maXbox

maXbox Starter 14

Start with Asynchronous Programming

1.1 Set a Sound

Today we spend another small time in programming with asynchronous programming. We will concentrate on creating an API call of a DLL using `PlaySound`() from the `winmm.dll` library. But, first of all I'll explain you what "blocking" and "non-blocking" calls are. In fact there are 2 programming models used in event driven applications:

- Synchronous or blocking
- Asynchronous or non blocking

Don't confuse it with parallel programming (simultaneous) in which we can have two synchronous functions but each in a separate thread to split and speed up the results.
Sure, two asynchronous functions can be started at the same time if they can handle parallel processing (More later on).

Hope you did already work with the Starter 1 to 13 available at:

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

Non Blocking means that the application will not be blocked when the application plays a sound or a socket read/write data. This is efficient, because your application don't have to wait for a sound result or a connection. Unfortunately, using this technique is little complicated to develop a protocol. If your protocol has many commands or calls in a sequence, your code will be very unintelligible.

☞ A function which returns no data (has no result value) can always be called asynchronously, cause we don't' have to wait for a result or there's no need to synchronise the functions.

Let's begin with the application structure process in general of an API call:
The `PlaySound` function plays a sound specified by the given file name, resource, or system event. (A system event may be associated with a sound in the registry or in the WIN.INI file.)

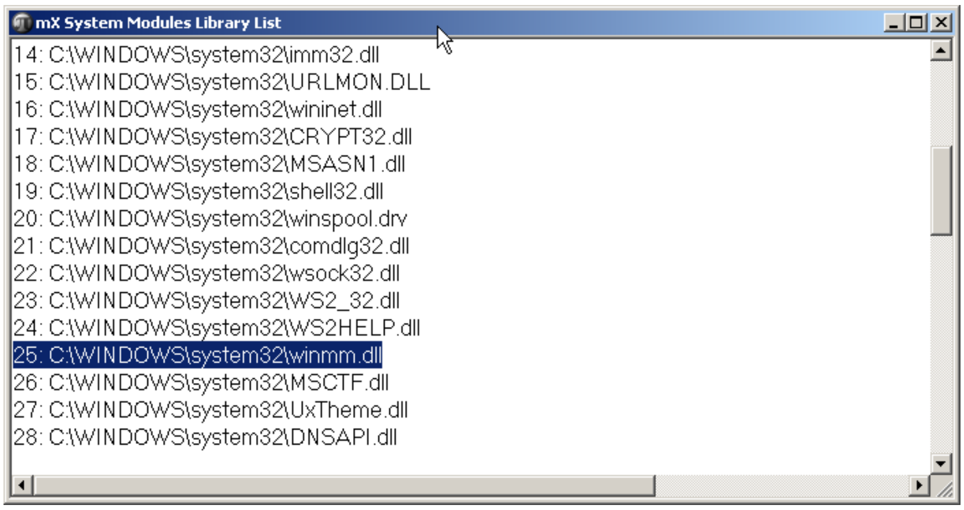
1. Header: Declared in `msystem`.h; include `Windows`.h.
2. Library: Use `Winmm.dll` or the lib.
3. Declaration of the external function
4. Load the DLL and call the API function
   - Static at start time
   - Dynamic at run time
5. Unload the DLL (loaded DLL: `C:\WINDOWS\system32\winmm.dll`)

So we call the API function dynamic at runtime and of course asynchronously. The sound is played asynchronously and `PlaySound` returns immediately after beginning the sound. To terminate an asynchronously played waveform sound, call `PlaySound` with `pszSound` set to NULL.
The API call to the function just works fine, doesn't have any glitch.

☞ On the other side a sound is played synchronously, and `PlaySound` returns after the sound event completes. This is the default behaviour in case you have a list of songs.

If you click on the menu `<Debug/Modules Count>` you can see all the libraries (modules) loaded by your app and of course also the `winmm.dll` with our function `PlaySound()` in it.

```
25 function BOOL PlaySound(
  LPCTSTR pszSound,
  HMODULE hmod,
  DWORD fdwSound
);
```


mX System Modules Library List

```
14: C:\WINDOWS\system32\imm32.dll
15: C:\WINDOWS\system32\URLMON.DLL
16: C:\WINDOWS\system32\wininet.dll
17: C:\WINDOWS\system32\CRYPT32.dll
18: C:\WINDOWS\system32\MSASN1.dll
19: C:\WINDOWS\system32\shell32.dll
20: C:\WINDOWS\system32\winspool.drv
21: C:\WINDOWS\system32\comdlg32.dll
22: C:\WINDOWS\system32\wsock32.dll
23: C:\WINDOWS\system32\WS2_32.dll
24: C:\WINDOWS\system32\WS2HELP.dll
25: C:\WINDOWS\system32\winmm.dll
26: C:\WINDOWS\system32\MSCTF.dll
27: C:\WINDOWS\system32\UxTheme.dll
28: C:\WINDOWS\system32\DNSAPI.dll
```

1: Result as of the loaded Modules

1.2 Code with API Call

As you already know the tool is split up into the toolbar across the top, the editor or code part in the centre and the output window shell like a song length protocol result at the bottom.
Included in the package maXbox is the integrated MP3 Songs utility player as an add-on listed on the menu `/Options/Add Ons/`. But we show the tool as a script extract with source, an add-on is a real compiled and integrated script.

There is a special function in Win32 which plays .wav files. You do not need the full Windows Media Control Interface. You can either play the sound by mentioning its filename (in which case the file needs to be distributed with the program) or by making the file into a resource with a line like

```
PlaySound("SoundName", hInst, SND_RESOURCE | SND_ASYNC);
```

The following example stops playback of a sound that is playing asynchronously:

```
PlaySound(NULL, 0, 0);
```

If you set the flag `SND_NOSTOP` we get an interesting constellation.
This specified sound event will yield to another sound event that is already playing. If a sound cannot be played because the resource needed to generate that sound is busy playing another sound, the function immediately returns `FALSE` without playing the requested sound.

If this flag is not specified, PlaySound attempts to stop the currently playing sound so that the device can be used to play the new sound (again).

Discussion:
I have an app that has a background sound and a sound when someone clicks something. But when they click the something sound it stops the background sound, which is supposed to loop. How can I play both sounds and leave the background looping?

Answer:
I'm 99% sure that the PlaySound API (which SoundPlayer.Play is wrapping) does not support this no matter how you do it. You will have to look at something like DirectX or multithreading if you really want to do this.
I'm trying to get the SoundPlayer to play two wav files at the same time. The documentation says is should play asynchronously but when I try, it seems to end the play of the first SoundPlayer as soon as the second one starts to play.
I read the documentation, and it doesn't imply that it will merge the two sounds, merely that the method call is asynchronous. Internally PlaySound calls the PlaySound API from mmsystem.h, with the parameters SND_ASYNC and SND_NODEFAULT. This means so far that "The sound is played asynchronously and PlaySound returns immediately after beginning the sound..." and "No default sound event is used. If the sound cannot be found, PlaySound returns silently without playing the default sound."

So, as far as I can see it implies that the API call queues up the sounds to be played, and the method returns when your sound starts playing, so there's no parallel processing possible!

Before this starter code will work you will need to download maXbox from the website. It can be down-loaded from http://sourceforge.net/projects/maxbox site. Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is. If you double-click maxbox3.exe the box opens a default program. Make sure the version is at least 3.8 because Modules Count use that. Test it with F2 / F9 or press **Compile** and you should hear a sound a browser will open. So far so good now we'll open the example:

263_async_sound.txt

If you can't find the file use the link (size 3 KB, 2666 Bytes):

http://www.softwareschule.ch/examples/263_async_sound.txt

Or you use the Save Page as… function of your browser[1] and load it from examples (or wherever you stored it). One important thing: The mentioned DLL must reside on your disk. Now let's take a look at the code of this project first with the declaration of the DLL. Our first line is

```
6 Program ASYNC_Sound_Parallel;
```

We have to name the program it's called ASYNC_Sound_Parallel; Next we jump to line 8:

```
7
8 function PlaySound(sres: pchar; hmod: HMODULE; syncflag_: dword): bool;
9   external 'PlaySound@winmm.dll stdcall';
```

First in line 8 we ask to set function parameters of a DLL. Parameters are important for all IT systems that offer interfaces for user or programming access. Particularly, this includes machines in the private and commercial sector as well as services like electronics or multimedia. In these cases the external declaration is usually done by entering a DLL name and the according function name.

[1] Or copy & paste

✌As we now know, `PlaySound` can't play two sounds at the same time, even if you do use `async` flag. You might be able to get this to work by having two separate threads both calling `PlaySound` synchronously.

The "MediaPlayer" object will not let you play two sounds at once, even if you create two instances. You will need to bring in the native windows API "`mciSendString`".

You can test it with F4 or menu `/Output/New Instance` which opens a second instance of maXbox (see the following screenshot) with the same script!

An example of the low-level `mciSendString()`:

```
mciSendString(@"open C:\Users\Desktop\applause.wav type waveaudio alias
applause", null, 0, IntPtr.Zero);
mciSendString(@"play applause", null, 0, IntPtr.Zero);


mciSendString(@"open C:\Users\Desktop\foghorn.wav type waveaudio alias
foghorn", null, 0, IntPtr.Zero);
mciSendString(@"play foghorn", null, 0, IntPtr.Zero);
```

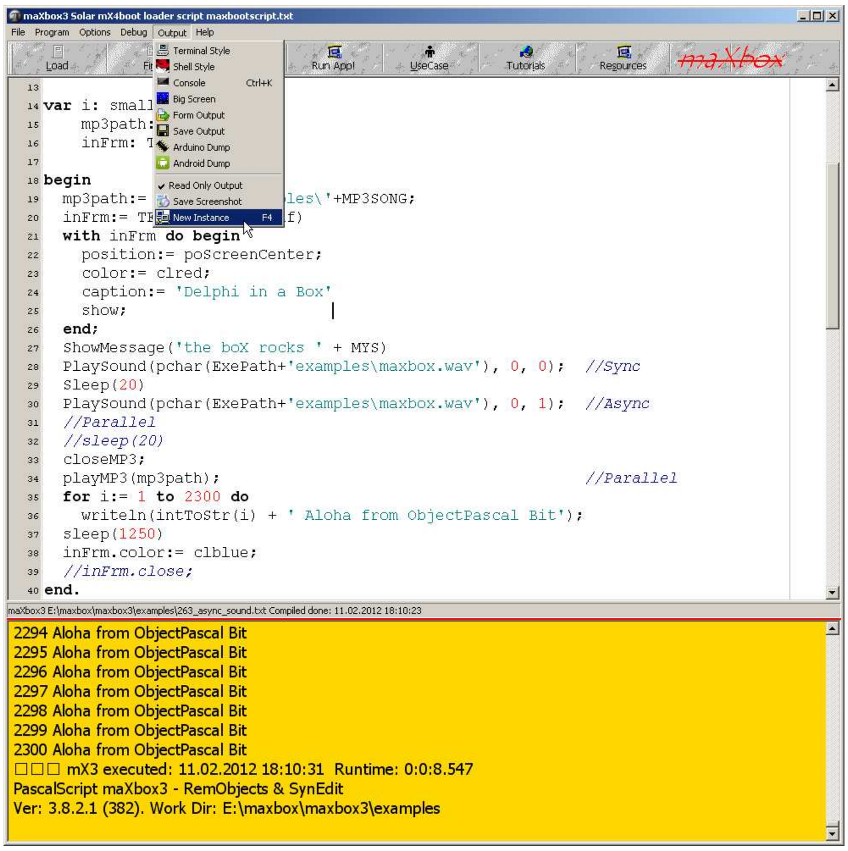
So your code will now show all three possibilities in a sequence. First with start with a dialog which stops the control flow because it’s a modal dialog we have to wait and pass it. Second on line 28 our function is called in sync-mode `sync` and we have to wait or in other words we are blocked. Third in line 30 we call the same function with the `async` flag set to 1 and now you can follow at the same time the music plays and a loop of numbers on the output can be seen.

In line 33 we start almost the same time a sound twice, yes it’s the same sound therefore you can hear the delay or an echo to prove its parallel! Ok. It’s a trick to open another function with the same song to show the simultaneous mode.
If you increase the delay of sound (for example with sleep or with a massive CPU payload), the echo effect space grows to more than milliseconds time of its start offset.

```
//Result:= Writeln(FloatToStr(IntPower(62,8))) = 218340105584896
27   ShowMessage('the boX rocks ' + MYS)
28   PlaySound(pchar(ExePath+'examples\maxbox.wav'), 0, 0);  //Sync
29   Sleep(20)
30   PlaySound(pchar(ExePath+'examples\maxbox.wav'), 0, 1);  //Async
31   //Parallel
32   //sleep(20)
33   closeMP3;
34   playMP3(mp3path);                                       //Parallel Trick
35   for i:= 1 to 2300 do
36     writeln(intToStr(i) + ' Aloha from ObjectPascal Bit');
37   sleep(1250)
38   inFrm.color:= clblue;
39   //inFrm.close;
40 End.
```

☞This `sleep` is here just so you can distinguish the two sounds playing simultaneously.

By the way: You can also set a hash function around a sound file to distinguish it.
A hash function is a (mathematical) function which receives an input of arbitrary length and returns a function value (hash value) of a fixed length (usually 128 or 160 bits).



```

var i: small
    mp3path:
    inFrm: T

begin
  mp3path:=                les\'+MP3SONG;
  inFrm:= TF               f)
  with inFrm do begin
    position:= poScreenCenter;
    color:= clred;
    caption:= 'Delphi in a Box'
    show;
  end;
  ShowMessage('the boX rocks ' + MYS)
  PlaySound(pchar(ExePath+'examples\maxbox.wav'), 0, 0);  //Sync
  Sleep(20)
  PlaySound(pchar(ExePath+'examples\maxbox.wav'), 0, 1);  //Async
  //Parallel
  //sleep(20)
  closeMP3;
  playMP3(mp3path);                                        //Parallel
  for i:= 1 to 2300 do
    writeln(intToStr(i) + ' Aloha from ObjectPascal Bit');
  sleep(1250)
  inFrm.color:= clblue;
  //inFrm.close;
end.
```

```
maXbox3 E:\maxbox\maxbox3\examples\263_async_sound.txt Compiled done: 11.02.2012 18:10:23
```

```
2294 Aloha from ObjectPascal Bit
2295 Aloha from ObjectPascal Bit
2296 Aloha from ObjectPascal Bit
2297 Aloha from ObjectPascal Bit
2298 Aloha from ObjectPascal Bit
2299 Aloha from ObjectPascal Bit
2300 Aloha from ObjectPascal Bit
□□□ mX3 executed: 11.02.2012 18:10:31  Runtime: 0:0:8.547
PascalScript maXbox3 - RemObjects & SynEdit
Ver: 3.8.2.1 (382). Work Dir: E:\maxbox\maxbox3\examples
```

Till now we are discussing the topics of `sync` and `async` calls and the way it processes the calls by the receiver parallel or not. Asynchronous calls or connections allow your app to continue processing without waiting for the process (function) to be completely closed but not always in a simultaneous mode. Therefore we come now to the discussion about real parallel programming with threads (multithreading) or multiprocessing programming.

1.3 Notes about Threads

Multi-threaded applications are applications that include several simultaneous paths of execution. While using multiple threads requires careful thought, it can enhance your programs by:

- Avoiding bottlenecks. With only one thread, a program must stop all execution when waiting for slow processes such as accessing files on disk, communicating with other machines, or displaying multimedia content. The CPU sits idle until the process completes. With multiple threads, your application can continue execution in separate threads while one thread waits for the results of a slow process.
- Organizing program behaviour. Often, a program's run can be organized into several parallel processes that function independently. Use threads to launch a single section of code simultaneously for each of these parallel cases.
- Multiprocessing. If the system running your program has multiple processors, you can improve performance by dividing the work into several threads and letting them run simultaneously on separate processors.

One word concerning a processing thread: In the internal architecture there are 2 threads categories.

- Threads with synchronisation (blocking at the end)
- Threads without synchronisation (non blocking at all)

In case you’re new to the concept, a thread is basically a very beneficial alternative (in most cases) to spawning a new process. Programs you use every day make great use of threads whether you know it or not but you have to program it.
The most basic example is whenever you’re using a program that has a lengthy task to achieve (say, downloading a file or backing up a database), the program (on the server) will most likely spawn a thread to do that job.
This is due to the fact that if a thread was not created, the main thread (where your main function is running) would be stuck waiting for the event to complete, and the screen would freeze.
For example first is a listener thread that “listens” and waits for a connection. So we don't have to worry about threads, the built in thread will be served by for example Indy though parameter:

```
   IdTCPServer1Execute(AThread: TIdPeerThread)
```

When our DWS-client is connected, these threads transfer all the communication operations to another thread. This technique is very efficient because your client application will be able to connect any time, even if there are many different connections to the server.
The second command "`CTR_FILE`" transfers the app to the client:

Or another example is AES cryptography which is used to exchange encrypted data with other users in a parallel way but not a parallel function. It does not contain functions for key management. The keys have to be exchanged between the users on a secure parallel channel. In our case we just use a secure password!

```
procedure EncryptMediaAES(sender: TObject);


 with TStopwatch.Create do begin
    Start;
    AESSymetricExecute(selectFile, selectFile+'_encrypt',AESpassw);
    mpan.font.color:= clblue;
    mpan.font.size:= 30;
    mpan.caption:= 'File Encrypted!';
    Screen.Cursor:= crDefault;
    Stop;
    clstBox.Items.Add('Time consuming: ' +GetValueStr +' of: '+
          inttoStr(getFileSize(selectFile))+' File Size');
    Free;
 end;
```

And that’s how to get a feeling of the speed of AES with a protocol extract:

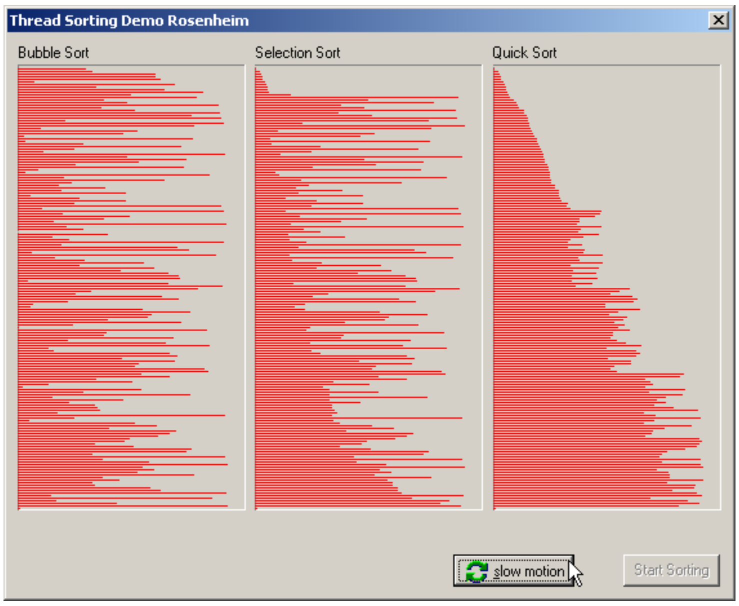
```
Use native.AES-256 cipher and native.CBC chaining mode.
Ciphertext size = 21726 bytes.
Decryption succeeded. 21726 bytes processed.
Speed of decryption was 1326 KiB per second.  //4 GByte about 40 Minutes
```

✌To understand threads one must think of several programs running at once. Imagine further that all these programs have access to the same set of global variables and function calls.

Each of these programs would represent a thread of execution and is thus called a thread. The important differentiation is that each thread does not have to wait for any other thread to proceed. If they have to wait we must use a synchronize mechanism.

All the threads proceed simultaneously.
To use a metaphor, they are like runners in a race, no runner waits for another runner. They all proceed at their own rate.

☞ Because Synchronize uses a form message loop, it does not work in console applications. For console applications use other mechanisms, such as a mutex (see graph below) or critical sections, to protect access to RTL or VCL objects.



2: 3 Threads at work with Log

The point is you can combine asynchronous calls with threads! For example asynchronous data fetches or command execution does not block a current thread of execution.
But then your function or object has to be thread safe. So what's thread-safe. Because multiple clients can access for example your remote data module simultaneously, you must guard your instance data (properties, contained objects, and so on) as well as global variables.

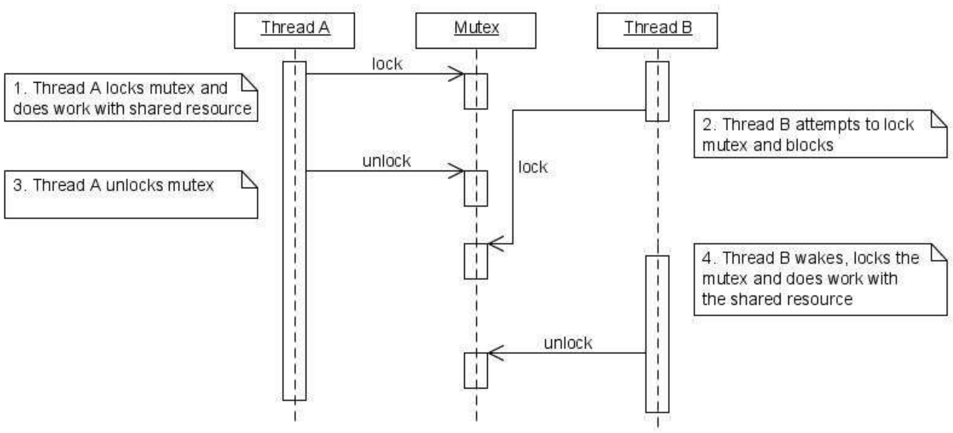
Tasks for advanced studies:

How can you crack a password with a massive parallel concept? Study all about a salt. A 512-bit salt is used by password derived keys, which means there are `2^512` keys for each password. So a same password doesn't generate always a same key. This significantly decreases vulnerability to 'off-line' dictionary' attacks (pre-computing all keys for a dictionary is very difficult when a salt is used). The derived key is returned as a hex or base64 encoded string. The salt must be a string that uses the same encoding.

We also use a lot of more multi scripts to teach and the wish to enhance it with a thread, simply take a look in the meantime at `141_thread.txt`, `210_public_private_cryptosystem.txt` and `138_sorting_swap_search2.txt`. At least let's say a few words about massive threads and parallel programming and what functions they perform.

Do not create too many threads in your apps. The overhead in managing multiple threads can impact performance. The recommended limit is 16 threads per process on single processor systems. This limit assumes that most of those threads are waiting for external events. If all threads are active, you will want to use fewer.

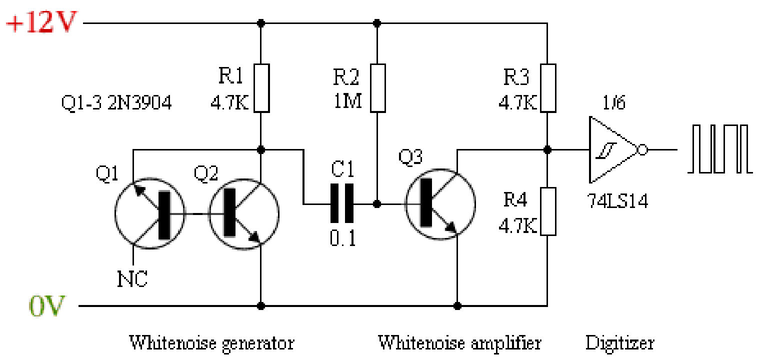
You can create multiple instances of the same thread type to execute parallel code. For example, you can launch a new instance of a thread in response to some user action, allowing each thread to perform the expected response.


Thread A
Mutex
Thread B
lock
1. Thread A locks mutex and does work with shared resource
2. Thread B attempts to lock mutex and blocks
unlock
lock
3. Thread A unlocks mutex
4. Thread B wakes, locks the mutex and does work with the shared resource
unlock

4: A real Thread with a synchronisation object

One note about async execution with fork on Linux with libc-commands; there will be better solutions (execute and wait and so on) and we still work on it, so I'm curious about comments, therefore my aim is to publish improvements in a basic framework on sourceforge.net depends on your feedback ;)

Try to find out more about symmetric/asymmetric encryption systems and the probability to crack it. Due to their large computational requirements, a asymmetric key system is used in practice only for encrypting so called "session keys". The pay load is encrypted with a symmetric algorithm using the session key. This combination of two methods is called hybrid encryption (asymmetric/symmetric). This implies no security deficit if you use a good random number generator for the session key, like a HW generator below!



Try to change the sound file in order to compare two sounds at the same time:

```
04 pchar(ExePath+'examples\maxbox.wav'));
```

Try to find out more about the HW schema above and the question if it works in a synchronous or asynchronous mode.

Check the source of LockBox to find out how a thread is used:

```
05 E:\maxbox\maxbox3\source\TPLockBoxrun\ciphers\uTPLb_AES.pas;
```

Try to change the form message with a string literal const and a modal form:

```
  with inFrm do begin
    position:= poScreenCenter;
    color:= clred;
    caption:= 'Delphi in a Box';
    show;
  end;
```

Next starter will be the number 15 with the topic function building.

max@kleiner.com
Links of maXbox and Asynchronous Threads of DelphiWebStart:

`059_timerobject_starter2_ibz2_async.txt`

http://sourceforge.net/projects/delphiwebstart

http://www.softwareschule.ch/maxbox.htm
http://sourceforge.net/projects/maxbox
http://sourceforge.net/apps/mediawiki/maxbox/

My Own Experience:
http://www.softwareschule.ch/download/armasuisse_components.pdf

SHA1 Hash of maXbox 3.8.2 Win: A47BD7EA26878DCEF2F74852DE688686A1D9523B