



Building Script Projects

Summary

Tutorial

TU0125 (v1.5) Nov 29, 2007

This tutorial covers the general aspects of the Scripting System and the use of Borland Delphi Objects and DXP Object Models in scripts.

Netlist, Board Outline Copier and Query Expression scripts are developed using the DXP Object Models to illustrate the power of the scripting system in Altium Designer.

This tutorial builds on the *Getting Started with Scripting* tutorial and is broken up into several sections:

- [Scripting Fundamentals](#)
- [Executing Scripts](#)
- [Debugging Scripts](#)
- [Using the Script Completion features](#)
- [Using Borland Objects in your scripts](#)
- [Using DXP Object Models in your scripts](#)
- [Building a Netlist project](#)
- [Building the Board Outline Copier project](#)
- [Using Query Expressions](#)

This tutorial examines the existing real world script examples:

1. A netlist generator script utilizes the WorkSpace Manager Object Model to generate a Protel netlist format.
2. A Board Outline Copier script utilizes the PCB object model to copy the existing PCB board outline as tracks and arcs onto a specified layer.
3. A basic query expression script to run in the *PCB Filter* panel that searches for specified objects on a PCB document.

The DelphiScript and VBScript language sets are used in this tutorial. The DelphiScript language set is based on Borland Delphi and VBScript is based on Microsoft Scripting technology.



For information on the differences between DelphiScript and Object Pascal (used in Borland Delphi) section, please refer to the *Delphi Script Reference* document.



For more information on VBScript, refer to the Scripting documentation by Microsoft at <http://msdn2.microsoft.com/en-us/library/d1et7k7c.aspx>


Scripting Fundamentals

Before we can start writing or debugging scripts successfully, we need to define terms that are related to scripting within Altium Designer.

- Altium Designer uses an internal DXP Run Time Library which is used by the Scripting System. In addition, Altium Designer uses a subset of the Borland Delphi Run Time Library
- DXP Run Time Library has a set of Application programming Interfaces (APIs). Each API represents an editor in Altium Designer. For example the PCB editor has its PCB API, the Schematic editor has its Schematic API and the Project Manager has its Workspace manager API and so on
- Each API has an Object Model which consist of object Interfaces. Each object interface can consist of methods and properties (but no variables).
- DXP Object Interfaces represent actual design objects in Altium Designer. You can use the DXP Object Model to extract and modify data from design documents open in Altium Designer from your scripts in DelphiScript, JScript, VBScript and EnableBasic
- Object Interfaces consist of methods and properties (but have no variables)
- You need to have the target design document open first before you can run a script using DXP Object Models. To deal with PCB documents and objects, you need to use the PCB Object Model, and to deal with Projects and documents in general, you will need to use the Workspace Manager Object Model
- You need to check the validity of Object Interfaces by either checking whether they are not Nil or using the `Assigned` function. For example `If Assigned(Board) = True Then // do something with the board which is a IPCB_Board type.`

Script Units and Forms

- A Script Form has two views – the Script **Code** view and the Script **Form** view. The Script Code view contains event handlers and procedures/functions. The Script Form view represents a dialog form (of different types) and have controls and their associated event handlers.
- A Script Form has two associated files. A * . `PAS` file that contains has event handlers and procedures/functions and a * . `DFM` file that has details of the script form itself, the components on this form, and their locations on this form as well.
- A component is a control that accepts the user's input such as a mouse click or a sequence of characters entered. Components are visual or non visual control objects on the *Tool Palette* panel that can be manipulated on a Script Form using the *Object Inspector* panel during design.
- A component has methods, properties, and events. Methods are the actions an object can perform. Properties represent the data contained in the Object Interface which can be accessed or modified. Events are conditions a component on a Script Form can react to. There are event handlers in a Script Form that process captured conditions. The Components (implemented with Borland DelphiTM) can be used in any Script Form in any language set supported by the Scripting system.

 Consult the Scripting Resources accessible through the bottom part of the *Knowledge Center* panel in Altium Designer. Navigate to the appropriate API reference document via **Configuring the System » Scripting in Altium Designer » Altium Designer RTL Reference**.

Executing Scripts


A Script Project helps you to manage your scripts. There are two script types, Script Units and Script Form. Any script (using the same language set) within a project has access to global variables and procedures so it is important to have unique procedure and global variable names.


To run a script that requires access to DXP Object Interfaces, you need to run the script on the currently focused and specific document type. For example, if your script has PCB functions to update PCB design objects and you attempt to run the script in the text editor, you will get undeclared identifier errors.


Therefore, it is imperative to check for the presence of a PCB document first before proceeding with the script that has the PCB Object Interfaces.

Use unique names for procedures/routines and global variables for scripts in the same project.

You can install script projects in the **Installed Projects** list which means these projects are seen as global to Altium Designer. The global variables and procedures from these scripts are available for use in other open script projects. To make your script project global, navigate to the **DXP » Preferences** menu and drill down to **Scripting System** folder. Click the **Install** button which brings up the *Select Script Project File* dialog. Navigate to your script projects and click **OK**. Repeat for all script project that you would like to make globally available.

 For an introduction to scripting, refer to the [Getting Started with Scripting](#) tutorial

 For more general information on resources in Altium Designer, refer to the tutorial, [Customizing the Altium Designer](#)

 Also refer to the script examples, mostly in DelphiScript, in the `\Examples\Scripts\` folder which are available for experimentation.

Running a Script from the Run Process dialog

To run a script from the *Run Process* dialog (**DXP » Run Process**), You will need to specify the full path to the script project and specify which script unit and procedure in order to execute the script. Two parameters need to be specified: the `ProjectName` and the `ProcName` parameters.

There are two parameters in this case: the `ProjectName` and the `ProcName`. For the `ProcName` parameter, you need to specify the script filename and the main procedure in this script. The format is as follows: `ProcName = ScriptFileName > ProcedureName`. Note the **Greater Than (>)** symbol used between the script file name and the procedure name.

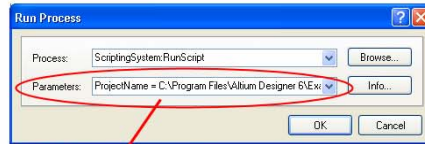
Thus, to execute a script from the *Run Process* dialog:

```
Process: ScriptingSystem:RunScript
```

```
Parameters: ProjectName = PathToProject | ProcName = ScriptFileName >
ProcedureName
```

Note (Parameter = Value) blocks are separated by a Pipe (Vertical Bar) symbol eg `Param1 = Val1 | Param2 = Val2`.

Building Script Projects



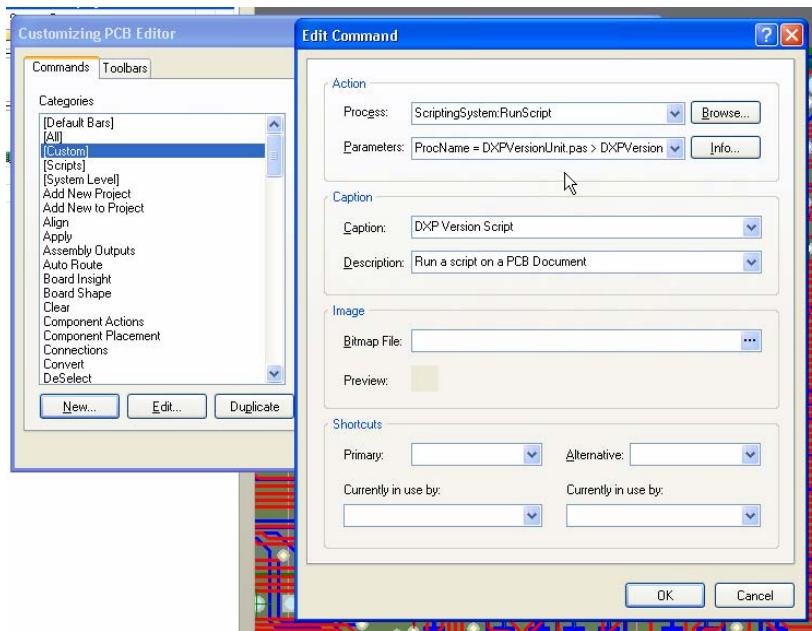
`ProjectName = C:\Program Files\Altium Designer 6\Examples\Scripts\Delphiscript Scripts\Dxp\DXPVersion.PrjScr | ProcName = DXPVersionUnit.pas > DXPVersion`

For example, to run a DXPVersion script in Altium Designer from the *Run Process* dialog, we need to have `ScriptingSystem:RunProcess` in the **Process:** field and `ProjectName = C:\Program Files\Altium Designer 6\Examples\Scripts\Delphiscript Scripts\Dxp\DXPVersion.PrjScr | ProcName = DXPVersionUnit.pas > DXPVersion` in the **Parameters:** field.


Running a Script from a key, toolbar or menu item


You can assign a script to a server menu, toolbar or hot key in Altium Designer and making it possible for you to run the script in your current document. For example to run the open the PCB document and then:

1. Double click on the menu or alternatively use the **DXP » Customize** command, the *Customizing PCB Editor* dialog appears.
2. Click on the **[Custom]** entry on the **Categories** part on the left side of the dialog. Then click on the **New** button and then the *Edit Command* dialog appears.
3. Click on the **Browse** button adjacent to the **Process** field and select the `ScriptingSystem:RunScript` process.
4. Define the `ProjectName` and `ProcName` parameters in the **Parameters** field of the dialog. For example; `ProjectName = C:\Program Files\Altium Designer 6\Examples\Scripts\Delphiscript Scripts\Dxp\DXPVersion.PrjScr | ProcName = DXPVersionUnit.pas > DXPVersion` in the **Parameters:** field.




5. Name this new command in the **Caption** field of this dialog and if desired, assign the text in the **Description** field and a new icon in the **Bitmap File** field in the *Edit Command* dialog. Click the **OK** button to close this dialog.
6. The new commands appear in the **[Custom]** category of the **Categories** list. Click on the **[Custom]** entry from the **Categories** list. The new command appears in the **Commands** list of this dialog.
7. Drag and drop the new command from the *Customizing PCB Editor* dialog onto the PCB menu. The command appears on the PCB menu (if you have not assigned a bitmap, then the Caption of this command will appear instead). Click on this command to run the DXP Version Script.

 For an introduction to scripting, refer to the [Getting Started with Scripting](#) tutorial

 For more general information on resources in Altium Designer, refer to the [Customizing the Altium Designer resources](#) tutorial

Debugging a Script

There are various tools in the scripting system to help you debug your scripts including using break points in your script, using the *Watches List* panel to monitor the variable values, using the bookmarks in the script to jump around more efficiently, and using step in or step out facilities to step through the scripts.

When a script contains an error, the script is stopped by the debugger. You can stop the script using the **Stop** button  on the menu and make changes to the script. Save changes and re-start the script.

To check the values of the variables in your scripts, you can utilize the following features in the scripting system:

- Modal dialogs (such as ShowMessage and MsgBox functions)

Building Script Projects

- Messages panel
- Breakpoints
- Expression Evaluation tooltips
- Symbol Insight tool tips
- Evaluate dialog



Consult the guide, [A Tour of the Scripting System](#) to learn more about debugging scripts and script panels.

Common Debugging Dialogs

The `ShowMessage`, `ShowInfo`, `ShowWarning` and `ShowError` procedures from DXP Run Time Library (part of Altium Designer application and exposed through the Scripting System) can be used as simple tools for displaying data values or to track the sequence of events of your script in any language set you use.

The VBScript in Altium Designer has built in functions that can be used in your VBScript script projects, one of the common functions is the `MsgBox` function.

ShowMessage Example

```
Procedure HelloWorld;
Begin
    ShowMessage('Hello world!');
End;
```

The Messages panel

The *Messages* panel is a very useful debugging tool for scripting, this panel is a workspace manager object. You can use the *Messages* panel as a debugging dialog that shows states of variables and properties from your script. An example of using the *Message* Panel in your script to capture data values or states is shown below.

Example

```
Var
    WSM : IWorkSpace;
Begin    WSM := GetWorkSpace;
    If WSM = Nil Then Exit;
    // Clear out messages and initialize the Message panel
    WSM.DM_ClearMessages;
    WSM.DM_MessageViewBeginUpdate;
    WSM.DM_ShowMessageView;

    // Display states or values of data with DM_AddMessage procedures
    WSM.DM_AddMessage('MessageClass 1', 'MessageText 1',
        'DXP Message 1', '1', '', 3, False);
```

```
WSM.DM_AddMessage('MessageClass 2', 'MessageText 2',
                  'DXP Message 2', '2', '', '', 3, False);
WSM.DM_MessageViewEndUpdate;
WSM.DM_ShowMessageView;
End;
```

Trapping Errors in your DelphiScript Scripts

The use of `Try Finally End` and `Try Except End` blocks are useful to catch exceptions in your DelphiScript scripts. The major limitation of a `Try Except End` block is that you cannot define custom exceptions but only use a text string for the `Raise` statement.

Example

```
Try
  X := Y/Z;
Except
  Raise('A divide by zero error!');
End;
```

The `Raise` keyword is related to the `Try` keyword. The `Raise` keyword can be used without parameters to re-raise the last exception. It can also be used with a string parameter to raise an exception using a specific message. An example is

```
Raise(Format('Invalid Value Entered : %d', [Height]));
```

Note, the `On` keyword is not supported in DelphiScript, and consequently you cannot use `Exception` objects in scripts unlike in Borland Delphi programming projects.

Trapping Errors in your VB scripts

The use of `On Error` statement can be used in your VB Scripts. Consult the Scripting documentation from the MSDN website (www.msdn.com) for more information on trapping errors.

Using the Code Completion features

The Scripting System supports the Code Completion facility. Server scripting panels include the Code Explorer, etc. The Code Explorer panel which displays the variables, methods and objects is used by the script in the tree-like panel.

Code Completion

The Code completion feature utilizes a script pop up window which shows available methods and objects once you have entered the **period** character (the dot/fullstop character) after the variable name in the script. You can also use the **CTRL + SPACE** shortcut keys to bring up the Code Completion options on demand.

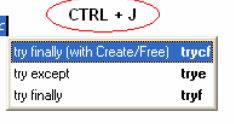
```
Var
  Board : IPCB_Board;
Begin
  If PCBServer.|
End;
```



The screenshot shows a code editor with a script. The line `If PCBServer.|` is highlighted. A red circle around the period character is labeled **CTRL + SPACE**. A pop-up window displays a list of completion options: `function GetCurrentComponent(const Board : IPCB_Board) : WideString;`, `function GetCurrentPCBBoard : IPCB_Board;`, `function GetPCBBoardByPath(APath : WideString) : IPCB_Board;`, and `property InteractiveRoutingOptions : IPCB_InteractiveRoutingOptions;`.

Script Templates

You can use the script template feature to automatically generate a complete statement for you when you press the **CTRL + J** shortcut keys after you have typed the first few letters of the statement.

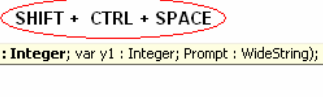


The screenshot shows a code editor with a script. The line `try` is highlighted. A red circle around the text `try` is labeled **CTRL + J**. A pop-up window displays a list of completion options: `try finally (with Create/Free) trycf`, `try except trye`, and `try finally tryf`.

Script Parameters

You can use the Script Parameters feature to inspect which parameters are used for a particular function/procedure/method of an object interface. Use the **SHIFT + CTRL + SPACE** keys to display the parameters pop up message.

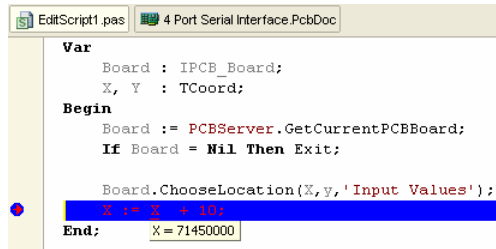
```
Var
  Board : IPCB_Board;
Begin
  Board := PCBServer.GetCurrentPCBBoard;
  If Board = Nil Then Exit;
  Board.ChooseLocation(
End;
function ChooseLocation(var x1 : Integer; var y1 : Integer; Prompt : WideString);
```



The screenshot shows a code editor with a script. The line `Board.ChooseLocation(` is highlighted. A red circle around the text `ChooseLocation` is labeled **SHIFT + CTRL + SPACE**. A pop-up window displays the signature of the `ChooseLocation` function: `function ChooseLocation(var x1 : Integer; var y1 : Integer; Prompt : WideString);`.

Expression Evaluation ToolTip

The Expression Evaluation ToolTip feature displays data values for the variable that the cursor is hovering over.



The screenshot shows a script editor window with two tabs: 'EditScript1.pas' and '4 Port Serial Interface.PcbDoc'. The script content is as follows:

```

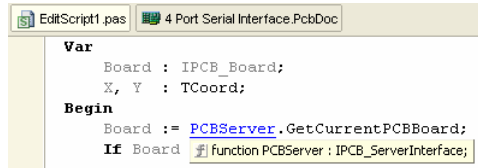
Var
  Board : IPCB_Board;
  X, Y : TCoord;
Begin
  Board := PCBServer.GetCurrentPCBBoard;
  If Board = Nil Then Exit;

  Board.ChooseLocation(X, Y, 'Input Values');
  X := X + 10;
End;      X = 71450000
  
```

The line `X := X + 10;` is highlighted in blue. A red dot is positioned at the end of this line, and a tooltip box displays the value `X = 71450000`.


Symbol Insight ToolTip


The Symbol Insight ToolTip feature shows the variable type when the mouse cursor is hovering over the variable. This information can also be accessed by pressing the **CTRL** key.



The screenshot shows the same script editor window as above. The script content is:

```

Var
  Board : IPCB_Board;
  X, Y : TCoord;
Begin
  Board := PCBServer.GetCurrentPCBBoard;
  If Board  Function PCBServer : IPCB_ServerInterface;
  
```

The line `If Board  Function PCBServer : IPCB_ServerInterface;` is highlighted in blue. A tooltip box is visible over the variable `Board`, displaying its type: `IPCB_Board`.

 Consult the guide, [A Tour of the Scripting System](#) for more information on debugging scripts and script panels.

Using Borland Delphi Objects in your Scripts

The Scripting System handles two types of components for script forms: Visual and Non-visual components. The visual components (Button, Edit and Memo components for example) are the ones you use to build the user interface. The non-visual components are used for different tasks such as Timer, OpenFileDialog and MainMenu components. You use the Timer non-visual component to activate specific code at scheduled intervals which is not visible to the user.

Components are also called controls. All Script Forms have one or more components. Any combination of components can be placed on a Script Form and while your script is running, a user can interact with any component on a form. It is your task to decide what happens when a user clicks a button or changes a text in an Edit box. Both visual and non-visual components appear at design time but non visual components are not visible at runtime. These components come from the Borland Run Time Library in Altium Designer.

Dropping Components onto a Script Form

The Scripting system supplies a number of components for your scripts. You can find all of the components you can place on a Script Form on the *Tool* palette. To place a component on a form, locate its icon on the *Tool Palette* panel and double-click it. This action places a component on the active form. When you drop a component onto a form, the Scripting system automatically generates the code necessary to use the component and updates the script form. You do not need to create or destroy these objects, the script form does them for you automatically.

Set the properties, add code in the event handlers and use methods as necessary manipulate your component on the Script Form. You can also use the Object Inspector to manipulate the properties of a component on the Script Form.

Creating Components from the Script Form

You can directly create and destroy components in a script – normally you don't need to pass in the handle of the form because the Script Form takes care of it automatically for you, so you normally pass a Nil parameter to the Constructor of a component.

Example of creating a TOpenDialog component

```
Function LoadAFile : String;
Var
    OpenDialog : TOpenDialog;
begin
    Result := '';
    OpenDialog := TOpenDialog.Create(nil);
    OpenDialog.InitialDir := 'C:\';
    // Display the OpenFileDialog component
    OpenDialog.Execute;
    // Obtain the file name of the selected file.
    Result := OpenDialog.FileName;
```

```

    OpenDialog.Free;
End;

```

Note, you can also use the **CreateObject** function as provided by the DelphiScript language and this function can take in multiple parameters, for example

```

Procedure DemoCreateObjectFn;
Var
    OpenDialog : TOpenDialog;
Begin
    // the CreateObject function takes in two parameters -
    // the TOpenDialog class and a Nil parameter
    OpenDialog := CreateObject(TOpenDialog, nil);
    OpenDialog.Execute;
End;

```

Instantiating Borland Delphi Classes

For other Borland Delphi objects such as the `TStringList` object, you can use some of the classes from the Borland Delphi Run Time Library to create them during the duration of the script. Normally in scripts, you can use these classes to create objects and store information in these objects then free them when they are no longer in use. It is your responsibility as a coder to create and free objects where appropriate.

```

Procedure Demo_TStringList_Class;
Var
    AList      : TStringList;
Begin
    Try
        AList := TStringList.Create;
        AList.Add('Line 1');
        AList.Add('Line 2');
        AList.SaveToFile(FileName);
    Finally
        AList.Free;
    End;
End;

```



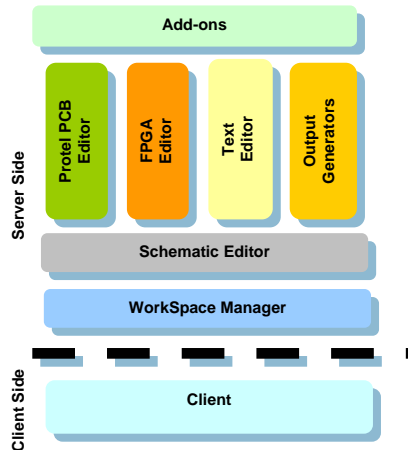
You can use a subset of the Borland Delphi classes in your scripts. Consult the [Helper Functions and Objects section in the System Reference](#) document for details on the use of Borland Delphi classes and functions in your scripts.

Using DXP Object Models in your scripts

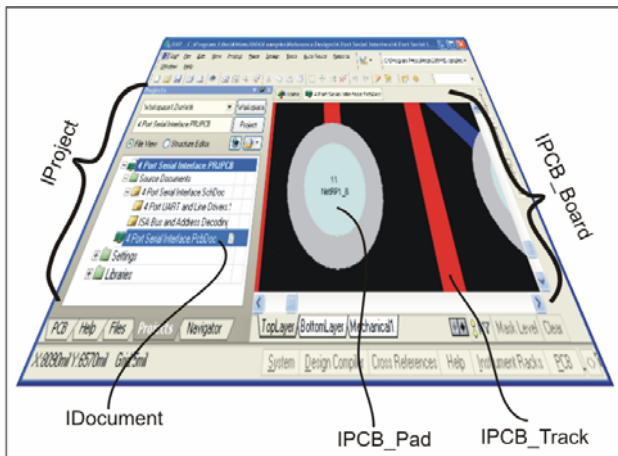
Altium Designer is composed of a single Client executable along with plugged in servers. The Client deals with actions generated by the user of the Altium Designer application. The servers provide specialized functionality depending upon the task requested by the user.

The scripting system has access to the DXP Object Models from the DXP Run Time Library as well as a subset of Borland Delphi Run Time Library. You need not worry about where the functions or objects come from – they are readily accessible for use in scripts. The DXP Object model consists of PCB Object Model, Schematic Object Model and the WorkSpace Manager Object Model etc.

From the diagram below, a Design project is represented by the `IProject` interface as part of the WorkSpace Manager Object model. The PCB document is represented by the `IPCB_Board` interface from the PCB Object Model.



The Client – Server architecture of DXP



In this tutorial, the Workspace Manager Object Model will be used to retrieve the connectivity information of the schematic project to generate a netlist. The PCB Object model is used to fetch PCB objects from a PCB document.



Consult the Scripting Resources accessible through the *Knowledge Center* panel in Altium Designer. Navigate to the appropriate API reference document via **Configuring the System » Scripting in Altium Designer » Altium Designer RTL Reference.**



Refer to the script examples, mostly in DelphiScript, in the `\Altium Designer 6\Examples\Scripts\` folder which are available for experimentation.

Building a Netlister Project

The aim of this Netlister is to generate a Protel netlist (either Version1 or Version 2 formats) for the Project containing schematics. A flat netlist of a schematic project is separated into two sections – component designators and the information associated with each component, and net names and the information associated with each net name along with pin connections (pins of a component).

From the WorkSpace Manager Object Model, there are interfaces that represent the project and its constituents – the documents, the components and its pins, and the nets. The WorkSpace Manager is a system extensions server coupled tightly with the Client module and deals with projects and their associated documents. This server 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 you the `IWorkspace` interface.

We need to concern ourselves with the `IWorkspace`, `IProject`, `IDocument`, `IComponent`, `IPin` and `INet` interfaces. You may notice that some of the interfaces, especially the design object interfaces correspond directly to Schematic Object interfaces. For example, `IPin` and `INet` correspond directly to `ISch_Pin` and `ISch_Net` interfaces respectively. It is because the logical documents in a project are schematic documents with connectivity information. In fact, we can use the Schematic Object model instead, but the Workspace Manager provides us the functionality to compile a project, extract documents of a project as well as retrieving data from schematic objects.

Main parts of a Netlister Script

The main parts of 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
- `WriteComponent` and its `WriteComponent_Version1` and `WriteComponent_Version2` routines
- `WriteNet` and its `WriteNet_Version1` and `WriteNet_Version2` routines
- `ConvertElectricToString` which converts a pin's electrical property to a string
- `GenerateNetlist` procedure

The Functionality of a Netlister

The netlister is based on the `ScripterProtelNetlist.pas` script from the `\Altium Designer 6\Examples\Scripts\DelphiScript Scripts\WSM\Protel Netlister\` folder. We will go through the netlister script step by step.

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

The two procedures in the Select Item to Run dialog

```
Procedure GenerateProtelV1FormatNetlist;
Var
```

Building Script Projects

```
    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;
```

2. The `GenerateNetList` procedure gets the workspace interface so that the project interface can be extracted for the current project. 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 this 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;
```

3. 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 Borland Delphi class part of the **Classes** unit of the Borland Delphi Run Time Library which is available to use in scripts.
4. A netlist is broken up into two sections so two procedures are required to write component data and net data separately. For nets, at least two nodes in a net will be written to a netlist, anything less than two nodes is 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 Protel Version 1 format. Remember component and net data are stored in the `NetList` object which is a `TStringList` type.
5. The `Generate` procedure obtains the output path of a project for the netlist file to go in. Then with all the schematic documents in a project, each document is checked for nets and components and they are then extracted to the `Netlist` object.
6. When a netlist is generated, it is composed of two sections; the component information section and the net information section.

Components Section

In this section, for each component found from the Project, it is checked if it is an actual component and then the physical designator, footprint and part type values are extracted.

```
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

In this section, if a net has two pins or more, then the NetName and the designators are extracted.

```
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(')');
```

7. Save the script and then execute the **RunScript** command from **DXP** menu. The *Select Item to Run* dialog appears. Choose either `GenerateProtelV1FormatNetlist` or `GenerateProtelV2FormatNetlist` procedure and a netlist will be generated and opened in Altium Designer automatically.



You can open the `Scripter_ProtelNetlist.PrjScr` project and execute the `ScripterProtelNetlist.pas` script from the `\Examples\Scripts\DelphiScript Scripts\WSM\Protel Netlister\` folder.

Building the Board Outline Copier Project

The aim of this Board Outline Copier is to copy an existing board outline from the PCB document to a different layer in the same document. Using the PCB Object model and its PCB interfaces from the PCB API, we can extract objects of a board outline and copy these objects onto a specified layer.

We will be using a Script Form so that user can input the width of the board outline and select the layer from a drop down menu.

Let's start dissecting the board outline copier script but this time in VBScript! The Board Outline Copier example can be found in `\Examples\Scripts\VB Scripts`, `\Examples\Scripts\JScript Scripts` and `\Examples\Scripts\DelphiScript Scripts\PCB folders`.

Main parts of a Board Outline Copier script

The main parts of the VB script script are

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

The Functionality of a Board Outline Copier script

Since we are using a script form, we need to have event handlers to capture the mouse clicks of individual controls such as the **Cancel** and **OK** buttons. The OK Click event handler obtains the Width of the outline in Internal coordinate values from the `StringToCoordUnit` function and the Layer enumerated value from the `String2Layer` function.

Both functions come from the Altium Designer Run Time Library which is available to use in scripts. Load the `CopyBoardOutlinePRJ.PRJSCR` project and check out the `CopyBoardOutlineForm` Script form.

IPCB_Board Interface

A board outline is obtained from the `IPCB_Board` interface via the `PCBServer` function and it needs to be initialized first before proceeding with copying and creating a new board outline.

Board Outline has Arc and Track segments

The board outline is represented by the `IPCB_BoardOutline` interface and this interface is inherited from the `IPCB_Group` interface. A `IPCB_Group` interface represents a group object that can store child objects. An example of `IPCB_Group` interface is a polygon or a board outline because they can store arcs and tracks.

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.

Each segment of this board outline needs to be checked for tracks and arcs with the `If PCB_Board.BoardOutline.Segments(I).Kind = ePolySegmentLine Then` statement.

For each segment found and depending on the segment kind, a new track or arc is created.

PCBObjectFactory and AddPCBObject Methods

Creating a PCB object employs the `PCBObjectFactory` method from the `IPCB_ServerInterface` interface. In this case, the `PCBServer.PCBObjectFactory(eArcObject, eNoDimension, eCreate_Default)` statement creates a new Arc object. A new copy of an arc object is added onto a specified layer of the PCB document with the `PCB_Board.AddPCBObject(NewObject)` statement.

PreProcess and PostProcess methods

For each PCB object creation, you need to apply the `PreProcess` method from the `IPCB_ServerInterface` object interface before creating a PCB object, and then after creating this PCB object, you need to apply the `PostProcess` method from the `IPCB_ServerInterface` interface. Below is a code snippet with `PreProcess` and `PostProcess` statements.

```
PCBServer.PreProcess
//create a PCB object
PCBServer.PostProcess
```

The `PreProcess` and `PostProcess` methods keep the Undo/Redo system up to date and synchronized in the PCB editor.

Setting the new PCB Layer

When objects are added to a layer that has not been displayed in the PCB document, we need to force this layer to be visible. This `PCB_Board.LayerIsDisplayed(ALayer) = True` statement does this job.

Refreshing the PCB Document

Finally the PCB document is refreshed with the new board outline on a new layer with the `PCB:Zoom` command and using the `Action = Redraw` parameters.



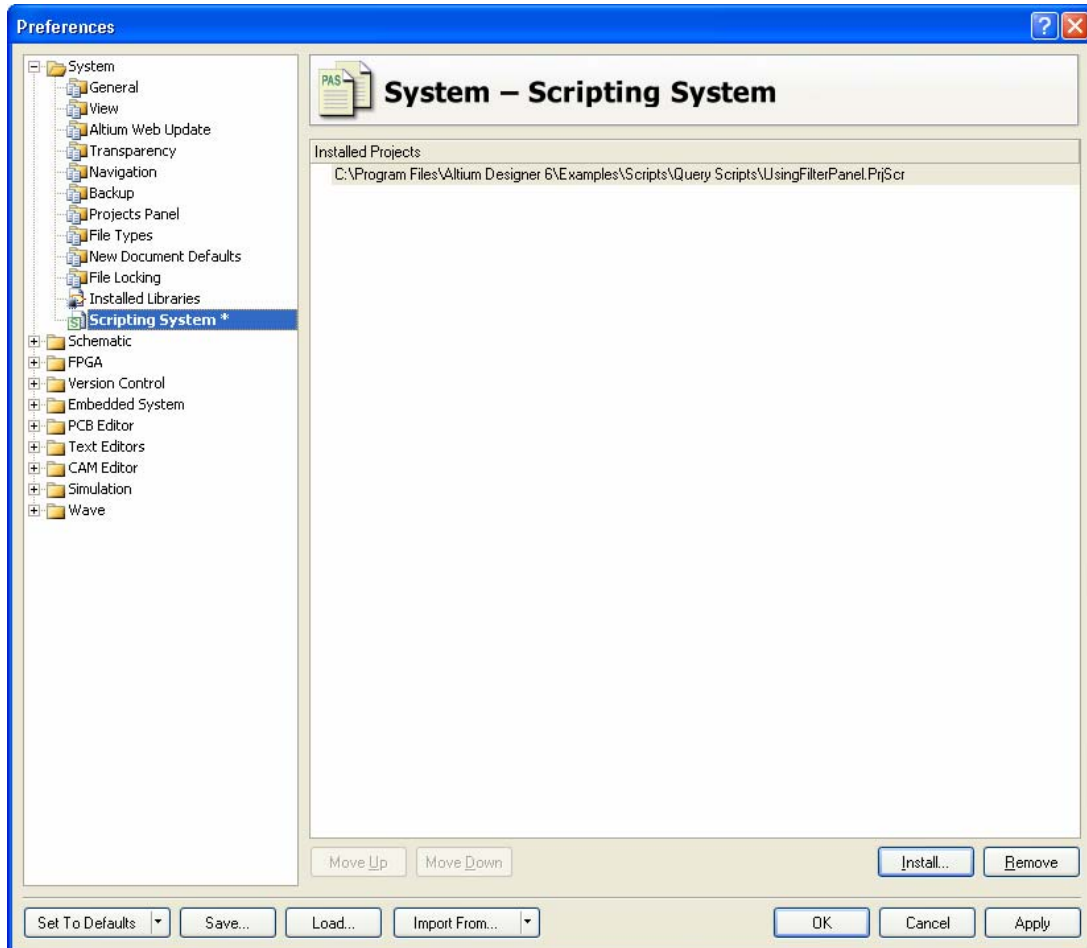
You can open the `CopyBoardOutlinePRJ.PRJSCR` project and execute the `CopyBoardOutlineForm.vbs` script from the `\Examples\Scripts\VB Scripts\` folder.

Using Query Expressions in Scripts

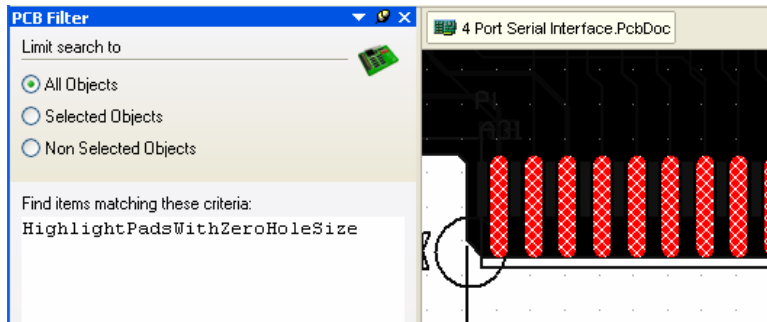
The *Filter* panel in Schematic and PCB has the ability to execute scripts that have query expressions to filter and select certain objects.

To be able to execute query scripts, these script projects must be installed in the **Installed Projects** list, so that they are made global to Altium Designer.

Go to the **DXP » Preferences** menu and when the Preferences dialog appears. Click on the **System** folder and drill down to **Scripting System** folder on the left pane of the *Preferences* dialog to install scripts in the **Installed Projects** list on the right side of the dialog.



A query script must have a function name that returns a boolean value. Inside the function block contains the query expressions. The function name is entered in the PCB *Filter* panel's expression window as shown below.



An example will be demonstrated how a query expression incorporated in a script can be executed in the query box on the *PCB Filter* panel. A simple query expression that looks for components with pads that have zero hole sizes is presented below. This script can only be executed on a PCB document.

```
Function HighlightPadsWithZeroHoleSize : Boolean;
Begin
    Result := False;
    // Check if PCB document exists, if not, exit.
    If UpperCase(Client.CurrentView.OwnerDocument.Kind) <>
'PCB' Then Exit;
    Result := IsComponentPad and (Holesize = 0);
End;
```

Query Expression scripts need to be installed in the **Installed Projects** list in the Scripting System page in the **Preferences** dialog.



Refer to the Query Expression Script examples which can be found in \Altium Designer 6\Examples\Scripts\Query Scripts\ folder.



Consult the [Query Language Reference](#) and [Introduction to the Query Language](#) documents to obtain more information on PCB and Schematic query expressions.

Conclusion

You have learnt how to create a script project and investigated three scripts that use Workspace Manager Object Interfaces, PCB Object Interfaces and Query Expressions.



Consult the Scripting Resources accessible through the *Knowledge Center* panel in Altium Designer. Navigate to the Scripting resources via **Configuring the System » Scripting in Altium Designer**.



Explore various script examples in the \Altium Designer 6\Examples\Scripts folder.

Revision History

Date	Version No.	Revision
30-Nov-2004	1.0	New product release
4-Apr-2005	1.1	Updated for Altium Designer
15-Dec-2005	1.2	Updated for Altium Designer 6
18-Jan-2006	1.3	Minor updates.
2-Mar-2006	1.4	Fixed the wrong title on page 11.
29-Nov-2007	1.5	Updated for Altium Designer 6.8

Software, hardware, documentation and related materials:

Copyright © 2007 Altium Limited.

All rights reserved. You are permitted to print this document provided that (1) the use of such is for personal use only and will not be copied or posted on any network computer or broadcast in any media, and (2) no modifications of the document is made. Unauthorized duplication, in whole or part, of this document by any means, mechanical or electronic, including translation into another language, except for brief excerpts in published reviews, is prohibited without the express written permission of Altium Limited. Unauthorized duplication of this work may also be prohibited by local statute. Violators may be subject to both criminal and civil penalties, including fines and/or imprisonment. Altium, Altium Designer, Board Insight, Design Explorer, DXP, LiveDesign, NanoBoard, NanoTalk, P-CAD, SimCode, Situs, TASKING, and Topological Autorouting and their respective logos are trademarks or registered trademarks of Altium Limited or its subsidiaries. All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same are claimed.