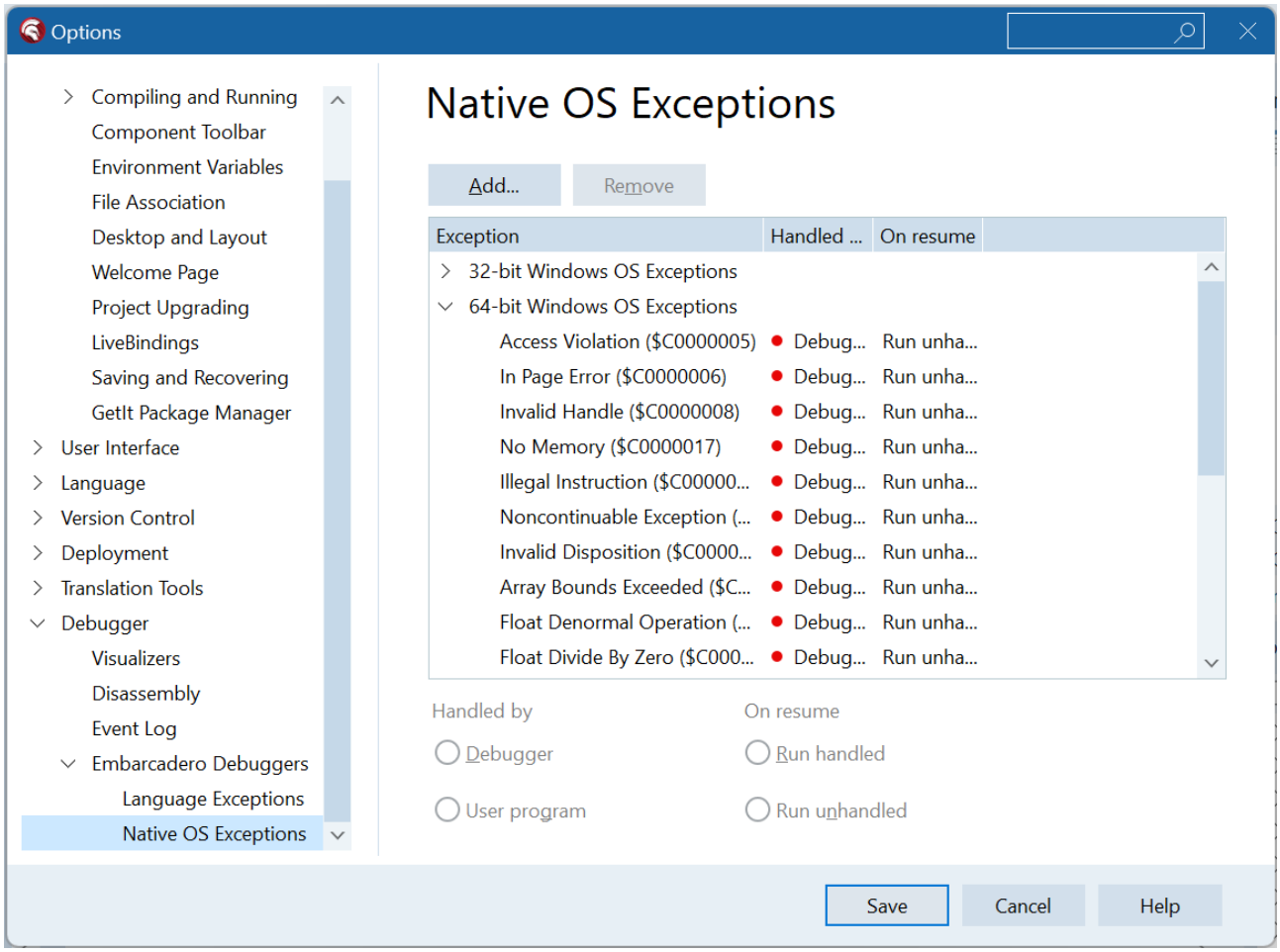
//

Debugging a 64-bit-box.

maXbox Starter 116 – Advanced Debugging under the hood with RAD Studio 11.3 Alexandria.

"Change favour the prepared mind.

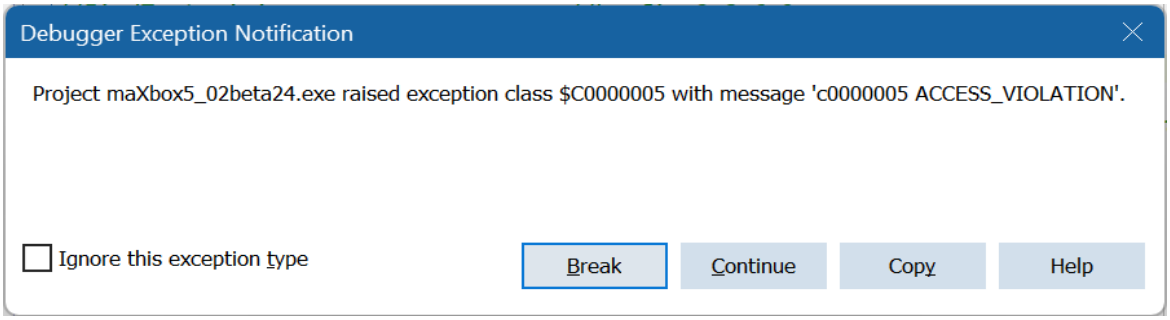
As you my know you can tell the debugger to ignore certain kinds of exceptions. Figure 1 shows Delphi's language-exception and native OS exception options. Add an exception class to the list in language-exceptions, and all exceptions of that type and of any descendant types will pass through to your program without Delphi interfering.



pic1: 1_116_Options_Screenshot2023-11-20160050.png

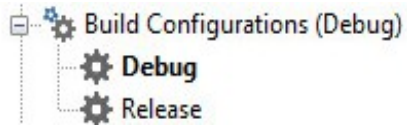
In its default settings, the Delphi IDE notifies you whenever an exception occurs in your program, as in Figure 2. What's important to realize is that at that point, none of your program's exception-handling code has run yet. It's all Delphi itself; its special status as a debugger allows it to get first notification of any exception in your program, even before your program knows about it.
So its not that easy open your 64-bit application in the IDE, add and

activate the 64-bit debugger options, and compile your application as a 64-bit Windows application with the right debugging.



Pic2: 2_116_except_Screenshot2023-11-20160519.png

So Embarcadero decided to have a debug or release mode (and Base!). Base, Debug, and Release are the three default build configurations. In the Project Manager, the Build Configurations node itself represents the Base configuration, but the Debug and Release configurations are listed in separate nodes. Hint: when you change from Debug to Release you have to set your version info separately, there's no bridge in Project Options > Version Info if you change the target!



Pic2_1: 2_1_116_2_1_BuildConfigsNodexe7mX5.png

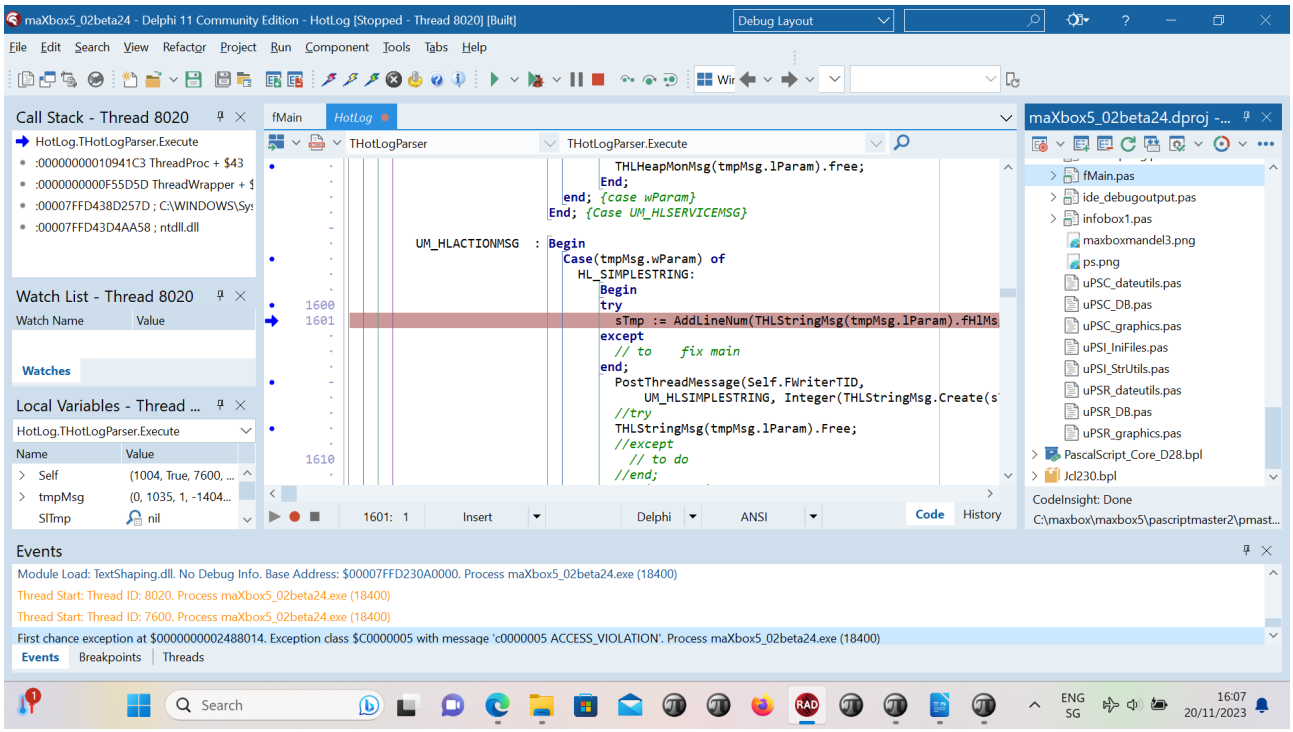
You can change option values in any configuration, including Base. You can delete the Debug and Release configurations, but you cannot delete the Base configuration or move it.

While digging or diving through the source code of maXbox4 it seems to be impossible to migrate over 3340 units (exactly 3354) in a decent and proper way to maXbox5 aka 64-bit Version.

Some, people were complaining about it as creating problems but without actually providing a clear example where this might happen. Additionally, Embarcadero recently added the recommendation to use *FreeAndNil* in their manual (finally!).
But ALWAYS compile the application in Release and Debug mode. Make sure the Project Options are correctly set for debug mode. The DEFAULT settings for Debug mode are NOT correct/complete – at last not in Delphi XE7 and Tokyo in my opinion. Maybe one day they will set the correct options for Debug mode if any exists. So, enable and study things like:

- "Stack frames"
- "Map file generation (detailed)"
- "Range checking",
- "Symbol reference info"
- "Debug information"
- "Overflow checking"
- "Assertions"
- "Debug DCUs"

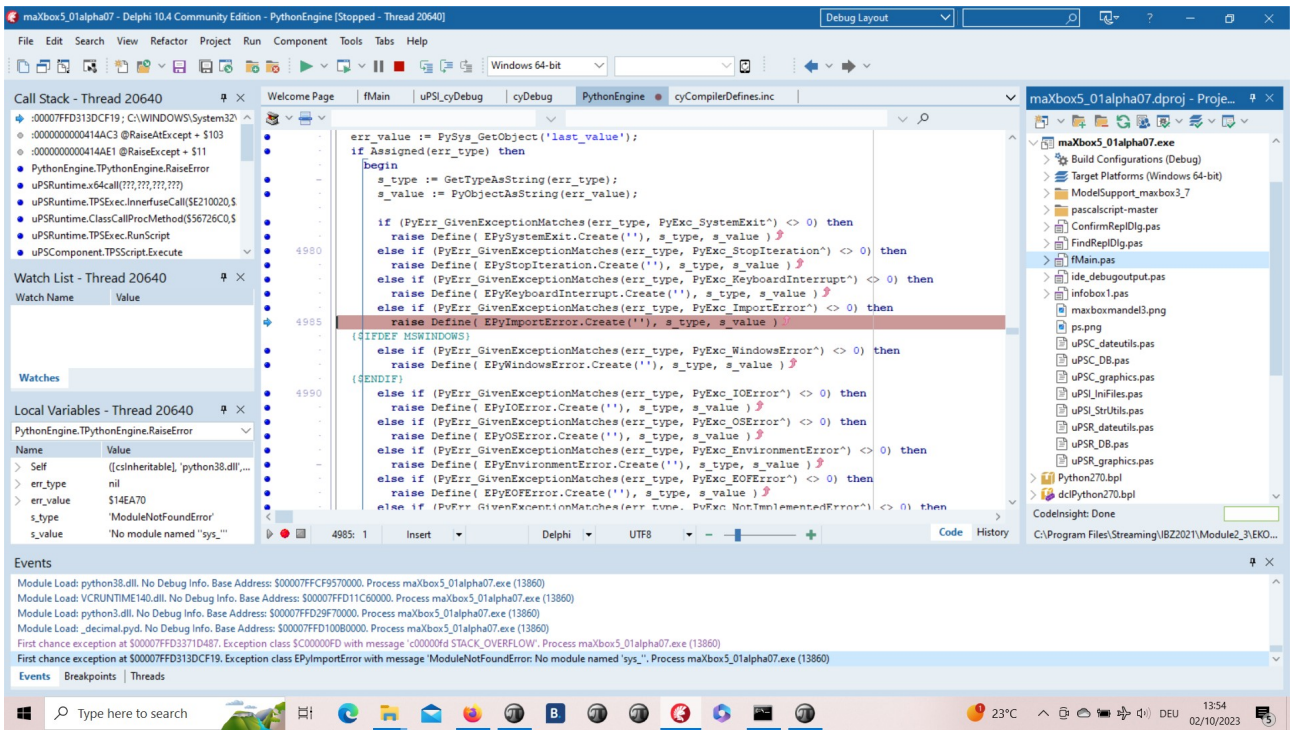
Most of the time you get some first chance exceptions.
After you break the debugger exception notification in Pic1 you get exactly the line to fix the bug (is was a Nil pointer):



Pic3: 3_116_brake_Screenshot2023-11-20160745.png

You can use Delphi's "advanced breakpoints" to disable exception handling around a region of code. To begin, set a breakpoint on the line of code where you want the IDE to ignore exceptions. Right-click on the breakpoint dot in the gutter and open the breakpoint-property dialogue. In the advanced section are some check boxes. Clear the "Break" box to prevent the debugger from interrupting your program at that line, and set the "Ignore subsequent exceptions" box. Afterward, set another breakpoint where you want the debugger to resume handling exceptions. Change its properties to handle subsequent exceptions.

What are **subsequent exceptions**: It handles all subsequent exceptions raised by the current process during the current debug session (the debugger will stop on exceptions based on the current exception settings in Tools > Options > Debugger Options > Embarcadero Debuggers > Language Exceptions. This option does stop on all exceptions. Use it to turn on normal exception behaviour after another breakpoint disabled normal behaviour using the Ignore subsequent exceptions option.
It makes sense in a block of code with the option **Ignore subsequent exceptions;** Ignores all subsequent exceptions raised by the current process during the current debug session (the debugger will not stop on any exception). (Normally you know **Halts execution;** the traditional and default action of a breakpoint). Use this Ignore subsequent exceptions **with** Handle subsequent exceptions as a pair. You can surround specific blocks of code with the Ignore/Handle pair to skip any exceptions which occur in that block of code like in the following Pic.



Pic4: 4_116_buildexceptioncatchinfo_mX5.png

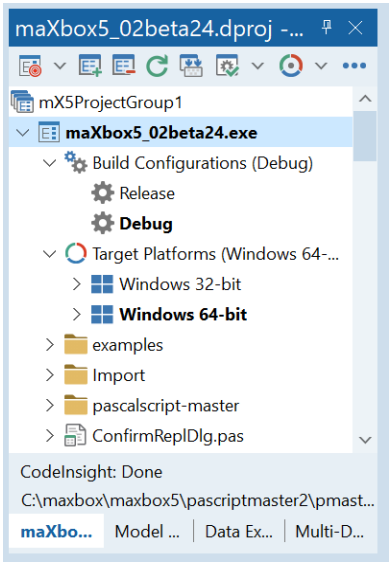
Prepare to debug the Debugger

In Delphi 11 – and probably most other versions – if an exception escapes from the Execute method without being handled, then it is caught by the function that called Execute and stored in the thread's FatalException property. (Look in Classes.pas, ThreadProc.) Nothing further is done with that exception until the thread is freed, at which point the exception is also freed.

```
procedure TForm1.Onterminate(Sender: TObject);
var ex: TObject;
begin
  Assert(Sender is TThread);
  ex:= TThread(Sender).FatalException;
  if Assigned(ex) then begin
  // Thread terminated due to an exception
    if ex is Exception then
    Application.ShowException(Exception(ex))
```

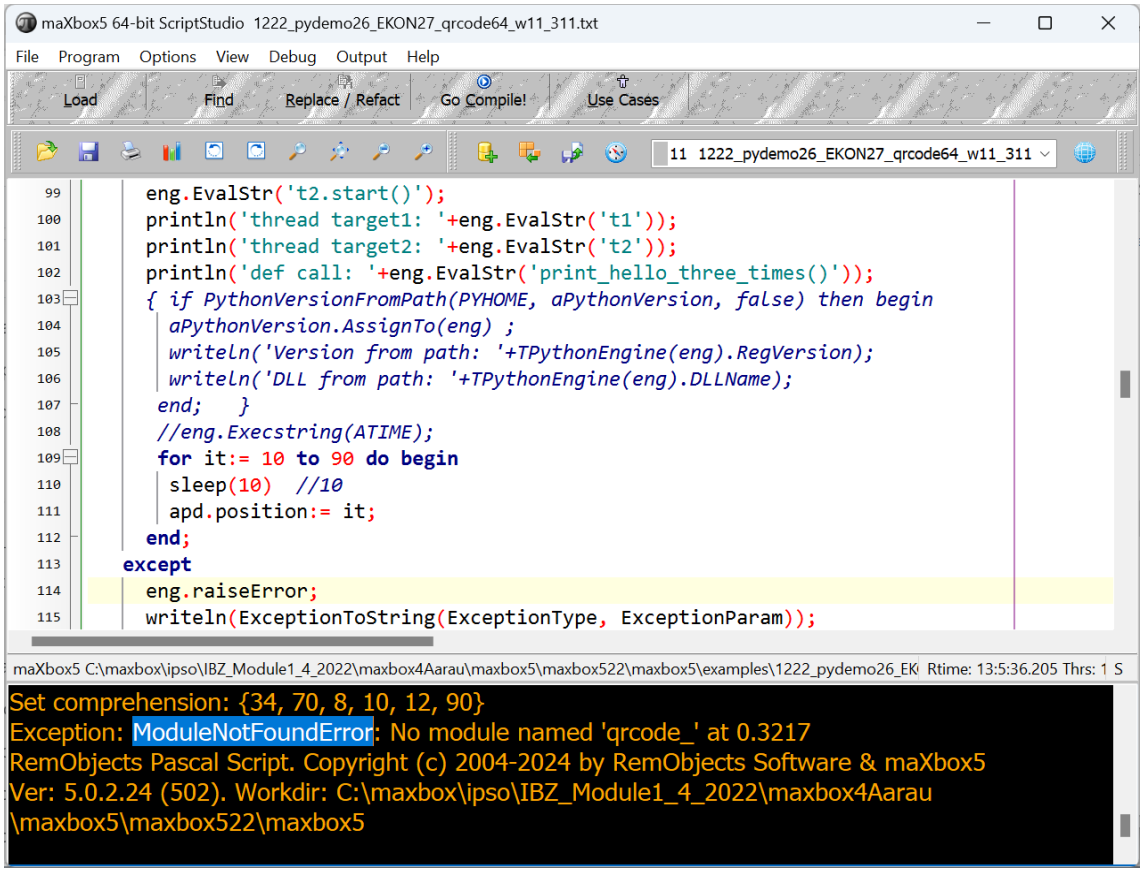
Unlike the Windows API TerminateThread MSDN, which forces the thread to terminate immediately, the Terminate method merely requests that the thread terminate. This allows you the thread to perform any cleanup and finalisation before it shuts down.
To catch the exceptions that occur inside your thread function, add a try...except block to the implementation of the Execute method!

So I prepared the Project and debugging as Debug mode like this:



Pic5: 5_116_prepare_Screenshot 2023-11-20 173110.png

The odd thing is that I have wrapped my pascal call in a try except, which has handlers for AccessViolationException, COMException and everything else, but when Delphi 10.4 or Studio 11.3 intercepts the AccessViolationException, the debugger breaks on the method call (doc.OCR), and if I step through, it continues to the next line instead of entering the catch or except block. So I decided to to catch the exception on the script in a runtime routine to get the most on my console from a dynamic debugbox maXbox:



Pic6: 6_116_handleexception_screenshot-2023-11-14-130654.jpg

I also activated the JITEnable variable, it controls when the just-in-time debugger is called.
So for the reorganisation of the sources I have the latest revision with patches from issue #202 (commit 86a057c) but I am unable to compile the files at first (Core_D27) that are part of the **PascalScript_Core_D27.dpk** for that platform for Linux64, Win64 nor MacOS64.

Here's some source output at first to show the internal exception handling for the 10.4 dccosx64 or dcc64 compiler (similar results exist for dcclinux64):

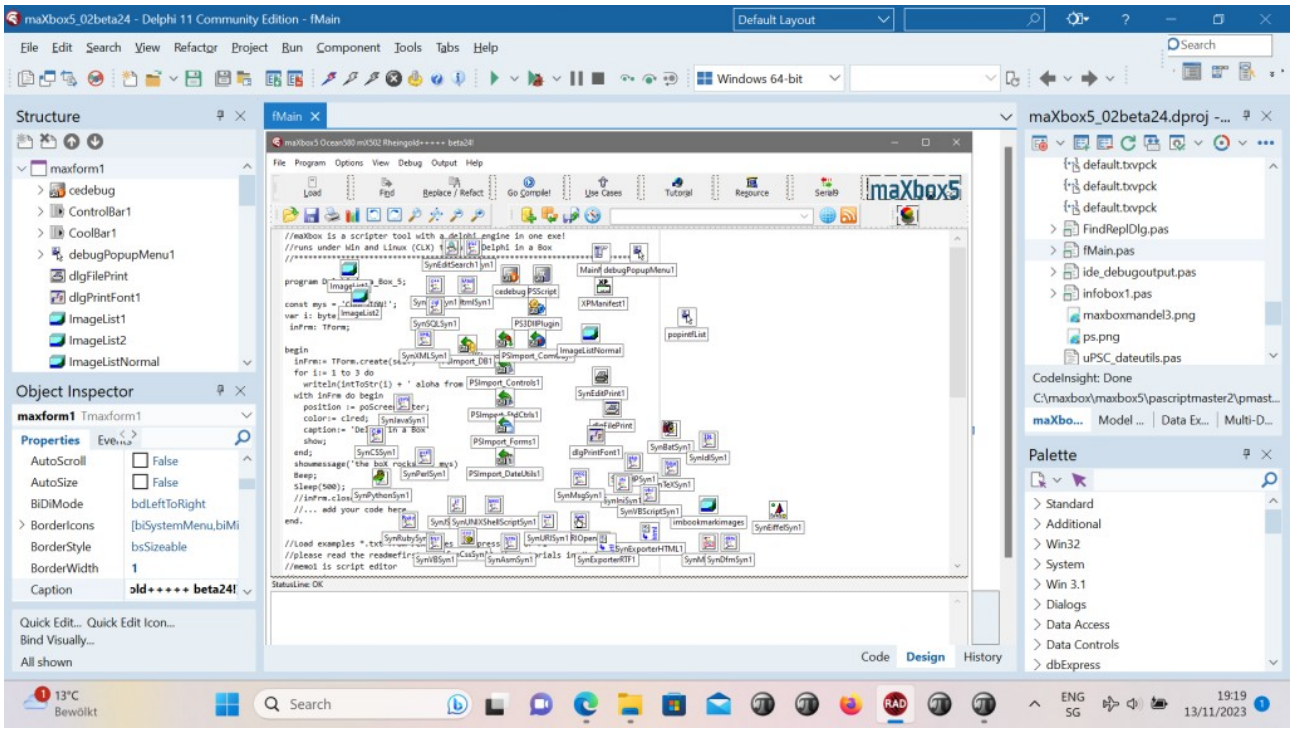
```
procedure TPSExec.ExceptionProc(proc, Position: Cardinal; Ex: TPSError;
                                const s: tbtString; NewObject: TObject);
var
  d, l: Longint;
  pp: TPSExceptionHandler;   //debcnt: integer
begin
  ExProc:= proc;
  ExPos:= Position;
  ExEx:= Ex;
  ExParam:= s;
  inc(debcnt);
  if maxform1.GetStatDebugCheck then
     maxform1.memo2.lines.add('debug: '+inttostr(debcnt)+'-'+s+'
               '+inttostr(proc)+' err:'+inttostr(ord(ex)));     //@fmain
  if ExObject <> nil then
    ExObject.Free;
  ExObject:= NewObject;
  //ShowMessage('We do not get this far: '+exparam);
  if Ex = eNoError then Exit;
   //maxform1.memo2.lines.add(s);
   // halt(1);
   // ShowMessage('We don't want not get this far');

  for d:= FExceptionStack.Count -1 downto 0 do begin
    pp:= FExceptionStack[d];
    if Cardinal(FStack.Count) > pp.StackSize then begin
      for l:= Longint(FStack.count) -1 downto Longint(pp.StackSize) do
        FStack.Pop;
    end;
```

Then I can see the **ExceptionProc** as a First chance exception at $0000000100419E9E. Exception class EAccessViolation with message 'Access violation at address 0000000100419E9E, accessing address 00000009017241F8'. Process TestApplication (5741)
Source Breakpoint at : C:\Program Files\Streaming\IBZ2021\Module2_3\EKON26\maxbox4\pascalscript-master\pascalscript-master\Source\uPSRuntime.pas line 2060. Process TestApplication (5741)

```
Upsruntime.TPSExec.Clear()(0x00000002017350d0)
Upsdebugger.TPSCustomDebugExec.Clear()(0x00000002017350d0)
Upscomponent.TPSScript.Compile()(0x0000000201734c20)
Fmain.TForm1.Compile1Click(System.TObject*)(0x00007ffeefbfe038)
Vcl.Menus.TMenuItem.Click()(0x0000000201734960)
Vcl.Menus.TMenu.DispatchCommand(unsigned short)(0x0000000201734340,2)
Vcl.Forms.TCustomForm.WMCommand(Winapi.Messages.TWMCommand&)
(0x0000000205039ff0,0x00007ffeefbfe878)
:000000010001132B System::Tobject::Dispatch(void*)
```

Another disturbing point by debugging was a redirect to debug;-). Since I use *SynPdf.pas* I have to include *SynCommons.pas* in my project. But it seems I've got more than I wanted. Sometimes when I debug in IDE and try to dig into some routine I get to Move procedure from *SynCommons.pas* instead of the normal system call where I want to come!
The solution is easy; FastCodem, Move and Fillchar are included within *SynCommons.pas* optimized for speed and SmartLinking won't make it big in your exe and logging and all the other won't be part of it.
You can get rid of it, by commenting the corresponding lines in the initialization block of this unit. Under new versions of the framework, you have a conditional setting to disable it.

```
RedirectCode(GetAddressFromCall(@RecordCopyInvoke),@RecordCopy);
RedirectCode(GetAddressFromCall(@FillCharInvoke),@FillChar);
RedirectCode(GetAddressFromCall(@MoveInvoke),@Move);
```



Pic7: 7_116_guiorg_screenshot-2023-11-13-191920.jpg

When I choose a second compile in a CrossVCL from the menu I always test also the debugger with two code functions as similarities but with different code solutions, for example GetBytes:

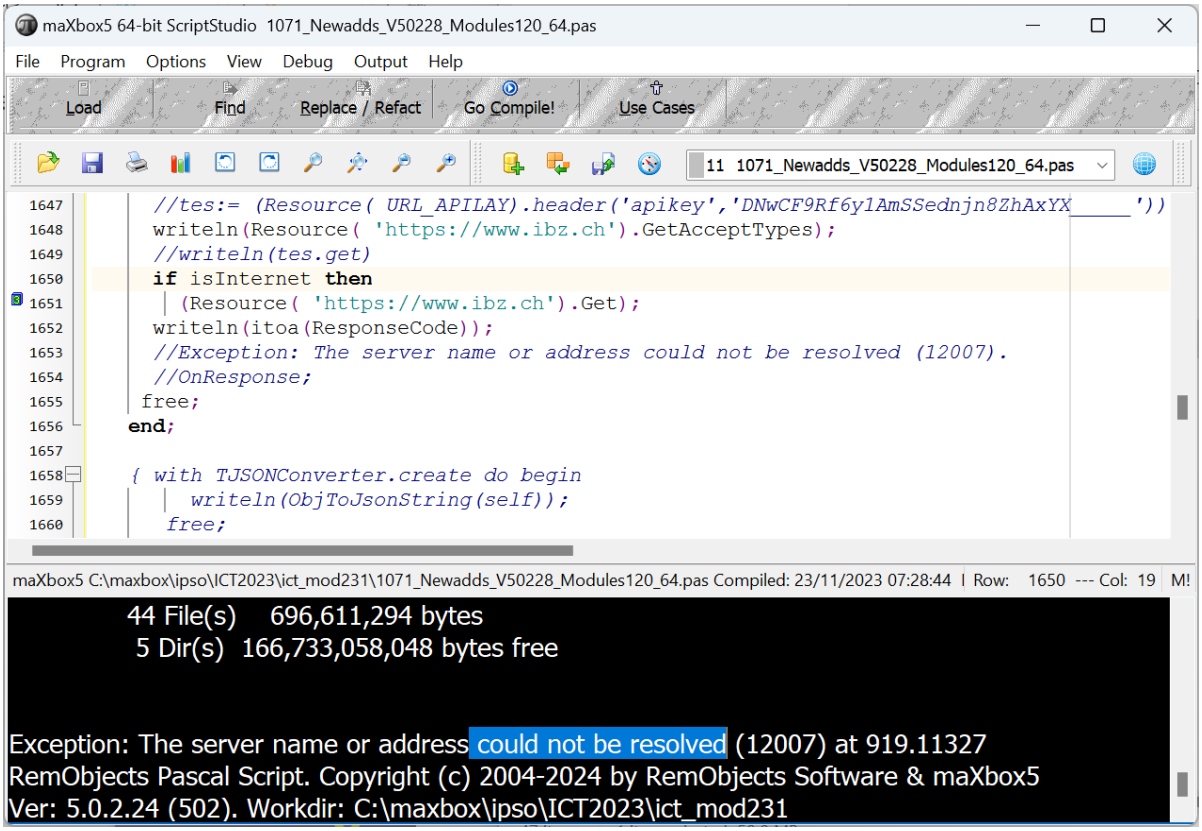
```
function GetBytes(value: string): TBytes;
begin
  SetLength(Result, SizeOf(value));
  Move(value, Result[0], SizeOf(value));
end;

function AnsiBytesOf(const S: string): Tbytes; //like GetBytes
begin
  Result := TEncoding.ANSI.GetBytes(S);
  //Result := GetBytes(S);
end;
```

And then maybe by intuition I made a build and the AD has gone away and I got my first screen, compiled and script executed:

One of the solved problems in the meantime is to catch an Access-violation instead of crash the app. Its like the code in *uPSRuntime.pas* could not catch cause of halt or exit like the following:

```
procedure TdynamicDll.Quit;
begin
  if not( csDesigning in ComponentState ) then begin
{$IFDEF MSWINDOWS}
    MessageBox( GetActiveWindow, PChar(GetQuitMessage), 'Error',
                                          MB_TASKMODAL or MB_ICONSTOP );
    ExitProcess( 1 );
{$ELSE}
    WriteLn(ErrOutput, GetQuitMessage);
    Halt( 1 );
{$ENDIF}
  end;
end;
```

After debugging I realized it is a first chance exception which works as long the debugger is running with break or continue but without debugger the app disappears without forwarding the AV on the output like AV at address xyz read of address 000.
You may know that The **Application.OnException** handler is only called for unhandled exceptions. An unhandled exception is one where no try..except block has caught the exception or where it has been caught and then re-raised!
Execution will continue in any outer except block. If there are none, or if any that may exist also re-raise the exception, then eventually the exception will reach the Application.OnException handler.
In fact: Application.OnException is your last chance to deal with an unhandled exception. It is not the first opportunity to respond to any exception.
Catching the exception and displaying the message is only helpful if you've already released the software to users and cannot reproduce the problem locally (and even then, it's not as good as using a real exception library, like **EurekaLog** or **MadExcept**). In this case, we already know which line caused the exception, which exception class was thrown, and what its message was. The suggested code would add no information to the investigation
So if you have a network error or exception in your web call from a Rest-Client, it could be wise to check the notwork-connection before you catch the possible exception, as seen in the next image.
Hint to Error 12007: I am attempting to fetch the HTML source behind an onion site via the WinINet API but the InternetOpenUrl() returns the error code of 12007 which suggests that there is an issue regrading the resolution of the web address. The unusual part of this problem is that the error is not reproducible when a non-onion site is used for the web link, which clearly means that there is no issue in the part concerning a possible proxy configurations.
Also *GetLastError* function returns error code like 12007. You can also check with netmon tool if an actual connection attempt has been made.



Pic7b: 7b_116_Handleit 2023-11-23 073156.png

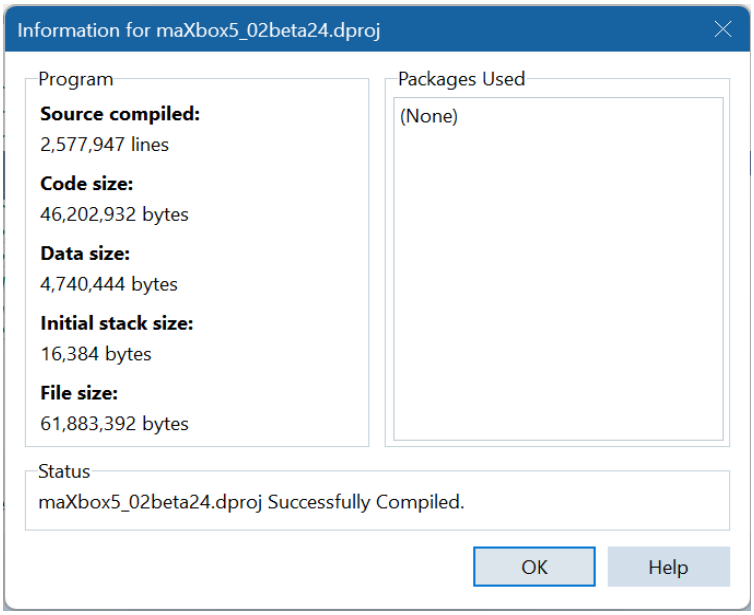
As we have seen you can tell the debugger to ignore certain kinds of exceptions. Add an exception class to the list, and all exceptions of that type and of any descendant types will pass through to your program without Delphi interfering. You can use Delphi's "advanced breakpoints" to disable exception handling around a region of code. To begin, set a breakpoint on the line of code where you want the IDE to ignore exceptions or the call is an outside API like the following User-Agent.

```
'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36(KHTML, like Gecko) Chrome/88.0.4324.104 Safari/537.36'}
try:
    request_result = requests.get(url, headers=headers).json()
    print(request_result)
    print('[In English]: ' + request_result['alternative_translations'][0]
['alternative'][0]['word_postproc'])
    print('[Language Dectected]: ' + request_result['src'])
except:
     pass
```

Pic8: 8_116_maxbox5_64logologo5.jpg

Conclusion:

Breakpoints pause program execution at a certain location or when a particular condition occurs. You can set source breakpoints and module load breakpoints in the Code Editor before and during a debugging session. Application.OnException is your last chance to deal with an unhandled exception. Basically exceptions are thrown to the debugger first and then to the actual program where if it isn't handled it gets thrown to debugger a second time, giving you a chance to do something with it in your IDE before and after the application itself.



Pic9: 9_116_boxbuild_Screenshot 2023-11-21 082511.png

References:

Compiled Project: https://github.com/maxkleiner/maXbox4/releases/download/V4.2.4.80/maxbox5.zip

Docs and Tool: https://maxbox4.wordpress.com

maXbox herunterladen | heise Download

Max Kleiner, 23/11/2023

maXbox4exe Digitally signed by maXbox4exe
Date: 2023.11.23 16:42:31 +01'00'