Using Innerfuse Pascal Script

Innerfuse Pascal Script (or IFPS) is build up from two different parts:
- Compiler
- Runtime

The Compiler, which can be found in the *ifpscomp.pas* unit and the Runtime, which can be found in the *ifps3.pas* unit have no dependencies on each other. You can use the compiler and runtime directly, or use the component wrapper for it, the *TIFPS3CompExec* component can be found in the *IFPS3CompExec* unit and wraps both the compiler and exec in one easy to use class.

To use the component version of IFPS3 you have to put it on your form, or data module. Set or assign the script property, call the *Compile* method and then the *Execute* method. Any errors, warnings or hints that occur while compiling can be found by looking through the C*ompilerMessages* array property and runtime errors can be found by reading the *ExecErrorToString* property.

The following example will compile and execute an empty script (begin end.):

```
var
  Messages: string;
  compiled: boolean;
begin
  ce.Script.Text := 'begin end.';
  Compiled := Ce.Compile;
  for i := 0 to ce.CompilerMessageCount -1 do
    Messages := Messages + ce.CompilerMessages[i].MessageToString + #13#10;
  if Compiled then
    Messages := Messages + 'Succesfully compiled'#13#10;
  ShowMessage('Compiled Script: '#13#10+Messages);
  if Compiled then
  begin
    if Ce.Execute then
        ShowMessage('Succesfully Executed')
      else
        ShowMessage('Error while executing script: '+Ce.ExecErrorToString);
  end;
end;
```

By default, the component only adds a few standard functions to the script engine (the exact list of those can be found at the top of *ifpscomp.pas*). Besides the standard functions, there are a few libraries included with IFPS3:

TIFPS3DllPlugin:	Allow scripts to use dll functions, the syntax is like: *function FindWindow(C1, C2: PChar): Longint; external 'FindWindowA@user32.dll stdcall';*
TIFPS3CE_Controls:	Import library to Controls.pas and Graphics.pas.
TIFPS3CE_DB:	Import library for db.pas.
TIFPS3CE_DateUtils:	Import library for date/time related functions.
TIFPS3CE_Std:	Import library for TObject and the Classes unit.
TIFPS3CE_StdCtrls:	Import library for ExtCtrls and Buttons.
TIFPS3CE_Forms:	Import library for the Forms & Menus units.

To use these libraries, add them to your form or data module, select the [...] next to the *plugins* property of the *TIFPS3CompExec* component and add a new item, set the Plugin property to the plugin component. Besides the standard libraries, you can easily add new functions to the script engine. To do that, create a new method you would like to expose to the script engine, for example:

```
procedure TForm1.ShowNewMessage(const Message: string);
begin
  ShowMessage('ShowNewMessage invoked:'#13#10+Message);
end;
```

Then, you need to assign an event handler to the *OnCompile* event and use the *AddMethod* method of *TIFPS3CompExec* to add the actual method:

```
procedure TForm1.CECompile(Sender: TIFPS3CompExec);
begin
  Sender.AddMethod(Self, @TForm1.ShowNewMessage, 'procedure ShowNewMessage
(const Message: string);');
end;
```

A sample script that uses this function could look like this:

```
begin
  ShowNewMessage('Show This !');
end.
```

Advanced Features

IFPS3 includes a preprocessor that allows you to use defines (*{$IFDEF}, {$ELSE}, {$ENDIF}*) and include other files in your script. (*{$I filename.inc}*), to enable these features, you have to set the *UsePreprocessor* property to true, you also have to set the *MainFileName* to the name of the script you put in the *Script* property. The *Defines* property specifies which defines are set by default, and the OnNeedFile event is called when an include file is needed.

```
function TForm1.ceNeedFile(Sender: TObject; const OrginFileName: String;
  var FileName, Output: String): Boolean;
var
  path: string;
  f: TFileStream;
begin
  Path := ExtractFilePath(ParamStr(0))  + FileName;
  try
    F := TFileStream.Create(Path, fmOpenRead or fmShareDenyWrite);
  except
    Result := false;
    exit;
  end;
  try
    SetLength(Output, f.Size);
    f.Read(Output[1], Length(Output));
  finally
    f.Free;
  end;
  Result := True;
end;
```

When these properties are set, the CompilerMessages array property will include the file name these messages occur in.

Another feature of IFPS3 is the possibility to call scripted functions from Delphi. The

following sample will be used as a script:

```
function TestFunction(Param1: Double; Data: String): Longint;
begin
  ShowNewMessage('Param1: '+FloatToString(param1)+#13#10+'Data: '+Data);
  Result := 1234567;
end;

begin
end.
```

Before this scripted function can be used, it has to be checked to match it's parameter and result types, this can be done in the *OnVerifyProc* event.

```
procedure TForm1.CEVerifyProc(Sender: TIFPS3CompExec;
  Proc: TIFPSInternalProcedure; const Decl: String; var Error: Boolean);
begin
  if Proc.Name = 'TESTFUNCTION' then
  begin
    if not ExportCheck(Sender.Comp, Proc, [btU8, btDouble, btString], [pmIn,
pmIn]) then
    begin
      Sender.Comp.MakeError('', ecCustomError, 'Function header for
TestFunction does not match.');
      Error := True;
    end else
    begin
      Proc.aExport := etExportDecl;
      Error := False;
    end;
  end else
    Error := False;
end;
```

The ExportCheck function checks if the parameters match, in this case, btu8 is a boolean (the result type), btdouble is the first parameter, and btString the second parameter. [pmIn, pmIn] specifies that both parameters are IN parameters.
To call this scripted function, you have to create an event declaration for this function and call that.

```
type
  TTestFunction = function (Param1: Double; Data: String): Longint of object;

...

var
  Meth: TTestFunction;

  Meth := TTestFunction(ce.GetProcMethod('TESTFUNCTION'));
  if @Meth = nil then
    raise Exception.Create('Unable to call TestFunction');
  ShowMessage('Result: '+IntToStr(Meth(pi, DateTimeToStr(Now))));
```

It's also possible to add variables to the script engine that can be used from within the script. To do this, you have to use the *AddRegisteredVariable* function. And in the OnExecute event you can set this:

```
procedure TForm1.ceExecute(Sender: TIFPS3CompExec);
begin
  CE.SetVarToInstance('SELF', Self); // For class variables
  VSetInt(CE.GetVariable('MYVAR'), 1234567);
end;
```

To read this variable back, after the script has finished executing, you can use the

OnAfterExecute event and use *VGetInt(CE.GetVariable('MYVAR'))*.

The previous functions will make a copy of the values, changes in these variables will not be reflected in the application. To make sure the script updates the real value, instead of a copy, you can use:

```
var
  s: string;

procedure TForm1.ceCompile(Sender: TIFPS3CompExec);
begin
  Sender.AddRegistedPTRVariable('s', 'String');
end;

procedure TForm1.ceExecute(Sender: TIFPS3CompExec);
begin
  Sender.SetPointerToData('s', @s, Sender.Exec.FindType2(btString));
end;
```

The component version of IFPS3 also supports executing scripted functions. This works by using the ExecuteFunction method.

```
ShowMessage(CompExec.ExecuteFunction([1234.5678, 4321, 'test'],
'TestFunction'));
```

This will execute the function called 'TestFunction' with 3 parameters, a float, an integer and a string. The result will be passed back to ShowMessage.

Notes:

- For some functions and constants, it might be needed to add: ifpscomp, ifps3 and/or ifps3utl to your uses list.
- The script engine never calls *Application.ProcessMessages* by itself, so it might be that your application will hang, while the script is running, to avoid this, you can add *Application.ProcessMessages* to the *TIFPS3CompExec.OnLine* event.
- It's possible to import your own classes in the script engine. IFPS3 includes a tool to create import libraries in the /Unit-Importing/ directory.
- The source code contains a small description for every method/function in the interface section. These descriptions can also be read from the /Help/Units directory.
- A few small samples can be found in the /Help/
- The /DUnit/ directory contains a DUnit test application for IFPS3. You will need http://dunit.sourceforge.net/ to run this.
- For examples on how to use the compiler and runtime seperately, see the Demo_Import/ and Demo_Kylix.
- The Ide-demo and unit-import requires SynEdit http://synedit.sourceforge.net/.