Script Example Analysis

Modified by Admin on Sep 13, 2017

Related information
Script API Objects
Altium Designer API

Parent page: Scripting

To provide further insight into the general aspects of the Scripting System, and the use of Delphi and DXP Object Models in scripts, two example projects are examined here from a functionality perspective — a board outline copier and a netlist generating script.

The Board Outline Copier and Netlister scripts are developed using the DXP Object Models to illustrate the capabilities of the scripting system in Altium Designer. These are existing scripts available in the example script collection as follows:

- The Board Outline Copier script utilizes the PCB Object Model to copy the existing PCB board outline as tracks and arcs onto a specified layer.
  See `\Scripts\VB Scripts\CopyBoardOutlinePRJ.PRJSCR.`
- The Netlist generator script utilizes the WorkSpace Manager Object Model to generate a netlist in the traditional Protel (v1 and v2) format.
  See `\Scripts\Delphiscript Scripts\WSM\Protel Netlister\ScripterProtelNetlist.PRJSCR`.

The DelphiScript and VBScript language sets are used in the illustrated examples.. The DelphiScript language set is based on Embarcadero Delphi and VBScript is based on Microsoft Scripting technology.

For information on the differences between DelphiScript and Object Pascal (used in Delphi) refer to the DelphiScript reference document.
For more information on VBScript, refer to the Microsoft VBScript documentation.

Board Outline Copier Project

The aim of the Board Outline Copier is to copy an existing board outline from the PCB document to a different layer in the same document.

The project uses a Script Form so the user can interact with a dialog to nominate the width of the board outline and select its target layer from a drop down menu. Using the PCB Object model and its PCB interfaces from the PCB API, the objects of a board outline are extracted and copied onto the specified layer.

Note that the Board Outline Copier script discussed here is in VBScript rather than DelphiScript.

The main parts of the script are:

- A global `PCB_Board` (of `IPCB_Board` type) variable.
- A `CopyBoardOutline` subroutine with `AWidth` and `ALayer` parameters.
- The `bCancelClick` event handler which closes the Board Outline script form (dialog).
- The `bOkClick` event handler which obtains the width and layer values from the script form and then executes the `CopyBoardOutline` subroutine.

Script functionality

Since the script uses a script form, there needs to be event handlers that capture the mouse clicks of individual controls such as the **OK** and **Cancel** buttons.

The **OK** button event handler is essentially as below:

```
Sub bOKClick(Sender)
    Dim Width
    Dim Layer

    Call StringToCoordUnit(eWidth.Text,Width,PCB_Board.DisplayUnit)
    Layer = String2Layer(cbLayers.Items(cbLayers.ItemIndex))

End Sub
```

Here, the form's Mouse|OnClick event handler (`bOKClick`) uses the the `StringToCoordUnit` function to obtain the nominated width of the outline (the `eWidth` string from the TEdit box) in internal coordinate values. This function applies the current board units (`PCB_Board.DisplayUnit` property) to the `Width` variable.

Similarly, the selected layer string (a `cbLayers` item from the form TComboBox) is passed to the `String2Layer` function to obtain an enumerated value for the `Layer` variable. Note that in this project the form's TComboBox is pre-populated with an indexed list of layer strings (`cbLayers.Items`).

✓ The DelphiScript version of the Board Outline Copier project uses an additional procedure to extract the list of the available layers from the current board.

The final step in the event handler (`bOKClick`) calls the `CopyBoardOutline` subroutine with the `Width` and `Layer` variables as passed parameters. The handler for the form's **Cancel** button (`bCancelClick`) simply closes the form.

IPCB_Board Interface

In this script the current board outline is obtained from the `IPCB_Board` interface called by the `PCBServer` function — also used to obtain the current board itself:

`PCBServer.GetCurrentPCBBoard`. The board outline needs to be initialized before proceeding with copying and creating a new outline.

A board's outline, represented by the `IPCB_BoardOutline` interface, can be initialized using the interface's rebuilding/validation methods.

```
PCB_Board.BoardOutline.Invalidate
PCB_Board.BoardOutline.Rebuild
PCB_Board.BoardOutline.Validate
```

Outline Arc and Track segments

The `IPCB_BoardOutline` interface, representing the board outline, is inherited from the `IPCB_Group` interface. An `IPCB_Group` interface represents a group object that can store child objects. An example of an `IPCB_Group` interface is a polygon or a board outline, since these can store arcs and tracks as child objects.

A board outline object stores two different type of segments – `ePolySegmentLine` and `ePolySegmentArc` which represent a track or arc object respectively. The number of segments is determined by the `PointCount` method from the `IPCB_BoardOutline` interface, which extracts each outline vertex to the `I` variable for the `For-To-Next` loop.

Each segment of the obtained board outline is checked for tracks and arcs with the `If PCB_Board.BoardOutline.Segments(I).Kind = ePolySegmentLine` Then statement. If not a track segment (`ePolySegmentLine`) the `Else` part of the statement assumes the segment is an arc.

For each segment found and depending on the segment type, a new track or arc object is created using the `PCBObjectFactory` method.

PCBObjectFactory method

Creating the new PCB objects employs the `PCBObjectFactory` method from the `IPCB_ServerInterface` interface.

```
'Create new Track object
PCBServer.PCBObjectFactory(eTrackObject, eNoDimension, eCreate_Default)

'Create new Arc object
PCBServer.PCBObjectFactory(eArcObject, eNoDimension, eCreate_Default)
```

The `PCBObjectFactory` method parameters set the object type (Track, Arc, Via etc), the dimension kind (Linear, Radial etc) and the object creation mode (to local default or a global preferences).

After each track or arc object is created by the `PCBObjectFactory` procedure, its properties are instantiated by the track/arc statements that follow.

The Track and Arc properties are represented by their respective interfaces, `IPCB_Track` and `IPCB_Arc`, where the relevant properties are implemented by the script. For example, the new track coordinates are obtained from the source outline segment vertices, where `Track.X1` and `Track.Y1` represent the track's initial coordinates and the X2 and Y2 properties are its final coordinates.

```
Track.X1       = PCB_Board.BoardOutline.Segments(I).vx
Track.Y1       = PCB_Board.BoardOutline.Segments(I).vy
Track.X2       = PCB_Board.BoardOutline.Segments(J).vx
Track.Y2       = PCB_Board.BoardOutline.Segments(J).vy
Track.Layer    = ALayer
Track.Width    = AWidth
```

As can also be seen in the above code snippet, the track Layer and Width properties are simply defined by the nominated values extracted from the user interface dialog.

Once fully defined, the new objects are added to a specified layer of the PCB document with the `PCB_Board.AddPCBObject(NewObject)` statement, where `NewObject` here is `Track` or `Arc`.

PreProcess and PostProcess

When creating a PCB object, the `PreProcess` method from the `IPCB_ServerInterface` object interface needs to be first invoked to ready the PCB server. After the object creation, the `PostPocess` method (also from the `IPCB_ServerInterface` interface) is applied to inform the server that the object additions are complete.

The `PreProcess` and `PostProcess` methods keep the Undo system and other sub systems of the PCB editor up to date and in synchronization. Below is a representative code snippet with the `PreProcess` and `PostProcess` statements.

```
PCBServer.PreProcess
'Create PCB objects
PCBServer.PostProcess
```

Setting the PCB Layer

When objects are added to a nominated layer that has not been displayed in the PCB document, the layer needs to be forced to be visible. This is handled by the `PCB_Board.LayerIsDisplayed(ALayer) = True` statement, where `ALayer` is the user selected layer.

Document Refresh

Lastly, the PCB document with its new board outline is refreshed by the `PCB:Zoom` command and its associated `Action = Redraw` parameters. The zoom command parameters are applied using the `AddStringParameter` procedure after the parameter buffer is first cleared with the `ResetParameters` method.

Netlister Project

The aim of this Netlister script project is to generate a standard Protel netlist (in either Version 1 or Version 2 formats) for an Altium Designer project containing schematics. A flat netlist of a schematic project is separated into two sections:

- Component designators and the information associated with each component,
- Net names and the information associated with each net name along with pin connections (pins

of a component).

The API's WorkSpace Manager Object Model provides interfaces that represent the project and its constituents – the documents, the components and its pins, and the nets. The WorkSpace Manager is a system server coupled tightly with the Client module that deals with projects and their associated documents. It provides compiling, multi sheet design support, connectivity navigation tools, multi-channel support, multiple implementation documents and so on. To retrieve the WorkSpace Manager interface, invoke the `GetWorkspace` function which yields the `IWorkspace` interface.

For the Netlister script the interfaces of interest are the `IWorkSpace`, `IProject`, `IDocument`, `IComponent` and `INet` interfaces.

Note that some of the interfaces, especially the design object interfaces, correspond to equivalent Schematic Object interfaces. This is because the logical documents in a project are schematic documents with connectivity information. In fact the Schematic Object model can be used instead of the WorkSpace Manager, but the latter provides the functionality to compile a project and extract documents from a project, as well as retrieving data from schematic objects.

The main parts of Netlister the script are:

- A global `TargetFileName` string which is the file name of the netlist.
- A global Netlist `TStringList` collection object which contains the data of the netlist.
- `The WriteComponent_Version1` and `WriteComponent_Version2` procedures.
- `The WriteNet_Version1` and `WriteNet_Version2` procedures.
- `A ConvertElectricToString` function which converts a pin's electrical property to a string
- `The GenerateNetlist` procedure which manages the data generation and the filename, path and directory housekeeping tasks.

The Netlister script is in DelphiScript.

Script functionality

The two parameter-less procedures, `GenerateProtelV1FormatNetlist` and `GenerateProtelV2FormatNetlist`, will appear in the *Select Item to Run* dialog offering the choice of generating a Protel V1 format netlist or a Protel V2 format netlist. These procedures will call the `GenerateNetlist` procedure with the netlist format choice (or 1) as a passed parameter.

```
Procedure GenerateProtelV1FormatNetlist;
Var
 Version : Integer;
Begin
 // Protel 1 Netlist format, pass 0
 GenerateNetlist(0);
End;
Procedure GenerateProtelV2FormatNetlist;
Var
 Version : Integer;
Begin
 // Protel 2 Netlist format, pass 1
 GenerateNetlist(1);
```

```
End;
```

GenerateNetList

The `GenerateNetList` procedure retrieves the workspace interface so that the project interface can be extracted for the current project (`IWorkspace.DM_FocusedProject`).

The project needs to be compiled before nets can be extracted, as the compile process builds the connectivity information of the project. The project interface's `DM_Compile` method performs is a applied (`IProject.DM_Compile`) as shown in the code snippet below.

```
WS := GetWorkspace;
 If WS = Nil Then Exit;
 Prj := WS.DM_FocusedProject;
 If Prj = Nil Then Exit;
 // Compile the project to fetch the connectivity info for design.
 Prj.DM_Compile;
```

The component and net information is stored in the Netlist object of `TStringList` type which is used later to generate a formatted netlist text file. The `TStringList` object is a Delphi class which is available to use in scripts.

The `Generate` procedure is called with passed parameters that define the current project path and filename, plus the netlist format version.

Generate

The `Generate` procedure obtains the project path as the target output path for the generated netlist file, determines the netlist file name and checks the flattened status of the project (`IProject.DM_DocumentFlattened`).

For all the schematic documents in a project, each document is the checked for nets and components with the `WriteNets` and `WriteComponents` procedures, and ultimately extracted to the Netlist object.

Write Nets and Components

A netlist is composed of component and net sections so two procedures are required to write component data and net data separately.

Only nets with greater than two nodes will be written to a netlist so those with less are discarded. For each net, the net name is based on the Net's `DM_CalculatedNetName` method, which extracts the net names from the connectivity information generated by the compile.

Two code snippets for the Components and Nets sections of a netlist are shown below for Version 1 netlist format. Note that the component and net data are stored in the NetList object which is a `TStringList` type.

The generated netlist is composed of two sections; the component information section and the net information section.

Components section

In the `WriteComponent` procedure section, each component found from the Project is checked if it is an actual component, and then the physical designator, footprint and part type values are extracted. These are added to the Netlist object container (`NetList.Add`) to build the netlist itself.

```
If Component <> Nil Then
Begin
 NetList.Add('[');
 NetList.Add(Component.DM_PhysicalDesignator);
 NetList.Add(Component.DM_FootPrint);
 NetList.Add(Component.DM_PartType);
 NetList.Add('');
 NetList.Add('');
 NetList.Add('');
 NetList.Add(']');
End;
```

Nets section

For the Nets section, the the NetName and the Designators are extracted if a net has two pins or more (`INet.DM_PinCount`). The net and pin information plus formatting characters are added to the Netlist container to build the netlist. The below snippet is the net writing procedure for the version 1 netlist.

```
If Net.DM_PinCount >= 2 Then
 Begin
  NetList.Add('(');
  NetList.Add(Net.DM_CalculatedNetName);
  For i := 0 To Net.DM_PinCount - 1 Do
  Begin
    Pin := Net.DM_Pins(i);
    PinDsgn := Pin.DM_PhysicalPartDesignator;
    PinNo := Pin.DM_PinNumber;
    NetList.Add(PinDsgn + '-' + PinNo);
  End;
  NetList.Add(')');
 End;
```

Note that the more verbose netlist format, Protel v2, includes the electrical properties for net pins (`In`, `Out`, `Passive`, `HiZ` etc).

The net writing procedure for the version 2 format (`WriteNet_Version2`) therefore interrogates each net pin's electical property (`INet.DM_Electical`), which is then converted by the called `ConvertElectricToString` procedure - essentially a string conversion lookup table. These are then added to a local string variable (`ElectricalString`), which is in turn added to the Netlist container object.

Create Netlist file

Finally with the Netlist container object fully populated with the project component and net information, in the chosen format, the Netlist data is written to a file (`TStringList.SaveToFile`). The file path and name are defined by the `TargetFileName` string variable, as determined in the `Generate` procedure.

Source URL: https://techdocs.altium.com/display/SCRT/Script+Example+Analysis