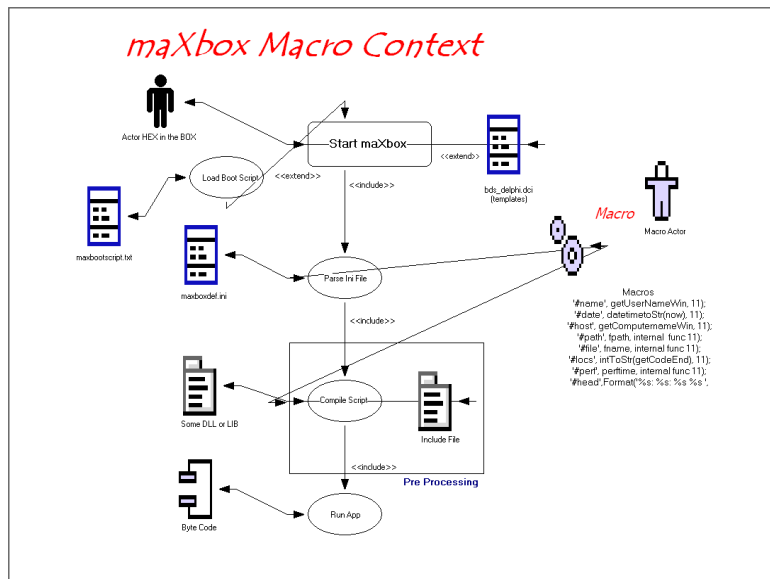
//-----
10181: //*****mX4 Macro Tags *****
10182: //-----
10183:
10184: #name, #date, #host, #path, #file, #head, #sign, #teach
10185:
10186: SearchAndCopy(memol.lines, '#name', getUsernameWin, 11);
10187: SearchAndCopy(memol.lines, '#date', datetimetoStr(now), 11);
10188: SearchAndCopy(memol.lines, '#host', getComputernameWin, 11);
10189: SearchAndCopy(memol.lines, '#path', fpath, 11);
10190: SearchAndCopy(memol.lines, '#file', fname, 11);
10191: SearchAndCopy(memol.lines, '#locs', intToStr(getCodeEnd), 11);
10192: SearchAndCopy(memol.lines, '#perf', perftime, 11);
10193: SearchAndCopy(memol.lines, '#head', Format('%s: %s: %s %s ',
10194: [getUsernameWin, getComputernameWin, datetimetoStr(now), Act_Filename]),11);

```

```

10195: SearchAndCopy(memol.lines, '#sign',Format('%s: %s: %s ',
      [getUserNameWin, getComputernamewin, datetimetToStr(now)]), 11);
10196: SearchAndCopy(memol.lines, '#tech',Format('perf: %s threads: %d %s %s',
      [perftime, numprocessthreads, getIPAddress(getComputerNameWin),
      timetoStr(time)]), 11);

```



Some macros produce simple combinations of one liner tags but at least they replace the content by reference in contrary to templates which just copy a content by value.

1.4 Build your own IDE

At last we go back to the magic boot script which will be the key to modify the IDE especially with the inbuilt SynEdit API (since V3.9.8.9). What does it mean. It means you can change or rebuild your IDE not just by fixed options or settings but also in a programmatic way in your boot script without compilation!

Imagine you want to set a vertical red line on the gutter to the left:

```

//memol.Gutter.BorderColor:= clred; //---> reflection to box!
//memol.Gutter.ShowLineNumbers:= true; //---> reflection to box!

```

You simply put the line above on the boot script and make sure the ini file has it set to Yes.
 BOOTSCRIPT=Y //enabling load a boot script

In combination with the Open Tools API you can tweak the GUI with new or change buttons, events and behaviour for example:

```

if extractFileName(maxform1.appname) = '370_synedit.txt' then begin
  Options:= +[eoShowSpecialChars];
  ActiveLineColor:= clyellow;
  maxform1.tbtnUseCase.caption:= 'SynScriptUC';
  maxform1.ShellStyle1Click(self)
end else
  ActiveLineColor:= clgreen;

```

☝ Be aware that the internal representation of `SynEdit TSynMemo` at `maXbox` editor is always `memo1`. and the console output as you know `memo2`!., so don't name an object var `memo1` otherwise your script will show or jump to unexpected content.
More secure is the namespace

```
maxform1.memo1.font.size:= 14; instead of memo1.font.size:= 14;

with CL.AddClassN(CL.FindClass('TForm'), 'TMaxForm1') do begin
10230: ('memo2', 'TMemo', iptrw);
10231: ('memo1', 'TSynMemo', iptrw);

maxform1.memo1.Options:=+[eoShowSpecialChars];
maxform1.memo1.ActiveLineColor:= clyellow;
```

More examples at `370_synedit.txt` and all the other changeable properties or methods you find at the bottom of the help file `<All Functions List> maxbox_functions_all.pdf`

```
10197: //-----
10198: //*****mX4 Public Tools API *****
10199: //-----
//-----
10702: file : unit uPSI_fMain.pas; OTAP Open Tools API Catalog
10703: // Those functions concern the editor and pre-processor, all of the IDE
10704: Example: Call it with maxform1.InfolClick(self)
10705: Note: Call all Methods with maxForm1., e.g.:
10706: maxForm1.ShellStyle1Click(self);
```

You can also enhance the API with functions like the example above `GetEnvironmentString`:

```
function getEnvironmentString2: string;
var
  list: TStringList;
  i: Integer;
begin
  list:= TStringList.Create;
  try
    GetEnvironmentVars(list, False);
    for i:= 0 to list.Count-1 do
      result:= result + list[i]+#13#10;
  finally
    list.Free;
  end;
end;
```

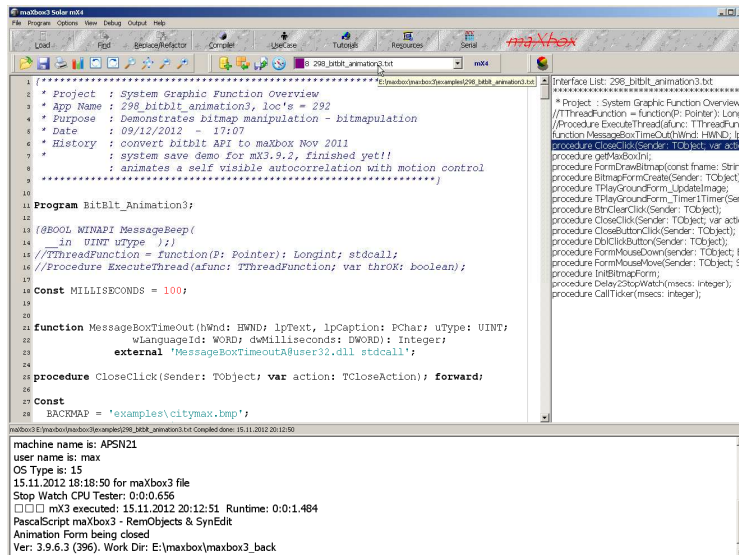
The Open Tools API is a collection of classes and functions of `SynEdit` and `VCL` components for extending and enhancing the design and your editor environment.

Unlike other development tools, you use `maXbox` (Delphi) to extend `maXbox`. You don't need to learn a new scripting language cause `PascalScript` works for you.

The Open Tools API puts you in control; reshape `maXbox` to match your needs.



~~maXbox~~



1.5 Appendix

EXAMPLE: List Objects

Working with Lists

The VCL/RTL includes many classes that represents lists or collections of items. They vary depending on the types

of items they contain, what operations they support, and whether they are persistent.

The following table lists various list classes, and indicates the types of items they contain:

Object Maintains

TList A list of pointers

TThreadList A thread-safe list of pointers

TBucketList A hashed list of pointers

TObjectBucketList A hashed list of object instances

TObjectList A memory-managed list of object instances

TComponentList A memory-managed list of components (that is, instances of classes descended from *TComponent*)

TClassList A list of class references

TInterfaceList A list of interface pointers.

TQueue A first-in first-out list of pointers

TStack A last-in first-out list of pointers

TObjectQueue A first-in first-out list of objects

TObjectStack A last-in first-out list of objects

TCollection Base class for many specialized classes of typed items.

TStringList A list of strings

THashedStringList A list of strings with the form Name=Value, hashed for performance.

The ProgID for each of the Shell objects is shown in the following table.

Object	Function
TList	A list of pointers
TThreadList	A thread-safe list of pointers
TBucketList	A hashed list of pointers
TObjectBucketList	A hashed list of object instances
TObjectList	A memory-managed list of object instances
TComponentList	A memory-managed list of components
TClassList	A list of class references
TInterfaceList	A list of interface pointers
TQueue	A first-in first-out list of pointers
TStack	A last-in first-out list of pointers
TObjectQueue	A first-in first-out list of objects
TObjectStack	A last-in first-out list of objects
TCollection	Base class for many specialized classes of typed items
TStringList	A list of strings
THashedStringList	A list of strings with the form Name=Value, hashed for performance.