



Using the Altium Designer RTL

Summary

Guide

GU0117 (v1.3) Dec 28, 2006

This guide defines what the Altium Designer RTL is and what role it plays in Altium Designer and how it can be used in scripts and server projects.

The Altium Designer Run Time Library (RTL) is composed of Application Programming Interfaces (APIs), specialized classes and system routines. Each API has an Object Model and in turn the object model is a hierarchical system of object interfaces. These object interfaces represent actual objects in the Altium Designer application.

In the scripting system within Altium Designer, we only refer to the Object Models because we are dealing only with object interfaces that represent objects of the Altium Designer application and their properties/methods from scripts.

In the server development, we are dealing with classes, objects and object interfaces. Therefore we will refer to APIs (remember each API also includes the specific Object model).

When we are dealing directly with object interfaces, then we will talk about the object models.

Therefore an object model is a subset of the API. For example, the PCB Object model is part of the PCB API.

The major APIs from the Altium Designer RTL are; Client/Server API, Work space manager API, PCB API, Schematic API and Integrated Library API.

This document covers the following areas:

- The software technology platform
- Altium Designer RTL used in Scripting
- Altium Designer RTL used in Server Development
- Using APIs in Scripts and Server Projects
- Examples

DXP Platform

The Altium Designer's open architecture which is the DXP platform allows scripters and third party developers to write and integrate their applications into the environment, with full Application Programming Interface (API) access to the Schematic editor, PCB editor and other plug in editors.

Several major software technologies are used in Altium Designer; the client/server architecture technology, DXP Object Models, and object oriented technologies.

The Altium Designer system provides a hybrid interface model, which exposes the functionality of a dynamic linked library (DLL) module to both a user and the client executable system. The servers themselves are built using the DXP Run Time Library.

The Client executable is a standalone executable system that collaborates with loaded dynamic library linked servers within the Altium Designer system. The Client module controls the user interface and delegates tasks (sends commands) to appropriate servers, while servers (as dynamic linked library files) concentrate solely on providing specific functionality based on the commands invoked by the user in Altium Designer.

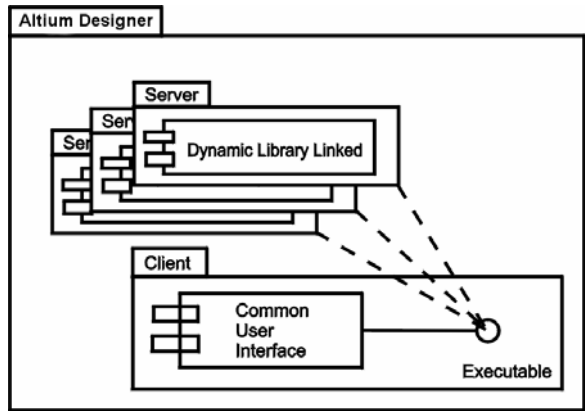


Figure 1: Client / Server – Exe / DLL models.

A server provides specialised services

A server provides its services in the Altium Designer application. The DXP platform interprets the tasks in terms of commands and then delegates these commands to the appropriate server.

The server responds by activating its required functionality on this currently focussed design document.

For example when a user is clicking on the Schematic menu to place a wire, the system interprets this action as a '**Sch:PlaceWire**' command and delegates the command to the Schematic Editor. The Schematic server responds by executing the command. The functionality of a server that is installed in the Altium Designer is exposed by that server's processes and its exposed functions.

For more information on server processes, please read the **Exposing the functions in your server to the DXP environment** section in the **GU0118 The Insiders Guide to Server Development** document.

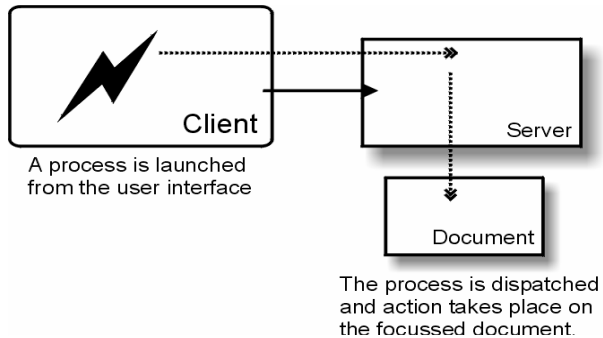


Figure 2: Server processes

Main servers in Altium Designer

The Work-Space Manager is a system extensions server coupled tightly with the client module part of the DXP Platform 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. The Work-space manager manages output generators such as netlisters.

The Schematic server and PCB server are two main document editors and they have their own document types (design and library documents).

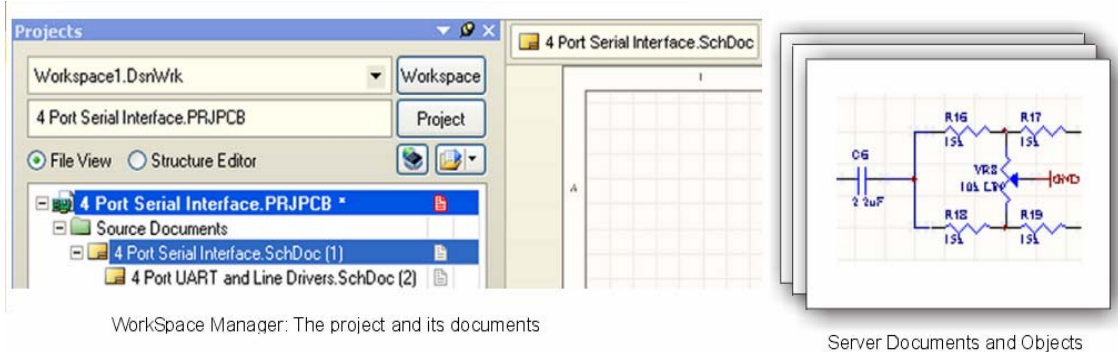


Figure 3 Projects and Design Documents in Altium Designer

Managing the locations of footprints or symbols from the library documents are done by the Integrated Library server.

There are two representations of a design document in Altium Designer: Projects and documents dealt by the Work-Space Manager or loaded server documents dealt by the servers (Figure 3).

How is Altium Designer RTL used in scripts?

The scripting engine in Altium Designer supports PCB, Schematic and Workspace Manager APIs enabling you to write scripts that act on PCB or Schematic documents or invoke one of the file management routines.

The Altium Designer RTL is automatically exposed in scripts, so you just code the object names and their method names with appropriate parameter values using one of the several supported scripting languages such as EnableBasic, Visual Basic, Javascript, TCL and as well as commonly used DelphiScript (which is very much like Borland Delphi).

DelphiScript example that looks for Pad objects on a PCB document

```
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;
```

VBScript example that creates a new via object on a PCB document

```
Sub ViaCreation
    Dim Board
    Dim Via

    Set Board = PCBServer.GetCurrentPCBBoard
    If Board is Nothing Then Exit Sub

    ' Create a Via object
    Via = PCBServer.PCBObjectFactory(eViaObject, eNoDimension,
eCreate_Default)
    Via.X = MilsToCoord(7500)
    Via.Y = MilsToCoord(7500)
    Via.Size = MilsToCoord(50)
    Via.HoleSize = MilsToCoord(20)
    Via.LowLayer = eTopLayer
    Via.HighLayer = eBottomLayer

    ' Put this via in the Board object
    Board.AddPCBObject(Via)
End Sub
```

See also



Refer to the **Scripting** help in Altium Designer by choosing the **Help » Contents** menu item and within the **Altium Designer Documentation Library** section, click on the **Inside the Altium Designer Environment » Scripting** and check out the following documents; **A Tour of the Scripting System**, **Getting Started with Scripting** and **Building Script Projects** as well as **Scripting References**.



Refer to the **Altium Designer RTL Online** help by choosing the **Help » Contents** menu item and within the **Altium Designer Documentation Library** section, click on the **Inside the Altium Designer Environment » Scripting and Server Development**.



There are many script examples in the `\Altium Designer 6\Examples\Scripts\` folder which demonstrate how scripts with different API functions work within the Altium Designer framework.

How is DXP Run Time Library used for Server Development?

The DXP Run Time Library is implemented using Borland Delphi 6™ tool kit. The Altium Designer RTL is made up of Delphi Compiled Units (files with a DCU extension). These Delphi Compiled Units are composed of different sets of APIs.

These DCUs contain compiled Delphi object information which gives you the ability to access to the object oriented structures (for example pad, track and text objects contained on a PCB document). These DCU files are Delphi compiler-version specific, and at the time of writing, DCUs are compiled with Borland Delphi 6. Thus you would need the matching Delphi version for your server development.

The units from Altium Designer RTL that you need to use in your server project are added to the `Uses` clause in one of the units from this server project. When you want to use a particular RTL function, you need to add the corresponding unit in the `Uses` clause of your server code project. For example, to use the `MessageRouter_SendCommandToModule` procedure, add `RT_API` unit in the `Uses` clause, or the compiler will report an Undeclared identifier error message.

The DCU files can be found in `\Developer Kit\RTL` subdirectory.

Server project example

`Uses`

```
    SysUtils, Windows, Dialogs, Rt_Util, Rt_Types, Rt_Param, Rt_Forms, Rt_Api,  
    Rt_ClientServerInterface, RT_PCB, RT_PCBProcs;
```

`Implementation`

```
Procedure DemoSimpleIterator;
```

`Var`

```
    Board      : IPCB_Board;  
    Pad        : IPCB_Primitive;  
    Iterator   : IPCB_BoardIterator;  
    PadNumber  : Integer;
```


`Begin`


```
    PadNumber      := 0;  
    // retrieve the current board's handle  
    Board          := PCBServer.GetCurrentPCBBoard;  
    If Board = Nil Then Exit;  
  
    // retrieve the iterator handle  
    Iterator       := Board.BoardIterator_Create;  
    Iterator.AddFilter_ObjectSet([ePadObject]);  
    Iterator.AddFilter_LayerSet(AllLayers);  
    Iterator.AddFilter_Method(eProcessAll);
```

```

// search and count pads
Pad := Iterator.FirstPCBObject;
While (Pad <> Nil) Do
Begin
    PadNumber := PadNumber + 1;
    Pad := Iterator.NextPCBObject;
End;
Board.BoardIterator_Destroy(Iterator);
// Display the count result on a dialog.
ShowMessage('Pad Count = ' + IntToStr(PadNumber));
End;
End.

```


 The DCUs can be found in \Developer Kit\RTL subdirectory and the source headers of DXP Run Time Library can be found in INT files in the \Developer Kit\RTL\RTL Interfaces folder.

 Check out the RTL Interfaces folder within the Developer Kit folder to check the object declarations/interfaces.

Using other languages for server development

Other versions of Borland Delphi can be used only if the DXP run time library has been translated into other versions. Other programming languages can be used, again only if the DXP run time library (RTL) has been implemented using different languages. At this time of writing, only the Borland Delphi 6™ programming language is supported.

See also

 Refer to the **Altium Designer RTL Online** help in Altium Designer by choosing the **Help » Contents** menu item and within the **Altium Designer Documentation Library** section, click on the **Inside the Altium Designer Environment » Scripting and Server Development**

Refer to the documents in the Altium Designer Developer Edition;

 GU0119 Getting started with developing Servers

 TU0127 Building your first server

 GU0118 The Insider's Guide to Sever Development

 TR0133 Altium Designer RTL Reference for Servers reference

Using the DXP Object Interfaces

The projects and the corresponding documents are managed by the Workspace Manager. A project open in Altium Designer is represented by the **IProject** object interface, and the documents from this project are represented by the **IDocument** interfaces.

The figure is an illustration of the relationship between objects in the Altium Designer application and the Object Interfaces supported by the Altium Designer RTL.

The PCB documents and PCB design objects are managed by the PCB Editor and its PCB API.

The PCB document open in Altium Designer is represented by its **IPCBoard** interface and the design objects for example the pad object and the track object are represented by **IPCBoard_Pad** and **IPCBoard_Track** interfaces.

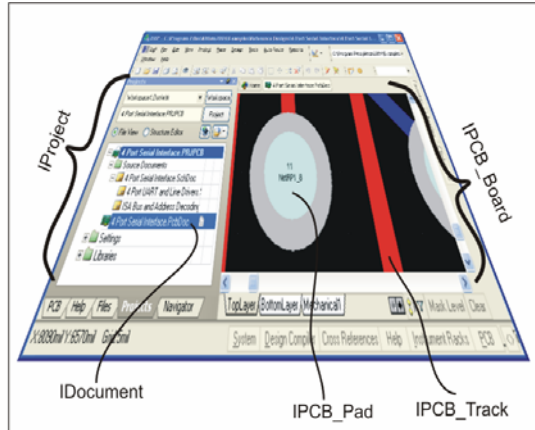


Figure 5: DXP Object Interfaces of Design objects

DXP Object Interfaces

Object Interfaces in Borland Delphi, are implementation independent declarations of functionality. To use interfaces in scripts and server projects, you need to have the associated Delphi classes and their methods implemented and instantiated first in Altium Designer. Thus you need to check if these interfaces are valid references to actual objects in memory before invoking the methods.

From a developer's perspective, there are high level object interfaces which encapsulate certain system objects in Altium Designer system such as **IProject**, **IWorkspace**, **IClient** and **IServerModule** interfaces which can be used to extract data for further processing by other plug-ins.

DXP Object Interfaces

- Client and Server interfaces – needed for dealing with Server Documents and Client objects.
- Work-Space Manager interfaces – needed for dealing with projects and documents.
- Schematic Interfaces – needed for dealing with schematic objects
- PCB interfaces – needed for dealing with PCB objects
- Integrated Library interfaces – needed for dealing with library links and building model editors.
- Other interfaces – such as the Output Generator and Nexar interfaces.

Client and Server Interfaces

The client module is represented by its `IClient` interface object, and you can have the ability to take a peek into the data structures through this `IClient` interface. The client maintains a list of loaded servers, opened server documents, loaded server processes and resources.

Servers are represented by its `IServerModule` interfaces. These interfaces are declared and partially implemented in the `RT_ClientServerInterfaces` unit.

Client function

The `Function Client : IClient;` returns the Client interface object within your server project. With this object interface, you can extract extra information about the client module and its associated servers.

Using the Client function

```
Var
    CurrentView : IserverDocumentView;
Begin
    CurrentView := Client.CurrentView;
    If CurrentView <> Nil Then
        OwnerDocument := Client.CurrentView.OwnerDocument;
    End;
```

IClient and its main composite interfaces


- `IClient` (the interface of the Client subsystem) and a few of its main composite interfaces
- `IServerModule` (Client has a handle on servers loaded in memory)
- `INotification` (Client can broadcast notification messages to servers)
- `ICommandLauncher` (deals with launching a server command)
- `IProcessControl` (determines the level of stacked processes for an opened document)
- `IGUIManager` (deals with the User interface of ALTIUM DESIGNER, the locations and state of panels)
- `IServerDocumentView` (deals with the current view as panels and documents in Altium Designer)
- `IOptionsManager` (deals with system wide Preferences dialog in Altium Designer)
- `IServerRecord` (information of server installation files in the `\Altium Designer 6\System` folder)
- `IServerPanelInfo` (panels information)
- `IOptionsManager` (Manage Option pages for each server)

IServerModule and its main composite interfaces

- `IServerModule` (deals with the shell of the server module)
- `IServerView` (represents a system panel that can have a view of the Altium Designer system)
- `IServerDocumentView` (represents a document view)


Using the Altium Designer RTL


- `IServerDocument` (represents a container of documents of the same type for a server)
- `IProcessControl` (determines the level of stacked processes for this focussed server document)
- `INotification` (a server can receive notifications from the Client system).
- `ICommandLauncher` (deals with launching server commands)

 For detailed information on `RT_ClientServerInterfaces` unit, refer to the [Altium Designer RTL Reference](#) document and the reference online help in Altium Designer's Help folder (`DXPRTLLRef.chm` file).

Report of Client and server object interfaces used in Altium Designer

There is a `DXPInfo` server example which generates a text file reporting the usage of `IClient` and `IServerModule`, `IServerDocument`, `IServerDocumentView`, and `IServerRecord` interfaces used in Altium Designer after invoking the **FetchClientServerInterfaces** process.

 The source code files for this add-on is in the `\Developer Kit\Examples\DXP\Server Information` folder. You can compile the `DXPInfo` project and drop the `dxpinfo.dll` and `dxpinfo.ins` file in the `\System` folder.

 For a tutorial of an addon development, refer to the **TU0127 Building your first server** document.

Workspace Manager interfaces

The Work-Space Manager is a system extensions server which is always running when Altium Designer is loaded in memory. The Workspace Manager server provides project functionality of grouping of files and provides a bridge between source documents (such as Schematic documents) and its corresponding primary implementation documents (such as PCB documents). This workspace manager provides you information on how a project is structured, and information on nets and its associated net aware objects of source and implementation documents.

The document interfaces in the Work Space Manager can refer to documents which may not be open in Altium Designer, whereas, the `IServerDocument` interfaces (from the `RT_ClientServerInterfaces` unit) only exist for loaded documents in Altium Designer.

The Work-Space Manager object derives some of its functionality from the `RT_Workspace` unit. To have access to data within the Work-space Manager, you need to have access to the `IWorkSpace` interface object which wraps the workspace manager.

This workspace manager provides you the ability to manipulate the contents of a design project in Altium Designer.

Main Workspace Manager Interfaces


- The `IDMObject` interface is a generic ancestor interface used for all other `WorkSpace` interfaces.
- The `IWorkSpace` interface is the top level interface and contains many interfaces within. For example the `IWorkSpace` interface has a `DM_OpenProject` function which returns a currently open or currently focussed `IProject` interface.
- The `IProject` interface represents the current project in Altium Designer.
- The `IPart` interface represents a part of a multi-part component. This component is represented by this `IComponent` interface.
- The `IDocument` interface represents a document in Altium Designer.
- The `IComponentMappings` interface is used for the Signal Integrity models mapping to Schematic components.
- The `IECO` interface represents the Engineering Change Order system in PCB and Schematic editors.
- The `INet` interface is a container storing Net aware objects (which are `INetItem` interfaces) that have the same net property. So there are `INet` interfaces representing nets on a document.
- The `INetItem` interface is the ancestor interface for the Cross, Pin, Port, Netlabel, Sheet entry and Power Object interfaces. These interface objects have a net property and thus these objects can be part of a net.

GetWorkspace function

The Function `GetWorkspace : IWorkspace;` returns the Work-Space Manager interface object within your server project. With this interface, you can extract extra information about a project and its associated documents and their design objects on them.

Using the `GetWorkspace` function in a DelphiScript

```
Var
    i          : Integer;
    Document  : IDocument;
    Project    : IProject;
Begin
    Project := GetWorkspace.DM_FocusedProject;
    If Project = Nil Then Exit;
    For i := 0 To Project.DM_LogicalDocumentCount - 1 Do
        Begin
            Document := Project.DM_LogicalDocuments(i);
            ShowMessage(Document.DM_DocumentKind);
        End;
    End;
End;
```

 Check out the `DXPInfo` server example in the `\Developer Kit\Examples\DXP\Server Information` and the **FetchWSMInterfaces** process when executed generates the details of the Workspace manager interfaces.

Compiling a project

A project needs to be compiled first so you can have access to the most current data which provides a snapshot of the latest status of a design project. There are two ways you can compile the project;

Compile with Project.DM_Compile example

```
Procedure CompileProject;  
Begin  
    Project := GetWorkspace.DM_FocusedProject;  
    If Project = Nil Then Exit;  
    Project.DM_Compile;  
    FileName := Project.DM_ProjectFullPath;  
End;
```

Compile example using the MessageRouter_SendCommandToModule call

```
GetMem(P, 4048);  
SetState_Parameter(P, 'Action', 'Compile');  
SetState_Parameter(P, 'ObjectKind', 'Project');  
MessageRouter_SendCommandToModule('WorkspaceManager:Compile', P, 4048, Nil);  
FreeMem(P);
```



For detailed information on Workspace Manager API, refer to the **TR0140 Workspace Manager API Reference** document.



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\DXP\` folder.

Nets of design documents

Schematic Design documents that have components and wires contain connectivity information which is captured in nets. A net is composed of connections and each connection is linked by a node as a pin. Thus nets are connected pins and a valid net has more than 1 pin.

To obtain connectivity information, we use the WorkSpace manager API to fetch net information. Basically you need to have a project open with valid schematics and then going through each schematic, the net count is obtained and for each net, the pin count is obtained and we have pins at our disposal. With pin interfaces, we can fetch pin designator, pin number and so on.

The design project needs to be compiled first, the documents (IDocument interfaces) are obtained and then the Nets (INet interfaces) are obtained. For each net, IPin interfaces are again obtained.

Fetch Nets example using the INet interface

```
//net information is stored in the NetList TStringList container
```

```
For i := 0 To Document.DM_NetCount - 1 Do  
    WriteNet(Document.DM_Nets(i));
```

```
Procedure WriteNet(Net : INet);
```

```
Var
```

```
    I      : Integer;
```

```
    Pin   : IPin;
```

```
    PinDsgn : String;
```

```
    PinNo  : String;
```

```
Begin
```

```
    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;
```

```
End;
```

 PCB Design documents also have net information but these nets are captured in IPCB_Net interfaces and please refer to the **TR0138 PCB API Reference** document for more information.

 Check out the Connectivity script example in the \Examples\Scripts\DelphiScript Scripts\Sch\Connectivity folder.

 Check out the script examples in the \Examples\Scripts\DelphiScript Scripts\WSM\ folder.

Schematic Editor interfaces

The Schematic API allows a programmer to fetch or modify Schematic objects and their attributes from a Schematic document. The objects shown on a document are stored in its corresponding design database.

SchServer function

When you need to deal with Schematic design objects in Altium Designer, the starting point is to invoke the `SchServer` function and with the `ISch_ServerInterface` interface, you can extract the all other derived schematic interfaces that are exposed from the `ISch_ServerInterface` interface.

- The `ISCH_ServerInterface` interface is the main interface in the Schematic API. To use Schematic interfaces, you need to obtain the `ISch_ServerInterface` object by invoking the `SchServer` function. The `ISch_ServerInterface` interface is the gateway to fetching other Schematic objects.
- The `ISch_GraphicalObject` interface is a generic interface used for all Schematic design object interfaces.
- The `ISch_Document` interface points to an existing Schematic document in Altium Designer.

GetCurrentSchDocument Example

```
If SchServer = Nil Then Exit;  
  
If Not Supports (SchServer.GetCurrentSchDocument, ISch_Document, CurrentSheet) Then  
Exit;
```

```
ParentIterator := CurrentSheet.SchIterator_Create;  
  
If ParentIterator = Nil Then Exit;
```



For detailed information on Schematic API, refer to the SCH API Reference online help.

Main Schematic interfaces

- The `ISCH_ServerInterface` interface is the main interface in the Schematic API. To use Schematic interfaces, you need to obtain the `ISch_ServerInterface` object by invoking the `SchServer` function. The `ISch_ServerInterface` interface is the gateway to fetching other Schematic objects.
- The `ISch_GraphicalObject` interface is a generic interface used for all Schematic design object interfaces.
- The `ISch_Document` interface points to an existing Schematic document in Altium Designer.

When you need to deal with Schematic design objects in Altium Designer, the starting point is to invoke the `SchServer` function and with the `ISch_ServerInterface` interface, you can extract the all other derived schematic interfaces that are exposed from the `ISch_ServerInterface` interface.

SchServer function example

```
If SchServer = Nil Then Exit;
```

```
If Not Supports (SchServer.GetCurrentSchDocument, ISch_Document, CurrentSheet) Then
Exit;
```

```
ParentIterator := CurrentSheet.SchIterator_Create;
```

```
If ParentIterator = Nil Then Exit;
```

Accessing properties and methods of Schematic interfaces

For each Schematic object interface, there will be methods and properties listed (not all interfaces will have both methods and properties listed, some will only have methods).

A property of an object interface is like a variable, you get or set a value in a property, but some properties are read only properties, meaning they can only return values but cannot be set. A property is implemented by its Get and Set methods for example, the `Selection` property has two methods

```
Function GetState_Selection : Boolean; and Procedure SetState_Selection(B : Boolean);
```

Property values example

```
Component.Selection := True //set the value
```

```
ASelected := Component.Selection //get the value
```

The `ISch_GraphicalObject` is the base interface for all descendant Schematic design object interfaces such as `ISch_Arc` and `ISch_Line`, therefore all the methods and properties from the base interface are available in the descendant design objects.

For example the `Selection` property and its associated `Function GetState_Selection : Boolean; and Procedure SetState_Selection (B : Boolean);` methods declared in the `ISch_GraphicalObject` interface are inherited in the descendant interfaces such as `ISch_Arc` and `ISch_Line` interfaces.

This `Selection` property is not in the `ISch_Arc` according to the `RT_Schematic` unit, but you will notice that the `ISch_Arc` interface is inherited from the base `ISch_GraphicalObject` interface and this interface has a `Selection` property along with its associated methods (`GetState` function and `SetState` procedure for example).

If you can't find a method or a property in an object interface that you expect it to be in, then the next step is to look into the base `ISch_GraphicalObject` interface. You can check the interfaces in `RT_Schematic.INT` file in the `\RTL\RTL Interfaces` folder. This file defines all the interfaces that are exposed and can be used to access most of the objects in Schematic.

Typecasting schematic interface objects

With server development, you need to deal with types of variables and objects. Interface types follow the same rules as class types in variable and value typecasts. Class type expressions can be cast to interface types, for example `ISch_Pin(SchPin)` provided the `SchPin` variable implements the interface. With scripts, they are typeless and you normally do not need to type cast objects.

You cannot directly typecast Delphi objects that are not directly implemented as interface objects if these Delphi objects do not have interfaces, for example, this code snippet produces an exception because you are typecasting a Delphi object that do not directly implement the interface.

Using the Altium Designer RTL

The solution to this problem is to use the `Support` keyword as part of Borland Delphi's VCL library. This `Supports` keyword checks whether a given object or interface supports a specified interface. You can call this `Supports` function to check if the Delphi object contains references to interface objects.

Illegal Typecasting example for server projects

```
SchComponent := CurrentLib.CurrentSchComponent;
If SchComponent = Nil Then Exit;
PinList := GetState_AllPins(SchComponent);
For i := 0 to PinList.Count - 1 do
Begin
    SchPin := ISch_Pin(PinList[i]);
    ShowMessage(SchPin.Designator+' '+SchPin.Name);
End;
```

Using the Supports function example for server projects

```
SchComponent := CurrentLib.CurrentSchComponent;
If SchComponent = Nil Then Exit;
PinList := GetState_AllPins(SchComponent);
For i := 0 to PinList.Count - 1 do
Begin
    If (Supports(TObject(ISch_Pin(PinList[i])),ISch_Pin, SchPin) Then
        ShowMessage(SchPin.Designator+' '+SchPin.Name);
End;
```

Another example of the Supports function for server projects

```
If Not Supports (SchServer.GetCurrentSchDocument, ISch_Lib, CurrentLib) Then Exit;
```

The Schematic database system

The Schematic editor uses a 32-bit database system and stores two types of objects - drawing and electrical objects. The database system has two different data structures which are the flat database and the spatial database. In this secondary database system, each container holds the same object kind, for example the bus entry container consists of a linear list of bus entry objects which are organized by their coordinates.

The type of database is selected automatically depending on how you wish to access schematic objects and every existing schematic object on a schematic sheet is identified by its `TSchObjectHandle` value. Each Schematic object wrapped by its interface has an `I_ObjectAddress` function.

Schematic documents

There are two types of documents in Schematic editor; the Schematic document and the Schematic Library document. Dealing with Schematic documents is straightforward, you just obtain the Schematic document in question, and then you can add or delete Schematic objects.

The concept of handling a Schematic Library document is a bit more involved. Since each Schematic symbol (a component with its designator undefined) is part of the one and same Schematic library document and there are library documents within a Schematic library file. Therefore you need the schematic library container before you can iterate through the symbols of a library or add/delete symbols.

Loading Schematic or Library documents in Altium Designer

There are other situations when you need to programmatically open a specific document. This is facilitated by using the `Client.OpenDocument` and `Client.ShowDocument` methods.

Opening a text document, you pass in the 'Text' string along with the full file name string. For Schematic and Schematic Library documents, the 'SCH' and 'SCHLIB' strings respectively need to be passed in along with the full file name string. For PCB and PCB Library documents, the 'PCB' and 'PCBLIB' strings respectively need to be passed in along with the full file name string.

Since the parameters are null terminated types for some of the functions, and often the strings are `TDynamicString` types, thus you will need to typecase these strings as `PChar` type.

In the code snippet below, the `Filename` parameter is a `TDynamicString` type and this parameter is typecasted as a `PChar` in the `Client.OpenDocument` method.

Opening a schematic document using `Client.OpenDocument` method

```
Var
    ReportDocument : IServerDocument;
Begin
    ReportDocument := Client.OpenDocument('SCH',PChar(Filename));
    If ReportDocument <> Nil Then
        Client.ShowDocument(ReportDocument);
End
```

See also



See the source code example in the `\Developer Kit\DXP\Server Information` folder.

Creating Schematic or Library documents in Altium Designer

There are situations when you need to programmatically create a blank stand-alone document. This is facilitated by using the `CreateNewDocumentFromDocumentKind` function from the `RT_ClientServerInterface` unit. For example, creating a text document, you pass in the 'Text' string.

`CreateNewDocumentFromDocumentKind` example

```
Var
    Document : IServerDocument;
    Kind      : TDynamicString;
Begin
    //The available Kinds are PCB, PCBLib, SCH, SchLib, TEXT,...
    Kind := 'SCH';
```

Using the Altium Designer RTL

```
Document := CreateNewDocumentFromDocumentKind(Kind);
```

```
End;
```

Create a blank schematic and add to the current project

```
Var
```

```
Doc      : IServerDocument;
```

```
Project  : IProject;
```

```
Path     : TDynamicString;
```

```
Begin
```

```
If SchServer = Nil then Exit;
```

```
Project := GetWorkSpace.DM_FocusedProject;
```

```
If Project <> Nil Then
```

```
Begin
```

```
Path := GetWorkSpace.Dm_CreateNewDocument('SCH');
```

```
Project.DM_AddSourceDocument(Path);
```

```
Doc := Client.OpenDocument(Pchar('SCH',Pchar(Path)));
```

```
Client.ShowDocument(Doc);
```

```
End;
```

```
If Not Supports (SchServer.GetCurrentSchDocument,ISchDocument,Doc) Then Exit;
```

```
End;
```

See also



See the source code example in the \Developer Kit\DXP\Server Information folder.



Check out the script examples in the \Examples\Scripts\DelphiScript Scripts\DXP\ folder.

However, generally you would like to create a document programmatically and put in the currently focussed project. To do this, you would need the interface access to the `WorkSpaceManager` in Altium Designer and invoke `DM_FocusedProject` and `DM_AddSourceDocument` functions.

Adding a document to a project

```
Procedure Command_CreateSchObjects(View : IServerDocumentView; Parameters : PChar);
```

```
Var
```

```
Doc      : IServerDocument;
```

```
Project  : IProject;
```

```
Path     : TDynamicString;
```

```
Begin
```

```
If SchServer = Nil Then Exit;
```

```
// create a blank schematic document and adds to the currently focussed project.
```

```

Project := GetWorkspace.DM_FocusedProject;
If Project <> Nil Then
Begin
    Path := Getworkspace.DM_CreateNewDocument('SCH');
    Project.DM_AddSourceDocument(Path);
    Doc := Client.OpenDocument(PChar('SCH'), PChar(Path));
    Client.ShowDocument(Doc);
End;

If Not Supports (SchServer.GetCurrentSchDocument, ISch_Document, SchDoc) Then
Exit;
    // do what you want with the schematic document!
End;

```

Checking the type of schematic documents in Altium Designer

You can use the `GetCurrentSchematicDocument` function from the `SchServer` object to check whether the current schematic document is a schematic type document.

ISch_Doc type code snippet

```

Var
    CurrentSch : ISch_Doc;
Begin
    If Not Supports (SchServer.GetCurrentSchDocument, ISch_Doc, CurrentDoc) Then
    Begin
        ShowInfo('Not a Schematic Library document. ');
        Exit;
    End;
//ditto
End;

```

This code snippet checks if the focused document is a schematic library type.

ISch_Lib type code snippet

```

Var
    CurrentLib : ISch_Lib;
Begin
    If Not Supports (SchServer.GetCurrentSchDocument, ISch_Lib, CurrentLib) Then
    Begin

```

Using the Altium Designer RTL

```
ShowInfo('Not a Schematic Librarydocument.');
```

```
Exit;
```

```
End;
```

```
End;
```

This code snippet checks if a document is a Schematic library format.

CurrentView.Kind example

```
If StrPas(Client.CurrentView.Kind) <> UpperCase('SchLib') Then Exit;
```

This code snippet uses the `Client.CurrentView.Kind` method to find out the current document's type.

Setting a document dirty

There are situations when you need to programmatically set a document dirty so when you close Altium Designer, it prompts you to save this document. This is facilitated by setting the `ServerDocument.Modified` to true.

Setting a document dirty example

```
Var
```

```
    AView          : IServerDocumentView;
```

```
    AServerDocument : IServerDocument;
```

```
Begin
```

```
    // grab the current document view using the Client's Interface.
```

```
    AView := Client.GetCurrentView;
```



```
    //grab the server document which stores views by extracting the ownerdocument
```

```
field.
```

```
    AServerDocument := AView.OwnerDocument;
```



```
    // set the document dirty.
```

```
    AServerDocument.Modified := True;
```

```
End;
```

See also



See the source code example in the `\Developer Kit\DXP\Server Information` folder.



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\DXP\` folder.

Refreshing a document programmatically

When you place or modify objects on a schematic document, you often need to do a refresh of this document (you will also need to use the PreProcess / PostProcess, and SchServer.RobotManager.SendMessage() methods).

- ISch_Document.GraphicallyInvalidate;
- RunProcess(); (Script code)
- Commands.LaunchCommand(); (Server code)

Few examples below demonstrate more than one way to update the document. You can use the ICommandLauncher.LaunchCommand method or the IProcessLauncher's SendMessage function from RT_CClientServer Interface unit.

GraphicallyInvalidate and RunProcess methods in a script example

```
//Using GraphicallyInvalidate method to refresh the screen
Var
    SchDoc : ISch_Document;
Begin
    // Refresh the screen
    SchDoc := SchServer.GetCurrentSchDocument;
    If SchDoc = Nil Then Exit;
    // modify the schematic document (new objects, objects removed etc)
    // Call to refresh the schematic document.
    SchDoc.GraphicallyInvalidate;
    ResetParameters;
    AddStringParameter('Action', 'Document');
    RunProcess('Sch:Zoom');
End;
```

Commands.LaunchCommand example in a server project

```
Procedure ZoomToDoc;
Var
    Commands      : ICommandLauncher;
    Parameters    : PChar;
    SchematicServer : IServerModule;
Begin
    GetMem (Parameters, 256);
    Try
        SetState_Parameter(Parameters, 'Action', 'Document');
        If Supports(SchServer, IServerModule, SchematicServer) Then
```

Using the Altium Designer RTL

```
Begin
    If Supports (SchematicServer.CommandLauncher, ICommandLauncher, Commands)
Then
    Commands.LaunchCommand('Zoom', Parameters, 255, Client.CurrentView);

End;
Finally
    FreeMem(Parameters);
End;
End;
```

Client.SendMessage in a script example

```
Client.SendMessage('SCH:Zoom', 'Action=Document' , 255, Client.CurrentView);
```

Note, in a server project, the `SendMessage` method is available from the `IProcessLauncher` interface in the `RT_ClientServerInterface` unit.

Schematic Objects

Schematic design objects are stored inside the database of the Schematic editor for the currently active schematic document and the basic Schematic objects are called primitives. There are two types of primitives in the Schematic editor: Electrical primitives and non-electrical primitives. Each design object has a unique object handle which is like a pointer. These handles allow you to access and change the design object's properties.

The Schematic editor includes the following electrical primitives- Bus, Bus Entry, Junction, Port, Power Port, PCB layout directive, Pin, No ERC Directive, Sheet Entry, Sheet Symbol, Stimulus Directive, Test Vector Directive, and Wire objects.

Non electrical primitives include- Annotation, Arc, Bezier, Ellipse, Elliptical Arc, Graphical Image, Line, Pie, Polygon, Rectangle, Rounded Rectangle, and Text Frame objects. The non-electrical primitives are used to add reference information to a sheet. They are also used to build graphical symbols, create custom sheet borders, title blocks or adding notes and instructions.

The schematic editor has other system objects such as a container for templates, preferences settings, a search facility, a font manager, a robot manager (capture events of the schematic editor) and so on.

The schematic objects that have objects within themselves are called group objects. The group objects are part objects and sheet symbol objects, that is, Part objects have pin objects. Sheet symbols have sheet entry objects.

`ISch_BasicContainer` interface is the ancestor interface for all Schematic design objects including schematic sheets and library documents. This interface has methods that return the unique object address and setup an iterator with filters to look for specific objects within a defined region.

`ISch_GraphicalObject` interface is the interface for all schematic design objects with graphical attributes.

The three interfaces, `ISch_MapDefiner`, `ISch_ModelDatafileLink`, `ISch_Implementation` all deal with the mapping of schematic components to its models such as PCB footprint, 3D Model, Signal Integrity model and so on.

Accessing schematic objects

An iterator provides a way of accessing the elements of an aggregate object sequentially without exposing its underlying representation. The use of iterators provides a compact method of accessing Schematic objects without creating a mirror database across the API. To retrieve objects on a schematic sheet or a library document, you will need to employ an iterator which is an efficient data retrieval method – there are three types of iterators – simple iterators, spatial iterators and group iterators.

- Object iterators are used to conduct global searches.
- Spatial iterators are used to conduct restricted searches.
- Group iterators are used to conduct searches for primitives inside certain schematic objects. These schematic objects which have objects within them are called group objects. Such group objects are sheet symbols and part objects.

In all, you can specify which objects to look for, in the specified region of a document with a spatial iterator. You also set up global iterators that look inside the child objects of a parent object, for example sheet entries of a sheet symbol or parameters of a schematic component.

Iterating for Schematic objects example

```

Var
    Pin          : ISch_Pin;
    PinIterator  : ISch_Iterator;
    PinFound     : Boolean;

Begin
    PinFound := False;
    PinIterator := AComponent.SchIterator_Create;
    PinIterator.SetState_IterationDepth(eIterateAllLevels);
    PinIterator.AddFilter_ObjectSet([ePin]);

    Try
        Pin := PinIterator.FirstSchObject As ISch_Pin;
        While Pin <> Nil Do
            Begin
                If Not PinFound Then
                    PinFound := True;
                // add pins to the PinsList container of a TStringList type.
                PinsList.Add('Pin ' + Pin.Designator +
                    ' located at (x=' + IntToStr(Pin.Location.X) +
                    ', y=' + IntToStr(Pin.Location.Y) + ')');
                Pin := PinIterator.NextSchObject As ISch_Pin;
            
```

Using the Altium Designer RTL

```
        End;  
    Finally  
        AComponent.SchIterator_Destroy(PinIterator);  
    End;  
    If Not PinFound Then  
        PinsList.Add('There are no pins for this component.');
```

End;

This code snippet demonstrates the method of fetching schematic objects from a schematic sheet using an iterator.

See also



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\Sch\` folder.



See the source code example in the `\Developer Kit\Sch\Library Utilities` folder.

Creating / deleting new schematic objects

Schematic objects created using the Schematic API will need to follow a few simple steps to ensure that the database system of the Schematic editor will successfully register new objects. The example below demonstrates the placement of a Port object onto a Schematic document programmatically.

Create and place a port object using a script

```

Procedure PlaceAPort;
Var
    AName      : TDynamicString;
    Orientation : TRotationBy90;
    AElectrical : TPinElectrical;
    SchPort     : ISch_Port;
    Loc         : TLocation;
    FSchDoc     : ISch_Document;
    CurView     : IServerDocumentView;
Begin
    If SchServer = Nil Then Exit;
    FSchDoc := SchServer.GetCurrentSchDocument;
    If FSchDoc = Nil Then Exit;

    SchPort := SchServer.SchObjectFactory(ePort,eCreate_GlobalCopy);
    If SchPort = Nil Then Exit;
    SchPort.Location := Point(100,100);
    SchPort.Style := ePortRight;
    SchPort.IOType := ePortBidirectional;
    SchPort.Alignment := eHorizontalCentreAlign;
    SchPort.Width := 100;
    SchPort.AreaColor := 0;
    SchPort.TextColor := $FFFFFF;
    SchPort.Name := 'Test Port';

    // Add a new port object in the existing Schematic document.
    FSchDoc.AddSchObject(SchPort);
    FSchDoc.GraphicallyInvalidate;

    // use of Server processes to refresh the screen.

```

Using the Altium Designer RTL

```
ResetParameters;  
AddStringParameter('Action', 'Document');  
RunProcess('Sch:Zoom');  
End;
```

How does this code work?

A new port object is created by the `SchObjectFactory` method. This function takes in two parameters, the `ePort` value of `TObjectID` type and the creation model parameter of `TObjectCreationMode` type. The port's attributes need to be set accordingly, then you will need to register this port object into the Schematic database.

The `AddSchObject` method needs to be invoked for the `SchServer` object which represents the schematic database. Finally the `GraphicallyInvalidate` call refreshes the schematic document

Note that a `RGB` function is used in the code to assign a color value to the `areacolor` and `textcolor` fields. This function is a Windows API function and takes in three parameters for the three color intensities (red, green and blue). In this case, (255,255,255) represents the white color, and (0,0,0) represents the black color.

See also



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\Sch\` folder.



See the source code example in the `\Developer Kit\Sch\Sch Objects` folder.

Creation of a new schematic object on a library document

To create a symbol on an existing library document, it is done the same way as you create objects on a schematic document.

Creating a new library component on a new library document

```

Procedure CreateALibComponent;
Var
    CurrentLib    : ISch_Lib;
    SchComponent  : ISch_Component;
    R              : ISch_Rectangle;
    Location      : TLocation;
    Corner        : TLocation;
Begin
    If SchServer = Nil Then Exit;
    If Not Supports (SchServer.GetCurrentSchDocument, ISch_Lib, CurrentLib) Then
Exit;

    If Not Supports (SchServer.SchObjectFactory(eSchComponent, eCreate_Default),
ISch_Component, SchComponent) Then Exit;

    If Not Supports (SchServer.SchObjectFactory(eRectangle, eCreate_Default),
ISch_Rectangle, R)          Then Exit;

    SchComponent.CurrentPartID := 1;
    SchComponent.DisplayMode   := 0;

    SchComponent.LibReference := 'Custom';
    CurrentLib.AddSchComponent(SchComponent);
    CurrentLib.CurrentSchComponent := SchComponent;

    R.LineWidth := eSmall;
    Location.X   := 10;
    Location.Y   := 10;
    R.Location   := Location;

    Corner.X     := 30;
    Corner.Y     := 20;

```

Using the Altium Designer RTL

```
R.Corner      := Corner;

R.Color       := $FFFF;    // YELLOW
R.AreaColor  := 0;        // BLACK
R.IsSolid    := True;

R.OwnerPartId      := CurrentLib.CurrentSchComponent.CurrentPartID;
R.OwnerPartDisplayMode := CurrentLib.CurrentSchComponent.DisplayMode;
CurrentLib.CurrentSchComponent.AddSchObject(R);

CurrentLib.CurrentSchComponent.Designator.Text := 'U';
CurrentLib.CurrentSchComponent.ComponentDescription := 'Custom IC';

ZoomToDoc;

End;
```

See also



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\Sch\` folder.



See the source code example in the `\Developer Kit\Sch\Component Utilities` folder.

Creating objects and refreshing the Undo/Redo system in the Schematic Editor

The simple creation of objects in the examples above does not refresh the Undo system in the Schematic Editor, and to do this, you will need to employ the `SchServer.RobotManager.SendMessage` method.

The sequence is as follows;

- PreProcess which initialize the robots in the Schematic server;
- Add new objects;
- PostProcess which cleans up the robots in the Schematic server

Creating schematic objects example

This example describes the correct method for allowing Undo/Redo at various different levels of objects (the first at adding components to the document, and the second at adding parameters to the pin of a placed component).

Specifically this will add a constructed component to the current sheet, and then a parameter to the pin. You will then be able to do undo, at the first press of 'Undo', the parameter being added to the pin and then, using undo a second time, adding the component to the document.

```
If Not Supports (SchServer.GetCurrentSchDocument, ISch_Sheet, SchDoc) Then Exit;
```

```
If Not Supports (SchServer.SchObjectFactory (eSchComponent, eCreate_Default),
                                                    ISch_Component, Component) Then
Exit;
If Not Supports (SchServer.SchObjectFactory (eRectangle, eCreate_Default),
                                                    ISch_Rectangle, Rect) Then Exit;
If Not Supports (SchServer.SchObjectFactory (ePin, eCreate_Default), ISch_Pin,
Pin)
                                                    Then Exit;
If Not Supports (SchServer.SchObjectFactory (eParameter, eCreate_Default),
                                                    ISch_Parameter, Param) Then Exit;
If Supports (SchServer, IServerModule, SchematicServer) Then
Begin
    Try
        // Initalize the Robot Manager
        SchematicServer.ProcessControl.PreProcess(Client.CurrentView, '');

        // Add the component to the current schematic sheet with undo stack
enabled
        Rect.OwnerPartId := Component.CurrentPartID;
        Rect.OwnerPartDisplayMode := Component.DisplayMode;
        Location.X := 0;
        Location.Y := 0;
        Rect.Location := Location;
        Location.X := 20;
        Location.Y := 20;
        Rect.Corner := Location;

        Pin.OwnerPartId := Component.CurrentPartID;
        Pin.OwnerPartDisplayMode := Component.DisplayMode;
        Location.X := 20;
        Location.Y := 10;
        Pin.Location := Location;

        Component.AddSchObject(Rect);
        Component.AddSchObject(Pin);
        SchDoc.AddSchObject(Component);
        Component.MoveByXY(100, 100);
```

Using the Altium Designer RTL

```
SchServer.RobotManager.SendMessage(SchDoc.I_ObjectAddress, c_BroadCast,
                                   SCHM_PrimitiveRegistration, Component.I_ObjectAddress);

Finally
    // Clean up the Robot Manager
    SchematicServer.ProcessControl.PostProcess(Client.CurrentView, '');

End;

End;
```

See also



Check out the script examples in the \Examples\Scripts\DelphiScript Scripts\Sch\ folder.



UndoRedo addon example in \Developer Kit\Examples\Sch\UndoRedo folder

Removing schematic objects example

```
Iterator := CurrentSheet.SchIterator_Create;
If Iterator = Nil Then Exit;
Iterator.AddFilter_ObjectSet([ePort]);

// Initalize the Robot Manager
SchematicServer.ProcessControl.PreProcess(Client.CurrentView, '');
Try
    Port := Iterator.FirstSchObject As ISch_Port;
    While Port <> Nil Do
        Begin
            OldPort := Port;
            Port := Iterator.NextSchObject As ISch_Port;
            CurrentSheet.RemoveSchObject(OldPort);
            SchServer.RobotManager.SendMessage(CurrentSheet.I_ObjectAddress,
                                               c_BroadCast,
                                               SCHM_PrimitiveRegistration,
                                               OldPort.I_ObjectAddress);

        End;
    Finally
        CurrentSheet.SchIterator_Destroy(Iterator);

    // Clean up the Robot Manager
    SchematicServer.ProcessControl.PostProcess(Client.CurrentView, '');

End;
```

See also

Check out the script examples in the \Examples\Scripts\DelphiScript Scripts\Sch\ folder.



DeletePort Objects addon example in \Developer Kit\Examples\Sch\UndoRedo folder

Modifying Schematic Objects

To modify Schematic objects on a current Schematic document, you will need to invoke certain methods in a certain order to ensure all the Undo/Redo system is up to date when a schematic object's attributes have been modified programmatically.

The sequence is as follows;

- Invoke the PreProcess method which initialize the robots in the schematic server
- Send a SCHM_BeginModify message
- Modify the Schematic object
- Send a SCHM_EndModify message
- Invoke the PostProcess method to clean up the robots in the Schematic server.

Changing Schematic object's attributes example

```
Procedure FetchAndModifyObjects;
```

```
Var
```

```
  AnObject      : ISch_GraphicalObject;
  Iterator      : ISch_Iterator;
  SchematicServer : IServerModule;
  Doc           : ISch_Document;
```

```
Begin
```

```
  SchematicServer := SchServer As IServerModule;
  Doc              := SchServer.GetCurrentSchDocument;
  Iterator         := Doc.SchIterator_Create;
  Iterator.AddFilter_ObjectSet([ePort,eWire]);

  SchematicServer.ProcessControl.PreProcess(Client.CurrentView, '');
  // Initialize the Robot Manager
  SchServer.ProcessControl.PreProcess(Doc, '');
```

```
If Iterator = Nil Then Exit;
```

```
Try
```

```
  AnObject := Iterator.FirstSchObject As ISch_GraphicalObject;
  While AnObject <> Nil Do
  Begin
```

Using the Altium Designer RTL

```
SchServer.RobotManager.SendMessage(AnObject.I_ObjectAddress, c_BroadCast,
                                     SCHM_BeginModify, c_NoEventData);

Case AnObject.ObjectId Of
    eWire    : AnObject.Color      := $FF0000;
    ePort    : AnObject.AreaColor := $00FF00;
End;

SchServer.RobotManager.SendMessage(AnObject.I_ObjectAddress, c_BroadCast,
                                     SCHM_EndModify , c_NoEventData);

AnObject := Iterator.NextSchObject As ISch_GraphicalObject;

End;

Finally
    Doc.SchIterator_Destroy(Iterator);

End;

// Clean up the Robot Manager
SchServer.ProcessControl.PostProcess(Doc, '');

End;
```

Notes

When you change the properties of a schematic object on a Schematic document, it is necessary to employ the `SchServer.RobotManager.SendMessage` function to update the various subsystems of the Schematic system such as the Undo/Redo system and setting the document as dirty so the document can be saved. Look for `SendMessage` method of the `ISch_RobotManager` in this document.

Remember, the Schematic API only can be used to work on a currently open Schematic document in the Altium Designer, if you need to fetch specific Schematic documents, you will need extra routines to open and display a document, please refer to the `DXP Developer Kit Guide` document in the `\Developer Kit\` folder and the `\Developer Kit\Examples\DXP\Server Information` source code for details.

See also



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\Sch\` folder.



Examples in the `\Developer Kit\Examples\Sch\UndoRedo` folder.

Schematic interactive feedback using the mouse

To monitor the mouse movement and clicks from your script, the `ISch_Document` document interface and its descendant interfaces, `ISch_Lib` and `ISch_Sheet` interfaces has several interactive feedback methods. For example the `ChooseRectangleInteractively` method can be used for the `SpatialIterator` where it needs the bounds of a rectangle on the schematic document to search within.

- `ChooseLocationInteractively` method
- `ChooseRectangleInteractively` method

ChooseRectangleInteractively example

Var

```
CurrentSheet      : ISch_Document;
SpatialIterator  : ISch_Iterator;
GraphicalObj     : ISch_GraphicalObject;
Rect             : TCoordRect;
```

Begin

```
If SchServer = Nil Then Exit;
CurrentSheet := SchServer.GetCurrentSchDocument;
If CurrentSheet = Nil Then Exit;

Rect := TCoordRect;
If Not CurrentSheet.ChooseRectangleInteractively(Rect,
    'Please select the first corner',
    'Please select the final corner') Then Exit;

SpatialIterator := CurrentSheet.SchIterator_Create;
If SpatialIterator = Nil Then Exit;
Try
    SpatialIterator.AddFilter_ObjectSet(MkSet(eJunction, eSchComponent));
    SpatialIterator.AddFilter_Area(Rect.left, Rect.bottom, Rect.right, Rect.top);
    GraphicalObj := SpatialIterator.FirstSchObject;
    While GraphicalObj <> Nil Do
        Begin
            // do what you want with the design object
            GraphicalObj := SpatialIterator.NextSchObject;
        End;
    Finally
        CurrentSheet.SchIterator_Destroy(SpatialIterator);
```

Using the Altium Designer RTL

```
End;
```

```
End;
```

See also

ISch_Document interface



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\Sch\` folder.



Spatial Iterator example in the `\Developer Kit\Examples\Sch\Iterators` folder

Activating Schematic API

There are situations when Altium Designer is in a blank editing session. When Schematic documents have not been opened, the Schematic API is not active. At these times, you need to manually start the server to have access to the Schematic API. Starting up the Schematic server to activate the Schematic API is done by invoking the `Client.StartServer` method with a 'SCH' parameter.

```
Client.StartServer('SCH');
```

PCB Editor interfaces

The PCB API allows a programmer to fetch or modify PCB objects and their attributes from a PCB document. The objects shown on a document are stored in its corresponding design database.

The PCB interfaces exposed by the PCB editor refer to opened PCB documents and the objects on them. The PCB API derives its functionality from the `RT_PCB` unit and this unit contains interface objects declarations and type declarations.

An interface is just a means of access to an object in memory. To have access to the PCB server and message certain PCB design objects, you need to invoke the `PCBServer` function which extracts the `IPCB_ServerInterface` interface. This is the main interface and contains many interfaces within. With this interface, you can proceed further by iterating for certain PCB objects.

The `IPCB_ServerInterface` and `IPCB_Board` interfaces to name the few are the main interfaces that you will be dealing with, when you are extracting data from a PCB or PCB Library document. Below is the simplified PCB object interface hierarchy.

A simplified PCB Objects Interfaces hierarchy

```

IPCB_Primitive
    IPCB_Arc
        IPCB_Via
        IPCB_Group
            IPCB_Net
  
```

PCBServer function

When you need to work with PCB design objects in Altium Designer, the starting point is to invoke the `PCBServer` function and with the `IPCB_ServerInterface` interface, you can extract the all other derived PCB interfaces that are exposed in the `IPCB_ServerInterface` interface. For example to get an access to the current PCB document open in Altium Designer, you would invoke the `GetCurrentPCBBoard` method from the `IPCB_ServerInterface` interface object.

- The `IPCB_ServerInterface` interface is the main interface in the PCB API. To use PCB interfaces, you need to obtain the `IPCB_ServerInterface` object by invoking the `PCBServer` function. The `IPCB_ServerInterface` interface is the gateway to fetching other PCB objects.
- The `IPCB_Primitive` interface is a generic interface used for all PCB design object interfaces.
- The `IPCB_Board` interface points to an existing PCB document in Altium Designer.

GetCurrentPCBBoard Example

```

TheServer := PCBServer.GetCurrentPCBBoard;

If TheServer = Nil Then Exit;

TheBoard := PCBBoard.GetCurrentPCBBoard
If TheBoard = Nil Then Exit;
  
```

Using the Altium Designer RTL

```
TheFileName := TheBoard. GetState_FileName;
```



For detailed information on PCB API, refer to the PCB API Reference online help.

Invoking properties and methods of PCB interfaces

For each PCB object interface, there will be methods and properties listed (not all interfaces will have both methods and properties listed, that is, some interfaces will only have methods).

A class method is a procedure or function that operates on a class rather than on specific instances of the class.

A property of an object interface is like a variable, you get or set a value in a property, but some properties are read only properties, meaning they can only return values but cannot be set. A property is implemented by its Get and Set methods

For example the `IPCB_Component` interface has a Height property and two associated methods `GetState_Height` and `SetState_Height`;

```
Function GetState_Height : TCoord;
```

```
Procedure SetState_Height (Value : TCoord);
```

```
Property Height : TCoord Read GetState_Height Write SetState_Height;
```

Another example is that the `Selected` property has two methods `Function GetState_Selected : Boolean`; and `Procedure SetState_Selected (B : Boolean)`;

Object Property example

```
PCBComponent.Selected := True //set the value
```

```
ASelected := PCBComponent.Selected //get the value
```

The `IPCB_Primitive` is the base interface for all other PCB design object interfaces such as `IPCB_Track` and `IPCB_Component`.

For example the `Selected` property and its associated `Function GetState_Selected : Boolean`; and `Procedure SetState_Selected (B : Boolean)`; methods declared in the `IPCB_Primitive` interface are inherited in the descendant interfaces such as `IPCB_Component` and `IPCB_Pad` interfaces.

This `Selected` property is not in the `IPCB_Component` according to the `RT_PCB` unit, but you will notice that the `IPCB_Component` is inherited from `IPCB_Primitive` and this interface has a `Selected` property along with its associated methods (`GetState` function and `SetState` procedure).

If you can't find a method or a property in an object interface that you expect it to be in, then the next step is to look into the base `IPCB_Primitive` interface. You can check the interfaces in `RT_PCB.INT` file in the `\RTL\RTL Interfaces` folder. This file defines all the interfaces that are exposed and can be used to access most of the objects in PCB.

The PCB database system

The database system is the heart of the PCB editor, and this system stores all of the information about the PCB document. The objects in the database system are a direct representation of the objects on the PCB document.

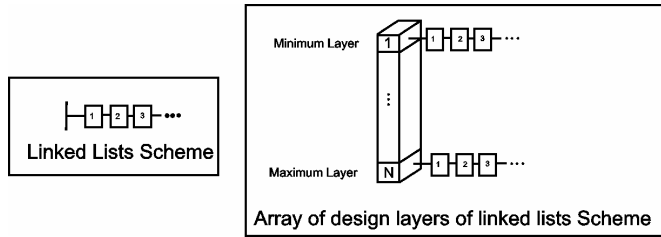


Figure 6: Storage scheme using linked lists

The database system of the PCB editor stores primitive objects and group objects such as pads, vias,

tracks, arcs, fills, text, components, nets, coordinates, dimensions, polygons, from tos, manual from-tos, classes, rules, violations and embedded objects. This primary database is a **flat** database, which is composed of two different data structures. The first structure on the left of this figure above consists of separate singular linked lists of vias, components, nets, coordinates, dimensions, classes, rules, from-tos, and connections. The second structure on the right of figure above consists of an array of linked lists of the same kind of PCB objects. Each array exists separately for track, arc, fill, text string, pad and polygon objects. This second data structure is optimized for performance reasons.

The secondary database system is a **spatial** database. For each layer in the spatial database, there is a container, which is a spatial or quad tree that can store any kind of PCB object. This secondary database is optimised for fast access of PCB objects. The type of database is chosen automatically depending on which specific PCB functions are being invoked.

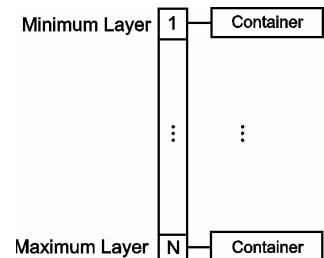


Figure 7: Spatial database

PCB Documents

There are two types of documents in PCB editor; the PCB document and the PCB Library document. Dealing with PCB documents is straightforward.

The concept of handling a PCB Library document is a bit more involved, since each PCB footprint (a component with an undefined designator) occupies one PCB library document within a PCB library file. Note that, you can only place tracks, arcs, fills, texts, pads and vias on a library document.

Loading PCB or PCB Library documents

There are other situations when you need to programmatically open a specific document. This is facilitated by using the Client object and invoking one of its methods such as `Client.OpenDocument` and `Client.ShowDocument` methods from the `RT_ClientServerInterface` unit.

Opening a text document, you pass in the 'Text' string along with the full file name string. For PCB and PCB Library documents, the 'PCB' and 'PCBLIB' strings respectively need to be passed in along with the full file name string. For Schematic and Schematic Library documents, the 'SCH' and 'SCHLIB' strings respectively need to be passed in along with the full file name string.

Using the Altium Designer RTL

Since the parameters for some of the functions are null terminated types, and often the strings are TPCBString types, thus you will need to typecase these strings as PChar type.

In the code snippet below, the `IServerDocument.DM_FullPath` method returns a TPCBString, and this parameter is typecasted as a PChar in the `Client.OpenDocument` method.

Opening a text document using Client.OpenDocument method

```
Var
    ReportDocument : IServerDocument;
Begin
    ReportDocument := Client.OpenDocument('Text', PChar(FileName));
    If ReportDocument <> Nil Then
        Client.ShowDocument(ReportDocument);
End
```

See also



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\PCB\` folder.



See the source code example in the `\Developer Kit\DXP\Server Information` folder.

Creating PCB or PCB Library documents

There are situations when you need to programmatically create a blank document, this is facilitated by using the `CreateNewDocumentFromDocumentKind` function from the `RT_ClientServerInterface` unit. For example, creating a text document, you pass in the 'Text' string.

CreateNewDocumentFromDocumentKind example

```
Var
    Document : IServerDocument;
    Kind      : TPCBString;
Begin
    //The available Kinds are PCB, PCBLib, SCH, SchLib, TEXT,...
    Kind := 'PCB';
    Document := CreateNewDocumentFromDocumentKind(Kind);
End;
```

Create a blank PCB and add to the current project

```
Var
    Doc      : IServerDocument;
    Project  : IProject;
    Path     : TPCBString;
Begin
    If PCBServer = Nil then Exit;
```

```

Project := GetWorkspace.DM_FocusedProject;
If Project <> Nil Then
Begin
    Path := GetWorkspace.Dm_CreateNewDocument('PCB');
    Project.DM_AddSourceDocument(Path);
    Doc := Client.OpenDocument(Pchar('PCB', PChar(Path)));
    Client.ShowDocument(Doc);
End;
// do what you want with the new document.

```

End;

See also



Check out the script examples in the \Examples\Scripts\DelphiScript Scripts\PCB\ folder.



See the source code example in the \Developer Kit\DXP\Server Information folder.

Checking the type of PCB documents

You can use the `GetCurrentPCBBoard` function from the `PCBServer` object and with the interface object (`IPCB_Board`) you can invoke the `IsLibrary` method to check whether the current PCB document is a PCB Library type document or not.

IsLibrary method example

```

// check if a PCB Library document exists.
Board := PCBServer.GetCurrentPCBBoard;
If Board = Nil Then Exit;
If Not (Board.IsLibrary) Then
    ShowMessage('This is not a PCBLIB document');

```

Document Kind method example

```

If StrPas(Client.CurrentView.Kind) <> UpperCase('PCBLib') Then Exit;

```

This code snippet uses the `Client.CurrentView.Kind` method to find out the current document's type. See the source code example in the \Altium Designer 6\Developer Kit\DXP\Server Information folder.

Setting a document dirty

There are situations when you need to programmatically set a document dirty so when you close Altium Designer, it prompts you to save this document. This is facilitated by setting the `ServerDocument.Modified` to true.

Document's Modified property example

```

Var
    AView          : IServerDocumentView;

```

Using the Altium Designer RTL

```
AServerDocument : IServerDocument;  
Begin  
    // grab the current document view.  
    AView := Client.GetCurrentView;  
  
    //grab the owner server document.  
    AServerDocument := AView.OwnerDocument;  
  
    // set the document dirty.  
    AServerDocument.Modified := True;  
End;
```

See also



Check out the script examples in the \Examples\Scripts\DelphiScript Scripts\PCB\ folder.



See the source code example in the \Developer Kit\DXP\Server Information folder.

Refreshing a document programmatically

When you place or modify objects on a PCB document, you often need to do a refresh of the document. An example below demonstrates one way to update the document.

Refresh PCB document example in a PCB server project

```
// Need RT_Param and RT_API units in a server project  
Procedure ReDrawCurrentBoard;  
Var  
    P : PChar;  
Begin  
    //need to refresh board.  
    GetMem(P, 255);  
    StrPCopy(P, '');  
    SetState_Parameter(P, 'Action', 'Redraw');  
    MessageRouter_SendCommandToModule('PCB:Zoom', P, 255,  
                                       Client.CurrentView);  
  
    FreeMem(P, 255);  
End;
```

Refresh a PCB document in a script

```
// Refresh PCB screen  
Client.SendMessage('PCB:Zoom', 'Action=Redraw' , 255, Client.CurrentView);
```

PCB Objects

The PCB design objects are stored inside the database of the currently active PCB document in PCB editor. Each design object has a handle. These handles allow you to access and change the object's properties and change the properties.

A PCB object is either a primitive or a group object. A primitive is a basic PCB object which could be one of the following: tracks, pads, fills, vias and so on.

A group object can be a component, dimension, coordinate, polygon or a net object and each group object is composed of primitives. Each group object also has its own small database that stores primitives. A component is a group object and thus is composed of primitives.

Accessing PCB objects

Iterators provide a way of accessing the elements of an aggregate object sequentially without exposing its underlying representation. With regards to the PCB editor's database system, the use of iterators provides a compact method of accessing PCB objects without creating a mirror database across the API.

The main function of an iterator is to traverse through the database to fetch certain PCB objects by traversing the database inside the PCB editor from the outside.

There are four types of iterators:

- Board iterators;
- Spatial iterators;
- Group iterators;
- Library iterators.

Board iterators are used to conduct global searches, while spatial iterators are used to conduct restricted searches within a defined boundary, group iterators are used to conduct searches for child objects within a group object such as a component, and finally a library iterator is used to conduct a search within a PCB library document within a PCB library.

Board Iterator example in a server project

```

Var
    Board      : IPCB_Board;
    Pad        : IPCB_Primitive;
    Iterator   : IPCB_BoardIterator;
Begin
    // retrieve the current board's handle
    Board      := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;

    // retrieve the iterator handle
    Iterator   := Board.BoardIterator_Create;

```

Using the Altium Designer RTL

```
Iterator.AddFilter_ObjectSet([ePadObject]);
Iterator.AddFilter_LayerSet(AllLayers);
Iterator.AddFilter_Method(eProcessAll);

// search and count pads
Pad := Iterator.FirstPCBObject;
While (Pad <> Nil) Do
Begin
    // do what you want with the fetched pad
    Pad := Iterator.NextPCBObject;
End;
Board.BoardIterator_Destroy(Iterator);
```

See also



See the iterator examples in the Developer Kit\Examples\PCB\PCB Iterators Pads folder.



Check out the script examples in the \Examples\Scripts\DelphiScript Scripts\PCB\ folder.

Creation of a new PCB object

PCB objects created using the PCB API will need to follow a few simple steps to ensure that the database system of the PCB editor will successfully register these objects. An example is shown to illustrate the steps involved in creating a new PCB object. The code will demonstrate the paramount role of the creation of a board object before any PCB objects are to be created, destroyed or modified. If there is no board object, the database system inside the PCB editor will not be updated and thus the current PCB document will not be affected either.

This code snippet demonstrates the registration and placement of a Via object (which is one of the PCB editor's design objects) onto a PCB document. This example will also illustrate the concept of object handles.

PCBObjectFactory method example

```
Procedure CreateAViaObject;
Var
    Board : IPCB_Board;
    Via    : IPCB_Via;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;

    (* Create a Via object*)
    Via := PCBServer.PCBObjectFactory(eViaObject) As IPCB_Via;
```

```
Via.X      := MilsToCoord(1000);  
Via.Y      := MilsToCoord(1000);  
Via.Size   := MilsToCoord(50);  
Via.HoleSize := MilsToCoord(20);  
  
Via.LowLayer := eTopLayer;  
Via.HighLayer := eBottomLayer;  
  
Board.AddPCBObject(Via);
```

```
End;
```

How does this code work?

The board is fetched which is a representation of an actual PCB document, and then a `PCBObjectFactory` method is called from the `IPCBoardServerInterface` (representing the PCB editor) and a copy of the `Via` object is created.

You are free to change the attributes of this new object and then you need to add this new object in the PCB database.

To ensure that the PCB editor's database system registers this new via object, we need to add this via object into the board object, by using the board object's `AddPCBObject` method the via object parameter. Once this method has been invoked, this new `Via` object is now a valid object on the current PCB document.

To actually remove objects from the database, invoke the `Board.RemovePCBObject` method and pass in the parameter of a reference to an actual PCB object.

See also



See the examples in the `\Developer Kit\Examples\PCB\PCB Objects` folder.



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\PCB` folder.

Creating/Deleting objects and updating the PCB Editor's Undo system

The simple CreateAVia addon above example does not refresh the Undo system in the PCB editor, and to do this, you will need to employ the `SendMessageToRobots` method to send messages into the PCB editor server to inform the datastructure that new PCB objects have been added to the PCB document.

One Undo sequence

The sequence is as follows for a one big undo operation

- Invoke the `PCBServer.PreProcess` method which initializes the robots in the PCB server once
- For each new object added, send a `PCBM_BoardRegistration` **message**
- Invoke the `PCBServer.PostProcess` method which cleans up the robots in the PCB server once

Multiple Undos sequence

The sequence is as follows for multiple step undo operation, ie for each PCB creation, do the four steps

1 Invoke the `PreProcess` method which Initializes the Robots in the PCB server

2 Add a new object

3 Send a `PCBM_BoardRegistration` **message**

4 Invoke the `PostProcess` method which cleans up the robots in the PCB server

Repeat Steps 1-4 for each PCB object creation.

Adding two fill objects and refreshing the Undo system in PCB editor

```
Board := PCBServer.GetCurrentPCBBoard;
If Board = Nil Then Exit;

PCBServer.PreProcess;

Fill      := PCBServer.PCBObjectFactory(eFillObject) as IPCB_Fill;
Fill.X1Location := MilsToCoord(4000);
Fill.Y1Location := MilsToCoord(4000);
Fill.X2Location := MilsToCoord(4400);
Fill.Y2Location := MilsToCoord(4400);
Fill.Rotation   := 0;
Fill.Layer     := eTopLayer;
Board.AddPCBObject(Fill1);

PCBServer.SendMessageToRobots(
    Board.I_ObjectAddress,
    c_Broadcast,
```

```

PCBM_BoardRegistration,
Fill1.I_ObjectAddress);

Fill2      := PCBServer.PCBObjectFactory(eFillObject);
Fill.X1Location := MilsToCoord(5000);
Fill.Y1Location := MilsToCoord(3000);
Fill.X2Location := MilsToCoord(5500);
Fill.Y2Location := MilsToCoord(4000);
Fill.Rotation  := 45;
Fill.Layer     := eTopLayer;
Board.AddPCBObject(Fill2);

PCBServer.SendMessageToRobots(
    Board.I_ObjectAddress,
    c_Broadcast,
    PCBM_BoardRegistration,
    Fill2.I_ObjectAddress);

PCBServer.PostProcess;

```

See also

See the examples in the \Developer Kit\Examples\PCB\PCB UndoRedo folder.



Check out the script examples in the \Examples\Scripts\DelphiScript Scripts\PCB\ folder.

Removal of objects example

```

PCBServer.PreProcess;

Try
    Track := Iterator.FirstPCBObject As IPCB_Track;
    While Track <> Nil Do
        Begin
            OldTrack := Track;
            Track := Iterator.NextPCBObject As IPCB_Track;

            CurrentPCBBoard.RemovePCBObject(OldTrack);
            PCBServer.SendMessageToRobots(Board.I_ObjectAddress,
                c_BroadCast,
                PCBM_BoardRegistration,

```

```
OldTrack.I_ObjectAddress);  
  
End;  
Finally  
    CurrentPCBBoard.BoardIterator_Destroy(Iterator);  
End;  
PCBServer.PostProcess;
```

See also



See the example in the `\Developer Kit\Examples\PCB\PCB UndoRedo` folder.



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\PCB\` folder.

Modifying PCB design objects

To modify PCB objects on a current PCB document, you will need to invoke certain methods in a certain order to ensure all the Undo/Redo system is up to date when a PCB object's attributes have been modified programmatically.

The Modification of a PCB object sequence is as follows

- Invoke the `PreProcess` method and Initialize the robots in the PCB server
- Send a `PCBM_BeginModify` message
- Modify the PCB object
- Send a `PCBM_EndModify` message
- Invoke the `PostProcess` method to clean up the robots in the PCB server

Changing PCB object's attributes example

```
Procedure ModifyObject;  
Begin  
    PCBServer.PreProcess;  
  
    PCBServer.SendMessageToRobots(Fill.I_ObjectAddress, c_Broadcast, PCBM_BeginModify  
,c_NoEventData);  
  
    Fill.Layer := eBottomLayer;  
  
    PCBServer.SendMessageToRobots(Fill.I_ObjectAddress, c_Broadcast, PCBM_EndModify ,  
        c_NoEventData);  
  
    //Clean up the robots in the PCB editor.  
  
    PCBServer.PostProcess;  
  
End;
```

Notes

When you change the properties of a PCB object on a PCB document, it is necessary to employ the `PCBServer.SendMessageToRobots` function to update the various subsystems of the PCB system such

as the Undo/Redo system and setting the document as dirty so the document can be saved. Look for `SendMessageToRobots` method of the `IPCB_ServerInterface` in the reference online help.

Remember, the PCB API only can be used to work on a currently open PCB document in the Altium Designer, if you need to fetch specific PCB documents, you will need extra routines to open and display a document and the `\Developer Kit\Examples\DXP\Server Information` source code for details.

See also



See examples in the `\Developer Kit\Examples\PCB\PCB UndoRedo` folder.



Check out the script examples in the `\Examples\Scripts\DelphiScript Scripts\PCB` folder.

PCB interactive feedback using the mouse

To monitor the mouse movement and clicks from your script, the `IPCB_Board` document interface has several interactive feedback methods;

- `GetObjectAtCursor`
- `GetObjectAtXYAskUserIfAmbiguous`
- `ChooseRectangleByCorners`
- `ChooseLocation`

The `GetObjectAtCursor` returns you the interface of an object where the PCB system has detected that this object has been clicked upon.

The `GetObjectAtXYAskUserIfAmbiguous` method does the same function as the `GetObjectAtCursor` except that if there are objects occupying the same region on the PCB document then this method prompts you with a dialog with a list of objects to choose before returning you the object interface. You have the ability to control which objects can be detected and which layers can be detected and what type of editing action the user has been doing.

The `ChooseRectangleByCorners` method prompts you to choose the first corner and the final corner and then the X1,Y1, X2 and Y2 parameters are returned.

The `ChooseLocation` method prompts you to click on the PCB document and then the X1,Y1 coordinates of the mouse click are returned.

Interactive Methods

`GetObjectAtCursor`

`GetObjectAtXYAskUserIfAmbiguous`

`ChooseRectangleByCorners`

`ChooseLocation`

Using the Altium Designer RTL

ChooseRectangleByCorners Example in a Server Project

Var

```
Board          : IPCB_Board;
SpatialIterator : IPCB_SpatialIterator;
X1,Y1,X2,Y2    : TCoord;
ASetOfLayers   : TLayerSet;
ASetOfObjects  : TObjectSet;
Track          : IPCB_Track;
TrackCount     : Integer;
```

Begin

```
(* A spatial iterator queries PCB data within X1,Y1,X2,Y2 constraints. *)
Board := PCBServer.GetCurrentPCBBoard;
If Board = Nil Then Exit;

TrackCount := 0;
If Not (Board.ChooseRectangleByCorners('Please select the first corner',
                                       'Please select the final corner',
                                       X1,Y1,X2,Y2)) Then Exit;

ASetOfLayers := [eTopLayer,eBottomLayer];
ASetOfObjects := [eTrackObject];

SpatialIterator := Board.SpatialIterator_Create;
SpatialIterator.AddFilter_Area(X1,Y2,X2,Y2);
SpatialIterator.AddFilter_ObjectSet(ASetOfObjects);
SpatialIterator.AddFilter_LayerSet(ASetOfLayers);
Track := SpatialIterator.FirstPCBObject as IPCB_Track;
While Track <> Nil Do
Begin
    TrackCount := TrackCount + 1;
    Track := SpatialIterator.NextPCBObject as IPCB_Track;
End;
Board.SpatialIterator_Destroy(SpatialIterator);
Showinfo(InTtoStr(TrackCount));
End;
```

See also



See examples in the \Developer Kit\Examples\PCB\PCB Iterators folder.



Check out the script examples in the \Examples\Scripts\DelphiScript Scripts\PCB\ folder.

Activating PCB API

There are situations when Altium Designer is in a blank editing session, that is when no PCB documents have not loaded yet, the PCB API is not active.

You will then need to manually start the server to have access to the PCB API. Starting up the PCB server to activate the PCB API is done by invoking the `Client.StartServer` method.

Starting up the PCB server in DXP

```
If PCBAPI_GetCurrentEditorWindow = 0 Then
Begin
    Client.StartServer('PCB');
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;
End;
```

Integrated Library interfaces

A schematic design is a collection of components which have been connected logically. To test or implement the design it needs to be transferred to another modelling domain, such as simulation, PCB layout, Signal Integrity analysis and so on.

Each domain needs some information about each component, and also some way to map that

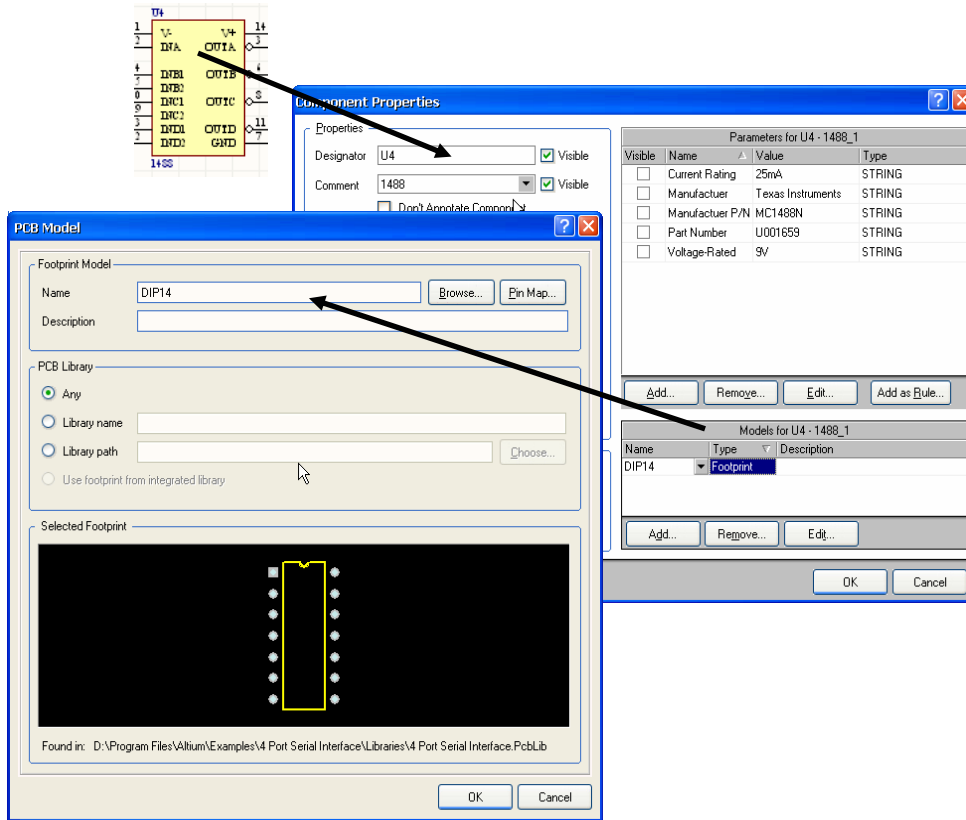


Figure 8: This diagram shows a DIP14 PCB footprint is linked to a 1488 type schematic component. The PCB model dialog represents the PCB Library Model Editor which provides the capability to

information to the pins of the schematic component. Some of the domain information resides in model files, the format of which is typically pre-defined, examples of these includes IBIS, MDL and CKT files. Some of the information does not reside in the model files for example the spice pin mapping and netlist data.

Models

Each schematic component can have models from one or more domains. A schematic component can also have multiple models per domain, one of which will be the current model for that domain.

A model represents all the information needed for a component in a given domain, while a datafile entity (or link) is the only information which is in an external file. See diagram below for a relationship between a Schematic component and its models. A model can be represented by external data sources called data file links. For example, pins of a component can have links to different data files, as for signal integrity models. We will consider each model type in respect to the data file links for the main editor servers supported in Altium Designer.

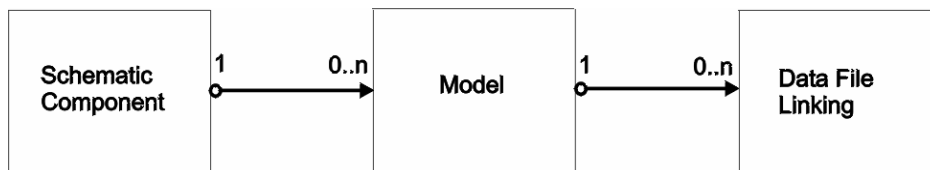


Figure 9: Schematic component, its models and its sub data file links.

For the PCB footprints, the model and the data file are both the same.

With the simulation models, you can have a simulation model which is a 4ohm resistor for example, there is a simulation model here, but there is no information is coming from an external file, therefore, a no external file is needed for this as the resistor model is built from spice. This is the case where you have a model with no data file entity. Thus the parameters are used for these types of simulation models that don't have data file links.

With signal integrity models, it can have information required for each pin. If we used IBIS datafiles, not the Altium Designer's central database, then each signal integrity model would then have multiple data files, each one for each type of pin.

Note that a model can also be called an implementation. For each implementation there are parameters and data file links.

There are different types of libraries in Altium Designer – normal standalone libraries like PCB Libraries and Schematic Libraries and another type called an integrated library which contains different libraries bundled together.

Using Integrated Library Interfaces

The `RT_IntegratedLibrary` unit provides a list of interfaces which you can use to have access to some of the features of an integrated library open in Altium Designer.

The `IModelEditor` interface represents the Model Editor hosted by a server which normally has a dialog that displays data about the model properties in Altium Designer.

The `IModelDatafile` interface represents the data file that is associated with a model. Each model can have multiple data files (different representations of the same model type). This interface is used within the `IServerModel` interface.

The `IServerModel` interface represents the model set up by the server to be used by the integrated library server.

The `IModelType` interface represents the model domain. Each model domain has at least one data file type or entity type.

The `IModelDatafileType` interface represents the data file as one of the models for that model type.

Using the Altium Designer RTL

The `IModelTypeManager` interface is like a repository of available model types in Altium Designer. The `IMP` files are collected and processed by this manager. This manager uses `IModelType` and `IModelDataType` interfaces.

The `IIntegratedLibraryManager` interface is the integrated library manager that can retrieve or set many parameters associated with schematic components and its models. This interface is implemented in the Integrated Library server and the functionality can be used to extract the information you require.

The Integrated Library server manages `IModelType`, `IModelDataFileType`, `IModelTypeManager` and `IIntegratedLibraryManager` interfaces for you. You can however still use them to find out the current state of these interfaces used within Altium Designer. The `IModelEditor`, `IModelDataFiles` and `IServerModel` interfaces are needed when you need to build a server that hosts a model editor that adds new model types in Altium Designer.

To have access to the Integrated Library manager, you simply invoke the `IntegratedLibraryManager` function from the `RT_IntegratedLibrary` unit, and to have access to the interfaces of the Model Type Manager, you simply invoke the `ModelTypeManager` function from the same unit.

Note there is a source code example that extracts data using the Integrated Library server's interfaces. It is located in the `\Developer Kit\Examples\DXP\IntLib Interfaces` folder.

Building a model editor in a server project

There is a provision for Server Developers to build their own model editors for their document editor servers to extend the range of models the schematic components can support. To build your own Model Editor, you would need to define the source code implementations for the `IModelEditor` and its associated `IModelDatafile` and `IServerModel` interfaces.

The `IServerModel` interface and the implementation configuration file (with an `.IMP` extension) are taken in by the Integrated Library server when you plug in the server that implements the `IModelEditor` interface.

A model implementation configuration file is needed that outlines the model type (particular model domain such as PCB, Simulation, 3D models etc), and the model libraries/files for that model domain.

The `Description` clause within the `[ImplementationEditor]` section in this implementation file gets listed in the **Add New Model** dialog (accessed when you click the **Add** button from the **Component Properties** dialog).

The model implementation configuration files for PCB editor and Simulation editor are as follows:

PCB footprint implementation script (advpcb.imp)

```
[ImplementationEditor]
ModelType                = PCBLIB
Module                   = AdvPCB.dll
Description               = Footprint
ServerName                = PCB
Process                   = PCB_RunSchModelEditor
```

```
[Datafiles]
Count                = 1
DatafileKind0       = PCBLib
DatafileDescription0 = Footprint Library
DatafileEntityType0 = Footprint
DatafileExtensionFilter0 = *.PCBLIB
```

Simulation model implementation script (advsim.imp)

```
[ImplementationEditor]
ModelType            = SIM
Module               = AdvSim.dll
Description          = Simulation
Process              = SimAPI_RunImplEditor
ServerName           = Sim
```

```
[Datafiles]
Count                = 2
DatafileKind0       = MDL
DatafileDescription0 = Sim Model File
DatafileEntityType0 = Sim Model
DatafileExtensionFilter0 = *.MDL
DatafileKind1       = CKT
DatafileDescription1 = Sim Subcircuit File
DatafileEntityType1 = Sim Subcircuit
DatafileExtensionFilter1 = *.CKT
```



For detailed information on Integrated Library API, refer to the Integrated Library API Reference online help

Altium Designer RTL Online Help

The information from the Altium Designer RTL reference is available for scripts and server projects. The information specific to the server development only is covered in the **Altium Designer RTL Reference for Servers** document and has details on the components of a server.



Refer to the **Altium Designer RTL Online** help in Altium Designer.

Revision History

Date	Version No.	Revision
20-Sept-2005	V1.0	New document for Altium Designer
15-Dec-2005	V1.1	Updated for Altium Designer 6
20-Dec-2006	V1.2	Updated for Altium Designer 6.6 and extra information Nets and Connectivity added.
28-Dec-2006	V1.3	Updated wording.

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.