Script API Objects

Modified by Admin on Sep 13, 2017

Parent page: Scripting

One of the more powerful features of the scripting system is accessibility to Altium Designer design objects through its Application Programming Interface (API). This allows scripts to read and modify design object properties in an open design document.

The object models are accessed in Altium Designer through the application of Embarcadero Delphi's Object Interfaces technology, which is used to implement Object Interfaces that are exposed to scripts. In essence, Object Interfaces reference existing objects in Altium Designer — so design objects in a Schematic document are updated using the Schematic Interface, those in a PCB document using the PCB Interface, and so on.

The scripting system itself is written in Embarcadero Delphi and offers various Delphi Objects along with a subset of the Delphi Visual Component Library (as available through the *Tool Palette* panel components).

> The same objects, functions and components are available regardless of the scripting language - DelphiScript, VBScript, JavaScript etc.

The Altium Designer system is composed of a single Client executable along with plugged in server extensions. The client module is part of the DXP software technology platform, and deals with the actions generated by an Altium Designer user.

Servers extensions provide specialized functionality depending upon the domain task requested by a user. For example, the Schematic and PCB servers are two main document editor servers used in the design process, where each has its own document types (design and library documents).

For both the Client and range of Servers in Altium Designer, Object interfaces are available for referencing their objects models. An Object Interface has methods and many also have properties:

- A *method* is a procedure or function that is invoked from an Object Interface — it represents something an object can do.
- A *property* of an object interface is an attribute that can be read or modified. A property can be implemented by its Get and Set methods, however some properties are read-only and can return values but not be set.

For example, the `IPCB_Component` interface from the PCB API object model has a `Height` property and associated methods supported by its Read and Write identifiers. A component's height property

can be accessed using the `GetState_Height` and `SetState_Height` methods or by directly referring to the property itself (`IPCB_Component.Height`). See the `CreateComponentOnPCB` script example (in the Example Scripts collection) for a working implemtentation.

The API source code is not shown in the scripting system, but you can see the methods and properties for an object interface using the context-sensitive Code Completion feature (**Ctrl+Space**) when writing a script.

See the Altium Designer API reference for detailed information on the Object Models, Interfaces, Methods and Properties available to the Scripting system.

DXP Object Interfaces

In Altium Designer scripting you use an Object Interface that references an existing object rather than creating instances of the objects. The Object Models and their supporting interfaces are functionally grouped as follows, where each Server (or the Client) API offers a specific hierarchy of scripting interfaces:

- Client Object Mode
- WorkSpace Manager Object Model
- Schematic Object Model
- PCB Object Model
- Integrated Library Object Model
- Nexus Object Model

For a typical PCB design document open in a Project...

The project and its corresponding documents are managed by the Workspace Manager API and its Object Models, where:

- An open project is represented by the `IProject` object interface.
- The documents from this project are represented by the `IDocument` interfaces.

The PCB documents and PCB design objects are managed by the PCB Editor API and its PCB Object Model, where:

- The PCB document open is represented by its `IPCB_Board` interface
- The design objects, for example, the pad object and the track object are represented by `IPCB_Pad` and `IPCB_Track` interfaces.

Interface names as a convention have an `I` character prefixed to their name, for example `IPCB_Board` represents an interface for an existing PCB document.

PCB Objects

The PCB Object Model enables scripts to have access to PCB objects and PCB documents. The gateway to PCB objects on a PCB document or in a PCB Editor Server is the `PCBServer` function. When invoked in a script, the `PCBServer` function returns the `IPCB_ServerInterface` interface,

which represents the PCB Editor server.

By way of example, the `GetCurrentPCBBoard` method from the `PCBServer` function is normally invoked in scripts to obtain the `IPCB_Board` interface. Moving down the hierarchy, PCB pad objects are encapsulated by the `IPCB_Pad` interface.

In the below DelphiScript example, the `GetCurrentPCBBoard` method is invoked to return the `IPCB_Board` interface — which represents the current PCB document in Altium Designer. The script uses an Iterator to look for specific PCB design objects on a PCB document. The Iterator can be configured to look for specified objects within a specified region or on a specified layer, but in this case, only pad objects are searched on the PCB Document layers.

Note that a PCB document that contains a number of pad objects needs to be open before the script is executed.

```
Procedure PadCount;
Var
    Board     : IPCB_Board;
    Pad       : IPCB_Primitive;
    Iterator  : IPCB_BoardIterator;
    PadNumber : Integer;
Begin
    PadNumber       := 0;

    // Retrieve the current board
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;

    // retrieve the iterator
    Iterator        := Board.BoardIterator_Create;
    Iterator.AddFilter_ObjectSet(MkSet(ePadObject));
    Iterator.AddFilter_LayerSet(AllLayers);
    Iterator.AddFilter_Method(eProcessAll);

    // Search and count pads
    Pad := Iterator.FirstPCBObject;
    While (Pad <> Nil) Do
    Begin
        Inc(PadNumber);
        Pad := Iterator.NextPCBObject;
    End;
    Board.BoardIterator_Destroy(Iterator);

    // Display the count result on a dialog.
    ShowMessage('Pad Count = ' + IntToStr(PadNumber));
End;
```

> The above script (`CountPads.pas`) can be found in the `\Delphiscript Scripts\PCB\PCB Iterators` folder of the Script Examples collection.

See the Technical Reference - PCB API entry and its subpages for detailed information on the PCB

Object Interfaces.

Unique Object Interfaces

In scripts, all objects of the same type have the same Object Interfaces, so for example, all PCB track objects in a open PCB document are represented by `IPCB_Track` interfaces.

To distinguish between the multiple object interfaces (where each represents a PCB track) an object's unique memory address, or handle, can be used. In this case, all PCB object addresses are represented by the `TPCBObjectHandle` type (a numeric pointer) and the PCB object interface offers the I_ObjectAddress method (`IPCB_Track.I_ObjectAddress : Pointer`)

Similarly, all Schematic object addresses are represented by the `TSchObjectHandle` type and the Schematic interfaces have the `Function I_ObjectAddress : TSCHObjectHandle` method.

Schematic Objects

The Schematic Object Model enables scripts to have access to Schematic objects and Schematic documents. The gateway to Schematic objects in a document or the Schematic Editor Server is the `SchServer` function, which returns the `ISch_ServerInterface` interface. This Object Interface obtains the Schematic editor server object, so you can then get the current schematic document, extract data from Schematic objects and invoke Schematic object's methods.

To examine the Schematic Object Interface used in a practical Altium Designer Delphi script, load the `UsingSpatialIterators` script from the `\Scripts\Delphiscript Scripts\SCH` folder of the Script Examples collection - this script is part of the `Sch_Scripts` project.

When executed, this script example uses a method that waits for the user to click on two points that define a boundary box on a schematic sheet. The spatial iterator, which is a search object, then looks for components inside the defined boundary on the sheet and reports their locations in terms of X,Y coordinates. Again, a schematic document needs to be open before running this script. The spatial iterator is encapsulated by the `ISch_Iterator` interface in the Schematic Object Model.

See the Technical Reference - Schematic API entry and its subpages for detailed information on the Schematic Object Interfaces.

Workspace Manager Objects

The WorkSpace Manager server deals with projects and their associated documents. This server provides project and multi-sheet design support, compiling, connectivity navigation tools and so on.

The Workspace manager is schematic-centric and also manages output generators such as netlisters. To retrieve the WorkSpace Manager interface, invoke the `GetWorkspace` function — `IWorkspace.GetWorkspace`.

To examine the Workspace Manager Object Interface used in a practical Altium Designer Delphi script, load the `UsingMessagePanel` script from the `\Scripts\Delphiscript Scripts\WSM` folder of the Script Examples collection - this script is part of the `WSM_Scripts` project.

The `UsingMessagePanel` example shows how to insert strings in the *Messages* panel in Altium Designer. The *Messages* panel is encapsulated by the IMessagesManager interface from the Workspace Manager object model (`IWorkspace.DM_MessagesManager`).

See the Technical Reference - Workspace Manager API entry and its subpages for detailed information on the Workspace Manager Object Interfaces.

Client Objects

The Client executable module within the Altium Designer application is responsible for dispatching commands to the appropriate server, and handles the user interfaces and system level functions. To retrieve the Client module interface, invoke the `Client` function. The `Client` function retrieves the `IClient` interface which encapsulates the Client executable module.

To examine Client Object Interface used in a practical Altium Designer Delphi script, load the `OpenADoc` script from the `\Scripts\Delphiscript Scripts\DXP` folder of the Script Examples collection - this script is part of the `DXP_Scripts` project.

In that example the `OpenDocumentShowOrHide` method of the `IClient` interface is used to open (but not show) a nominated text file in Altium Designer. Here, the `OpenDocumentShowOrHide` method returns an `IServerDocument` interface, which is used as a parameter for the `ShowDocumentDontFocus` method that displays the document in Altium Designer.

Before executing the script ensure that a matching text file is available (eg: C:\Report.txt) and the script is edited accordingly — `FileName := 'C:\Report.Txt';` at code line 15.

See the Technical Reference - System API entry and its subpages for detailed information on the Client Object Interfaces.

Executing Processes from a Script

Altium Designer's Client module interprets commands in terms of server processes and then delegates these processes to the appropriate servers. A command represents an action performed and is supported by a process string via a packaged process launcher.

A process is implemented as a `server name:server process` string and are stored in a command launcher table maintained by the server. Each time a process is excecuted via the Graphical User Interface (GUI) in Altium Designer, it consults the appropriate server's command table to fetch the process string, and then sends this string to the correct server to determine which process to execute. The functionality of a server is exposed by that server's processes.

For example, when the Schematic menu is used to place a wire, the Client interprets this action as a '`SCH: PlaceWire`' command and delegates the command to the Schematic Editor server. The Schematic server responds by executing the command. In a nutshell, a server is supported by its set of processes and these processes act as a link between the GUI and this server.

To see the processes available in Altium Designer open the *Run Process* dialog (**DXP » Run Process**) and open the *Process Browser* list dialog from the **Browse** button.

The simple example shown below places a new Schematic Port on an open Schematic document. To try the example, open the `Process_Scripts` project from `\Scripts\DelphiScript Scripts\Processes` folder of the Script Examples collection, and then open the `PlaceSchPort.pas` script.

```
Begin
```

```
   ResetParameters;
   AddIntegerParameter('Location.X', 20000000   );
   AddIntegerParameter('Location.Y', 20000000   );
   AddIntegerParameter('Style'     , 2          );
   AddIntegerParameter('IOType'    , 3          );
   AddIntegerParameter('Alignment' , 0          );
   AddIntegerParameter('Width'     , 10000000   );
   AddStringParameter ('Name'      , 'Test Port');
   AddIntegerParameter('AreaColor' , $FFFFFF    );
   AddIntegerParameter('TextColor' , $000000    );
   RunProcess         ('Sch:PlacePort'          );
End.
```

The above `PlaceSchPort` script uses Schematic Server processes. The `ResetParameters` method is invoked first to clear Altium Designer's process buffer. The server process parameters are set up using the `AddIntegerParameter` and `AddStringParameter` methods, and the server process is finally invoked with the `RunProcess` method.

See the Server Process Reference documentation for more information about Server Processes.

Delphi Objects in Scripts

The Scripting System handles two types of component objects for script forms: visual and non-visual components. The visual components (Button, Edit and Memo components for example) are used to build the user interface, whereas the non-visual components are used for a range behind the scenes tasks — such as the Timer, OpenDialog and MainMenu components. You would use the Timer non-visual component to activate specific code at scheduled intervals, and this will not be visible to the user. In Altium Designer these components come from the Delphi Run Time Library.

All Script Forms have one or more components, which are also called controls. Any combination of components can be placed on a Script Form, and while the script is running a user can interact with any of the form components. The script code determines 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.

Placing Form Components

Altium Designer's *Tool Palette* panel exposes all of the available components that can be placed on a Script Form. To place a component on the active form, double-click the component icon on the *Tool Palette* panel. When a component is placed on a form, the Scripting system automatically generates the code necessary to use the component and updates the script form. The objects do not need to be created or destroyed, as the script form does this automatically.

Set the properties, add code in the event handlers (see the **Event** tab in the *Object Inspector* panel) and use methods as necessary manipulate the component on the Script Form. You can also use the *Object Inspector* panel to simply manipulate the properties of a component on the Script Form.

See Writing Scripts for a guide to implementing components (controls) on a script form.

Creating form Components

Components can be directly created and destroyed in a script. Normally, there's no need to pass in the handle of the form because the Script Form takes care of it automatically, so a Nil parameter is passed to the Constructor of a component, as shown below.

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

Note that Delphi's `CreateObject` function can also be used to create the dialog instance. This function takes multiple parameters, and in the case of above example, the `TOpenDialog` class and a `nil` parameter. The line highlighted in gray (above) would change to; `OpenDialog := CreateObject(TOpenDialog, nil);`

Instantiating Delphi Classes

For other Delphi objects such as the `TStringList` object, some of the Delphi run-time classes can be used to create them during the duration of the script. In scripts, these classes are normally used to create objects and store information in the objects then free them when they are no longer in use. The objects need to be created and freed where appropriate.

An example that demonstrates creating and using a `TStringList` class:

```
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;
```

See the 'Helper Functions and Objects' section in the System API Server Processes document for details on the available subset of Delphi classes and functions for scripts.