



maXbox Starter 28

Start with DLL

1.1 A Library for All

The purpose of writing programs is communication. And communication needs structure. A DLL can help to structure the system.

A DLL is not XML or UML those L stands for language. A DLL is a library, short for Dynamic Link Library, a library of executable functions or data that can be used by a Windows or Linux¹ application. Typically, a DLL provides one or more particular functions and a program accesses the functions by creating either a static or dynamic link to the DLL.

A static link remains constant during program execution while a dynamic link is created by the program as needed. DLLs can also contain just data. DLL files usually end with the extension *.dll, *.exe, drv, or *.fon.

This is how we declare a function we want to use from a DLL:

```
Function OpenProcess2(dwDesiredAccess: DWORD; bInheritHandle: BOOL;
                    dwProcessId:DWORD): THandle;
    External 'OpenProcess@kernel32.dll stdcall';
```

Suppose you want to use the function `OpenProcess` of the `'kernel32.dll'`.

All you have to do is to declare above statement and you get access to the kernel!

With `external` you made these functions available to callers external to the DLL, so we must export them or at least say the function we use is from **External**. This means also to use the modifier `stdcall` because this is a C convention.

The function name `OpenProcess2` is different from the original name `OpenProcess`! This is an alias to prevent name conflicts or name it you like because you do have conventions you are free to rename the function in your script.

That's all for the meantime, now you can call the function:

```
ProcessHandle:= OpenProcess2(PROCESS_QUERY_INFORMATION or
                            PROCESS_VM_READ, false, ProcessID);
```

And you get back the process handle like: > 904

```
Writeln('Process Handle: '+intToStr(ProcessHandle));
```

A DLL can be used by several applications at the same time. Some DLLs are provided with the Windows operating system and available for any Win application. Other DLLs are written for a particular application and are loaded within the application.

¹ In Linux we say `.so` for shared objects

Oh wait a minute, you can't call the function because there is one parameter missing:

```
ProcessHandle:= OpenProcess2(PROCESS_QUERY_INFORMATION or  
                             PROCESS_VM_READ, false, ProcessID);
```

The first is set of constants already declared, the second is just false but by the third you need the `ProcessID` to get the corresponding handle, that's life.


So you get the current `ProcessId` by simply call the function `GetCurrentProcessID`.

```
ProcessHandle:= OpenProcess2(PROCESS_QUERY_INFORMATION or  
                             PROCESS_VM_READ, false, GetCurrentProcessID);
```

The const `PROCESS_QUERY_INFORMATION` retrieves certain information about a process, such as its token, exit code, and priority class and the other const `PROCESS_VM_READ` is required to read memory in a process using. By the way the script you get all this is:

http://www.softwareschule.ch/examples/440_DLL_Tutor2.txt

With our DLL that just copied and pasted from the script folder, we will use a next example to retrieve version information. Until this call here some more topics.

 Windows DLLs usually have no safety whatsoever. The DLLs that interface with HW are kernel mode drivers that can wreak havoc and gone. Extreme care has to be used when passing parameters to the DLLs.

Usually you have an interface module that establishes "mapping" between DLL and Pascal functions so there should be a tested environment in your box.

The advantage of a DLL is:

For most applications, packages in an object oriented sense provide greater flexibility and are easier to create than DLLs. However, there are several situations where DLLs would be better suited to your projects than packages:

- Your code module will be called from non-Pascal application.
- You are extending the functionality of a Web server.
- You are creating a code module to be used by third-party developers.
- Your project is an OLE container.

I don't explain those points just for your insights. Let me say a few words about building a DLL which is not theme of this tutorial. When we update a DLL (change function's implementation), we simply compile it, export some new routines and ship the new version. All the applications using this DLL will still work (unless, of course, you've removed or ruined existing exported routines).

This solution does not support static binding. From a C or C++ application, static binding is achieved by linking import records (either import libraries or entries in the IMPORTS section of the application's Module Definition File) to the calling application. Using standard Delphi, static binding is achieved using a declaration such as:

```
function xFoobox (param: Integer): Integer; stdcall; external  
    'mytools.dll' name 'Foobox';
```

Therefore we use in `maXbox` also static binding but in the runtime loading of the script engine. You can use either `c++` or Structured Exception Handling in the DLL. For best results, exceptions should be caught within the scope of the DLL throwing the exception. If the calling application is built with Borland `c++`, Delphi or MS `c++`, the exception can be caught in the application calling the DLL. However, VB does not seem to have an exception handling syntax.

O.k. enough of technical grabbing let's chose the second function called `GetProcessMemoryInfo()` to glue it with the first function, hope you remember, `OpenProcess()` together. In the end we put those 2 DLL functions in our script function! This as an overview:

We started with `OpenProcess` from `kernel32.dll`

```
Function OpenProcess2(dwDesiredAccess:DWORD; bInheritHandle:BOOL;
dwProcessId: DWORD):THandle; External 'OpenProcess@kernel32.dll stdcall';
```

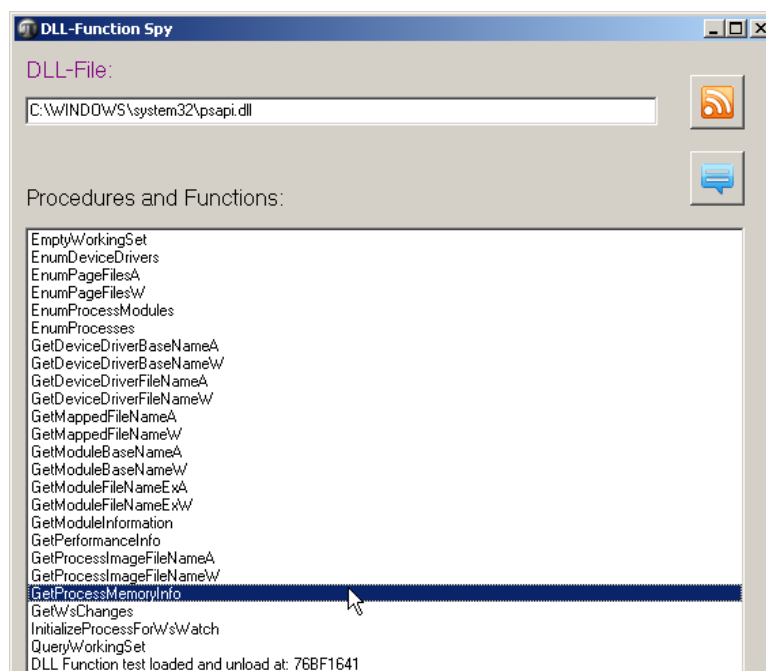
Now we go on with `GetProcessMemoryInfo` from `psapi.dll`

```
function GetProcessMemoryInfo(Process: THandle; var MemoryCounters:
    TProcessMemoryCounters;
    cb: DWORD): BOOL; //stdcall;;
External 'GetProcessMemoryInfo@psapi.dll stdcall';
```

Interesting point is the function which we use came from another DLL called `psapi.dll`. It's on of my favour DLL it's so small you can count the functions inside. (See picture below) You find a DLL Spy of the app in menu

../Options/Add Ons/DLL Spy.....

With this tool you can open a 32-bit DLL and explore which functions are inside.



To determine the efficiency of your application, you may want to examine its memory usage. The following sample code uses the `GetProcessMemoryInfo` function to obtain information about the memory usage of a process.

You find a lot of discussion concerning this function. Seeing strange results when using `GetProcessMemoryInfo()`. The same application ran on different machines, reports different memory usage. On both machines the process starts with `WorkingSetSize` of 10.2 MB. However, on one of the machines the number quickly drops down to 1 MB, while on the other it stays close to where it started. If I measure the memory consumed in the end, I see that on the first machine I've used close to 700 KB from the time the memory dropped; on the second machine it's only 41 KB more or less.

Answer: It is reliable, but you have to very careful when interpreting the results.

☞ The memory dynamics of a process is subject to lots of variables. A quick rundown might be:

- how much physical memory is there
- how much context switching is going on
- how many processes are running
- are there processes that need a lot of memory
- Does your process running in the foreground or not.....

☞ I mentioned earlier that a DLL is wrapped by a unit so here is the look behind the function `GetProcessMemoryInfo()`:

```
unit PsAPI;
interface
{$WEAKPACKAGEUNIT}
uses Windows;

function EnumDeviceDrivers(lpImageBase: PPointer; cb: DWORD;
    var lpcbNeeded: DWORD): BOOL;
{$EXTERNALSYM GetProcessMemoryInfo}
function GetProcessMemoryInfo(Process: THandle;
    ppsmemCounters: PPROCESS_MEMORY_COUNTERS; cb: DWORD): BOOL;

implementation
function CheckPSAPILoaded: Boolean;
begin
    if hPSAPI = 0 then begin
        {$IFDEF MSWINDOWS}
            hPSAPI := LoadLibrary('PSAPI.dll');
            if hPSAPI < 32 then begin
                hPSAPI := 0;
                Result := False;
                Exit;
            end;
        {$ENDIF}
        @_EnumProcesses := GetProcAddress(hPSAPI, 'EnumProcesses');
        @_EnumProcessModules := GetProcAddress(hPSAPI, 'EnumProcessModules');
        @_EnumDeviceDrivers := GetProcAddress(hPSAPI, 'EnumDeviceDrivers');
        @_GetProcessMemoryInfo := GetProcAddress(hPSAPI, 'GetProcessMemoryInfo');
    end;
    Result := True;
end;
```

In maXbox you can also check if a DLL is loaded. You find this function `CheckPSAPILoaded` in the script. And you may guess that `LoadLibrary('PSAPI.dll')` is also a DLL function and so on. Therefore a DLL provides a way to communicate between applications with different technologies and programming languages.

☞ DLL file is stored in a binary PE format. This prevents a software- and hardware-independent way of accessing functions or storing data.

The file formats for DLLs are the same as for Win EXE files - that is called, Portable Executable (PE). Another topic is maintenance. In most cases, when you get a message that a DLL file is missing it is because a program has been installed or uninstalled incorrectly.

Often you can tell what program it is by looking carefully at the name of the file. MSWKS, for example, refers to Microsoft Works.

But you may have to use your imagination -- or your memory to figure it out!

☞ In any case, if the DLL is associated with a program you are using and want to go on using, in most cases just reinstall the program. That should restore the file.

You see DLL is a large topic but we won't go that far. Just one last example and then we dive into the code example to bind those functions together.

Let's go back to the `OpenProcess()`. We use for most of your configuration an XML file to describe the build or install process with the necessary DLL files mentioned. A file for example you can get over internet and it has the purpose of an ini-file:

<http://max.kleiner.com/myconfig.xml>

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 and mobile devices where you can't always count on a system registry for storing configuration data or save the user options.

It shows the implementation of 2 databases (Interbase and `mySQL`) and the needed DLL which communicate with a client over `TCP` and sockets. I made it scriptable to demonstrate the fascination of 2 database boxes communicate with each other at the same time.

```
<?xml version="1.0" ?>
= <databases>
  = <database db="Firebird22">
    = <connection>
      <name>TInterbaseTrueSoft</name>
      <path>\\testserver\SuperPath</path>
      <dll >GDS32.DLL</user>
    </connection>
  </database>
  = <database db="mySQL55">
    = <connection>
      <name>maXSQL</name>
      <server>././kylix05/mXbase</server>
      <dll>mySQL.DLL</user>
    </connection>
  </database>
</databases>
```

So far so dude, let's start scripting. If a maXbox script or app is programmed to assume to get a file, it is always started relative to the path where the `maXbox3.exe` as the host itself is:

So for example you want to store or load a file, all your external files (we need one file below mentioned) or resources in a script will be found relative to `ExePath()`:

```
E:\Program Files\maxbox\maxbox3\sampligen3.xml
```

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 installation². You find all info concerning run environment of the app, dll and script in menu

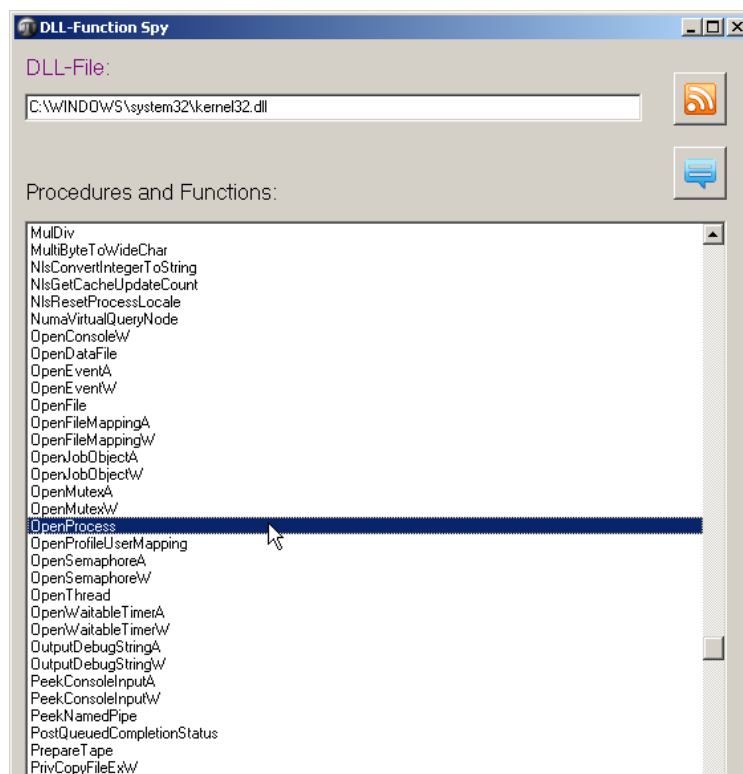
```
../Program/Information/.....  
../Debug/Modules Count/.....{list of all loaded DLLs!}
```

I am assuming you did have encountered DLL problems on your disk like this:

Exception: Access violation at address 772C5349 in module 'ntdll.dll'. Read of address 00000003.

Then most of support says either changes the path, check environment vars or reinstall the OS or get the files from the installation disk. The files are usually under i386 folder and are with .DLL_ extension or Lib in mobile; if you have service packs you can try to install them since they might have the .dll file and will override the existing one.

Next you see the DLL App which shows the source of our first function `OpenProcess()`:



1. DLL Spy at work

Many system DLLs are called key system components (`kernel32.dll`, `advapi32.dll`, `ntdll.dll`, `hal.dll`, `user32.dll`, `gdi32.dll`, and so forth). If you have ever programmed in Pascal or C, then you would know that unlike any object-oriented programming language like Object Pascal, the basic unit of organization is the function; that C or Pascal is a function, or procedural (not object) oriented language.

Next we dive into scripting our long ago introduced process information with the help of 3 DLL. You need the following script file ready to download and start with compile:

http://www.softwareschule.ch/examples/440_DLL_Tutor2.txt

² You don't have to install maXbox or a script anyway, just unzip and drop the script

Now we want to produce some data similar to the Task Manager below. In reality, a DLL file can be large enough so the structure generates some overhead you don't have with a direct link. That's the price for modularity at runtime.

Abbildname	PID	Benutzername	CPU	Arbeitssa...	Max. Arbeitssa...	Arbeitssa...	Arbeitssp...	Zugesich...	Ausgelag...	Threads	Beschreibung
Leerlaufprozess	0	SYSTEM	87	24 K	0 K	0 K	24 K	0 K	0 K	2	Zeit in Prozent, die der Pro...
sqlservr.exe	2244	SYSTEM	08	24,556 K	105,800 K	8 K	19,344 K	93,060 K	304 K	49	SQL Server Windows NT
FlashPlayerPlugin_12_0_0_44.exe	9080	Administrator	01	64,628 K	90,112 K	0 K	56,792 K	73,556 K	213 K	32	Adobe Flash Player 12.0 r...
firefox.exe	7520	Administrator	01	315,492 K	594,308 K	1,220 K	288,312 K	630,808 K	469 K	43	Firefox
taskmgr.exe	3824	Administrator	01	7,872 K	17,320 K	0 K	2,444 K	4,400 K	191 K	6	Windows Task-Manager
audiodg.exe	1408	LOKALER DIENST	01	9,960 K	18,752 K	0 K	7,692 K	18,144 K	94 K	8	Windows Graphisolerung...
wisptis.exe	10236	Administrator	00	7,020 K	7,020 K	0 K	1,968 K	2,328 K	101 K	9	Microsoft-Komponente für...
maxbox3.exe	10184	Administrator	00	154,184 K	850,924 K	0 K	100,388 K	135,684 K	469 K	28	maxBox Delphi VM
explorer.exe	10088	Administrator	00	42,624 K	64,352 K	0 K	26,840 K	70,064 K	399 K	13	Windows-Explorer
SnippingTool.exe	9972	Administrator	00	13,748 K	13,808 K	4 K	2,676 K	3,064 K	138 K	12	Snipping Tool
explorer.exe	9944	Administrator	00	30,100 K	59,952 K	0 K	15,984 K	38,796 K	384 K	12	Windows-Explorer
taskeng.exe	9632	SYSTEM	00	3,596 K	3,792 K	0 K	832 K	988 K	48 K	4	Aufgabenplanungsmodul
AcroRd32.exe	9224	Administrator	00	9,996 K	125,232 K	0 K	5,440 K	50,592 K	261 K	8	Adobe Reader
WINWORD.EXE	7916	Administrator	00	109,112 K	113,308 K	0 K	47,464 K	57,720 K	579 K	7	Microsoft Office Word

2. DLL and Process Info

Programs that must run on earlier versions of Windows as well as Win 7 and later versions should always call this function as `GetProcessMemoryInfo`.

Such a standard way of describing process data would enable a user to send an intelligent agent (a program) to each computer maker's web site, gather data, and then make a valid comparison. A DLL can be used by software or companies that want to share information in a consistent way.

The output from the script represents the data of the Task Manager:

```
workset mem from dll 157876224
workset page from dll 140021760
workset memproc test in bytes 157876224 //unit test
proc peak max. workingset in K 850924
proc peak max. paged use in K 647060
proc page peak file usage K 845200
!workset! mem from dll in K 154176 //=(157876224/1024)
True
OS Verinfo Major 6
OS Verinfo Minor 1
OS VerSizeinfo 148
```

So the app 440_DLL_Tutor2.txt has the following functions:

- `[OpenProcess]` : Opens an existing local process object. If the function succeeds, the return value is an open handle to the specified process.
- `[GetProcessMemoryInfo]` : Retrieves information about the memory and page usage of the specified process.
- `[GetVersionEx]` : Structure that receives the operating system information. Load the data for database or build transaction.

With the release of Win 8.1, the behavior of the `GetVersionEx` API has changed in the value it will return for the operating system version. The value returned by the `GetVersionEx` function now depends on how the application is manifested.

So I think it's time for practice. Load the script and press the button `[Compile]` and you should see the output above based on the lines below and compare it with the Task Manager.

```

verinfo:= GetOsVersionInfo2; //mX script
writeln('OS Verinfo Major: '+intToStr(Verinfo.dwMajorVersion))
writeln('OS Verinfo Minor: '+intToStr(Verinfo.dwMinorVersion))
writeln('OS Versizeinfo: '+intToStr(Verinfo.dwOSVersionInfoSize))

aProcessHandle:= OpenProcess2(PROCESS_QUERY_INFORMATION or
PROCESS_VM_READ, false, GetCurrentProcessID);

```

This first project 440_DLL_Tutor2 has been designed to be as simple as possible. But what does it mean running or loading at runtime?



Libraries can be loaded at runtime and thus shared between different concurrent applications. This is called dynamic linking.

Plus if a DLL is shared between applications, the OS can optimize how the library is loaded and share it between apps. Instead of loading a large single EXE into memory, the OS can load only the portions and parts needed.

1.2 Get the Process

Now it's time to put those pieces together and serve the highlight. How about the code of our Process Information? The procedure will accept a structure of TProcessMemoryCounters commands, until a Result delivers the specific information back.

It is a pointer³ to the PROCESS_MEMORY_COUNTERS or PROCESS_MEMORY_COUNTERS_EX structure that receives information about the memory usage of the process.

GetProcessMemoryInfo in turn calls the OpenProcess function to obtain the process handle. If OpenProcess fails, the output shows only the process identifier or in our case ProcessID.

```

function ProcessPageUsage(ProcessID: DWORD): DWORD;
var ProcessHandle : THandle;
    MemCounters : TProcessMemoryCounters;
begin
    Result:= 0;
    ProcessHandle:= OpenProcess2(PROCESS_QUERY_INFORMATION or
PROCESS_VM_READ,false, ProcessID);
    try
        if GetProcessMemoryInfo(ProcessHandle,
MemCounters, sizeof(MemCounters))
then Result:= MemCounters.PagefileUsage;
    finally
        CloseHandle(ProcessHandle);
    end;
end;

```

Finally, ProcessPageUsage calls the GetProcessMemoryInfo function to obtain the memory usage information for this kind of page file usage.



Note that only pieces of the DLL are "virtually loaded": not the entire DLL. This can be a misunderstanding: a loaded executable means that only pieces of it are loaded, not the entire image. Only the needed pieces of it are loaded.

As more features are used in application, more pieces of it are loaded, as well as the appropriate DLLs. This is part of the Win Memory Management mechanism - any memory that can be shared will be shared.

Alternative or as an added value we use at Nr. 3 of DLL functions the GetVersion function also as a DLL but the mXRTL does also deliver the function from inside:

³ In maXbox we use a reference, no pointers allowed


```

procedure GetVersionEx3(out verinfo: TOSVersionInfo); //ExA case sensitive!
    External 'GetVersionExA@kernel32.dll stdcall';

```



Before calling the `GetVersionEx` function, set the `dwOSVersionInfoSize` member of the `OSVERSIONINFO` data structure to `sizeof(OSVERSIONINFO)`.

The `GetVersionEx` function was developed because many existing applications got an error when examining the packed `DWORD` value returned by `GetVersion`, transposing the major and minor version numbers.

```

function GetOsVersionInfo3: TOSVersionInfo;
var
    verInfo: TOSVersionInfo;
begin
    verinfo.dwOSVersionInfoSize:= sizeof(verinfo);
    GetVersionEx3(Verinfo);
    result:= Verinfo;
end;

```

You can test for the presence of the functions associated with a feature.

To test for the presence of a function in a system DLL, call the `LoadLibrary` function to load the DLL. Then call the `GetProcAddress` function to determine whether the function of interest is present in the DLL. Use the pointer returned by `GetProcAddress` to call the function. Even if the function is present, it may be a stub that just returns an error code such as `ERROR_CALL_NOT_IMPLEMENTED`.

```

function CheckPSAPILoaded: Boolean;
var hPSAPI: THandle;
begin
    if hPSAPI = 0 then begin
    { $IFDEF MSWINDOWS }
        hPSAPI:= LoadLibrary('PSAPI.dll');
        //writeln('debug DLL handle '+inttostr(hPSAPI));
        if hPSAPI > 32 then result:= true;
        if hPSAPI < 32 then begin
            hPSAPI:= 0;
            Result:= False;
            Exit;
        end;
    { $ENDIF }
    end;
end;

```

When using the `GetVersionEx` function to determine whether your application is running on a particular version of the OS, check for version numbers that are greater than or equal to the version numbers you want in our case to test Vista.

This verification ensures that the test succeeds for later or mobile versions of the OS. Those are just words it follows the code:

```

function IsWindowsVista: boolean;
var
    verInfo: TOSVersionInfo;
begin
    verinfo.dwOSVersionInfoSize:= Sizeof(verinfo);
    GetVersionEx(Verinfo);
    result:= Verinfo.dwMajorVersion >=6;
end;

```

1.3 Calling Conventions

When you call a DLL written in C or C++, you have to use the `stdcall` or `cdecl` convention.

```
External 'GetVersionExA@kernel32.dll stdcall';
```

Otherwise, you will end up in violation troubles and from time to time the application may crash. But for the rest of the world you can use `pascal`! By the way the DLL, you are calling, should be on the search path already mentioned ;).

So these conventions (`stdcall`, `cdecl`) pass parameters from right to left. With `cdecl`, the caller (that's `maxbox` or `Delphi`) has to remove the parameters from the stack when call returns, others clean-up by the routine.

☞ Tip in C++: `DLL_IMPORT_EXPORT` means after all `stdcall`.

To understand which operations can be performed on which expressions, we need to distinguish several kinds of compatibility among types and values. These include:

- Type identity
- Type compatibility
- Assignment compatibility

The common type system defines how types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for cross-language integration.

A last word about the import unit:

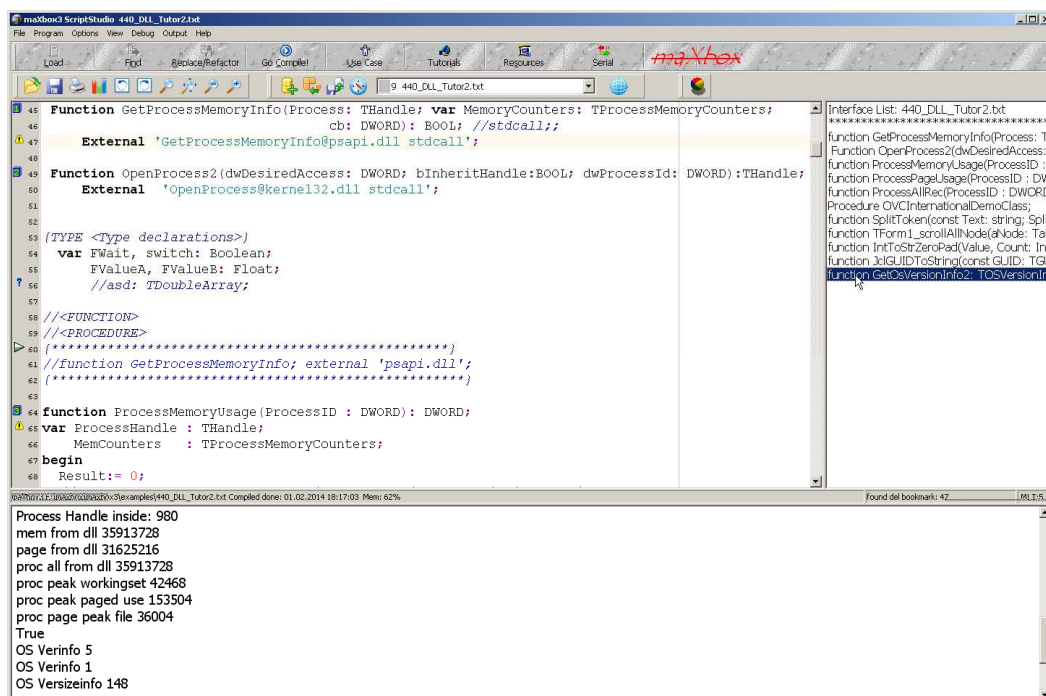
☞ Import Units wraps the functions of the API in DLLs for example:

```
const  
  {$EXTERNALSYM SC_SCREENSAVE}  
  SC_SCREENSAVE = $F140;
```

```
mmsyst = 'winmm.dll';
```

implementation

```
function auxGetDevCaps; external mmsyst name 'auxGetDevCapsA';  
function auxGetDevCapsW; external mmsyst name 'auxGetDevCapsW';  
function auxGetNumDevs; external mmsyst name 'auxGetNumDevs';  
function auxOutMessage; external mmsyst name 'auxOutMessage';  
function CloseDriver; external mmsyst name 'CloseDriver';
```



As I stated in an earlier article/session, it's possible to get an object-reference out from a DLL. This technique is known under the name DLL+.

Component Download:

<http://max.kleiner.com/download/cplusplus.zip>

initialization

```
MyIntObject := TMyObject.Create;
```

finalization

```
MyIntObject.Free;
```

But how about the DLL is written in C++?

At the very end we call a C++ DLL from maXbox script 38_pas_box_demonstrator.txt / 39_pas_dllcall.txt: of the DLL income.dll or income_c.dll

First of all, you have to translate the header-file (should be delivered with the DLL), which is like an interface-section in OP. Headers in C usually contain all sorts of definitions which are relevant outside the module. In our C++ example it looks like:

```
/*FILE: income.h */
class CIncome
{
public:
    virtual double __stdcall GetIncome( double aNetto ) = 0 ;
    virtual void __stdcall SetRate( int aPercent, int aYear ) = 0 ;
    virtual void __stdcall FreeObject() = 0 ;
};
```

Then you translate it to an Abstract Class in a unit of her own:

```
//FILE: income.pas
interface
type
CIncome = class
public
    function GetIncome(const aNetto: double): double;
                                virtual; stdcall; abstract;
    procedure SetRate(const aPercent: Integer; aYear: integer);
                                virtual; stdcall; abstract;
    procedure FreeObject; virtual; stdcall; abstract;
end;
```

In the C++ DLL, there is a procedure `FreeObject`, this is necessary because of differences in memory management between C++ and OP:

```
void __stdcall FreeObject()
{
    delete this ;
}
```

At least the DLL-call is simple:

```
incomeRef: CIncome; //member of the reference
function CreateIncome: IIncome;
                                stdcall; external 'income_c.dll';
```

```
function _SayHello2: boolean;
    external '_SayHello2@income.dll stdcall';
```

```

if _SayHello2 then
    writeln('dll invocation realised');

```

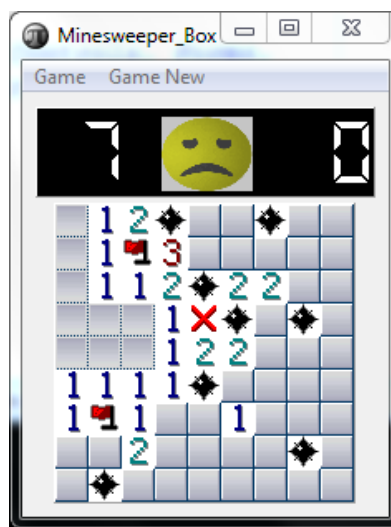
I got also 2 DLL tools available as my advise:

DCC32 compiler with JHPNE as option will generate C++ headers (with .hpp extension) for your units!

- DCC32 -JHPHN -N D:\DEV\PPT -O D:\DEV\PPT -U
D:\COMPONENTS\SC2_2\securePtBase.pas

“rundll” is a small shell program that makes a DLL run as a normal program would!

- rundll32 income.dll '_SayHello2' //for short tests



3: DLL Lib in Action

German: Unter einer dynamisch ladbaren Bibliothek versteht man in Windows eine dynamische Linkbibliothek (DLL) und in Linux eine Bibliotheksdatei mit gemeinsamen Objekten. Es handelt sich dabei um eine Sammlung von Routinen, die von Anwendungen und von anderen DLLs bzw. gemeinsamen Objekten aufgerufen werden können.

Dynamisch ladbare Bibliotheken enthalten wie Units gemeinsam genutzten Code und Ressourcen.

Sie stellen jedoch eine separat kompilierte, ausführbare Datei dar, die zur Laufzeit zu den Programmen gelinkt wird, die sie verwenden.

~~maXbox~~

□

Links of maXbox and DLL Programming:

<http://www.softwareschule.ch/maxbox.htm>

<http://sourceforge.net/projects/maxbox>

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

<https://github.com/maxkleiner/maXbox3.git>

http://en.wikipedia.org/wiki/Dynamic-link_library

<http://max.kleiner.com/download/cplusplus.zip>

<http://msdn.microsoft.com/en-us/library/windows/desktop/ms682050%28v=vs.85%29.aspx>

```
1 //maxbox is a graphics tool with an advanced dialog engine in one shot!
2 //demo shows the functions of maxbook in images, sound and text, loc's = 19
3 //maxbox: 2003: 31.05.2013 15:41:25  E:\maxbox\maxbox3\examples\102_maxbook_v4.tst
4
5
6
7
8
9 *program maxbook;
10
11 const logo = "welcome to maxbook look!";
12
13
14 imgPIC = "http://www.softwareschule.ch/images/max1";
15 imgPIC2 = "http://www.softwareschule.ch/images/max2";
16 imgMusic = "http://www.softwareschule.ch/download/no";
17 imgSFX = "maxbox.mp3";
18
19
20
21
22
23 var
24   iForm1 TForm1;
25   z: Integer;
26
27 procedure DrawText(iForm1 TForm1; wcolor: TColor);
28
29 var
30   theRect: TRect;
31
32 begin
33   with iForm1.Canvas do begin
34     brush.Color:= cWhite;
35     //iForm1.Canvas.FillRect(theRect);
36     theRect:= Rect(150,150,600,800);
37     font.Color:= wColor;
38     font.Size:= 22;
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Compiling maxbook.v40 files
CodeLines in window: 30
Memory Expanded: 200 files
E:\maxbox\maxbox3\examples\102_maxbook_v4.tst File stored
102_maxbook_v4.tst in my stored
maxbook.v40_maxbook.v4.tst Compiled done: 31.05.2013 15:41:25
----- MS-DOS Executable: 31.05.2013 15:41:28 Runtime: 0.0:1.31 Memoryload: 67% use
PascalScript maxbook3 - RemObjects & SymEdit

1.4 Appendix I

```
{*****}
{      Borland Delphi Run-time Library      }
{      WinNT process API Interface Unit    }
{                                          }
{      Copyright (c) 1985-1999, Microsoft Corporation  }
{      Translator: Borland Software Corporation    }
{*****}

unit PsAPI;
interface
{$WEAKPACKAGEUNIT}
uses Windows;
{$SHPPEMIT '#include <psapi.h>'}
type
    PPointer = ^Pointer;
    TEnumProcesses = function (lpidProcess: LPDWORD; cb: DWORD; var cbNeeded:
DWORD): BOOL stdcall;
    TEnumProcessModules = function (hProcess: THandle; lphModule: LPDWORD;
cb: DWORD;
    var lpcbNeeded: DWORD): BOOL stdcall;
    TGetModuleBaseNameA = function (hProcess: THandle; hModule: HMODULE;
    lpBaseName: PAnsiChar; nSize: DWORD): DWORD stdcall;
    TGetModuleBaseNameW = function (hProcess: THandle; hModule: HMODULE;
    lpBaseName: PWideChar; nSize: DWORD): DWORD stdcall;
    TGetModuleBaseName = TGetModuleBaseNameA;
    TGetModuleFileNameExA = function (hProcess: THandle; hModule: HMODULE;
    lpFilename: PAnsiChar; nSize: DWORD): DWORD stdcall;
    TGetModuleFileNameExW = function (hProcess: THandle; hModule: HMODULE;
    lpFilename: PWideChar; nSize: DWORD): DWORD stdcall;
    TGetModuleFileNameEx = TGetModuleFileNameExA;
    {$EXTERNALSYM _MODULEINFO}
    _MODULEINFO = packed record
        lpBaseOfDll: Pointer;
        SizeOfImage: DWORD;
        EntryPoint: Pointer;
    end;

    {$EXTERNALSYM MODULEINFO}
    MODULEINFO = _MODULEINFO;
    {$EXTERNALSYM LPMODULEINFO}
    LPMODULEINFO = ^_MODULEINFO;
    TModuleInfo = _MODULEINFO;
    PModuleInfo = LPMODULEINFO;

    TGetModuleInformation = function (hProcess: THandle; hModule: HMODULE;
    lpmodinfo: LPMODULEINFO; cb: DWORD): BOOL stdcall;
    TEmptyWorkingSet = function (hProcess: THandle): BOOL stdcall;
    TQueryWorkingSet = function (hProcess: THandle; pv: Pointer; cb: DWORD):
BOOL
```

1.5 Appendix II

DLL Loaded Inspector of the Tool in maxbox

You find a DLL Loader List of the app in menu

../Debug/Modules Count/.....

```
0: D:\kleiner2005\TestApp\maxbox2\maxbox2\source_2007\maxbox29\maxbox3.exe
1: C:\WINDOWS\system32\ntdll.dll
2: C:\WINDOWS\system32\kernel32.dll
3: C:\WINDOWS\system32\oleaut32.dll
4: C:\WINDOWS\system32\msvcrt.dll
5: C:\WINDOWS\system32\USER32.dll
6: C:\WINDOWS\system32\GDI32.dll
7: C:\WINDOWS\system32\ADVAPI32.dll
8: C:\WINDOWS\system32\RPCRT4.dll
9: C:\WINDOWS\system32\ole32.dll
10: C:\WINDOWS\system32\opengl32.dll
11: C:\WINDOWS\system32\GLU32.dll
12: C:\WINDOWS\system32\DDRAW.dll
13: C:\WINDOWS\system32\DCIMAN32.dll
14: C:\WINDOWS\system32\version.dll
15: C:\WINDOWS\system32\mpr.dll
16: C:\WINDOWS\system32\IMAGEHELP.DLL
17: C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9\comctl32.dll
18: C:\WINDOWS\system32\SHLWAPI.dll
19: C:\WINDOWS\system32\imm32.dll
20: C:\WINDOWS\system32\URLMON.DLL
21: C:\WINDOWS\system32\wininet.dll
22: C:\WINDOWS\system32\CRYPT32.dll
23: C:\WINDOWS\system32\MSASN1.dll
24: C:\WINDOWS\system32\shell32.dll
25: C:\WINDOWS\system32\winspool.drv
26: C:\WINDOWS\system32\comdlg32.dll
27: C:\WINDOWS\system32\wsock32.dll
28: C:\WINDOWS\system32\WS2_32.dll
29: C:\WINDOWS\system32\WS2HELP.dll
30: C:\WINDOWS\system32\winmm.dll
31: C:\WINDOWS\system32\winhttp.dll
32: C:\WINDOWS\system32\AVICAP32.dll
33: C:\WINDOWS\system32\MSVFW32.dll
34: C:\WINDOWS\system32\iphlpapi.dll
35: C:\WINDOWS\system32\usp10.dll
36: C:\WINDOWS\system32\oledlg.dll
37: C:\WINDOWS\system32\MSCTF.dll
38: C:\WINDOWS\system32\setupapi.dll
```

39: C:\WINDOWS\system32\olepro32.dll
40: C:\WINDOWS\system32\uxtheme.dll
41: C:\WINDOWS\system32\appwiz.cpl
42: C:\WINDOWS\system32\DUSER.dll
43: C:\WINDOWS\system32\MSIMG32.dll
44: C:\WINDOWS\system32\OLEACC.dll
45: C:\WINDOWS\system32\MSVCP60.dll
46: C:\WINDOWS\system32\msi.dll
47: C:\WINDOWS\system32\RICHED20.DLL
48: C:\WINDOWS\WinSxS\x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.2180_x-ww_522f9f82\gdiplus.dll
49: C:\WINDOWS\System32\mswsock.dll
50: C:\WINDOWS\system32\DNSAPI.dll
51: C:\WINDOWS\System32\winnr.dll
52: C:\WINDOWS\system32\WLDAP32.dll
53: C:\WINDOWS\system32\rasadhlp.dll
54: C:\WINDOWS\system32\Secur32.dll
55: C:\WINDOWS\system32\psapi.dll
56: C:\WINDOWS\system32\netapi32.dll
57: C:\WINDOWS\system32\appHelp.dll
58: C:\WINDOWS\system32\CLBCATQ.DLL
59: C:\WINDOWS\system32\COMRes.dll
60: C:\WINDOWS\system32\shdocvw.dll
61: C:\WINDOWS\system32\CRYPTUI.dll
62: C:\WINDOWS\system32\WINTRUST.dll
63: C:\WINDOWS\System32\cscui.dll
64: C:\WINDOWS\System32\CSCDLL.dll
65: C:\WINDOWS\system32\browseui.dll
66: C:\WINDOWS\system32\ntshrui.dll
67: C:\WINDOWS\system32\ATL.DLL
68: C:\WINDOWS\system32\USERENV.dll
69: C:\WINDOWS\system32\xpsp2res.dll
70: C:\WINDOWS\System32\drprov.dll
71: C:\WINDOWS\System32\ntlanman.dll
72: C:\WINDOWS\System32\NETUI0.dll
73: C:\WINDOWS\System32\NETUI1.dll
74: C:\WINDOWS\System32\NETRAP.dll
75: C:\WINDOWS\System32\SAMLIB.dll
76: C:\WINDOWS\System32\davclnt.dll
77: C:\WINDOWS\system32\wpdshext.dll
78: C:\WINDOWS\system32\PortableDeviceApi.dll
79: C:\WINDOWS\system32\Audiodev.dll
80: C:\WINDOWS\system32\WMVCore.DLL
81: C:\WINDOWS\system32\WMASF.DLL
82: C:\WINDOWS\system32\wiasext.dll
83: C:\WINDOWS\System32\sti.dll
84: C:\WINDOWS\System32\CFGMGR32.dll