EKON 14

PascalScript Development

Goal: Build a script in your App to be more "other language friendly" and less IDE or platform dependent (interpreter).

Kleiner
Kommunikation.....

What’s a Script Engine ?

- Why use a scripting engine?

- A scripting engine allows an end user to customize an application to his or her needs without having to recompile it. In addition, you can update your applications by just sending a new script file that could even be compiled to byte code, which cannot easily be transformed back to source code.

- Advantages of Scripting:
- scripting is specific to runtime
- testing or simulate becomes more plan able
- no installation, setup's or configuration needed
- scalable in product, e.g. a light- or professional version

How it works ?

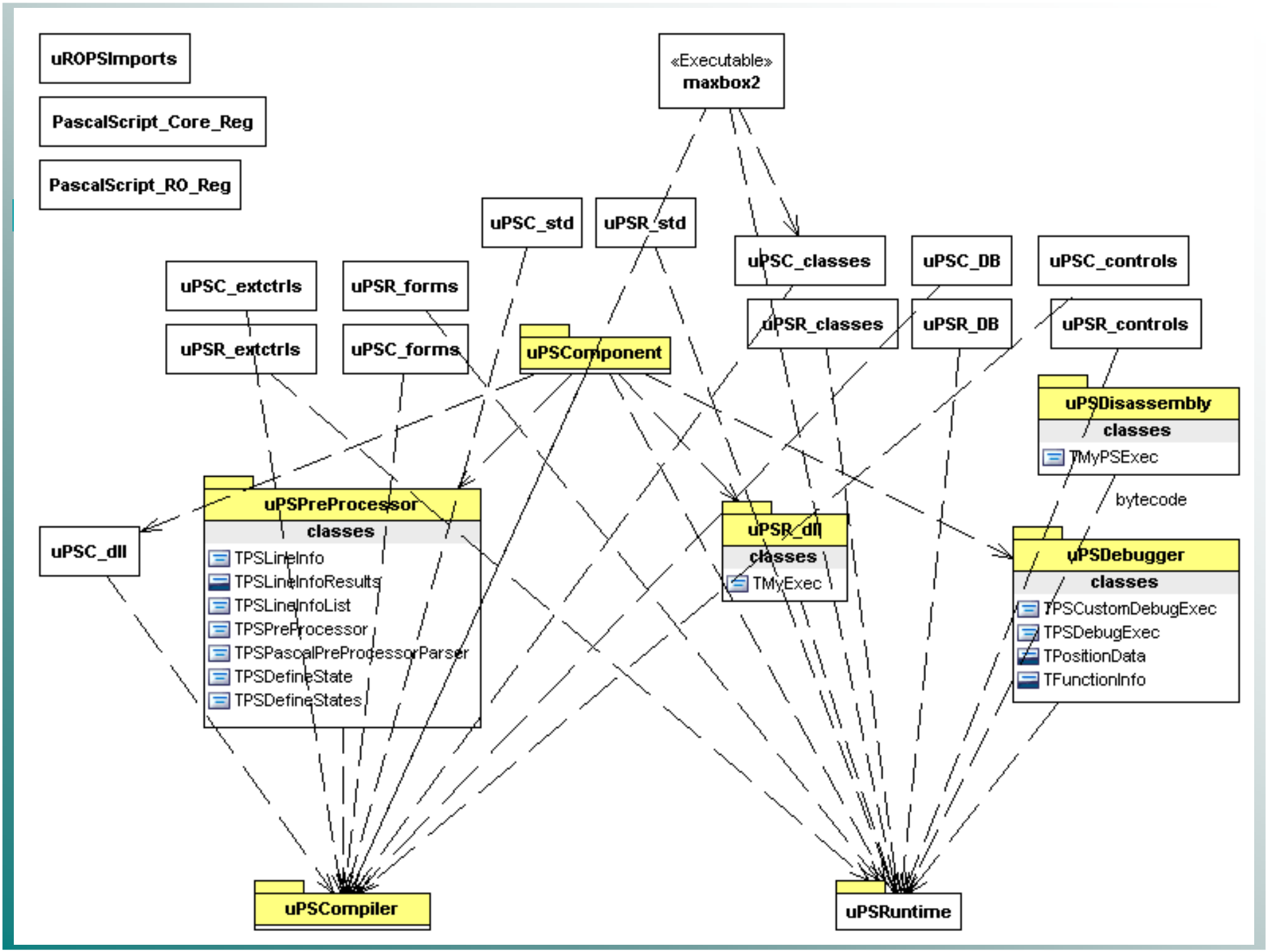
- procedure ExecuteScript(const Script: string);
- var

  Compiler: TPSPascalCompiler;

  { TPSPascalCompiler is the compiler part of the scriptengine. This translates a (macro stack machine) script into a compiled form the executer understands. }

- Exec: TPSExec; //Runtimer

  { TPSExec is the executer part of the scriptengine. It uses the output of the compiler to run a script. }


uROPSImports
PascalScript_Core_Reg
PascalScript_RO_Reg
«Executable»
maxbox2
uPSC_std
uPSR_std
uPSC_classes
uPSC_DB
uPSC_controls
uPSC_extctrls
uPSR_forms
uPSR_classes
uPSR_DB
uPSR_controls
uPSR_extctrls
uPSC_forms
uPSComponent
uPSDisassembly
classes
TMyPSExec
bytecode
uPSC_dll
uPSPreProcessor
classes
TPSLineInfo
TPSLineInfoResults
TPSLineInfoList
TPSPreProcessor
TPSPascalPreProcessorParser
TPSDefineState
TPSDefineStates
uPSR_dll
classes
TMyExec
uPSDebugger
classes
TPSCustomDebugExec
TPSDebugExec
TPositionData
TFunctionInfo
uPSCompiler
uPSRuntime

Data types

Variables, Constants

Standard language constructs:

Functions inside the script

Calling any external DLL function (no special function headers required)

Calling registered external methods

All common types like Byte, Shortint, Char, Word, SmallInt, Cardinal, Longint, Integer, String, Real, Double, Single, Extended, Boolean, Array, Record, Enumerations, Variants

Minimal Class of Compiler / Executer

```
type
  TPSCE = class
   protected
     FScr: TPSScript;
     procedure SaveCompiled(var Data: String);
     procedure SaveDissasembly(var Data: String);
     procedure OnCompile(Sender: TPSScript);
     procedure OnExecImport(Sender: TObject; se: TPSExec; x:
                                                TPSRuntimeClassImporter);
   public
     constructor Create;
     function Compile(const FileName: string): Boolean;
     function Execute: Boolean;
  end;
```

Pre-processor included

- The $I parameter directive instructs the compiler to include the named file in the compilation. In effect, the file is inserted in the compiled text right after the {$I filename} directive.
- type
  TPSPreProcessor = class;
- TPSPascalPreProcessorParser = class;
- {Event}
- TPSOnNeedFile = function (Sender: TPSPreProcessor; const callingfilename: string; var FileName, Output: string): Boolean;
- { Line info structure (internal debug info)
- To specify a filename that includes a space, surround the file name with single quotation marks: {$I 'My file'}.
- Ex.: maXbox pascalscript .inc

The Executer

- CI: TPSRuntimeClassImporter;
- RuntimeClass: TPSRuntimeClass;
- ftest: TMyTestObject;
  begin
- Compiler:= TPSPascalCompiler.Create;
  // create an instance of the compiler.
- Compiler.OnUses:= ScriptOnUses;
  // assign the OnUses event.
- if not Compiler.Compile(Script) then
  // Compile the Pascal script into bytecode.
- Compiler.Free;

Or not going…

```
if not Exec.LoadData(Data) then
// Load the data from the Data string.

{ For some reason the script could not be loaded. This is
usually the case when a library that has been used at
compile time isn't registered at runtime. }

Exec.Free; // You could raise an exception here.

Exit; end;

Exec.RunScript; // Run the script.
Exec.Free; // Free the executer.
end;
```

…to The Debugger (into the night)

- Debug Data:
- \#0+ Proc0Name+#1+Proc1Name+#1+Proc2Name+#1#0
- \#1+ Var0Name+#1+Var1Name+#1+Var2Name+#1#0
- \#2+ MI2S(FuncNo)+ Param0Name+#1+Param0Name+#1#0
- \#3+ MI2S(FuncNo)+ Var1Name+#1+Var1Name+#1#0
- \#4+ FileName + #1 + MI2S(FuncNo)+ MI2S(Pos)+
- MI2s(Position)+MI2S(Row)+MI2s(Col)

The Byte Code

Bytecode Format:

Address Space

0..GlobalVarCount -1 = GlobalVars

IFPSAddrStackStart div 2 .. IFPSAddrStackStart -1 = neg. stack
IFPSAddrStackStart... = positive stack

TPSVariable = packed record
TPSHeader = packed record
TPSAttributes = packed record, TPSAttribute = packed record
TPSType = packed record
TPSProc = packed record
TPSVar = packed record

The call of a procedure

- \<i\> Flags:
- 1 = Imported; (nameLen: Byte; Name: array[0..namelen-1] of byte) else (Offset, Length: Longint);
- 2 = Export; (only for internal procs); Name, Decl: MyString;
- 3 = Imported2; nameLen: Byte; Name: array[0..namelen-1] of byte; ParamsLength: Longint; Params: array[0..paramslength-1] of byte;
- 4 = With attributes (attr: TPSAttributes)

Your own Functions

Exec.RegisterDelphiFunction(@MyOwnFunction, 'MYOWNFUNCTION', cdRegister);

{ This will register the function to the executer. The first parameter is a pointer to the function. The second parameter is the name of the function (in uppercase).

And the last parameter is the calling convention (usually Register). }

Demo and Example: maxboxkonsole.dpr

Your own Classes

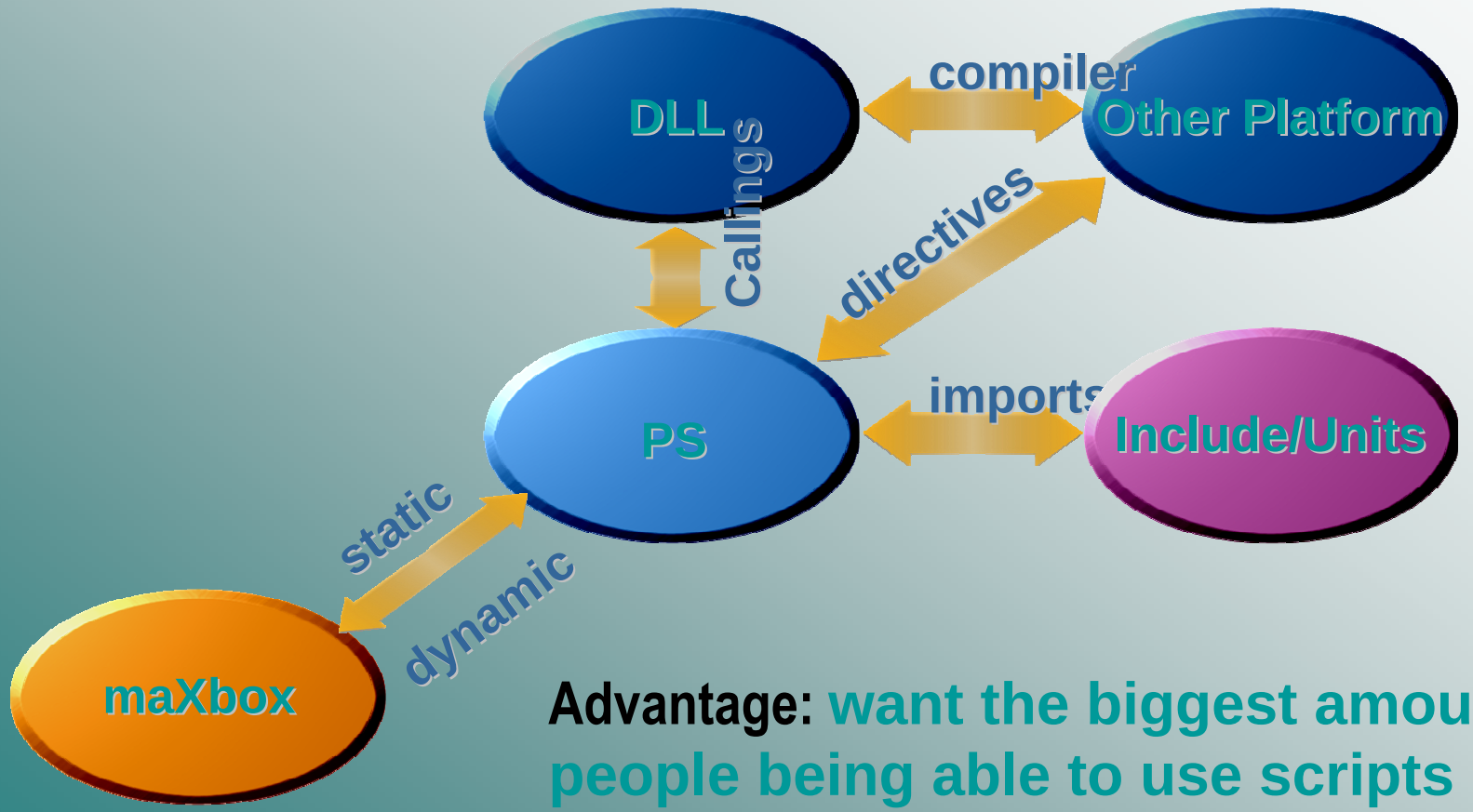
Exec:= TPSExec.Create; // an instance of the executer.

CI:= PSRuntimeClassImporter.CreateAndRegister(Exec, false); RIRegisterTObject(CI);

RuntimeClass:= CI.Add(TMyTestObject);
RuntimeClass.RegisterConstructor(@TMyTestObject.Create, 'Create');
RuntimeClass.RegisterMethod(@TMyTestObject.Print,'Print');

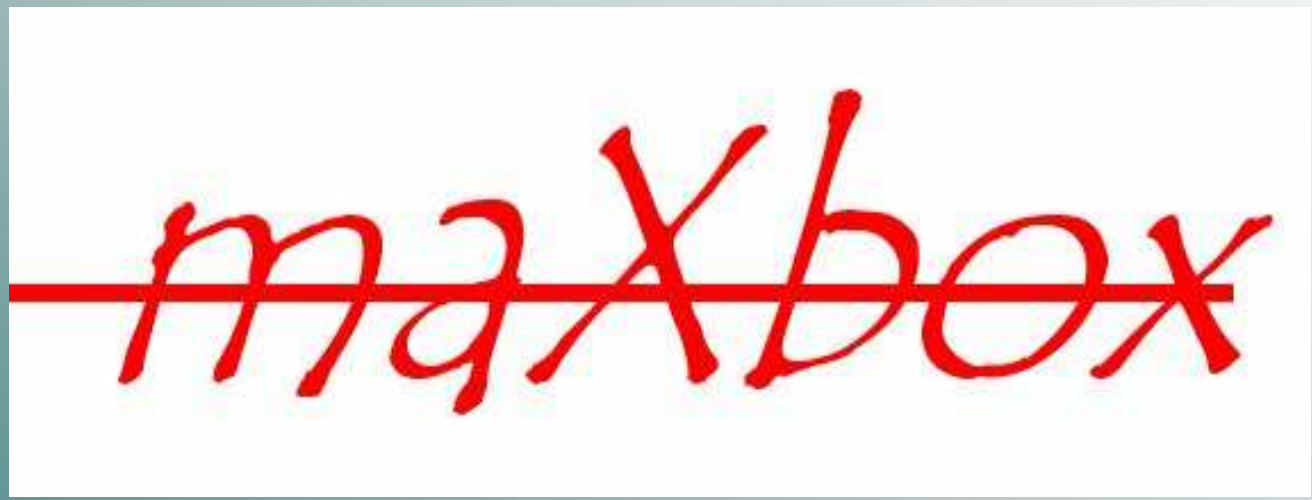
ftest:= TMyTestObject.Create();
SetVariantToClass(Exec.GetVarNo(Exec.GetVar('T')), fTest);
Demo of BinImportTool PSUnitImporter.exe

PS Multi Environment



Advantage: **want the biggest amount of people being able to use scripts**

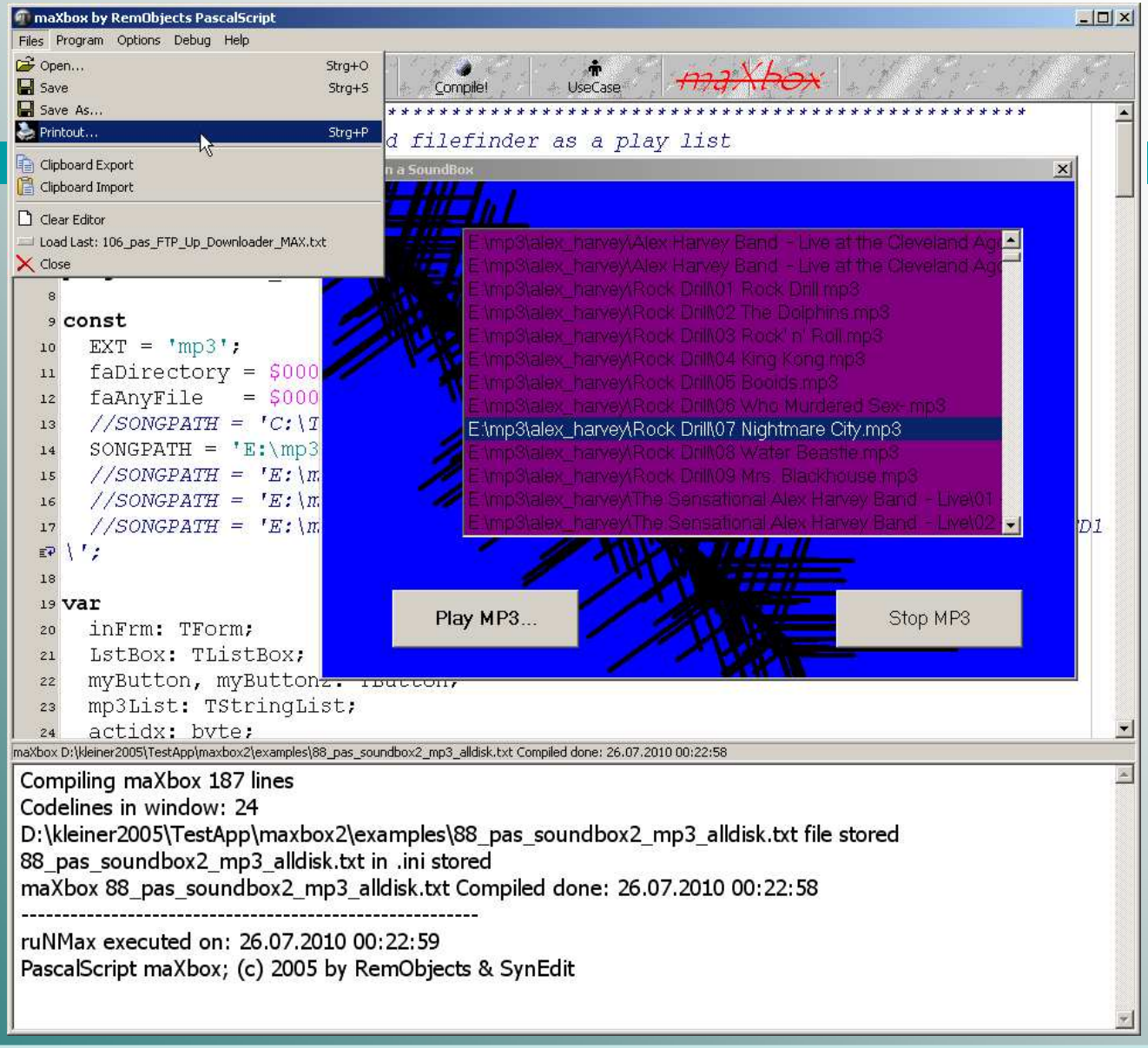
This is maXbox



- softwareschule.ch/maxbox.htm

or Delphi in a Box…

GUI


maXbox by RemObjects PascalScript
Files Program Options Debug Help
Open... Strg+O
Save Strg+S
Save As...
Printout... Strg+P
Clipboard Export
Clipboard Import
Clear Editor
Load Last: 106_pas_FTP_Up_Downloader_MAX.txt
Close
Compile!
UseCase
maXbox
d filefinder as a play list
n a SoundBox
E:\mp3\alex_harvey\Rock Drill\07 Nightmare City.mp3
Play MP3...
Stop MP3
const
EXT = 'mp3';
var
inFrm: TForm;
LstBox: TListBox;
mp3List: TStringList;
maXbox D:\kleiner2005\TestApp\maxbox2\examples\88_pas_soundbox2_mp3_alldisk.txt Compiled done: 26.07.2010 00:22:58
Compiling maXbox 187 lines
Codelines in window: 24
D:\kleiner2005\TestApp\maxbox2\examples\88_pas_soundbox2_mp3_alldisk.txt file stored
88_pas_soundbox2_mp3_alldisk.txt in .ini stored
maXbox 88_pas_soundbox2_mp3_alldisk.txt Compiled done: 26.07.2010 00:22:58
ruNMax executed on: 26.07.2010 00:22:59
PascalScript maXbox; (c) 2005 by RemObjects & SynEdit

MP3 example I

In EXE:

```
procedure playMP3(mp3path: string);
begin
mciSendString(PChar('open '" + mp3path + '" alias MP3'),
    NIL, 0, 0);
  //player.handle
  mciSendString('play MP3', NIL, 0, 0);
end;
```

In Script:

- playMP3(maXboxpath+'examples\maxbox.mp3');

- Demo and Example: 109_pas_mp3_download.txt

Import functions with DLL's

Import Units wraps the functions of the API in DLL's

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

  mmsyst = 'winmm.dll';

implementation

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

As Testing Tool example II

```
Procedure ShuffleList(var vQ: TStringList);
var j, k: integer;
     tmp: String;
begin
   randomize;
   for j:= vQ.count -1 downto 0 do begin
      k:= Random(j+1);
      tmp:= (vQ[j]);
      vQ[j]:= vQ[k];
      vQ[k]:= tmp;
   end;
end;
```

Source Code Versioning

Author: ck
Date: 2010-05-18 10:14:09 +0200 (Tue, 18 May 2010)
New Revision: 224

Modified:
Source/uPSComponent.pas
Source/uPSRuntime.pas
Log:
0: memory leak
Modified: Source/uPSComponent.pas
==========
--- Source/uPSComponent.pas 2010-05-11 15:08:04 UTC (rev 223)
+++ Source/uPSComponent.pas 2010-05-18 08:14:09 UTC (rev 224)
@@ -204,6 +204,8 @@

Some Notes 1

For some functions / constants, it might be necessary to add: uPSCompiler.pas, uPSRuntime.pas and/or uPSUtils.pas to your uses list.

It's possible to import your own classes in the script engine. PascalScript includes a tool to create import libraries in the Bin directory.

For examples on how to use the compiler and runtime separately, see the Import samples.

The Debug requires SynEdit http://synedit.sourceforge.net/.

PS License 2 with FPC Interop

PascalScript 3 is free and comes with source code.

You can also access the latest version of PS directly in our open-source SVN version control system, at code.remobjects.com. Supported Tools and Platforms:

PascalScript supports Delphi 4 through 7 and Delphi 2006 through 2009, as well as the latest Free Pascal 2.x.

Supported Platforms are Win32 Ansi and Unicode (Delphi + FPC), Win64 (FPC), Linux x86 (FPC or CLX) and Linux x64 (FPC).

Call from various Clients 3

```
begin
    {$IFDEF LINUX}
    dllhandle:= dlopen(PChar(s2), RTLD_LAZY);
    {$ELSE}
    dllhandle:= LoadLibrary(Pchar(s2));
    {$ENDIF}
    if dllhandle = {$IFDEF LINUX} NIL {$ELSE} 0 {$ENDIF} then

 {$IFDEF LINUX}
  p.Ext1:= dlsym(dllhandle, pchar(copy(s, 1, pos(#0, s)-1)));
  {$ELSE}
  p.Ext1:= GetProcAddress(dllhandle, pchar(copy(s, 1, pos(#0, s)-1)));
  {$ENDIF}
```

Conclusion

- PascalScript is for Delphi /.NET
- PascalScript is a free scripting engine that allows you to use most of the Object Pascal language within your Delphi or Free Pascal projects at runtime.
- Written completely in Delphi, it is composed of a set of units that can be compiled into your apps, eliminating the need to distribute any external files or components.

Links and Tools on board

- softwareschule.ch/maxbox.htm
- http://delphi-jedi.org/
- http://www.remobjects.com/ps.aspx
- http://sourceforge.net/projects/maxbox
- http://sourceforge.net/apps/mediawiki/maxbox/

- {SAFECODE ON};)
  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

Questions and hope answers ?

max@kleiner.com