DXP RTL Reference

Summary

Technical Reference

TR0126 (v1.0) December 01, 2004

This reference manual describes the DXP Run Time Library Reference.

The scripting system implements the DXP Run Time Library which is a very large Application Programming Interface (API). Since the DXP application is written in Borland Delphi, thus all the functions and objects are defined using the Borland Delphi language set, however you can use one of the scripting language sets to have access to the DXP Object Model or Borland Delphi Run Time Library.

DXP Run Time Library Reference

Language Sets supported by the Scripting system:

- DelphiScript
- EnableBasic
- VBScript
- JScript
- TCL

This DXP Run Time Library implements the following sections:

- The DXP Object Model (which is composed of Client, Server, PCB, Nexar, Schematic, WorkSpace Manager and Integrated Library Object Models)
- Components from the Tool Palette (which is based on Borland Delphi's Visual Component Library)
- Routines and objects exposed from Borland Delphi units (in Supported Borland Delphi Units section)
- Routines and objects exposed from DXP RTL units (in General DXP RTL Reference)
- Server Process routines (in Server Process Parameters Reference).

Please note that the scripting system implements a subset of the Borland Delphi version 6 run time library. Refer to the Supported Borland Delphi Units section for more information.

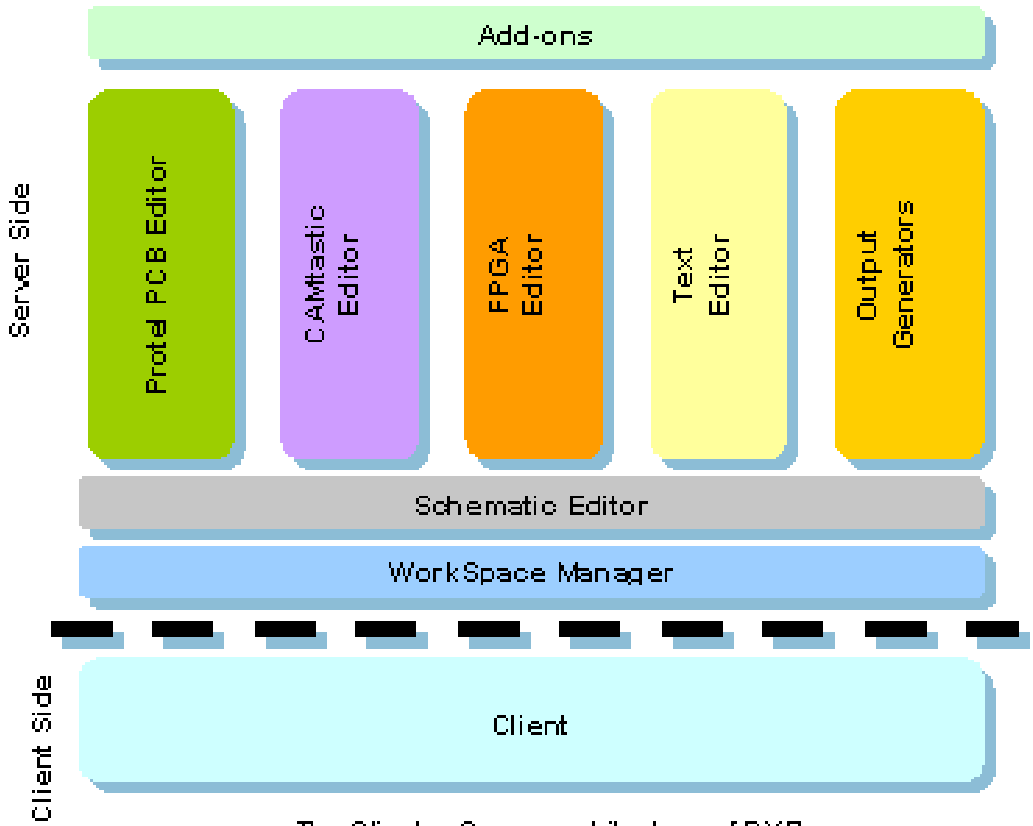
Also included are the following sections in the DXP Run Time Library Reference:

- DXP Object Models
- Client Server API Reference
- Integrated Library API Reference

- Nexar API Reference
- PCB API Reference
- Schematic API Reference
- Workspace Manager API Reference
- General API RTL Reference
- Supported Borland Delphi Units.

DXP Object Models

The DXP application is a large system as shown by the diagram below which illustrates the architecture of this DXP system. The DXP system is composed of a single Client executable along with plugged in servers.



The Client – Server architecture of DXP

The Client executable deals with actions generated by the user of the DXP application. The servers provide specialized functionality depending upon the task requested by the user. The Schematic server and PCB server are two main document editor servers used for the design process in DXP and these Schematic and PCB servers have their own document types (design and library documents).

DXP Object Models

The DXP Application supports PCB, Schematic, Workspace Manager, Integrated Library Manager, Model Type Manager and Client Object Models which makes it possible to write scripts that manipulate objects in DXP.

All scripting languages supported by the DXP application such as DelphiScript, EnableBasic, JScript and VB script have access to DXP object models. A Properties Methods Events (PME) dot-notation model is used in DXP and this model can be used the same way for any scripting language supported by DXP.

The DXP Object Model has one Client Object model and several server object models:

- • Client Object Model
- • Integrated Library Object Model
- • WorkSpace Manager Object Model
- • Schematic Object Model
- • PCB Object Model
- • Nexar Object Model.

The **Client** module and the **servers** plugged in DXP is exposed through the use of Interfaces. To use DXP Object models in your script, you will need to invoke the function for this particular object model. Usually you need to have the specific document open first before you can run the script that is written to deal with that server document.

Methods and Properties of Object Interfaces

For each object interface in DXP, 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 method is a procedure or function that is invoked from its interface.

• 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.

PCB Object Model Example

```
Var
    Board : IPCB_Board;
    Via   : IPCB_Via;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
    // etc
```

This example can be used in DelphiScript, JScript, EnableBasic, VBScript and TCL because DXP supports the Object Model dot notation and the Properties Methods Events method.

Notes

When you are using DXP interfaces in your scripts, it means these interfaces require access to the specific design document. You cannot just run these type of scripts when you are in the Text Editor environment within DXP. For example, using PCB interfaces in your script, means you have to run the script on an opened PCB document in DXP.

Otherwise, if you have developed scripts that don't use the DXP Object Interfaces, then you can run the script from within the Text Editor environment in DXP.

There are simple script examples for each object model exposed by Client, Workspace Manager, Schematic and PCB servers in **\Examples\Scripts** folder

Client Server API Reference

The Client/Server Application Programming Interface reference covers interfaces for Client/Server objects in the Client/Server Object Model.

What are Interfaces?

Each method in the interface is implemented in the corresponding class. Interfaces are declared like classes but cannot be directly instantiated and do not have their own method definitions. Each interface, a class supports is actually a list of pointers to methods. Therefore, each time a method call is made to an interface, the interface actually diverts that call to one of it's pointers to a method, thus giving the object that really implements it, the chance to act.

The Client/Server interfaces exist as long there are associated existing objects in memory, thus when writing a script, you have the responsibility of checking whether the interface you wish to query exists or not before you proceed to invoke the interface's methods.

You can obtain the **IClient** interface object by calling the **Client** function in a script and execute methods from this function directly for example calling this **Client.OpenDocument('Text',FileName);** method is valid.

The empty workspace or the shell of DXP is the top level client window. The client module is represented by its **IClient** interface object, and you can have the ability to take a peek into a loaded server's data structures through this **IClient** interface. Servers are represented by its **IServerModule** interfaces which are plug in modules in DXP.

Example

```
Var
    ReportDocument : IServerDocument;
Begin
    If Client = Nil Then Exit;
    // Opens and shows a text file in DXP
    ReportDocument  := Client.OpenDocument('Text',FileName);
    If ReportDocument <> Nil Then
          Client.ShowDocument(ReportDocument);
End;
```

Main Client's interfaces

- **ICommandLauncher** (deals with process launchers)
- **IServerDocumentView** (deals with panels or server documents)
- **IProcessControl** (determines the level of stacked processes)
- **IGUIManager** (deals with the User interface of DXP, the locations and state of panels)
- **IServerModule** (deals with loaded servers in DXP)
- **INotification** (broadcast or dispatch notification messages to servers or to a specified server)

Main Server Interfaces

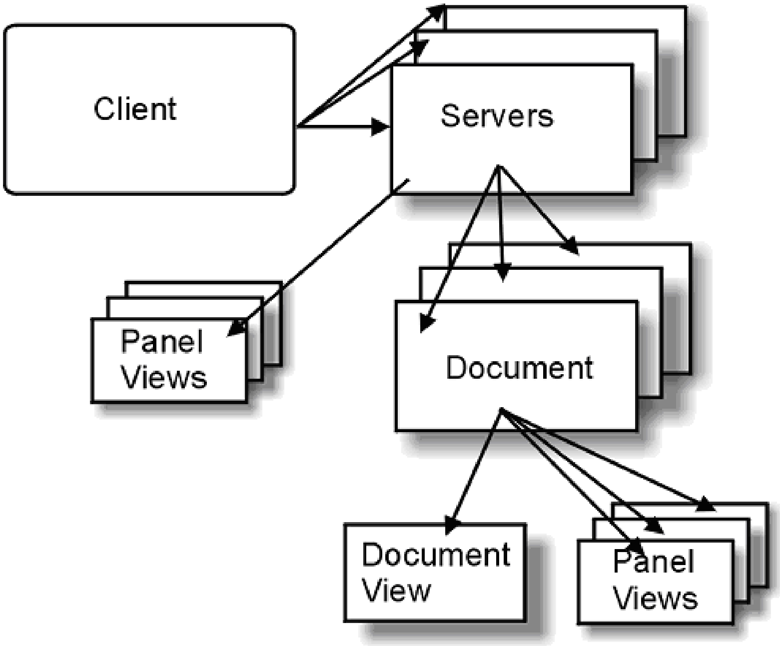
The **IServerModule** interfaces represent loaded servers in DXP. To obtain the server module and invoke the methods from this module, you can use the ModuleName property with the name of the server passed in, and if alls well, you can then launch the process for that server.

Script Examples

There are Client / Server script examples in the **\Examples\Scripts\DXP** folder

Using Client / Server interfaces

Central to the DXP architecture is the concept of a single client module as the controller collaborating with loaded servers. Each server manages their own documents. This is a big picture view of the Design Explorer – there is one Client executable and several servers as loaded dynamic library linked modules as shown in the diagram below.



Client Interfaces

The **IClient** interface represents the Client subsystem of the Design Explorer application and the Client subsystem manages the commands (pre packaged process launchers), process depths and documents of loaded servers. Every server module loaded in Design Explorer is linked to the client subsystem of DXP, so you have access to the specific loaded documents in DXP.

The client module maintains a list of loaded servers, that is this module stores many lists of opened server documents, loaded server processes, loaded server resources.

You can obtain the **IClient** interface object by calling the **Client** function in a script and execute methods from this function directly for example calling this **Client.OpenDocument('Text',FileName);** method is valid.

The **Client** function returns you the **IClient** interface object.

Client's interfaces

- **ICommandLauncher** (deals with process launchers)
- **IServerDocumentView** (deals with panels or server documents)
- **IProcessControl** (determines the level of stacked processes)
- **IGUIManager** (deals with the User interface of DXP, the locations and state of panels)
- **IServerModule** (deals with loaded servers in DXP)
- **INotification** (broadcast or dispatch notification messages to servers or to a specified server)

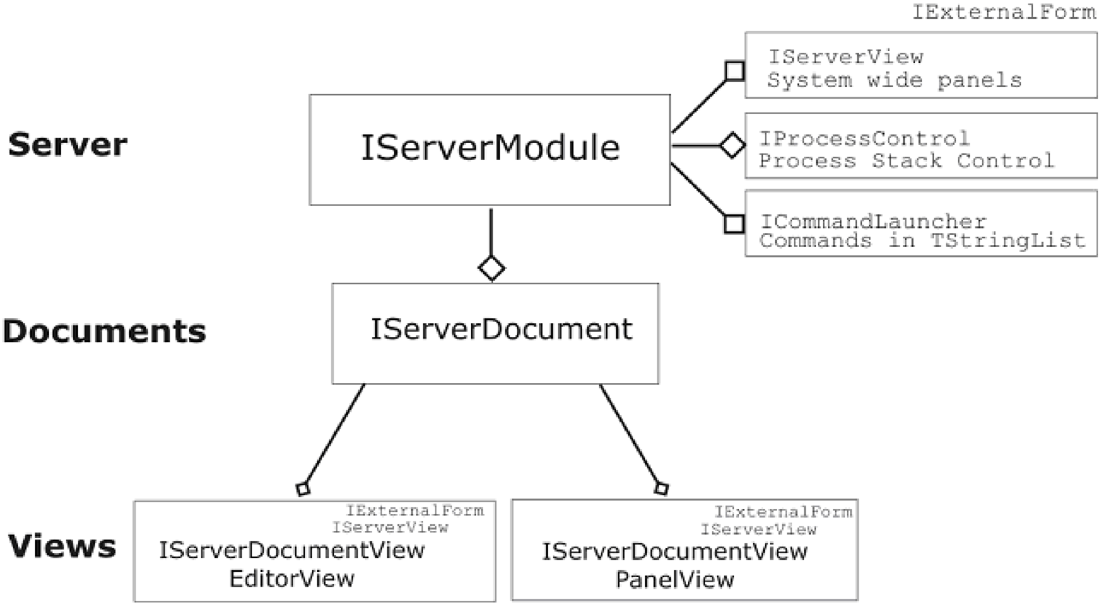
Server Interfaces

The **IServerModule** interfaces represent loaded servers in DXP. To obtain the server module and invoke the methods from this module, you can use the ModuleName property with the name of the server passed in, and if alls well, you can then launch the process for that server. An example is shown below;

Example

```
If StringsEqual(ServerModule.ModuleName,'TextEdit') Then
Begin
    ServerModule.CommandLauncher.LaunchCommand('TextEdit:MoveCursorToTopOfDo
cument',
                                                       Nil,0,ServerDocument.View[0]
);
End;
```

The relationship of a server and its documents



An **IServerModule** interface has the following interfaces:

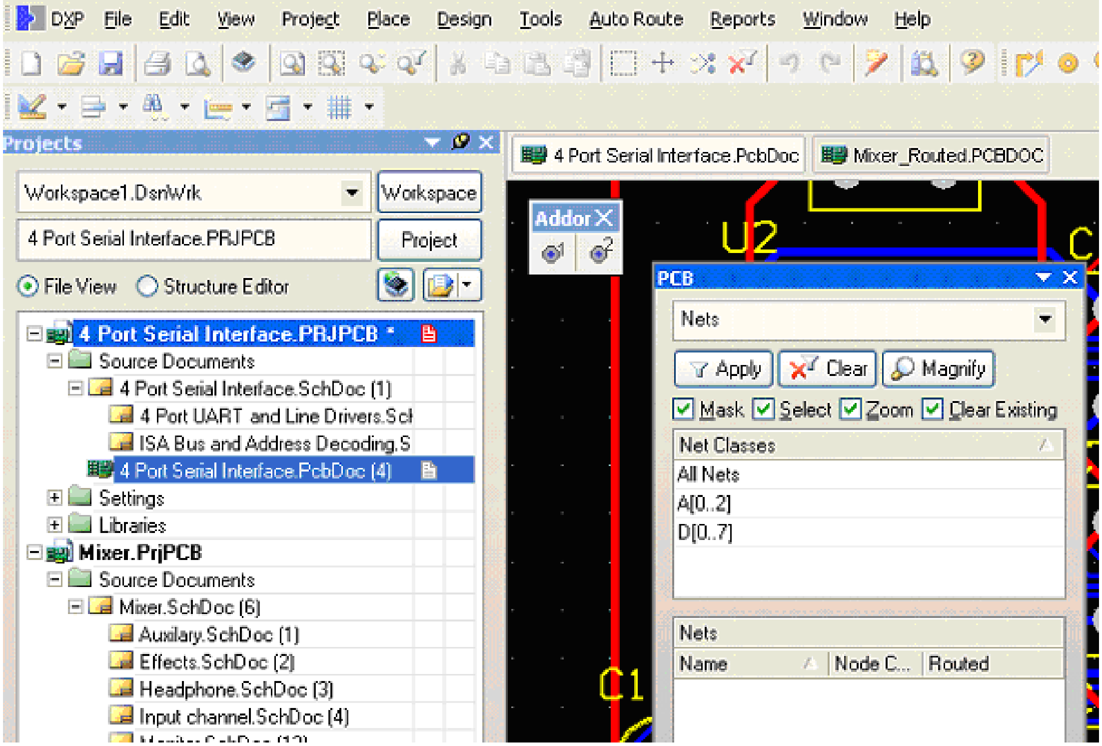
- **ICommandLauncher** (deals with a server's processes table)
- **IServerDocument** (represents a loaded design document in DXP)
- **IServerView** (represents a panel that can have a view of the DXP system)
- **IServerDocumentView** (deals with a document view (either the document window or panel window))
- **IExternalForm** (represents a DXP aware Delphi form either as a document form or a panel form. These forms are wrapped by the IServerDocumentView or IServerView interface object. This IExternalForm interface object has low level methods such as resizing and displaying the form)
- **IProcessControl** (represents the level of stacked processes for this focussed server document)
- **INotification** represents the system notifications from the Client system and all server modules receive these notifications. There is an ability to handle a notification and take it from there. Documents and associated panels can be synchronized through the use of notifications as well).

Servers Documents and Panels Interfaces in DXP

The concept of documents and panels are central to understanding how servers work in DXP. The servers manage their own panels and documents. DXP has access to the currently active panels and documents and manage the size and position of these panels and documents. Basically there are two types of panels – panels associated with documents and standalone panels such as the Messages panel.

Each server loaded in DXP store their own documents (there can be different document kinds, for example PCB and PCB library documents) and each document has its corresponding panel for example the PCB panel and the PCB document. Now, a server has its own document container which stores the same document kind, thus for different document kinds, there are document containers for each document kind. Each document container stores views of documents and associated panels along with standalone panels if any.

In the screen shot below, there are two PCB documents open in DXP with the Projects panel on the left and a floating PCB panel visible on top of a PCB document. The add-on's floating toolbar is visible as well.



We will consider the main interfaces used to represent the servers, documents and panels in the DXP as shown in figure above.

The Client system within DXP has access to an active document and panel views directly, therefore a panel's boundaries and visibility can be set programmatically via the **IClient** and its composite **IClientGUIManager** interfaces. The Client and the Server module has its own Command Launcher functionality which is used to execute a server process. This is encapsulated as the **ICommandLauncher** interface.

The Work-space manager server in DXP has several **IServerView** interfaces – Files panel, Projects panel, Messages panel, Navigator panel, Errors panel, Differences panel, To Do panel and so on.

There are three main interfaces, **IServerModule**, **IServerView** and **IServerDocumentView** interfaces that we will go over in respect to the figure above.

IServerModule interfaces

Each loaded server in DXP is encapsulated by the **IServerModule** interface, so from figure, there is a **IServerModule** interface for the PCB editor server, another one for the Work-space Manager server, one for the Help Advisor server, and finally another interface for the add-on for the PCB editor and so on.

IServerView interfaces

An **IServerView** interface points to a global (standalone) panel that can deal with multiple types of documents, for example the Projects panel. This Projects panel is controlled by the Work-space manager server and is represented by the **IServerView** interface.

IServerDocumentView interfaces

A PCB document has an editor (document) view and three panel views (PCB Navigator, Expression Filter and Object Inspector panels) all represented by the same **IServerDocumentView** interface. Therefore in the figure, there are eight **IServerDocumentView** interfaces representing the two PCB documents and the two sets of three PCB panels (the Expression Filter as the List panel, Object Inspector as Inspector panel, and the PCB Navigator as the PCB panel). Note that only the PCB panel is displayed but all panels are active in computer's memory.

Client Server Interfaces

Client/Server Object Model

The major interfaces that are used in the client – server architecture within DXP are:

IClient shell and its interfaces:

- ICommandLauncher (deals with client's process launchers table)
- IProcessLauncher (deals with launching a server process from the client)
- IServerDocumentView (deals with panels or server documents)
- IProcessControl (determines the level of stacked processes)
- IGUIManager (deals with the User interface of DXP, the locations and state of panels)
- IServerModule (deals with a loaded server in DXP)
- INotification (Client can broadcast or dispatch notification messages to servers or to a specified server)

DXP's configuration interfaces:

- IServerRecord (collects servers information at DXP's start up – not loaded servers)
- IServerWindowKind (determines which document kinds open in DXP)
- IServerProcess (contains the information of a current server process)

IServerModule interfaces represent loaded servers in DXP.

An IServerModule interface has the following interfaces:

- ICommandLauncher (deals with a server's processes table)
- IServerDocument (represents a loaded design document in DXP)
- IServerView (represents a panel that can have a view of the DXP system)
- IServerDocumentView (deals with a document view (either the document window or panel window))
- IExternalForm (represents a DXP aware Delphi form either as a document form or a panel form. These forms are wrapped by the IServerDocumentView or IServerView interface object. This IExternalForm interface object has low level methods such as resizing and displaying the form)
- IProcessControl (represents the level of stacked processes for this focussed server document)
- INotification receive system notifications from the Client system and all server modules receive these notifications. There is an ability to handle a notification and take it from there. Documents and associated panels can be synchronized through the use of notifications as well).

IClient interface

Overview

The **IClient** interface represents the Client subsystem of the Design Explorer application and the Client manages the commands (pre packaged process launchers), process depths and documents. The every server module loaded in Design Explorer has hooks to the single client executable subsystem, so you have access to the specific documents of any server loaded in DXP and launch server commands.

The **IClient** shell and its interfaces;

- ICommandLauncher (deals with process launchers)
- IProcessLauncher (deals with launching a server process)
- IServerDocumentView (deals with panels or server documents)
- IProcessControl (determines the level of stacked processes)
- IGUIManager (deals with the User interface of DXP, the locations and state of panels)
- IServerModule (deals with loaded servers in DXP)
- INotification (broadcast or dispatch notification messages to servers or to a specified server)

You can obtain the **IClient** interface object by calling the **Client** function directly in your script.

IClient methods

```
AddServerView
ApplicationIdle
BeginDisableInterface
BeginDocumentLoad
BeginRecoverySave
BroadcastNotification
CanServerStarted
CloseDocument
DispatchNotification
EndDisableInterface
EndDocumentLoad
EndRecoverySave
HideDocument
GetApplicationHandle
GetCommandLauncher
GetCount
GetCurrentView
GetDocumentByPath
```

IClient Properties

```
ApplicationHandle
MainWindowHandle
CommandLauncher
ProcessControl
CurrentView
GUIManager
NavigationSystem
TimerManager
Count
ServerModule
ServerModuleByName
```

```
GetDocumentKindFromDocumentPath
GetDefaultExtensionForDocumentKind
GetEncryptedTechnologySets
GetGUIManager
GetMainWindowHandle
GetNavigationSystem
GetOptionsSetCount
GetOptionsSet
GetOptionsSetByName
GetProcessControl
GetPanelInfoByName
GetRealMainWindowHandle
GetServerModule
GetServerModuleByName
GetServerNameByPLID
GetServerRecordCount
GetServerRecord
GetServerRecordByName
GetServerViewFromName
GetTimerManager
GetWindowKindByName
IsDocumentOpen
IsQuitting
InRecoverySave
LastActiveDocumentOfType
LicenseInfoStillValid
OpenDocument
QuerySystemFont
RemoveServerView
RegisterNotificationHandler
SetCurrentView
ShowDocument
ShowDocumentDontFocus
StartServer
StopServer
UnregisterNotificationHandler
```

IClient interface methods

AddServerView method

(IClient interface)

Syntax

```
Procedure AddServerView (AView : IServerView);
```

Description

Adds a document view especially a custom panel other than the standard server panel in the Client object within DXP.

See also

IServerView interface

ApplicationIdle method

(IClient interface)

Syntax

```
Procedure ApplicationIdle;
```

Description

When the ApplicationIdle method is invoked, the procedure puts the DXP in a mode where DXP has a chance to process

Window and DXP messages.

BeginDisableInterface method

(IClient interface)

Syntax

```
Procedure BeginDisableInterface;
```

Description

These BeginDisableInterface and EndDisableInterface methods are invoked when the User Interface of Client need to be disabled, for example there might be extensive processing going on, and you do not want the user's intervention. This is a DXP wide method.

See also

EndDisableInterface method

BeginDocumentLoad method

(IClient interface)

Syntax

```
Procedure BeginDocumentLoad;
```

Description

The BeginDocumentLoad and EndDocumentLoad procedures are used to load a group of documents in DXP.

Example

```
Client.BeginDocumentLoad;
ServerDocument1 := Client.OpenDocument('Text',FileName1);
ServerDocument2 := Client.OpenDocument('Text',FileName2);
ServerDocument3 := Client.OpenDocument('Text',FileName3);
Client.EndDocumentLoad(True);
```

See also

EndDocumentLoad method

BeginRecoverySave method

(IClient interface)

Syntax

```
Procedure BeginRecoverySave;
```

Description

The BeginRecoverySave and EndRecoverySave properties can be used to suppress the client notification of document name changes when doing a backup of a current design document in DXP. To check if the recovery save is in progress, invoke the InRecoverySave method.

See also

EndRecoverySave method

InRecoverySave method

BroadcastNotification method

(IClient interface)

Syntax

```
Procedure BroadcastNotification     (ANotification : INotification);
```

Description

This procedure broadcasts a notification message in DXP where all active design documents / servers have an opportunity to respond. A BoardcastNotification is a DispatchNotification (Nil, ANotification); There are five types of Notification interfaces; ISystemNotification, IDocumentNotification, IDocumentFormNotification, IViewNotification and IModuleNotification.

See also

DispatchNotifiaction method

INotification interface

Client_CanServerStarted method

(IClient interface)

Syntax

```
Function  CanServerStarted (AModuleName : PChar) : LongBool;
```

Description

This function checks if a server module can be loaded in DXP. Use this before invoking the StartServer function.

CloseDocument method

(IClient interface)

Syntax

```
Procedure CloseDocument(ADocument : IServerDocument);
```

Description

This procedure fetches the IServerDocument parameter to close the specified document (if it is loaded and opened in DXP already). Note the document is not removed from DXP, that is, the document still exists on the **Projects** panel for example.

See also

OpenDocument method

Count property

(IClient interface)

Syntax

```
Property  Count : Integer Read GetCount;
```

Description

This property returns the number of active servers in a current session of DXP. Use this property in conjunction with the ServerModule property to fetch Server Module interfaces.

See also

GetCount method

IServerModule interface

DispatchNotification method

(IClient interface)

Syntax

```
Procedure DispatchNotification      (AServerModule : IServerModule;
ANotification : INotification);
```

Description

This procedure dispatches a notification message to the targeted server in DXP. There are four types of Notification interfaces; IDocumentNotification, IDocumentFormNotification, IViewNotification and IModuleNotification.

See also

INotification interface

EndDisableInterface method

(IClient interface)

Syntax

```
Procedure EndDisableInterface;
```

Description

These BeginDisableInterface and EndDisableInterface methods are invoked when the User Interface of Client need to be disabled, for example there might be extensive

processing going on, and you do not want the user's intervention. This is a DXP wide method.

See also

BeginDisableInterface method;

EndDocumentLoad method

(IClient interface)

Syntax

```
Procedure EndDocumentLoad(AShow : LongBool);
```

Description

The **BeginDocumentLoad** and **EndDocumentLoad** procedures are used to load a group of documents in DXP.

Example

```
Client.BeginDocumentLoad;
ServerDocument1 := Client.OpenDocument('Text',FileName1);
ServerDocument2 := Client.OpenDocument('Text',FileName2);
```

```
ServerDocument3 := Client.OpenDocument('Text',FileName3);
Client.EndDocumentLoad(True);
```

See also

BeginDocumentLoad method

EndRecoverySave method

(IClient interface)

Syntax

```
Procedure EndRecoverySave;
```

Description

The **BeginRecoverySave** and **EndRecoverySave** methods can be used to suppress the client notification of document name changes when doing a backup of a current design document in DXP.

To check if the recovery save is in progress, invoke the **InRecoverySave** method.

See also

BeginRecoverySave method

InRecoverySave method

GetApplicationHandle method

(IClient interface)

Syntax

```
Function  GetApplicationHandle : Integer;
```

Description

You can use the application handle into server code if dialogs need to be created dynamically from your server and so that when a dialog that appears on Design Explorer will inherit Design Explorer's icon and appear as one whole application on the task bar. This ApplicationHandle property can be passed as a parameter for the create constructor of the dialog. The GetMainWindowHandle function is its equivalent.

See also

GetMainWindowHandle method

ApplicationHandle property

GetCommandLauncher method

(IClient interface)

Syntax

```
Function  GetCommandLauncher   : ICommandLauncher;
```

Description

This function fetches the ICommandLauncher interface which represents Client's process launcher which can be used to launch a server process and its parameters. See the IProcessLauncher interface as well.

See also

ICommandLauncher interface

IProcessLauncher interface

GetCount method

(IClient interface)

Syntax

```
Function  GetCount : Integer;
```

Description

This method returns the number of active (loaded) servers in a current session of DXP. Use this method (or the Count property) in conjunction with the **ServerModule** property to fetch Server Module interfaces.

See also

Count property

GetCurrentView method

(IClient interface)

Syntax

```
Function  GetCurrentView       : IServerDocumentView;
```

Description

This function fetches the current view (ie the open document in focus in DXP). See the CurrentView property and the IServerDocumentView interface.

Example

```
Procedure GrabACurrentDocumentView;
Var
    ServerDocumentView : IServerDocumentView;
    CurrentDirectory   : AnsiString;
Begin
    ServerDocumentView := Client.GetCurrentView;
    CurrentDirectory :=
ExtractFileDir(ServerDocumentView.GetOwnerDocument.FileName);
```

```
End;
```

See also

CurrentView property

GetDefaultExtensionForDocumentKind method

(IClient interface)

Syntax

```
Function  GetDefaultExtensionForDocumentKind(DocumentKind : PChar) : PChar;
```

Description

This function returns the default extension for the specific document kind based on the document kind parameter.

GetDocumentByPath method

(IClient interface)

Syntax

```
Function  GetDocumentByPath(Const AFilePath : WideString) : IServerDocument;
```

Description

This function fetches the full file path to a design document and if the path is valid, an IServerDocument object interface is returned representing the whole design document and its panels.

GetDocumentKindFromDocumentPath method

(IClient interface)

Syntax

```
Function GetDocumentKindFromDocumentPath   (Path : PChar) : PChar;
```

Description

This function returns the document kind based on the valid and full document path.

GetEncryptedTechnologySets method

(IClient interface)

Syntax

```
Function GetEncryptedTechnologySets (Var ValidAtTimestamp : Cardinal) :
WideString;
```

See also

IClient interface

GetGUIManager method

(IClient interface)

Syntax

```
Function  GetGUIManager : IGUIManager;
```

Description

Returns the GUI Manager interface. Use the GUIManager property instead. This Interface object deals with the User Interface of DXP such as controlling the status bars of DXP, the locations and the state of panels in DXP.

See also

IGUIManager interface

GetLicenseManager function

(IClient interface)

Syntax

```
Function  GetLicenseManager : ILicenseManager;
```

See also

IClient interface

ILicenseManager interface

GetMainWindowHandle method

(IClient interface)

Syntax

```
Function  GetMainWindowHandle : Integer;
```

Description

You can use the application handle into server code if dialogs need to be created dynamically from your server and so that when a dialog that appears on Design Explorer will inherit Design Explorer's icon and appear as one whole application on the task bar. This ApplicationHandle property is also its equivalent.

See also

GetApplicationHandle method.

ApplicationHandle property.

GetNavigationSystem method

(IClient interface)

Syntax

```
Function GetNavigationSystem : INavigationSystem;
```

Description

The function returns the Navigation system interface.

See also

INavigationSystem interface

GetOptionsManager function

(IClient interface)

See also

IClient interface

GetOptionsSetByName method

(IClient interface)

Syntax

```
Function GetOptionsSetByName (Const AName : Widestring) :
IDocumentOptionsSet;
```

See also

GetOptionsSetCount method

GetOptionsSet method

IDocumentOptionsSet interface

GetOptionsSetCount method

(IClient interface)

Syntax

```
Function  GetOptionsSetCount : Integer;
```

See also

GetOptionsSet method

GetOptionsSetByName method

GetOptionsSet method

(IClient interface)

Syntax

```
Function GetOptionsSet (Index : Integer) : IDocumentOptionsSet;
```

See also

GetOptionsSetCount method

GetOptionsSetByName method

GetPanelInfoByName method

(IClient interface)

Syntax

```
Function  GetPanelInfoByName (Const APanelName  : Widestring)
: IServerPanelInfo;
```

Description

This function obtains the IServerPanelInfo interface for the specified panel.

See also

IServerPanelInfo interface

GetProcessControl method

(IClient interface)

Syntax

```
Function  GetProcessControl : IProcessControl;
```

Description

Returns the Process Control interface. This Process Control determines the number of "re-entrant" processes occurring, ie one client's process occurring stacked on top of another active client's process – this is the process depth. If a process control's process depth is zero, it indicates that nothing is taking place in DXP.

See also

IProcessControl interface

GetRealMainWindowHandle method

(IClient interface)

Syntax

```
Function  GetRealMainWindowHandle : THandle;
```

Description

Returns the window handle of the main window in DXP.

GetServerNameByPLID method

(IClient interface)

Syntax

```
Function  GetServerNameByPLID(APLID : PChar) : PChar;
```

Description

This function returns you the server name based on the PLID identifier string (a string extracted from the server's resources file).

GetServerModule method

(IClient interface)

Syntax

```
Function  GetServerModule(Index : Integer) : IServerModule;
```

Description

The ServerModule property is used in conjunction with the Count property to retrieve active (loaded) servers. The ServerModule property returns the IServerModule interface for the loaded server module in DXP.

Note, that PCB server and Schematic server have their own IPCB_ServerInterface and ISch_ServerInterface interfaces respectively.

IServerModule example

This example gets the Schematic's IServerModule interface and returns the number of document views open in DXP

```
Var
    ServerModule : IServerModule;
Begin
    If Client = Nil Then Exit;

    ServerModule := Client.ServerModuleByName('SCH');
    ShowMessage('Doc Count = ' + IntToStr(ServerModule.DocumentCount));
End;
```

See also

Count property

IServerModule property

ServerModuleByName property

GetServerModuleByName method

(IClient interface)

Syntax

```
Function  GetServerModuleByName (Const AModuleName : Widestring) :
IServerModule;
```

Description

The function returns the server module interface depending on the validity of the AModuleName parameter. Examples include 'PCB' or 'SCH'. Use the ServerModuleByName property instead to return the indexed server module.

Example

```
Var
    ServerModule : IServerModule;
Begin
    If Client = Nil Then Exit;

    ServerModule := Client.ServerModuleByName('SCH');
    ShowMessage('Doc Count = ' + IntToStr(ServerModule.DocumentCount));
End;
```

See also

GetServerModule method

ServerModule property

GetServerRecord method

(IClient interface)

Syntax

```
Function  GetServerRecord (Index : Integer) : IServerRecord;
```

Description

The GetServerRecord function reports the number of installed servers based on the INS files in the Altium\System folder). Use this in conjunction with the GetServerRecordCount function.

The **IClient** interface has **GetServerRecord** and **GetServerModule** methods. The difference between these two methods is that the **GetServerRecord** function reports the number of installed servers (INS files in the **\Altium2004\System** folder).

The **GetServerModule** merely returns the active (loaded) server in Design Explorer and to get each active server, you need to invoke the **GetCount** function and pass the count parameter into the **GetServerModule** function.

See also

GetServerRecordCount method

GetServerModule method

GetServerRecordCount method

(IClient interface)

Syntax

Function GetServerRecordCount : Integer;

Description

This function returns the number of server records that represent the server installation files found in the \Altium2004\System folder. This is to be used in conjunction with the **GetServerRecord** function.

GetServerRecordByName method

(IClient interface)

Syntax

```
Function  GetServerRecordByName(AModuleName : WideString) : IServerRecord;
```

Description

This function returns the **IServerRecord** interface based on the **AModuleName** parameter. This **IServerRecord** interface represents the installation file for the server (with an INS extension).

See also

IServerRecord interface

GetServerViewFromName method

(IClient interface)

Syntax

```
Function  GetServerViewFromName (Const ViewName : Widestring) : IServerView;
```

Description

Returns the Server view (server document view) depending on the name of the server view.

See also

IServerView interface

GetTimerManager Interface

(IClient interface)

Syntax

```
Function  GetTimerManager : ITimerManager;
```

Description

Returns the timer manager interface associated with the client sub system. Refer to the **ITimerManager** interface for details.

See also

ITimerManager interface

GetWindowKindByName method

(IClient interface)

Syntax

```
Function  GetWindowKindByName  (AWindowKindName : Widestring :
IServerWindowKind
```

Description

This function returns the IServerWindowKind interface based on the AWindowKindName parameter which denotes the document kind. For example, there are two document kinds in the PCB editor – PCB and PCBLIB documents.

See also

IServerWIndowKind interface

HideDocument method

(IClient interface)

Syntax

```
Procedure HideDocument (Const ADocument        : IServerDocument);
```

Description

This procedure hides the document which means this document loses focus but it is not closed or destroyed.

See also

CloseDocument method

OpenDocument method

ShowDocument method

IServerDocument interface

InRecoverySave method

(IClient interface)

Syntax

```
Function  InRecoverySave : LongBool
```

Description

This function checks whether DXP is in the process of Recovery Save mode. before you can invoke the BeginRecoverySave or EndRecoverySave methods.

See also

BeginRecoverySave method

EndRecoverySave method

IsDocumentOpen method

(IClient interface)

Syntax

```
Function  IsDocumentOpen (Const AFilePath : PChar) : LongBool;
```

Description

Returns a boolean value whether the document is open in DXP or not and is dependent on whether the AFilePath parameter is valid or not.

IsQuitting method

(IClient interface)

Syntax

```
Function  IsQuitting : Boolean;
```

Description

Returns a boolean value that represents the state DXP is in: True if DXP is about to quit or in the process of quitting, False if DXP is still active.

LastActiveDocumentOfType method

(IClient interface)

Syntax

```
Function  LastActiveDocumentOfType (Const AType : Widestring) :
IServerDocument;
```

Description

Returns the last active loaded document in DXP by the document type. Types include PCB, SCH, TEXT, WAVE, PCBLIB, SCHLIB.

IsInitialized function

(IClient interface)

See also

IClient interface

LicenseInfoStillValid method

(IClient interface)

Syntax

```
Function  LicenseInfoStillValid (Const RetrievedAt : Cardinal) : LongBool;
```

MainWindowHandle property

(IClient interface)

Syntax

```
Property MainWindowHandle : Integer Read GetMainWindowHandle;
```

Description

The MainWindowHandle property returns the handle of the main window in DXP which can be used for addon dialogs that will be attached to DXP and have a single DXP icon on the Taskbar for example.

See also

GetMainWindowHandle method

ApplicationHandle property

OpenDocument method

(IClient interface)

Syntax

```
Function  OpenDocument (Const AKind, AFileName : PChar) : IServerDocument;
```

Description

The **OpenDocument** method returns the **IServerDocument** interface depending on the valid **DocumentKind** and **FileName**

parameters.

Example

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

See also

ShowDocument method

QuerySystemFont method

(IClient interface)

Syntax

```
Procedure QuerySystemFont (    QueryMode     : TFontQueryMode;
                           Var AUseSysFont   : Boolean;
                           Var AFontName     : WideString;
                           Var AFontSize     : Integer;
                           Var AFontStyle    : TFontStyles;
                           Var AFontColor    : TColor;
                           Var AFontCharset : TFontCharset);
```

Description

Query the system font used.

RegisterNotificationHandler method

(IClient interface)

Syntax

```
Procedure RegisterNotificationHandler(Const Handler : INotificationHandler);
```

See also

BroadcastNotification method

DispatchNotification method

UnRegisterNotificationHandler method

INotificationHandler interface

RemoveServerView method

(IClient interface)

Syntax

```
Procedure RemoveServerView (Const AView : IServerView);
```

Description

This procedure removes a server view (representing a server document window)from DXP.

See also

GetCurrentView method

ShowDocumentDontFocus method

(IClient interface)

Syntax

```
Procedure ShowDocumentDontFocus(ADocument : IServerDocument);
```

Description

This procedure fetches the IServerDocument parameter and then displays this design document but leaves the previously focussed document in focus. If there are not design documents open already, then this design document will still be displayed but not focussed.

See also

OpenDocument method

ShowDocument method

IServerDocument interface

ShowDocument method

(IClient interface)

Syntax

```
Procedure ShowDocument (ADocument : IServerDocument);
```

Description

This procedure fetches the IServerDocument parameter which represents the Server Document loaded in DXP and then displays the design document in DXP.

IServerDocument example

This example gets the client interface and then open./show a document from the MyDesigns folder

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

See also

OpenDocument method

IServerDocument interface

SetCurrentView method

(IClient interface)

Syntax

```
Procedure SetCurrentView(Value : IServerDocumentView);
```

Description

This procedure fetches the IServerDocumentView parameter to set this document form as the current view in DXP.

See also

GetCurrentView method

CurrentView property

StopServer method

(IClient interface)

Syntax

```
Function  StopServer (AModuleName : WideString) : Boolean;
```

Description

The StartServer and StopServer properties can be used to load a server in Design Explorer if it has not already before you can invoke this server's processes and to stop this server once you have done with these server processes. This can be used to conserve computer's memory.

The StartServer function is usually used if you need to load a design document and execute the server's processes or its API functions if the server has not been loaded yet. Example, during a blank session of Design Explorer where there are no PCB documents open, and you need to use the PCB API to manipulate the contents on a PCB document, you would need to “start” the PCB server first so the PCB API is made active.

Example of the StopServer method

```
Client.StopServer('PCB');
```

See also

StartServer method

StartServer method

(IClient interface)

Syntax

```
Function  StartServer (AModuleName : WideString) : Boolean;
```

Description

The **StartServer** and **StopServer** properties can be used to load a server in Design Explorer if it has not already before you can invoke this server's processes and to stop this server once you have done with these server processes. This can be used to conserve computer's memory.

The **StartServer** function is usually used if you need to load a design document and execute the server's processes or its API functions if the server has not been loaded yet. Example, during a blank session of Design Explorer where there are no PCB documents open, and you need to use the PCB

API to manipulate the contents on a PCB document, you would need to “start” the PCB server first so the PCB API is made active.

Example of the StartServer method

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

See also

StopServer method

UnregisterNotificationHandler method

(IClient interface)

Syntax

```
Procedure UnregisterNotificationHandler(Const Handler :
INotificationHandler);
```

Description

Procedure UnregisterNotificationHandler(Const Handler : INotificationHandler); Internal operation for handling notifications. The INotificationHandler object interface is responsible for handling notifications raised in DXP.

See also

BroadcastNotification

DispatchNotification

RegisterNotificationHandler method

INotificationHandler interface

IClient interface properties

ApplicationHandle property

(IClient interface)

Syntax

```
Property  ApplicationHandle : Integer
```

Description

The ApplicationHandle property sets the application handle in a server if dialogs need to be created dynamically from your server and every time a dialog that appears in front of Design Explorer will inherit Design Explorer’s icon and appear as one whole application on the task bar. This ApplicationHandle property can be passed as a parameter for the create constructor of a dynamic dialog for example.

See also

GetMainWindowHandle method

GetApplicationHandle method

CommandLauncher property

(IClient interface)

Syntax

```
Property CommandLauncher : ICommandLauncher Read GetCommandLauncher;
```

Description

The CommandLauncher property returns the Command Launcher interface. This interface contains the table of client's process launchers that can be used to launch a client's command.

Example

```
If StringsEqual(ServerModule.ModuleName,'TextEdit') Then
Begin
    Client.CommandLauncher.LaunchCommand('TextEdit:MoveCursorToTopOfDocument
',
                                                Nil,0,ServerDocument.View[0]
);
End;
```

GetCommandLauncher example

```
ACommandLauncher := Client.GetCommandLauncher;
If ACommandLauncher <> Nil Then
Begin
    ACommandLauncher.GetCommandState(Command,
                                     Parameters,
                                     View,
                                     Enabled,
                                     Checked,
                                     Visible,
                                     Caption,
                                     Image);
End;
```

See also

GetCommandLauncher method

IProcessLauncher interface

ICommandLauncher interface

CurrentView property

(IClient interface)

Syntax

```
Property  CurrentView : IServerDocumentView Read GetCurrentView Write
SetCurrentView;
```

Description

This property returns the current document view interface of **IServerDocumentView** type which represents the current design document view in DXP.

SendMessage example

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

CurrentView example

```
Procedure GrabACurrentDocumentView;
Var
    ServerDocumentView : IServerDocumentView;
    CurrentDirectory   : AnsiString;
Begin
    ServerDocumentView := Client.CurrentView;
    CurrentDirectory :=
ExtractFileDir(ServerDocumentView.GetOwnerDocument.FileName);
End;
```

ViewName example

```
If UpperCase(Client.CurrentView.OwnerDocument.Kind) <> 'PCBLIB' Then Exit;
```

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

See also

GetCurrentView method

SetCurrentView method

IServerDocumentView interface

GUIManager Property

(IClient interface)

Syntax

```
Property GUIManager : IGUIManager Read GetGUIManager;
```

Description

The GUIManager property returns the GUIManager interface. This Interface object deals with the User Interface of DXP such as controlling the status bars of DXP, the locations and the state of panels in DXP.

See also

GetGUIManager method

IGUIManager interface

NavigationSystem property

(IClient interface)

Syntax

```
Property  NavigationSystem : INavigationSystem   Read GetNavigationSystem;
```

See also

GetNavigationSystem method

INavigationSystem interface

ProcessControl property

(IClient interface)

Syntax

```
Property ProcessControl : IProcessControl Read GetProcessControl;
```

Description

This property returns the pointer to the ProcessControl interface. This Process Control interface determines the number of “re-entrant” processes occurring, ie one client’s process occurring stacked on top of another active client’s process – this is the process depth. If a process control’s process depth is zero, it indicates that nothing is taking place in DXP. Refer to the IProcessControl interface for details.

ProcessDepth Example

```
ShowMessage('Current process depth
',IntToStr(Client.ProcessControl.ProcessDepth));
```

See also

GetProcessControl method

IProcessControl interface

ServerModule property

(IClient interface)

Syntax

```
Property ServerModule [Index : Integer] : IServerModule Read
GetServerModule;
```

Description

The ServerModule property is used in conjunction with the Count property to retrieve active (loaded) servers. The ServerModule property returns the IServerModule interface for the loaded server module in DXP.

Note, that PCB server and Schematic server have their own IPCB_ServerInterface and ISch_ServerInterface interfaces respectively.

IServerModule example

This example gets the Schematic's IServerModule interface and returns the number of document views open in DXP

```
Var
    ServerModule : IServerModule;
Begin
    If Client = Nil Then Exit;

    ServerModule := Client.ServerModuleByName('SCH');
    ShowMessage('Doc Count = ' + IntToStr(ServerModule.DocumentCount));
End;
```

See also

Count property

GetServerModule method

IServerModule interface

ServerModuleByName property

(IClient interface)

Syntax

```
Property ServerModuleByName[Const AModuleName : Widestring] : IServerModule
Read GetServerModuleByName;
```

Description

The ServerModuleByName property returns the IServerModule interface if the module name is found in the Client's table of active servers. For a PCB editor, module name is PCB, for a Schematic Editor, the module name is SCH etc.

Example

```
Var
```

```
    ServerModule : IServerModule;
Begin
    If Client = Nil Then Exit;

    ServerModule := Client.ServerModuleByName('SCH');
    ShowMessage('Doc Count = ' + IntToStr(ServerModule.DocumentCount));
End;
```

See also

GetServerModuleByName method

IServerModule interface

TimerManager property

(IClient interface)

Syntax

```
Property TimerManager : ITimerManager Read GetTimerManager;
```

Description

Returns the timer manager object interface.

See also

GetTimerManager method

ITimerManager interface

OptionsManager property

(IClient interface)

Syntax

```
Property  OptionsManager : IOptionsManager Read GetOptionsManager;
```

See also

IClient interface

IOptionsManager interface

IServerModule interface

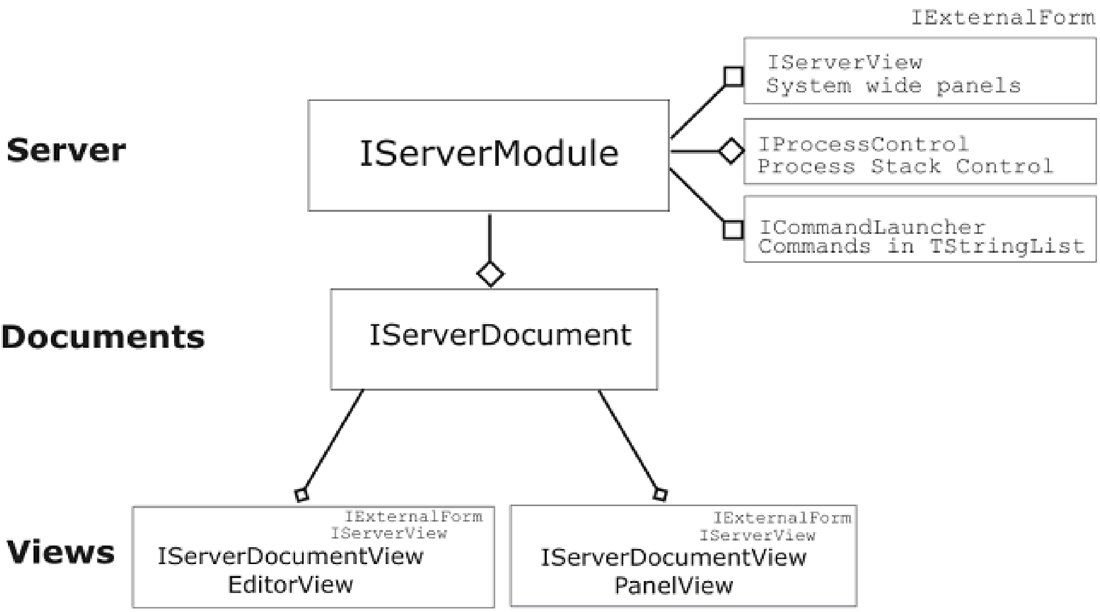
Overview

A server deals with its own server documents. There can be different design document types, for example the Schematic Editor has two Schematic and Schematic Library document types.

Each design document, in turn stores views which can be a document window or a panel window. A server has the ability to host multiple panel views for a single document view, see the diagram above. A server also has the ability to host multiple global panel views that represent some system state and are not necessarily tied to a particular design document (for example the Work-Space Manager server has Message, Differences and Errors panels). This document view / multiple panel views structure is the foundation of DXP client / server architecture.

These **IServerModule** interfaces represent loaded servers in DXP. The DXP application manages single instances of different server modules. Each server can have multiple server document kinds, for example the PCB server supports two server document kinds – PCB and PCBLIB design documents. A loaded server in DXP typically hosts documents and each document in turn hosts a document view and panel views.

The diagram below represents a server module with server documents. Each document has views - the document view and the associated panel view.



Notes

An **IServerModule** interface has the following interfaces:

- **ICommandLauncher** deals with a server's processes table
- **IServerDocument** represents a loaded design document in DXP

- **IServerView** represents a panel that can have a view of the DXP system
- **IServerDocumentView** (deals with a document view (either the document window or panel window)
- **IExternalForm** represents a DXP aware Delphi form either as a document form or a panel form.

These forms are wrapped by the **IServerDocumentView** or **IServerView** interface object. This **IExternalForm** interface object has low level methods such as resizing and displaying the form and is the ancestor interface for **IServerDocumentView** and **IServerView** interfaces.

- **IProcessControl** represents the level of stacked processes for this focussed server document
- **INotification** receive system notifications from the Client system and all server modules receive

these notifications. There is an ability to handle a notification and take it from there. Documents and associated panels can be synchronized through the use of notifications as well.

Notes

The PCB server module also has its **IPCB_ServerInterface** interface.

The Schematic Server module also has its **ISCH_ServerInterface** interface.

However both servers also have this **IServerModule** interface.

IServerModule methods	IServerModule Properties
`ApplicationIdle`	`Client`
`ReceiveNotification`	`CommandLauncher`
`CreateDocument`	`Handle`
`DestroyDocument`	`ModuleName`
`CreateOptionsView`	`ProcessControl`
`CreateServerView`	`DocumentCount`
`CreateServerDocView`	`Documents`
`RemoveServerView`	`ViewCount`
`AddServerView`	`Views`

See also

IPCB_ServerInterface interface

ISCH_ServerInterface interface

IServerModule interface methods

AddServerView method

(IServerModule interface)

Syntax

```
Procedure AddServerView            (Const AView : IServerView);
```

See also

IClient interface

IServerModule interface

ApplicationIdle method

(IServerModule interface)

Syntax

```
Procedure ApplicationIdle;
```

See also

IClient interface

IServerModule interface

CreateDocument method

(IServerModule interface)

See also

IClient interface

IServerModule interface

CreateServerDocView method

(IServerModule interface)

Syntax

```
Function  CreateServerDocView     (Const AName : Widestring; Const ADocument
: IServerDocument): IServerDocumentView;
```

See also

IClient interface

IServerModule interface

CreateServerView method

(IServerModule interface)

Syntax

```
Function  CreateServerView (Const AName : Widestring) : IServerView;
```

See also

IClient interface

IServerModule interface

DestroyDocument method

(IServerModule interface)

Syntax

```
Procedure DestroyDocument (Const ADocument : IServerDocument);
```

See also

IClient interface

IServerModule interface

ReceiveNotification method

(IServerModule interface)

Syntax

```
Procedure ReceiveNotification(Const ANotification : INotification);
```

See also

IClient interface

IServerModule interface

RemoveServerView method

(IServerModule interface)

Syntax

```
Procedure RemoveServerView (Const AView : IServerView);
```

See also

IClient interface

IExternalForm interface

IServerModule interface properties

Client property

(IClient interface)

Syntax

```
Property Client : IClient Read GetClient;
```

Description

The Client property returns the pointer to IClient interface of the client subsystem of Design Explorer. This IClient interface can be used to invoke its methods.

Example

```
    ServerModule.Client.BeginDisableInterface
```

See also

IClient Interface

CommandLauncher property

(IServerModule interface)

Syntax

```
Property CommandLauncher : ICommandLauncher Read GetCommandLauncher;
```

Description

The CommandLauncher property returns the pointer to the ICommandLauncher interface. It is used to launch a process from its server module.

The CommandLauncher object contains a command table which binds a process name to the actual function that implements the process at run-time. Whenever a process is called within the server, this table is looked up in order to find the actual function pointer. If a process name is not found within this table nothing will happen. This CommandLauncher object is initialized in the main.pas unit of a server project.

Example

```
If StringsEqual(ServerModule.ModuleName,'TextEdit') Then
Begin
    ServerModule.CommandLauncher.LaunchCommand('TextEdit:MoveCursorToTopOfDo
cument',
                                                              Nil,0,ServerDocument.View[0]
);
```

End;See also

GetCommandLauncher method

ICommandLauncher interface

CreateOptionsView property

(IServerModule interface)

See also

IClient interface

IServerModule interface

DocumentCount property

(IServerModule interface)

Syntax

```
Property DocumentCount : Integer Read GetDocumentCount;
```

Description

The DocumentCount property returns you the number of Document Kinds for the server. An important note is that a View is the actual design document window. Note that a IServerDocument interface is a container that stores specific Views.

IServerModule example

This example gets the Schematic's IServerModule interface and returns the number of Schematic documents open in DXP. Be aware of the terminology used for documents and views.

```
Var
    ServerModule : IServerModule;
Begin
    If Client = Nil Then Exit;
    ServerModule := Client.ServerModuleByName('SCH');
    ShowMessage('Schematic Document Count = ' +
IntToStr(ServerModule.DocumentCount));
    // Should be 2 kinds for Schematic editor, sheets and library documents.
End;
```

See also

IServerDocument interface

IServerDocumentView interface

DocumentCount property

Documents property

ViewCount property

Views property

Documents property

(IDocuments interface)

Syntax

```
Property  Documents[Index : Integer] : IServerDocument  Read GetDocuments;
```

Description

An editor type of server will have different document types, such as Schematic Editor and PCB Editor. Both servers have two document types - SCH/SCHLIB and PCB/PCBLIB respectively. An add-on type of server will normally have no document containers, because they work with an editor server.

See also

IClient interface

IServerModule interface

DocumentCount property

Handle property

(IServerModule interface)

Syntax

```
Property  Handle : THandle Read GetHandle;
```

See also

IClient interface

IServerModule interface

ModuleName property

(IServerModule interface)

Syntax

```
Property ModuleName : Widestring Read GetModuleName;
```

Description

The ModuleName property returns the name string of the server.

See also

IClient interface

IServerModule interface

ViewCount property

(IServerModule interface)

Syntax

```
Property  ViewCount : Integer Read GetViewCount;
```

Description

The ViewCount property returns you the number of views for the specified server. A View object encapsulates a form/window object in DXP normally as a global panel supported by its associated server. It is a Read Only property.

See also

IClient interface

IServerModule interface

Views property

(IServerModule interface)

Syntax

```
Property  Views[Index : Integer]     : IServerView      Read GetViews;
```

Description

The Views property in conjunction with the ViewCount property returns you the indexed View object. A view is a form supported by its associated server.

See also

IClient interface

IServerModule interface

Document and Panel View Interfaces

IExternalForm interface

Overview

The **IExternalForm** interface represents a DXP aware Delphi form either as a document form or a panel form. This **IExternalForm** interface object has low level methods such as resizing and displaying the form.

The **IServerDocumentView** and **IServerView** interfaces are inherited from this interface.

IExternalForm methods	IExternalForm properties
`SetParentWindow`	`Caption`
`ParentWindowCreated`	`Handle`
`ParentWindowDestroyed`	
`GetBounds`	
`Hide`	
`SetBounds`	
`SetFocus`	
`Show`	
`FocusFirstTabStop`	

See also

IServerView interface

IServerDocumentView interface

IExternalForm methods

FocusFirstTabStop method

(IExternalForm interface)

Syntax

```
Procedure FocusFirstTabStop;
```

GetBounds method

(IExternalForm interface)

Syntax

```
Procedure GetBounds (Var ALeft, ATop, AWidth, AHeight : Integer);
```

Hide method

(IExternalForm interface)

Syntax

```
Procedure Hide;
```

Description

This hides the dialog from view in DXP.

See also

IClient interface

IExternalForm interface

ParentWindowCreated method

(IExternalForm interface)

Syntax

```
Procedure ParentWindowCreated;
```

See also

IClient interface

IExternalForm interface

ParentWindowDestroyed method

(IExternalForm interface)

Syntax

```
Procedure ParentWindowDestroyed;
```

See also

IClient interface

IExternalForm interface

SetBounds method

(IExternalForm interface)

Syntax

```
Procedure SetBounds (ALeft, ATop, AWidth, AHeight : Integer);
```

See also

IClient interface

IExternalForm interface

SetFocus method

(IExternalForm interface)

Syntax

```
Procedure SetFocus;
```

Description

Invoking this method sets the dialog in focus in DXP.

See also

IClient interface

IExternalForm interface

SetParentWindow method

(IExternalForm interface)

Syntax

```
Procedure SetParentWindow (Const ParentWindow : IExternalFormHolder);
```

See also

IClient interface

IExternalForm interface

Show method

(IExternalForm interface)

Syntax

```
Procedure Show;
```

Description

This procedure displays the hidden dialog.

Example

See also

IClient interface

IExternalForm interface

IExternalForm properties

Caption property

(IExternalForm interface)

Syntax

```
Property  Caption : Widestring
```

Description

A read only property that returns you the caption of the external form that the dialog is associated with.

See also

IClient interface

IExternalForm interface

Handle property

(IExternalForm interface)

Syntax

```
Property  Handle   : HWND
```

Description

A read only property that returns you the handle of the dialog.

See also

IClient interface

IExternalForm interface

IServerView interface

Overview

The **IServerView** interface is the ancestor interface for a document or panel view object interface. This interface is inherited from the immediate ancestor interface, **IExternalForm** interface.

The hierarchy is as follows;

- **IExternalForm**
  - **IServerView** interface

IExternalForm methods

`SetParentWindow`

`ParentWindowCreated`

`ParentWindowDestroyed`

`GetBounds`

`Hide`

`SetBounds`

`SetFocus`

`Show`

`FocusFirstTabStop`

IExternalForm properties

`Caption`

`Handle`

IServerView Methods

`GetViewState`

`SetViewState`

`ReceiveNotification`

IServerView Properties

`IsPanel`

`ViewName`

IServerView interface Methods

GetViewState method

(IServerView interface)

Syntax

```
Function  GetViewState        : Widestring;
```

See also

IClient interface

IExternalForm interface

SetViewState method

ReceiveNotification method

(IServerView interface)

Syntax

```
Procedure ReceiveNotification (Const ANotification : INotification);
```

See also

IClient interface

IExternalForm interface

INotification interface

SetViewState method

(IServerView interface)

Syntax

```
Procedure SetViewState(Const Astate : Widestring);
```

IClient interface

IExternalForm interface

GetViewState method

IServerView interface properties

IsPanel property

(IServerView interface)

Syntax

```
Property IsPanel : LongBool Read GetIsPanel;
```

Description

The IsPanel property returns a boolean value denoting whether the view is a panel or a document view. The IsPanel property is a read only value. A document consists of a document view and at least one panel view. There also can be global or system views such as Message panel which is a global panel view.

See also

IServerView interface

ViewName property

(IServerView interface)

Syntax

```
Property ViewName     : Widestring    Read GetViewName;
```

Description

This property returns the document kind name such as PCBLIB for a PCB library document PCB for PCB document, SCH for schematic document and SCHLIB for schematic library document.

Example

```
If UpperCase(Client.CurrentView.ViewName) <> 'PCBLIB' Then Exit;
```

See also

IClient interface

IServerView interface

IServerDocumentView interface

Overview

The **IServerDocumentView** represents either the document view or one of the associated panel views in Design Explorer. This interface is inherited from the **IServerView** interface.

The **IServerDocument** interface contains **IServerDocumentView** interfaces, that is, a design document open in DXP contains links to a document view and at least one panel view.

The hierarchy is as follows:

- IExternalForm
    - IServerView interface
        - IServerDocumentView interface.

IExternalForm methods

`SetParentWindow`

`ParentWindowCreated`

`ParentWindowDestroyed`

`GetBounds`

`Hide`

`SetBounds`

`SetFocus`

`Show`

`FocusFirstTabStop`

IExternalForm properties

`Caption`

`Handle`

IServerView Methods

`GetViewState`

`SetViewState`

`ReceiveNotification`

IServerView Properties

`IsPanel`

`ViewName`

IServerDocumentView Methods

`GetOwnerDocument`

`PerformAutoZoom`

`UpdateStatusBar`

IServerDocumentView Properties

`OwnerDocument`

See also

IClient interface

IServerModule interface

IServerDocument interface

IServerView interface

IExternalForm interface

IServerDocumentView interface methods

PerformAutoZoom method

(IServerDocumentView interface)

Syntax

```
Procedure PerformAutoZoom;
```

Description

This method performs a auto zoom action which is in effect a forced refresh of the currently focussed design document in DXP.

See also

IClient interface

IServerDocumentView interface

UpdateStatusBar method

(IServerDocumentView interface)

Syntax

```
Procedure UpdateStatusBar;
```

Description

This UpdateStatusBar method forces a refresh of the Status bar on the DXP.

See also

IClient interface

IServerDocumentView interface

GetOwnerDocument method

(IServerDocumentView interface)

Syntax

```
Function  GetOwnerDocument : IServerDocument;
```

Description

This function returns you the server document container of **IServerDocument** interface type. This **IServerDocument** container stores server's document and panel views that are associated with the document container. For example PCB and PCB library documents are in this PCB Server's **IServerDocument** container.

Example

```
Procedure FetchCurrentDirectoryPath;
Var
    ServerDocumentView : IServerDocumentView;
    CurrentDirectory   : AnsiString;
Begin
    If Client = Nil Then Exit;
    ServerDocumentView := Client.GetCurrentView;
    CurrentDirectory   :=
ExtractFileDir(ServerDocumentView.GetOwnerDocument.FileName);
End;
```

See also

OwnerDocument Property

IServerDocument interface

IServerDocumentView interface properties

OwnerDocument property

(IServerDocumentView interface)

Syntax

```
Property OwnerDocument : IServerDocument Read GetOwnerDocument;
```

Description

This property returns you the server document container of IServerDocument interface type. This IServerDocument container stores server's document and panel views that are associated with the document container. For example PCB and PCB library documents are in this PCB Server's IServerDocument container.

Example

```
Procedure FetchTheCurrentDirectory;
Var
    ServerDocumentView : IServerDocumentView;
    CurrentDirectory   : WideString;
Begin
    If Client = Nil Then Exit;
```

```
    ServerDocumentView := Client.CurrentView;
    CurrentDirectory   :=
ExtractFileDir(ServerDocumentView.OwnerDocument.FileName);
End;
```

See also

GetOwnerDocument method

IHighlightedDocument interface

Overview

This **IHighlightedDocument** interface represents a mechanism in the DXP platform that deals with highlighting of objects on a design document in DXP when objects are being selected or deselected and when being masked or not.

Notes

The **IHighlightedDocument** interface is inherited from the **IServerDocument** interface.

IHighlightedDocument methods

```
HL_Begin
HL_End
HL_Perform
HL_HighlightMethod_Add
HL_HighlightMethod_Remove
HL_HighlightMethod_Clear
HL_HighlightMethod_IsApplicable
HL_Register_DMObject
HL_Register_NetItem
HL_Register_Net
HL_Register_Bus
HL_Register_Part
HL_Register_Component
HL_Register_VHDLEntity
HL_UnRegister_Object
HL_UnRegister_AllObjects
HL_ObjectCount
HL_Objects
HL_SetHighlightedNet
HL_GetHighlightedNet
HL_GetLinkedObject
HL_ChooseObjectGraphically
HL_XProbeChooseObject
HL_HighlightedNet
```

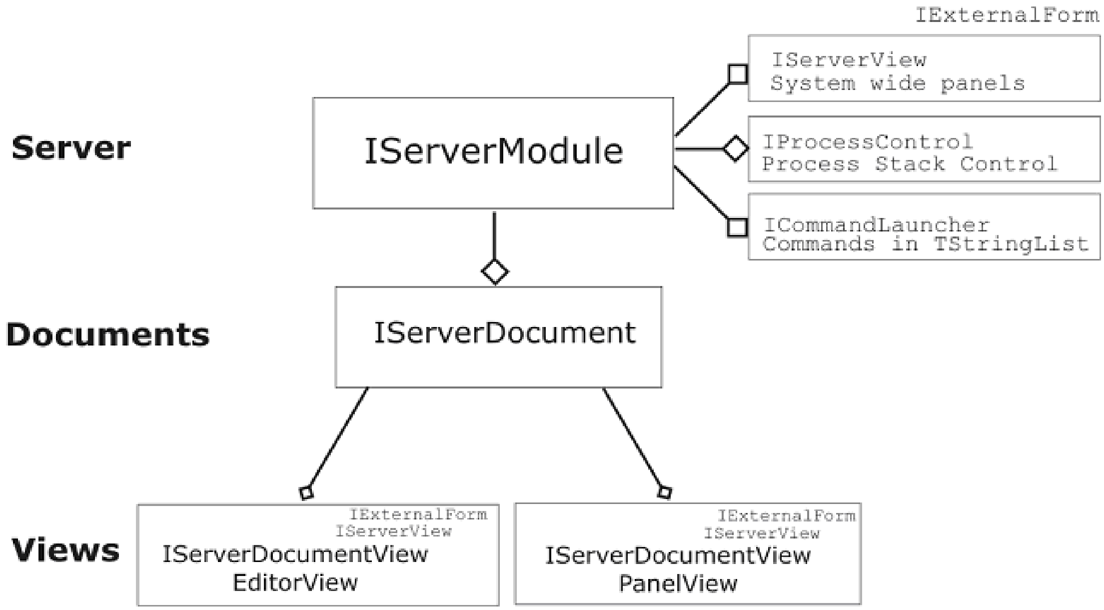
IHighlightedDocument properties

```
Property  HL_HighlightedNet :
INet
```

IServerDocument interface

Overview

The **IServerDocument** interface represents the document container. Each **IServerDocument** container is a document made up of views. A view can be a design document form or a panel form. Every server module (encapsulated by the **IServerModule** interface) that supports creation of documents will have a **IServerDocument** interface.



IServerDocument Methods	IServerDocument Properties
`AddView`	`CanClose`
`Focus`	`Count`
`DoFileLoad`	`FileName`
`DoFileSave`	`Kind`
`SupportsReload`	`Modified`
`SetFileName`	`IsShown`
`GetKind`	`BeingClosed`
`GetFileModifiedDate`	`ServerModule`
`UpdateModifiedDate`	`View`
`GetContextHelpTopicName`	`SupportsOwnSave`
`SetFileModificationDate`	

IServerDocument example

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

See also

IClient interface

IServerDocument interface Methods

AddView method

(IServerDocument interface)

Syntax

```
Procedure AddView (Const AView : IServerDocumentView);
```

Description

The AddView method adds a new view associated with the document in question. A server hosts multiple views and a view can be a document or a panel.

See also

IClient interface

IServerDocument interface

DoFileLoad

`(IServerDocument interface)`

Syntax

```
Function  DoFileLoad : LongBool;
```

Description

The DoFileLoad function when invoked re-loads the document and hereby refreshing the document in DXP.

See also

IClient interface

IServerDocument interface

DoFileSave

`(IServerDocument interface)`

Syntax

```
Function DoFileSave (Const AKind : Widestring)  : LongBool;
```

Description

The DoFileSave function when invoked with the supplied kind string saves the document itself. The AKInd parameter can be one of the following (PCB,PCBLIB, SCH, SCHLIB, TEXT...)

See also

IClient interface

IServerDocument interface

Focus method

`(IServerDocument interface)`

Syntax

```
Procedure Focus;
```

Description

The Focus method sets the focus for the design document in question, and the other document that has the focus is de-focused.

See also

IClient interface

IServerDocument interface

GetFileModifiedDate

`(IServerDocument interface)`

Syntax

```
Function  GetFileModifiedDate: TDateTime;
```

Description

The GetFileModifiedDate function returns the date and time associated with the document in regards to the last time it was modified in DXP.

See also

IClient interface

IServerDocument interface

SetFileModifiedDate

`(IServerDocument interface)`

Syntax

```
Procedure SetFileModifiedDate(Const AValue : TDateTime);
```

Description

The **SetFileModifiedDate** accepts a **TDateTime** type parameter to set the modified date for the current document.

See also

IClient interface

IServerDocument interface

GetFileModifiedDate method

SupportsReload

`(IServerDocument interface)`

Syntax

```
Function  SupportsReload : LongBool;
```

Description

The SupportsReload function returns a boolean value whether the server that the document is associated with supports reloading of same documents in DXP (same as refreshing).

See also

IClient interface

IServerDocument interface

UpdateModifiedDate

`(IServerDocument interface)`

Syntax

```
Procedure UpdateModifiedDate;
```

Description

The **UpdateModifiedDate** function forces a refresh of the date associated with the document.

See also

IClient interface

IServerDocument interface.

IServerDocument interface properties

BeingClosed property

(IServerDocument interface)

Syntax

```
Property  BeingClosed : LongBool Read GetBeingClosed Write SetBeingClosed;
```

Description

The **BeingClosed** property denotes that this design document is being closed before this design document can be successfully destroyed. This property is a read only property. You can check the status of the document before you attempt to modify or update the document before it is being closed.

See also

IClient interface

IServerDocument interface

CanClose property

(IServerDocument interface)

Syntax

```
Property CanClose : LongBool Read GetCanClose;
```

Description

The CanClose determines whether the document is ready to be closed and the document disappears from the workspace in DXP. Returns a boolean value.

Count Property

(IServerDocument interface)

Syntax

```
Property Count : Integer Read GetCount;
```

Description

The Count property returns the number of associated design documents stored in the current IServerDocument container. This property is a read-only property and it can be used in conjunction with the View property to return the Server Views associated with this IServerDocument object.

Note documents of a particular kind are stored in the same server document container.

See also

GetCount method

Filename property

(IServerDocument interface)

Syntax

```
Property FileName : Widestring Read GetFileName;
```

Description

The FileName property returns the filename for the specified design document. This property is a read-only property.

See also

IClient interface

IServerDocument interface

IsShown property

(IServerDocument interface)

Syntax

```
Property IsShown : LongBool Read GetIsShown Write SetIsShown;
```

Description

This property denotes whether or not this document is displayed in DXP. You can set or get a boolean value for this property.

See also

IClient interface

IServerDocument interface

Kind property

(IServerDocument interface)

Syntax

```
Property  Kind : Widestring Read GetKind;
```

Description

The Kind reports the type of the document opened in DXP. Examples include ‘PCB’, ‘PCBLIB’,’SCH’,’SCHLIB’ etc. This property is a read-only property.

Example

```
If UpperCase(Client.CurrentView.OwnerDocument.Kind) <> 'PCB' Then Exit;
```

See also

IClient interface

IServerDocument interface

Modified property

(IServerDocument interface)

Syntax

```
Property Modified : LongBool Read GetModified Write SetModified;
```

Description

This property denotes whether this document has been modified or not, and can be taken as a “dirty” flag, that is a document has been modified and it has been marked dirty. You can set or get a boolean value for this property.

See also

IClient interface

IServerDocument interface

ServerModule property

(IServerDocument interface)

Syntax

```
Property  ServerModule : IServerModule Read GetServerModule;
```

Description

The ServerModule is a read-only property which returns the pointer to the IServerModule interface which represents the server object installed and running in Design Explorer. Refer to the IServerModule interface entry for details. Read only property.

See also

IClient interface

IServerDocument interface

SupportsOwnSave property

(IServerDocument interface)

Syntax

```
Property SupportsOwnSave : LongBool Read GetSupportsOwnSave;
```

Description

The **SupportsOwnSave** property returns a boolean value whether a save routine has been provided to save these documents associated with the server.. Read only property.

See also

IClient interface

IServerDocument interface

View property

(IServerDocument interface)

Syntax

```
Property  View[Index : Integer] : IServerDocumentView Read GetView;
```

Description

The View property is an indexed property. It returns the list of views (which could be document or panel windows).

See also

IClient interface

IServerDocument interface.

IServerPanelInfo interface

Overview

The **IServerPanelInfo** interface returns information for the panel in question.

IServerPanelInfo Methods	IServerPanelInfo Properties
`GetName`	`DocumentKindCount`
`GetCategory`	`DocumentKinds`
`GetBitmap`	`ProjectKindCount`
`GetHotkey`	`ProjectKinds`
`GetButtonVisible`	
`GetMultipleCreation`	
`GetCreationClassName`	
`GetCanDockVertical`	
`GetCanDockHorizontal`	
`SupportsDocumentKind`	
`SupportsProjectKind`	

See also

IServerView interface

IProcessLauncherInfo interface

Overview

This **IProcessLauncherInfo** interface encapsulates details about a server process in Design Explorer. See **ICommandLauncher** and **IServerProcess** interfaces as well.

Since a server has a set of processes and these process identifiers are stored in an installation file (which ends with an INS extension) and the process launchers that link to specific user interface elements (also called resources) and the layout of user interface elements are defined in the resources file (which ends with a RCS extension).

IProcessLauncherInfo methods

IProcessLauncherInfo properties

```
Caption
Parameters
Description
ImageFile
Key
Shift
Key2
Shift2
ShortcutText
ServerCommand
```

See also

ICommandLauncher interface

IClient interface

IServerProcess interface

IProcessLauncher interface

ICommandLauncher interface

IProcessLauncher interface

Overview

This **IProcessLauncher** interface is a mechanism that launches a server process in Design Explorer. See **ICommandLauncher** and **IServerProcess** interfaces as well.

Since a server has a set of processes and these process identifiers are stored in an installation file (which ends with an INS extension) and the process launchers that link to specific user interface elements (also called resources) and the layout of user interface elements are defined in the resources file (which ends with a RCS extension).

IProcessLauncher methods

```
PostMessage
SendMessage
GetCommandState
```

IProcessLauncher interface Methods

GetCommandState method

(IProcessLauncher interface)

Syntax

```
Procedure GetCommandState(Const AProcess, AParameters : PChar;
                                AContext      : IServerDocumentView;
                          Var   Enabled, Checked, Visible : LongBool;
                          Const Caption, ImageFile     : PChar);
```

Description

This procedure returns the status of the server process or command. Check the ICommandLauncher interface for details.

See also

ICommandLauncher interface

PostMessage method

(IProcessLauncher interface)

Syntax

```
Procedure PostMessage (AProcess         : PChar;
                       AParameters      : PChar;
                       MaxParameterSize : Integer;
```

```
                              Const AContext    : IServerDocumentView);
```

Description

A PostMessage method sends a server process and parameters to its designated object in DXP as represented by the AContext parameter. Normally the AContext parameter is the current design view which is captured by the Client.CurrentView property.

Example

```
Client.ProcessLauncher.PostMessage('Client:CustomizeResources', 'Action=Show
| ObjectKind=Panel | ID=LibraryBrowser', 0, Client.CurrentView);
```

See also

SendMessage method

IClient interface

IServerProcess interface

SendMessage method

(IProcessLauncher interface)

Syntax

```
Function SendMessage(Const AProcess          : PChar;
                           AParameters       : PChar;
                           MaxParameterSize : Integer;
                           AContext          : IServerDocumentView) :
LongBool
```

Description

A SendMessage method is a faster and more direct method than the PostMessage method and returns a LongBool whether the message has arrived at its designated object or not in DXP.The designated object is represented by the AContext parameter. Normally the AContext parameter is the current design view which is captured by the Client.CurrentView property.

Example

```
Client.ProcessLauncher.SendMessage('Client:CustomizeResources', 'Action=Show
| ObjectKind=Panel | ID=LibraryBrowser', 0, Client.CurrentView);
```

IProcessControl interface

Overview

The **IProcessControl** interface controls the process depth for each design document in Design Explorer. Every time a process is launched on a document, the process depth is increased by one and once this same process has finished executing, the process depth is decreased by one. When the process depth is zero, it denotes that nothing is taking place on the current design document. This is necessary if you wish to keep the environment synchronized, especially the Undo system.

When you are using Schematic API or PCB API to modify/manipulate objects on a Schematic or PCB document respectively, you will need to set the **PreProcess** and **PostProcess** methods so that the environment is updated correctly when you are adding, deleting or modifying objects on a Schematic or PCB document.

IProcessControl Methods

`PostProcess`

`PreProcess`

IProcessControl Properties

`ProcessDepth`

See also

IPCB_ServerInterface for PostProcess and PreProcess methods

ISch_ServerInterface for PostProcess and PreProcess methods

IProcessControl interface methods

PostProcess method

(IProcessControl interface)

Syntax

```
Procedure PostProcess (Const AContext : IInterface; AParameters : PChar);
```

Description

Performs pre processing within in a main server which could involve resetting the environment of the server such as the Undo system. The AContext parameter is usually the focussed document in DXP such as the ISch_Document and IPCB_Board interfaces.

Example

```
// Initialize the robots in Schematic editor.
SchServer.ProcessControl.PreProcess(Doc, '');

// Create a new port and place on current Schematic document.
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 := $FFFF00;
    SchPort.Name      := 'New Port 1';

    // Add a new port object in the existing Schematic document.
    Doc.RegisterSchObjectInContainer(SchPort);
    SchServer.RobotManager.SendMessage(Doc.I_ObjectAddress,c_BroadCast,
                                   SCHM_PrimitiveRegistration,SchPort.I_
ObjectAddress);

    // Clean up the robots in Schematic editor
    SchServer.ProcessControl.PostProcess(Doc, '');
```

See also

PreProcess method

ProcessDepth property

(IProcessControl interface)

Syntax

Property ProcessDepth : Integer;

Description

Sets or gets the process depth. The depth value is an integer value.0 = inactive, and 1 onwards denotes the number of stacked processes.

ProcessDepth Example

```
ShowMessage('Current process depth
',IntToStr(Client.ProcessControl.ProcessDepth));
```

PreProcess method

(IProcessControl interface)

Syntax

```
Procedure PreProcess      (Const AContext : IInterface; AParameters :
PChar);
```

Description

Performs pre processing within in a main server which could involve resetting the environment of the server. The AContext parameter is usually the focussed document in DXP such as the ISch_Document and IPCB_Board interfaces

Example

```
    // Initialize the robots in Schematic editor.
    SchServer.ProcessControl.PreProcess(Doc, '');

    // Create a new port and place on current Schematic document.
    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 := $FFFF00;
    SchPort.Name      := 'New Port 1';

    // Add a new port object in the existing Schematic document.
    Doc.RegisterSchObjectInContainer(SchPort);
    SchServer.RobotManager.SendMessage(Doc.I_ObjectAddress,c_BroadCast,
                                       SCHM_PrimitiveRegistration,SchPort.I_
ObjectAddress);

    // Clean up the robots in Schematic editor
    SchServer.ProcessControl.PostProcess(Doc, '');
```

ICommandLauncher interface

Overview

The **ICommandLauncher** interface encapsulates the functionality of launching a command (which is a pre packaged process) in DXP. A command is associated with a user interface item in the server (Text Editor, Schematic Editor etc) such as a hot key button, menu item or a toolbar bitmap. In essence, a server is supported by its set of processes and the processes act as a link between Design Explorer and this server.

The **LaunchCommand** method launches a process from the server that this **ICommandLauncher** interface function is associated with.

The **GetCommandState** method retrieves information for the specified command.

Since a server has a set of processes and these process identifiers are stored in an installation file (which ends with an INS extension) and the process launchers that link to specific user interface elements (also called resources) and the layout of user interface elements are defined in the resources file (which ends with a RCS extension).

ICommandLauncher Methods	ICommandLauncher Properties
`LaunchCommand`	
`GetCommandState`	

Notes

All the functions in a server available to the user, such as placing a primitive, changing the zoom level and so on are performed by commands which are pre-packaged process launchers. The pre-packaged process launchers bundle together the process that runs when the command is selected, plus any parameters, bitmaps (icons), captions (the name of an item that displays on a resource), descriptions and associated shortcut keys.

When you select a menu item or click on a toolbar button, you are launching a process. Processes are launched by passing the process identifier to the appropriate server and the server then executes the process. Processes are defined and implemented in the Commands unit of a server source code project. The processes are declared in an Installation File (with an INS extension).

Each process has a process identifier. The process identifier is made up of two parts separated by a colon. The first part of the process identifier indicates the server that defines the process, and the second part is the process name.

For example, the process Sch:ZoomIn is provided by Schematic server. When this process is launched, either by selecting a menu item, pressing a hot key or activating a toolbar button (which are all defined as process launchers in the Design Explorer), it will perform the task of zooming in on the currently active schematic sheet.

When a server is started up for the first time in DXP, process procedures or commands registered in the CommandLauncher object within the server module are loaded in DXP.

ICommandLauncher interface Methods

GetCommandState

(ICommandLauncher interface)

Syntax

```
Procedure GetCommandState(      ACommandName,
                                AParameters      : PChar;
                          Const AContext         : IServerDocumentView;
                          Var   Enabled,
                                Checked,
                                Visible          : LongBool;
                                Caption,
                                ImageFile        : PChar);
```

Description

Example

```
ACommandLauncher := AServerModule.GetCommandLauncher;
If ACommandLauncher <> Nil Then
Begin
    ACommandLauncher.GetCommandState(Command,
                                     Parameters,
                                     View,
                                     Enabled,
                                     Checked,
                                     Visible,
                                     Caption,
                                     Image);
    // do what you want with the parameters
    // after you have supplied the Command parameter.
End;
```

See also

IServerModule interface

LaunchCommand

(ICommandLauncher interface)

Syntax

```
Function  LaunchCommand  (Const ACommandName     : PChar;
                                AParameters      : PChar;
                                MaxParameterSize : Integer;
                                AContext         : IServerDocumentView) :
LongBool;
```

Description

This function launches a command from a server module or from Client. (Client also has its own command launcher table since Client has its own processes as well). The AContext parameter denotes which **IServerDocumentView** interface to launch the process onto. If the command can be launched, the function returns a true value.

Example

```
If StringsEqual(ServerModule.ModuleName,'TextEdit') Then
Begin
    ServerModule.CommandLauncher.LaunchCommand('TextEdit:MoveCursorToTopOfDo
cument',
                                               Nil,0,ServerDocument.View[0]
);
End;
```

See also

IServerDocumentView interface

IServerRecord interface

Overview

This interface extracts the servers installation files information from \Altium2004\System folder which has a list of which servers are installed (not running), and this list is populated at DXP start up.

Since this **IServerRecord** interface is inside the Client object, invoke the **Client.GetServerRecordCount** to get the number of server installation files, and then assign the **Client.GetServerRecord(RecordCount)** to a **IServerRecord** variable where you can retrieve data associated with an installation file.

To find more information about each server module installed in Design Explorer, invoke the **IClient.GetServerModule** interface.

IServerRecord Methods	IServerRecord Properties
`GetVersion`	
`GetCopyRight`	
`GetDate`	
`GetSystemExtension`	
`GetGeneralInfo`	
`GetName`	
`GetInsPath`	
`GetExePath`	
`GetDescription`	
`GetServerFileExist`	
`GetRCSFilePath`	
`GetWindowKindCount`	
`GetCommandCount`	
`GetCommand`	
`GetWindowKind`	
`GetWindowKindByName`	
`GetPanelInfo`	
`GetPanelInfoByName`	
`GetPanelInfoCount`	

See also

IClient interface

IServerModule interface

IServerRecord interface Methods

GetCommand method

(IServerRecord interface)

Syntax

```
Function GetCommand(Index : Integer) : IServerProcess;
```

Description

The method returns the IServerProcess interface. Used in conjunction with the GetCommandCount function.

See also

IServerRecord interface

GetCommandCount method

(IServerRecord interface)

Syntax

```
Function GetCommandCount : Integer;
```

Description

The method returns the number of commands (Process launchers) this server supports. Used in conjunction with the GetCommand function

See also

IServerRecord interface

GetCopyRight method

(IServerRecord interface)

Syntax

```
Function GetCopyRight : PChar;
```

Description

The method returns the copyright string.

See also

IServerRecord interface

GetDescription method

(IServerRecord interface)

Syntax

```
Function GetDescription : PChar;
```

Description

The method returns the description string.

See also

IServerRecord interface

GetExePath method

(IServerRecord interface)

Syntax

```
Function GetExePath : PChar;
```

Description

The method returns the path to the server file.

See also

IServerRecord interface

GetDate method

(IServerRecord interface)

Syntax

```
Function GetDate : PChar;
```

Description

The method returns the date string associated with the server installation file.

See also

IServerRecord interface

GetGeneralInfo method

(IServerRecord interface)

Syntax

```
Function GetGeneralInfo : PChar;
```

Description

The method returns the general info string for the server record associated with a server.

See also

IServerRecord interface

GetInsPath method

(IServerRecord interface)

Syntax

```
Function GetInsPath : PChar;
```

Description

The method returns the path to the installation file.

See also

IServerRecord interface

GetName method

(IServerRecord interface)

Syntax

```
Function GetName : PChar;
```

Description

The method returns the name of the server.

See also

IServerRecord interface

GetPanelInfo method

(IServerRecord interface)

Syntax

```
Function GetPanelInfo (Index : Integer) : IServerPanelInfo;
```

Description

The method returns the indexed panel information. This is to be used in conjunction with the GetPanelInfoCount method.

See also

IServerRecord interface

GetPanelInfoByName method

(IServerRecord interface)

Syntax

```
Function GetPanelInfoByName (Const Name  : Widestring) : IServerPanelInfo;
```

Description

The method returns the panel information interface by the panel name.

See also

IServerRecord interface

GetPanelInfoCount method

(IServerRecord interface)

Syntax

```
Function GetPanelInfoCount : Integer;
```

Description

The method returns the number of panels used for the server module. This is to be used in conjunction with the GetPanelInfo method.

See also

IServerRecord interface

GetRCSFilePath method

(IServerRecord interface)

Syntax

```
Function GetRCSFilePath : PChar;
```

Description

The method returns the path to the resources file.

See also

IServerRecord interface

GetSystemExtension method

(IServerRecord interface)

Syntax

```
Function GetSystemExtension : LongBool;
```

Description

The method returns the file system extension string.

See also

IServerRecord interface

GetVersion method

(IServerRecord interface)

Syntax

```
Function GetVersion : PChar;
```

Description

The method returns the version string associated with the server installation file..

See also

IServerRecord interface

GetServerFileExist method

(IServerRecord interface)

Syntax

```
Function GetServerFileExist : LongBool;
```

Description

The method returns the Boolean value whether the server file (with a DLL) exists or not.

See also

IServerRecord interface

GetWindowKind method

(IServerRecord interface)

Syntax

```
Function GetWindowKind      (Index : Integer) : IServerWindowKind;
```

Description

The method returns the IServerWindowKind interface. Used in conjunction with the GetWindowKindCount function.

See also

IServerRecord interface

GetWindowKindCount method

(IServerRecord interface)

Syntax

```
Function GetWindowKindCount : Integer;
```

Description

The method returns the number of document kinds the server supports.

See also

IServerRecord interface

GetWindowKindByName method

(IServerRecord interface)

Syntax

```
Function GetWindowKindByName(Name   : PChar   ) : IServerWindowKind
```

Description

The method returns the IServerWindowKind interface depending on the DocumentKind Name parameter.

See also

IServerRecord interface

IServerWindowKind interface

IServerWindowKind interface

Overview

This **IServerWindowKind** interface reports the type of a design document in Design Explorer and it is a composite object used in **IServerRecord** and **IClient** interface objects

IServerWindowKind Methods	IServerWindowKind Properties
`GetServerRecord`	
`GetName`	
`GetNewWindowCaption`	
`GetNewWindowExtension`	
`GetWindowKindDescription`	
`GetIconName`	
`GetIsDomain`	
`GetIsDocumentEditor`	
`FileLoadDescriptionCount`	
`FileSaveDescriptionCount`	
`GetFileLoadDescription`	
`GetFileSaveDescription`	
`GetWindowKindClassCount`	
`GetWindowKindClass`	
`IsOfWindowKindClass`	

See also

IClient interface

IServerRecord interface

IServerWindowKind interface methods

FileLoadDescriptionCount method

(IServerWindowKind interface)

Syntax

```
Function FileLoadDescriptionCount : Integer;
```

Description

The method returns the number of File Load Descriptions for the document editor type of server. A document editor can support multiple document types and thus facilitate multiple load functions.

See also

IClient interface

IServerWindowKind interface

FileSaveDescriptionCount method

(IServerWindowKind interface)

Syntax

```
Function FileSaveDescriptionCount : Integer;
```

Description

The method returns the number of File Save Descriptions for the document editor server. A document editor can have multiple document types and thus have multiple corresponding file save functions.

See also

IClient interface

IServerWindowKind interface

GetFileLoadDescription method

(IServerWindowKind interface)

Syntax

```
Function GetFileLoadDescription(Index  : Integer) : Widestring;
```

Description

The method returns the indexed file load description. To be used in conjunction with the FileLoadDescriptionCount function.

See also

IClient interface

IServerWindowKind interface

GetFileSaveDescription method

(IServerWindowKind interface)

Syntax

```
Function GetFileSaveDescription(Index  : Integer) : Widestring;
```

Description

The method returns the indexed file save description. To be used in conjunction with the FileSaveDescriptionCount function.

See also

IClient interface

IServerWindowKind interface

GetIconName method

(IServerWindowKind interface)

Syntax

```
Function GetIconName : Widestring;
```

Description

The method returns the name of the icon associated with the server window of a document in DXP.

See also

IClient interface

IServerWindowKind interface

GetIsDocumentEditor method

(IServerWindowKind interface)

Syntax

```
Function GetIsDocumentEditor : Boolean;
```

Description

The method returns a Boolean value whether this server is a document editor or not. Addons are not document editors. A document editor is a server that hosts its own documents and provide editing facilities. For example the PCB Editor is a Document Editor.

See also

IClient interface

IServerWindowKind interface

GetIsDomain

(IServerWindowKind interface)

Syntax

```
Function GetIsDomain : LongBool;
```

Description

The method returns the Boolean value for this Domain. Normally false.

See also

IClient interface

IServerWindowKind interface

GetName method

(IServerWindowKind interface)

Syntax

```
Function GetName : Widestring;
```

Description

Returns the name of the window kind.

See also

IClient interface

IServerWindowKind interface

GetNewWindowCaption method

(IServerWindowKind interface)

Syntax

```
Function GetNewWindowCaption : Widestring;
```

Description

The GetNewWindowCaption method returns the new document caption string for the new document in DXP.

See also

IClient interface

IServerWindowKind interface

GetNewWindowExtension method

(IServerWindowKind interface)

Syntax

```
Function GetNewWindowExtension : Widestring;
```

Description

The method returns the new document's extension string in DXP.

See also

IClient interface

IServerWindowKind interface

GetServerRecord method

(IServerWindowKind interface)

Syntax

```
Function GetServerRecord : IServerRecord;
```

Description

Returns the IServerRecord interface that the IServerWindowKind interface is associated with. Since the server installation file defines document kinds (window kinds) and the IServerRecord interface represents this installation file.

See also

IClient interface

IServerWindowKind interface

GetWindowKindClass

(IExternalForm interface)

Syntax

```
Function GetWindowKindClass (Index  : Integer) : Widestring;
```

Description

The method returns the indexed window kind class.

See also

IClient interface

IServerWindowKind interface

GetWIndowKindClassCount

(IServerWindowKind interface)

Syntax

```
Function GetWindowKindClassCount : Integer;
```

Description

The method returns the number of window kind classes.

See also

IClient interface

IServerWindowKind interface

GetWindowKindDescription method

(IServerWIndowKind interface)

Syntax

```
Function GetWindowKindDescription : Widestring;
```

Description

The method returns the window kind description string for a window in DXP.

See also

IClient interface

IServerWindowKind interface

IsOfWindowKindClass method

(IServerWindowKind interface)

Syntax

```
Function IsOfWindowKindClass(Const AClass : Widestring) : Boolean;
```

Description

The method returns a boolean value whether the class string is part of a window kind class or not.

See also

IClient interface

IServerWindowKind interface

IServerProcess interface

Overview

The IServerProcess returns information for a server process.

IServerProcess Methods	IServerProcess Properties
`GetOriginalId`	
`GetLongSummary`	
`GetParameter`	
`GetParameterCount`	

Notes

All the functions in a server available to the user, such as placing a primitive, changing the zoom level and so on are performed by commands which are pre-packaged process launchers. The pre-packaged process launchers bundle together the process that runs when the command is selected, plus any parameters, bitmaps (icons), captions (the name of an item that displays on a resource), descriptions and associated shortcut keys.

When you select a menu item or click on a toolbar button, you are launching a process. Processes are launched by passing the process identifier to the appropriate server and the server then executes the process. Processes are defined and implemented in the Commands unit of a server source code project. The processes are declared in an Installation File (with an INS extension).

Each process has a process identifier. The process identifier is made up of two parts separated by a colon. The first part of the process identifier indicates the server that defines the process, and the second part is the process name.

For example, the process **Sch:ZoomIn** is provided by Schematic server. When this process is launched, either by selecting a menu item, pressing a hot key or activating a toolbar button (which are all defined as process launchers in the Design Explorer), it will perform the task of zooming in on the currently active schematic sheet.

When a server is started up for the first time in DXP, process procedures or commands registered in the CommandLauncher object within the server module are loaded in DXP.

See also

IServerRecord interface

IServerProcess interface methods

GetLongSummary method

(IServerProcess interface)

Syntax

```
Function GetLongSummary : WideString;
```

Description

The GetLongSummary function returns the Long Summary identifier string.

See also

IServerProcess interface

IServerRecord interface

GetOriginalId method

(IServerProcess interface)

Syntax

```
Function GetOriginalId : WideString;
```

Description

The GetOriginalID method returns the Process Identifier string for the specified server process.

See also

IClient interface

IServerProcess interface

GetParameter method

(IServerProcess interface)

Syntax

Function GetParameter(Index : Integer) : WideString;

Description

The GetParameter function returns the indexed parameter string depending on the index parameter. This is to be used in conjunction with the GetParmeterCount method. A server process can be parametric, and thus can have a number of parameters.

See also

IClient interface

IServerProcess interface

GetParameterCount method

(IServerProcess interface)

Syntax

```
Function GetParameterCount : Integer;
```

Description

The **GetParameterCount** function returns the number of parameters for the current Process Identifier (GetOriginalID). This is to be used in conjunction with the **GetParameter** method.

See also

IClient interface

IServerProcess interface

System Interfaces

IGUIManager interface

Overview

This is a composite Interface of IClient. This IGUIManager interface that deals with the DXP graphical user interface such as dealing with transparent toolbars, setting panels as floating or visible.

IGUIManager Methods

```
LaunchCurrentHotKey
AddKeyStrokeAndLaunch
AddKeyToBuffer
ProcessMessage
ShowTreeAsPopup
InitTransparentToolbars
DoneTransparentToolbars
UpdateTransparentToolbars
StatusBar_GetState
StatusBar_SetState
IsPanelVisibleInCurrentForm
IsPanelValidInCurrentForm
SetPanelVisibleInCurrentForm
SetPanelActiveInCurrentForm
GetPanelIsFloatingInCurrentForm
GetPanelIsOpen
CanResizePanel
ResizePanel
IsSysLevelHotKey
RegisterFloatingWindow
UnregisterFloatingWindow
UpdateInterfaceState
SetFocusLock
GetFocusedPanelName
BeginDragDrop
GetProcessLauncherInfoByID
GetActivePLByCommand
```

IGUIManager Properties

ITimerHandler

Overview

The ITimerHandler interface

ITimerHandler methods

HandleTimerEvent

See also

ITimerManager interface

INotification interface

Caption property

(IExternalForm interface)

See also

IClient interface

IExternalForm interface

IDocumentFormNotification interface

See also

IClient interface

IExternalForm interface

IDocumentNotification interface

Overview

The IDocumentNotification interface represents

IDocumentNotification Methods	IDocumentNotification Properties
	`Code`
	`ServerDocument`
	`OldFileName`

See also

IClient interface

IExternalForm interface

IDocumentRequest interface

See also

IClient interface

IExternalForm interface

IFastCrossSelectNotification interface

IFastCrossSelectionNotification Methods	IFastCrossSelectNotification Properties
	`ObjectType`
	`ObjectDesignator`
	`SourceKind`
	`SelectionState`

See also

IClient interface

IExternalForm interface

IDragDropNotification interface

Notes

Inherited from INotification interface.

IDragDropNotification Methods	IDragDropNotification Properties
`GetCode`	
`GetDragDropObject`	

See also

IDragDropObject interface

IMessagesNotification interface

Overview

The IMessagesNotification interface

IMessagesNotification methods	IMessagesNotification properties
	`Code`

See also

IClient interface

IExternalForm interface

IModuleNotification interface

See also

IClient interface

IExternalForm interface

ISystemNotification interface

(IExternalForm interface)

See also

IClient interface

IExternalForm interface

IViewNotification interface

See also

IClient interface

IExternalForm interface

INotificationHandler

Overview

The **INotificationHandler** interface handles notifications broadcasted in the DXP system. The notifications could be a document loading notification, workspace being loaded, an object being navigated, and a server module being loaded.

Notifications as event messages can be broadcasted by the Client system, and any open server documents can receive them and act on them accordingly. The Broadcast Notification is a system wide notification, and the Dispatch Notification is a server specific notification.

INotificationHandler methods

HandleNotification

See also

IClient interface

ITimerManager interface

Overview

The ITimerManager interface

ITimerManager methods

AddHandler

RemoveHandler

GetHandlerEnabled

SetHandlerEnabled

SetGlobalEnabled

ITimerManager Properties

See also

ITimerHandler interface

IOptionsManager interface

Overview

The IOptionsManager interface deals with generating and reading INI based text files which is used to store settings.

IOptionsManager methods

GetOptionsReader

GetOptionsWriter

OptionsExist

IOptionsManager properties

See also

IOptionsWriter interface

IOptionsReader interface

Overview

The IOptionsReader interface reads data from a text file that has settings.

IOptionsReader methods	**IOptionsReader properties**
ReadBoolean	
ReadDouble	
ReadInteger	
ReadString	
ReadSection	
SectionExists	
ValueExists	

See also

IOptionsWriter interface

IOptionsWriter interface

Overview

The IOptionsWriter interface generates a text file with sections and within each section fields of various types which is like an INI file.

IOptionsWriter methods	**IOptionsWriter properties**
EraseSection	
WriteBoolean	
WriteDouble	
WriteInteger	
WriteString	

See also

IOptionsReader interface

IOptionsManager interface

ITranslationManager interface

Overview

The ITranslationManager interface

ILicenseManager interface

Overview

The ILicenseManager interface deals with licenses associated with servers plugged in DXP.

ILicenseManager methods

UseLicense
ReleaseLicense
ChangeToNetwork
ChangeToStandalone
UseLicenseByName
GetLicenses

ILicenseManager properties

IServerSecurity interface

Overview

The IServerSecurity interface

IServerSecurity methods

IsTechnologySetSupported

IServerSecurity properties

INavigationSystem

Overview

The **INavigationSystem** interface handles .

INavigationSystem methods

```
RegisterNavigationProvider
UnregisterNavigationProtocol
RegisterSpecialURLString
UnregisterSpecialURLString
ParseDestinationString
NavigateTo
ExpandTargets
ValidatedTarget
```

INavigationSystem Properties

See also

IClient interface

Client Enumerated Types

The enumerated types are used for many of the client interfaces and methods which are covered in this section.

See also

Client API Reference

THighlightMethod

THighlightMethodSet

TServerModuleFactory function type

TCommandProc procedure type

TGetStateProc procedure type

TCommandProc procedure type

Syntax

```
TCommandProc  = Procedure(Const AContext : IServerDocumentView; AParameters
: PChar);
```

TGetStateProc procedure type

Syntax

```
TGetStateProc = Procedure(Const AContext : IServerDocumentView; AParameters
: PChar; Var Enabled, Checked, Visible : LongBool; Caption, ImageFile :
PChar);
```

THighlightMethod type

Syntax

```
THighlightMethod =
(eHighlight_Filter,eHighlight_Zoom,eHighlight_Select,eHighlight_Graph,eHighl
ight_Dim,eHighlight_Thicken, eHighlight_ZoomCursor);
```

THighlightMethodSet type

Syntax

```
THighlightMethodSet = Set Of THighlightMethod;
```

TServerModuleFactory function type

Syntax

```
TServerModuleFactory = Function (Const AClient : IClient) : IServerModule;
```

Client Constants

Client Constant values covered in this section.

See also

Client API Reference

Document Notification Codes

View Notification Codes

Module Notification Codes

System Notification Codes

Message Notification Codes

Document Notification codes

```
cDocumentLoading             = 0;
cDocumentOpening             = 1;
cDocumentClosing             = 2;
cDocumentActivating          = 3;
cDocumentNameChanging        = 4;
cDocumentCompiled            = 6;
cDocumentCompiling           = 7;
cDocumentBeforeClose         = 8;
cDocumentProjectChanged      = 9;
cDocumentSaved               = 10;
cDocumentModifiedChanged     = 11;
cDocumentHidden              = 12;
cDocumentProjectActivating   = 15;
cDocumentScrapCompiling      = 16;
cDocumentScrapCompiled       = 17;
cProjectClosing              = 18;
cDocumentWorkspaceLoad_Begin = 101;
cDocumentWorkspaceLoad_End   = 102;
cDocumentWorkspaceSave_Begin = 103;
cDocumentWorkspaceSave_End   = 104;
cDocumentRouterStarted       = 200;
cDocumentRouterStopped       = 201;
```

View Notification codes

```
cDocumentDataInserted             = 0;
```

```
cDocumentDataDeleted              = 1;
cDocumentDataModified             = 2;
cDocumentDataRefresh              = 3;
cApplicationStartupComplete       = 6;
cApplicationShutdownStarted       = 7;
cLicenseDetailsChanged            = 8;
cObjectNavigated                  = 150;
cGroupNavigated                   = 155;
cNavigationHistory                = 160;
cRefreshNavigationPanels          = 170;
cObjectCrossprobed                = 180;
cGroupCrossprobed                 = 185;
cBeginRefreshNavigationPanels     = 190;
```

Module Notification codes

```
cModuleLoaded    = 0;
```

System Notification codes

```
cLibrariesUpdated                 = 0;
cSystemPreferencesChanged         = 1;
cTextEditPreferencesChanged       = 2;
cPCBPreferencesChanged            = 3;
cSchPreferencesChanged            = 4;
cSchPreferencesChangedWithUpdate  = 5;
cCamtasticPreferencesChanged      = 6;
cPCB3DPreferencesChanged          = 7;
cVersionControlPreferencesChanged= 8;
```

Message notification codes

```
cMessagesAdd             = 0;
cMessagesReplaceLast     = 1;
cMessagesFullUpdate      = 2;
cMessagesClearAll        = 3;
```

Client Functions

```
Function Client : IClient;
Function Server : IServerModule;
```

```
Procedure SetClient (Const AClient : IClient);
Procedure SetServer (Const AServer : IServerModule);

Function CreateNewDocumentFromDocumentKind    (Const DocumentKind :
AnsiString) : IServerDocument;
Function CreateNewFreeDocumentFromDocumentKind(Const DocumentKind :
AnsiString) : IServerDocument;

Function GetSceneManager : ISceneManager;
```

Integrated Library API Reference

The Integrated Library Application Programming Interface reference covers interfaces for the Integrated Library objects in the Integrated Library Object Model.

What are Interfaces?

Each method in the interface is implemented in the corresponding class. Interfaces are declared like classes but cannot be directly instantiated and do not have their own method definitions. Each interface, a class supports is actually a list of pointers to methods. Therefore, each time a method call is made to an interface, the interface actually diverts that call to one of it's pointers to a method, thus giving the object that really implements it, the chance to act.

The Integrated Library interfaces exist as long there are associated existing objects in memory, thus when writing a script, you have the responsibility of checking whether the interface you wish to query exists or not before you proceed to invoke the interface's methods.

There are two main interfaces from the Integrated Library Object Model. To obtain the Integrated Library Manager interface that points to the Integrated Library manager object, invoke the **IntegratedLIbraryManager** function in your script which returns you the **IIntegratedLibraryManager** interface. To obtain the model type manager, invoke the **ModelTypeManager** function in your script which returns you the **IModelTypeManger** interface.

Example

```
IntMan := IntegratedLibraryManager;
If IntMan = Nil Then Exit;
```

Main Nexar Interfaces

- IModelTypeManager interface
- IIntegratedLibraryManager interface

Script Examples

There are script examples in the **\Examples\Scripts** folder

See Also

Client/Server Interfaces Overview

Client Server Interfaces

Integrated Library API Reference

Nexar API Reference

PCB API Reference

Schematic API Reference

Work Space Manager API Reference

Server Process Parameters Reference

Integrated Library Overview

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.

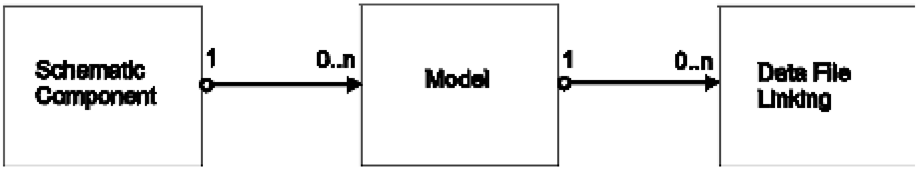
Each domain needs some information about each component, and also some way to map that information to the pins of the schematic component. Some of the domain information resides in model files, the format of which is typically pre-defined, examples of these includes IBIS, MDL and CKT files. Some of the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

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

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 DXP.



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 DXP'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.

See also

IModelTypeManager interface

IIntegratedLibraryManager interface

Integrated Library Interfaces

IModelEditor interface

Overview

The **IModelEditor** interface represents the Model Editor hosted by a server which normally has a dialog that displays data about the model properties in DXP. This **IModelEditor** interface is the front end for the actual implementation of a Model Editor for a specific model domain (PCB, Signal Integrity and other model types).

IModelEditor methods	IModelEditor Properties
EditModel	
CreateDatafile	
StartingLibraryCompile	
FinishedLibraryCompile	
PrepareModel	
CreateServerModel	
GetExternalForm	
DrawModel	
GetEntityParameters	
SetDefaultModelState	
CrossProbeEntity	
DrawModelToMetaFile	

IModelEditor interface methods

CreateDatafile method

(IModelEditor interface)

Syntax

```
Function CreateDatafile (ADatafilePath : PChar) : IModelDatafile;
```

Description

Full data file path is needed to create a model data file and give the entity a name.

See also

IModelEditor interface

EditModel

(IModelEditor interface)

Syntax

```
Function EditModel (SchModel  : ISch_Implementation;
                    SchComp   : ISch_Component;
                    IsLibrary : Boolean) : Boolean;
```

Description

This function is the starting point to invoke the Model Editor supported by a server. A model editor is normally a dialog that provides details about the models that can be linked to the current schematic component.

See also

IModelEditor interface

StartingLibraryCompile method

(IModelEditor interface)

Syntax

```
Procedure   StartingLibraryCompile;
```

Description

Provide functionality during the state that leads up to the compiling process in DXP.

See also

IModelEditor interface

IModelDataFile interface

Overview

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.

IModelDataFile methods	IModelDataFile Properties
FullPath	
EntityCount	
EntityName	
AddEntity	
EntityNames	

IServerModel interface

Overview

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

IServerModel methods

Name

PortCount

PortName

AddPort

CheckSchPins

CheckModelPins

PortNames

IServerModel Properties

IModelType interface

Overview

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

IModelType methods

Name

Description

ServerName

PortDescriptor

Editor

Previewable

IModelType Properties

IModelDataFileType interface

Overview

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

IModelDataFileType methods

FileKind

ExtensionFilter

Description

EntityType

ModelType

SupportsParameters

IModelDataFileType Properties

IModelTypeManager interface

Overview

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

Invoke the **ModelTypeManager** function to fetch the **IModelTypeManager** interface.

IModelTypeManager methods

ModelTypeCount

ModelTypeAt

ModelTypeFromName

ModelTypeFromServerName

ModelDatafileTypeCount

ModelDatafileTypeAt

ModelDatafileTypeFromKind

ModelTypes

ModelDatafileTypes

IModelTypeManager Properties

IIntegratedLibraryManager interface

Overview

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 DXP. 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 DXP.

Invoke the **IntegratedLibraryManager** function to fetch the **IIntegratedLibraryManager** interface.

IIntegratedLibraryManager methods	IIntegratedLibraryManager Properties
CreateIntegratedLibrary	
ExtractSources	
ExtractSourcesToPath	
InstallLibrary	
UninstallLibrary	
AddRemoveLibraries	
GetComponentLocation	
GetComponentDatafileLocation	
FindDatafileInStandardLibs	
ModelCount	
ModelName	
BrowseForComponent	
BrowseForComponentAndPart	
BrowseForDatafile	
PlaceLibraryComponent	
InstalledLibraryCount	
InstalledLibraryPath	
MakeCurrentProject	
AvailableLibraryCount	
AvailableLibraryPath	

AvailableLibraryType

GetComponentCount

ComponentHasModelOfType

GetComponentName

GetModelCount

GetModelType

GetModelName

GetDatafileEntityCount

GetDatafilePath

See also

IModelType

IModelDataFileType

IModelTypeManager

ILibObjectReportInfo interface

Overview

The **ILibObjectReportInfo** interface represents the

ILibObjectReportInfo methods

GetTableCount

GetTableInfo(AIndex : Integer)

GetChildrenTitle

GetChildrenIterator

ILibObjectReportInfo Properties

Integrated Library Enumerated Types

```
TLibraryType = (eLibIntegrated, eLibSource, eLibDatafile, eLibNone,
eLibQuery);
```

Integrated Library Constants

```
cModelType_PCB   = 'PCBLIB';
cModelType_Sim   = 'SIM';
cModelType_PCB3D = 'PCB3DLib';
cModelType_PCAD  = 'PCADLib';
```

```
cModelType_SI    = 'SI';
```

Integrated Library Functions

```
Function ModelTypeManager          : IModelTypeManager;
Function IntegratedLibraryManager : IIntegratedLibraryManager;
```

Nexar API Reference

The Nexar Application Programming Interface reference covers interfaces for Nexar objects in the Nexar Object Model.

What are Interfaces?

Each method in the interface is implemented in the corresponding class. Interfaces are declared like classes but cannot be directly instantiated and do not have their own method definitions. Each interface, a class supports is actually a list of pointers to methods. Therefore, each time a method call is made to an interface, the interface actually diverts that call to one of it's pointers to a method, thus giving the object that really implements it, the chance to act.

The Nexar interfaces exist as long there are associated existing objects in memory, thus when writing a script, you have the responsibility of checking whether the interface you wish to query exists or not before you proceed to invoke the interface's methods.

To obtain the Nexar interface that points to the Nexar editor object, invoke the **GetNexusWorkbench** function in your script which returns you the **INexusWorkbench** interface. Which you can then extract data from Nexar objects and invoke Nexar object's methods.

Example

```
NexusWorkBench := GetNexusWorkbench;
If NexusWorkBench.GetSoftDeviceCount > 0 Then
    SoftDeviceCount := NexusWorkBench.GetSoftDeviceCount
Else Exit;
```

Main Nexar Interfaces

- **INexusWorkbench** interface is the main interface in Nexar API. To use Nexar interfaces, invoke the **GetNexusWorkbench** function to retrieve the **INexusWorkbench** interface.

Script Examples

There are Nexar script examples in the **\Examples\Scripts\DelphiScript\Nexar** folder which demonstrate the use of Nexar interfaces.

See also

Nexar Interfaces Overview

Nexar Interfaces

Nexar Enumerated Types

Nexar functions

Client API Reference

Integrated Library API Reference

PCB API Reference

Schematic API Reference

Work Space Manager API Reference

Server Process Reference

Nexar Interfaces Overview

When you need to deal with the objects associated with the NanoBoard and the Nexar software, the starting point is to invoke the **GetNexusWorkbench** function in your script. This function returns you the **INexusWorkbench** interface object.

Examples

```
Procedure ShowAllInstruments;
Var
    i                : Integer;
    SoftDeviceCount : integer;
    NexusWorkBench  : INexusWorkbench;
Begin
    NexusWorkBench := GetNexusWorkbench;
    If NexusWorkBench.GetSoftDeviceCount > 0 Then
        SoftDeviceCount := NexusWorkBench.GetSoftDeviceCount
    Else Exit;

    For i := 1 To SoftDeviceCount Do
    Begin
        ResetParameters;
        AddStringParameter('Target', 'SoftDevice');
        AddStringParameter('Action', 'ShowViewer');
        AddStringParameter('Index' ,  IntToStr(i));
        RunProcess('FPGAFlow:DeviceAction');
    End;
End;
```

See also

Nexar Interfaces Object Model

Nexar Enumerated Types

Nexar Functions

Nexar Interfaces

Nexar Object Interfaces Model

The GetNexusWorkBench function returns you the INexusWorkBench interface. With this interface you are able to obtain different Nexar interfaces representing the FPGA project.

See also

IBoundaryCell interface

IBSDLEntity interface

IBSDLObject interface

ICoreGenerator interface

IDeviceLink interface

IDeviceInformation interface

IDeviceManager interface

IDevicePin interface

IDeviceIOStandard interface

IInstructionOpCode interface

IInstrumentView interface

IJTagChannel interface

IJTagDevice interface

IJTagPort interface

IMemorySpace interface

INexusBreakPoint interface

INexusCore interface

INexusDevice interface

INexusDriver interface

INexusNotification interface

INexusWorkBench interface

IPCBProjectLink interface

IPinMapping interface

IProcessorRegister interface

IProjectLink interface

IProcessFlow interface

IProcessFlowRunner interface

IRegisterAssocation interface

ISchComponentLink interface

ISoftBreakpoint interface

IBoundaryCell interface

IBoundaryCell's Methods

```
Function    GetCellNumber          : Integer;
Function    GetCellKind            : TCellKind;
Function    GetPortId              : WideString;
Function    GetCellFunction        : TCellFunction;
Function    GetSafeBit             : TBitValue;
Function    GetControlCellNumber   : Integer;
Function    GetDisableValue        : TBitValue;
Function    GetDisableResult       : TDisableResult;
Function    GetPin                 : IScanPin;
Function    GetControlCell         : IBoundaryCell;
Function    GetTargetCell          : IBoundaryCell;
Function    GetCurrentValue        : Integer;
```

IBSDLEntity interface

IBSDLEntity's Methods

```
Function    GetName                 : WideString;
Function    GetInstructionLength : Integer;
Function    GetIdCode               : LongWord;
Function    GetBoundaryLength       : Integer;
Function    GetPin                  (Index : Integer) : IScanPin;
Function    GetPinMapping           (Index : Integer) : IPinMapping;
Function    GetBoundaryCell         (Index : Integer) : IBoundaryCell;
Function    GetInstructionOpcode    (Index : Integer) : IInstructionOpCode;
Function    GetRegisterAssociation(Index : Integer) : IRegisterAssociation;
Function    GetPinCount                   : Integer;
Function    GetPinMappingCount            : Integer;
Function    GetBoundaryCellCount          : Integer;
Function    GetInstructionOpcodeCount     : Integer;
Function    GetRegisterAssociationCount : Integer;
Function    GetPin_TDI                    : IScanPin;
Function    GetPin_TDO                    : IScanPin;
Function    GetPin_TMS                    : IScanPin;
Function    GetPin_TCK                    : IScanPin;
Function    GetPin_TRST                   : IScanPin;
```

```
Function    GetInstruction_BYPASS        : LongWord;
Function    GetInstruction_EXTEST        : LongWord;
Function    GetInstruction_SAMPLE        : LongWord;
Function    GetInstruction_IDCODE        : LongWord;
Function    GetInstruction_USERCODE      : LongWord;
Function    GetInstruction_CLAMP         : LongWord;
Function    GetInstruction_HIGHZ         : LongWord;
Function    GetScanClockMaxSpeed         : Double;
Function    GetScanClockHaltMode         : TScanClockHaltMode;
Function    GetConformanceStatement      : WideString;
Function    GetCompliancePattern         : WideString;
Function    GetDesignWarning             : WideString;
Function    GetFilename                  : WideString;
```

IBSDLObject interface

IBSDLObject's Methods

```
Function    GetDescription      : WideString;
Function    GetShortDescription : WideString;
Function    GetEntity           : IBSDLEntity;
```

ICoreGenerator interface

ICoreGenerator's Methods

```
Function GenerateModule(Params : WideString) : LongBool;
```

IDeviceLink interface

IDeviceLink's Methods

```
Function   ProcessFlow                : IProcessFlow;
Function   DownloadFlow               : IProcessFlow;
Function   AnyFlowsRunning            : Boolean;
Function   GetParent                  : IDeviceLink;
Function   GetChildCount              : Integer;
Function   GetChild(Index : Integer)  : IDeviceLink;
Function   GetNexusDevice             : INexusDevice;
Function   GetNexusCore               : INexusCore;
Function   GetProject                 : IProject;
Function   GetConfiguration           : WideString;
```

```
Function   GetJTAGDevice              : IJTAGDevice;
Function   GetDeviceName              : WideString;
Function   GetUniqueDescription       : WideString;
Function   GetIndex                   : Integer;
Function   GetParentIndex             : Integer;
Function   GetJTAGIndex               : Integer;
Function   GetShortDescription        : WideString;
```

Example

```
Function GetDeviceName(AIndex : Integer) : String;
Var
    NexusWorkBench    : INexusWorkbench;
    DeviceLink        : IDeviceLink;
Begin
    NexusWorkBench := GetNexusWorkbench;
    DeviceLink     := NexusWorkBench.GetSoftDevice(AIndex);
    Result         := DeviceLink.GetDeviceName;
End;
```

IDeviceInformation interface

IDeviceInformation's Methods

```
Function    GetIdCode                         : LongWord;
Function    GetDeviceName                     : WideString;
Function    GetDeviceDescription              : WideString;
Function    GetFamilyName                     : WideString;
Function    GetFamilyDescription              : WideString;
Function    GetLibraryName                    : WideString;
Function    GetVendorPrimitiveLibraryName     : WideString;
Function    GetBaseName                       : WideString;
Function    GetEquivalentGatesForBase         : Integer;
Function    GetMaxIOForBase                   : Integer;
Function    GetPackageTypeId                  : WideString;
Function    GetPackageTypeName                : WideString;
Function    GetPackageTypeDescription         : WideString;
Function    GetPackageId                      : WideString;
Function    GetPackageName                    : WideString;
Function    GetPackageDescription             : WideString;
```

```
Function    GetPackageDimensions           : WideString;
Function    GetTemperatureGradeId          : WideString;
Function    GetTemperatureGradeName        : WideString;
Function    GetTemperatureGradeDescription : WideString;
Function    GetSpeedGradeId                : WideString;
Function    GetSpeedGradeName              : WideString;
Function    GetSpeedGradeDescription       : WideString;
Function    GetOptionDesignatorId          : WideString;
Function    GetOptionDesignatorName        : WideString;
Function    GetOptionDesignatorDescription : WideString;
Function    GetBSDLFilename                : WideString;
Function    GetBSDLEntity                  : IBSDLEntity;
Function    GetVendorName                  : WideString;
Function    GetInstructionLength           : Integer;
Function    GetDevicePinCount              : Integer;
Function    GetDeviceIOCount               : Integer;
Function    GetDevicePin(Index : Integer)  : IDevicePin;
Function    GetDownloadFileExtensions      : WideString;
Function    GetDSPMultiplierCount          : Integer;
Function    GetClockManagerCount           : Integer;
Function    GetMemorySize                  : Integer;
Function    GetTerminationSupport          : Boolean;
Function    GetGlobalClockCount            : Integer;
Function    GetConfigMemorySize            : Integer;
Function    GetDiffPairCount               : Integer;
Function    GetHighSpeedDiffPairCount      : Integer;
Function    GetIOStandardCount             : Integer;
Function    GetIOStandard(Index : Integer) : IDeviceIOStandard;
```

IDeviceManager interface

IDeviceManager's Methods

```
Function    Initialize : LongBool;
Procedure   Finalize;
Function    ChoosePhysicalDevice   (DeviceName  : WideString;
FilterParameters : PChar) : WideString;
```

```
Function    GetDeviceInformation   (DeviceName  : WideString) :
IDeviceInformation;
Function    GetDeviceFromDeviceId  (AnId        : WideString) :
INexusDevice;
Function    GetDevice              (DeviceName  : WideString) :
INexusDevice;
Function    GetDeviceNamesForIdCode(IdCode      : LongWord) : WideString;
Function    AreDevicesPinCompatible(DeviceName1 : WideString; DeviceName2 :
WideString) : LongBool;
Function    GetDefaultDevice : WideString;
Procedure   SetDefaultDevice       (DeviceName        : WideString);
Procedure   ReleaseDevice          (NexusDevice       : INexusDevice);
```

IDevicePin interface

IDevicePin's Methods

```
Function    GetPinNumber                    : WideString;
Function    GetIOBank                       : WideString;
Function    GetIsIOPin                      : Boolean;
Function    GetIsCLKPin                     : Boolean;
Function    GetIsVREFPin                    : Boolean;
Function    GetIsSpecialPin                 : Boolean;
Function    GetIOStandardCount              : Integer;
Function    GetIOStandard(Index : Integer)  : IDeviceIOStandard;
Function    GetIsInputOnlyPin               : Boolean;
Function    GetIsDCIP_Pin                   : Boolean;
Function    GetIsDCIN_Pin                   : Boolean;
Function    GetIsDiffP_Pin                  : Boolean;
Function    GetIsDiffN_Pin                  : Boolean;
Function    GetIsHDiffP_Pin                 : Boolean;
Function    GetIsHDiffN_Pin                 : Boolean;
Function    GetIsTransmitDiffPin            : Boolean;
Function    GetIsReceiveDiffPin             : Boolean;
Function    GetIsClkDiffP_Pin               : Boolean;
Function    GetIsClkDiffN_Pin               : Boolean;
Function    GetIsDLLPin                     : Boolean;
Function    GetDifferentialId               : WideString;
Function    GetClkDifferentialId            : WideString;
```

IDeviceIOStandard interface

IDeviceIOStandard's Methods

```
Function    GetDisplayName                         : WideString;
Function    GetParamName                           : WideString;
Function    GetSlewCount                           : Integer;
Function    GetSlewValue    (Index : Integer)      : WideString;
Function    GetDriveCount                          : Integer;
Function    GetDriveStrength(Index : Integer)      : WideString;
```

IInstructionOpCode interface

IInstructionOpCode's Methods

```
Function    GetInstructionName   : WideString;
Function    GetOpCodePattern     : WideString;
```

IInstrumentView interface

Overview

The Virtual instruments are used for synthesizing into the FPGA design, which can be used to debug the design once it has been implemented on the NanoBoard.

IInstrumentView's Methods

```
Procedure UpdateDisplay;
Procedure SetProjectDetails  (AProjectPath, ATarget : WideString);
Procedure GetInstrumentBounds(Var ALeft, ATop, AWidth, AHeight : Integer);
Procedure SetInstrumentBounds(    ALeft, ATop, AWidth, AHeight : Integer);
Function  LiveUpdateNet      (Const ANet : INet; AUpdateEnabled : Boolean) :
Boolean;
Procedure ReceiveNotification(ANotification : INotification);
```

IJTagChannel interface

IJTagChannel's Methods

```
{-----------------------------------------------------------------}
{Chain Functions                                                  }
Function    OpenChannel : LongBool;

Procedure   CloseChannel;
```

```
Procedure   PlaceAllDevicesInBypass;

Function    GetChainLength : Integer;

Procedure   ScanChain;

Procedure   DisplayStatus;

Function    GetDevice(Index : Integer) : IJTagDevice;

Function    GetDeviceCount : Integer;

Function    GetStringForDevicesOnChain          : WideString;


{-----------------------------------------------------------------}
{General Shift Register Functions - Applied to chain              }
Procedure   DoFlush(BitCount : Integer);

Function    ShiftIn32BitPartialData(Data   : LongWord) : LongWord;

Function    ShiftInNBitPartialData (Data   : LongWord;
                                   Length : Integer ) : LongWord;


{-----------------------------------------------------------------}
{General Shift Register Functions - Applied to Single Device      }
Function    ShiftInOutNBitData     (Data        : LongWord;
                                   BitCount    : Integer;
                                   DeviceIndex : Integer): LongWord;

Function    ShiftInOutString       (Data        : WideString;
                                   DeviceIndex : Integer): WideString;

Function    ShiftInOutPartialString(Data        : WideString;
                                   DeviceIndex : Integer): WideString;

Function    ShiftInOutString_IR    (Data        : WideString;
```

```
                                   DeviceIndex : Integer): WideString;

Function    DataRegisterLoad       (Data        : LongWord;
                                    BitCount    : Integer;
                                    DeviceIndex : Integer) : LongWord;

Procedure   LoadInstruction        (Instruction : LongWord; DeviceIndex :
Integer);
Procedure   SelectInstruction      (Instruction : LongWord; DeviceIndex :
Integer);
Function    ReadCaptureIR          (DeviceIndex : Integer) : LongWord;

Function    DeviceOnline           (DeviceIndex : Integer) : LongBool;

Function    CaptureInstructionRegister(DeviceIndex : Integer) : LongWord;

{-----------------------------------------------------------------}
{TAP State Machine Controller Functions                            }
Procedure   StartShiftDataIn;

Procedure   NextState(ATMS : Integer);

Procedure   ResetTAP;

Procedure   ForceTo_RUN_TEST_IDLE;

Function    CurrentTapState : TTapState;

Procedure   MoveToState_SHIFT_IR;

Procedure   MoveToState_SHIFT_DR;

Procedure   MoveFromState_EXIT1_IR_ToState_SHIFT_DR;

Procedure   MoveFromState_SHIFT_IR_ToState_RUN_TEST_IDLE;

Procedure   MoveFromState_SHIFT_DR_ToState_RUN_TEST_IDLE;

Procedure   MoveFromState_EXIT1_IR_ToState_RUN_TEST_IDLE;
```

```
Procedure   MoveFromState_EXIT1_DR_ToState_RUN_TEST_IDLE;

Procedure   SetTCK(N : Byte);

Procedure   SetTDO(N : Byte);

Procedure   SetTDI(N : Byte);

Procedure   SetTMS(N : Byte);

Function    GetTCK : Byte;

Function    GetTDO : Byte;

Function    GetTDI : Byte;

Function    GetTMS : Byte;

Procedure   Write;

Procedure   Read;


{ This SPI functionality should be in a seperate interface
                       }
{ Perhaps we would have IDevice and IDeviceChannel, from which we could
inherit   }
{ IJTagDevice, ISPIDevice, II2CDevice and their corresponding channel
objects.    }
Procedure   SPI_SetDataPin  (N : Byte);

Procedure   SPI_SetClockPin (N : Byte);

Procedure   SPI_SetSelectPin(N : Byte);

Function    SPI_GetDataPin : Byte;

Function    SPI_SendReceiveByte(DataByte : Byte   ) : Byte;
```

```
Procedure   SPI_SendAddress24  (Address  : Integer);

Procedure   SPI_SendAddress32  (Address  : Integer);

Procedure   SPI_BitDelay;

Function    SPI_SelectDevice(BoardAddress  : Byte;
                             DeviceAddress : Byte) : LongBool;
```

IJTAGChannel's Properties

```
Property    TCK : Byte Read GetTCK Write SetTCK;
Property    TDO : Byte Read GetTDO Write SetTDO;
Property    TDI : Byte Read GetTDI Write SetTDI;
Property    TMS : Byte Read GetTMS Write SetTMS;
```

IJTagDevice interface

IJTagDevice's Methods

```
Function    GetBSDL                 : IBSDLEntity;
Function    GetDeviceInformation : IDeviceInformation;
Function    GetIdCodeFromDevice  : LongWord;
Function    GetInstructionLength : Integer;
Function    GetIdCodeFromBSDL    : LongWord;
Function    GetBoundaryLength    : Integer;
Function    GetIndex             : Integer;
Function    GetFilename          : WideString;
Function    GetCurrentDeviceName : WideString;
Procedure   SetIndex   (Index : Integer   );
Function    UpdateWithPreferredDeviceName(NewName : WideString) : LongBool;
Function    GetPin                 (Index : Integer) : IScanPin;
Function    GetPinMapping          (Index : Integer) : IPinMapping;
Function    GetBoundaryCell        (Index : Integer) : IBoundaryCell;
Function    GetInstructionOpcode   (Index : Integer) : IInstructionOpCode;
Function    GetRegisterAssociation(Index : Integer) : IRegisterAssociation;
Function    ReadPinStatesFromChain(BSDLEntity : IBSDLEntity) : LongBool;
Function    GetPinCount                  : Integer;
Function    GetPinMappingCount           : Integer;
```

```
Function     GetBoundaryCellCount        : Integer;
Function     GetInstructionOpcodeCount   : Integer;
```

IJtagParallelPort_ChannelMapping interface

IJTagParallelPort_ChannelMapping Methods

IJTagParallelPort_ChannelMapping Properties

```
Property     Mask_TCK : Integer
Property     Mask_TDO : Integer
Property     Mask_TDI : Integer
Property     Mask_TMS : Integer
```

IJtagParallelPort interface

IJtagParallelPort's Methods

```
Function     InitializeParallelPort         : LongBool;
Function     GetParallelPort                : IParallelPort;
```

See also

Nexar Interfaces

IJTagPort interface

IJTagPort's Methods

```
Function     Scan(ForceFullScan : Boolean)  : LongBool;
Function     IsConnected                    : LongBool;
Function     GetJTagChannel_Hard            : IJTagChannel;
Function     GetJTagChannel_Soft            : IJTagChannel;
Function     GetJTagChannel_Board           : IJTagChannel;
Function     GetJTagChannel_SPI             : IJTagChannel;
```

IJTagPort's Properties

```
Property     BoardMask_TCK : Integer
Property     BoardMask_TDO : Integer
Property     BoardMask_TDI : Integer
Property     BoardMask_TMS : Integer
```

IMemorySpace interface

IMemorySpace's Methods

```
Function     GetNexusDevice          : INexusDevice;
Function     GetIndex                : Integer;
Function     GetName                 : WideString;
Function     GetShortName            : WideString;
Function     GetWidth                : Integer;
Function     GetLength               : Int64;
Function     GetKind                 : TMemoryKind;
Function     GetBytesPerUnit         : Integer;
Procedure    Read (Buffer   : PMemoryArray;
                   Address  : TMemoryAddress;
                   Length   : Integer);
Procedure    Write(Buffer   : PMemoryArray;
                   Address  : TMemoryAddress;
                   Length   : Integer);
```

INexusBreakPoint interface

INexusBreakPoint's Methods

```
Function     GetNexusDevice   : INexusDevice;
Procedure    Import_FromCore;
Procedure    Export_ToCore;
Function     GetKind          : TNexusBreakpointKind;
Function     GetEnabled       : LongBool;
Function     GetBreakOnRead   : LongBool;
Function     GetBreakOnWrite  : LongBool;
Function     GetData          : Byte;
Function     GetAddressLow    : Word;
Function     GetAddressHigh   : Word;
Function     GetDataMask      : Byte;
Function     GetStopped       : LongBool;
Function     GetIndex         : Integer;
Function     GetDescription   : WideString;
Procedure    SetKind          (Value : TNexusBreakpointKind);
Procedure    SetEnabled       (Value : LongBool            );
Procedure    SetBreakOnRead   (Value : LongBool            );
```

```
Procedure   SetBreakOnWrite (Value : LongBool            );
Procedure   SetData         (Value : Byte                );
Procedure   SetAddressLow   (Value : Word                );
Procedure   SetAddressHigh  (Value : Word                );
Procedure   SetDataMask     (Value : Byte                );
```

INexusCore interface

Overview

An INexusCore interface represents a core in a FPGA device. A core could be one of the following: OCDS processor cores, standard processor cores, memory program core, memory data core or instrument cores.

INexusCore's Methods

```
Function    GetEmbeddedProject           : IProject;
Function    GetFPGAProject               : IProject;
Procedure   DownloadProgramMemory(NexusDevice : INexusDevice);
Function    GetNexusWorkbench            : INexusWorkbench;
Function    GetEmbeddedProjectPath       : WideString;
Function    GetFPGAProjectPath           : WideString;
Function    GetCoreKind                  : TNexusCoreKind;
Function    GetComponentDesignator       : WideString;
Function    GetBaseComponentDesignator   : WideString;
Function    GetLibraryReference          : WideString;
Function    GetComponentDocumentPath     : WideString;
Function    GetMemory_Depth              : Integer;
Function    GetMemory_Width              : Integer;
Function    GetMemory_DepthA             : Integer;
Function    GetMemory_WidthA             : Integer;
Function    GetMemory_DepthB             : Integer;
Function    GetMemory_WidthB             : Integer;
Function    GetMemory_Type               : WideString;
Function    GetMemory_Use                : WideString;
Function    GetMemory_ClockEdge          : TEdgePolarity;
Function    GetMemory_EnablePin          : Boolean;
Function    GetHexFilePath               : WideString;
Function    GetJTagIndex_FromCore        : Integer;
Function    GetHexFilename               : WideString;
```

```
Function    GetHexFileState           : TFlowState;
Function    GetUniqueId               : WideString;
Function    GetDescription            : WideString;

Function    MatchesOtherCore      (ACore      : INexusCore) : Boolean;
Procedure   Import_FromParameters (Parameters : PChar);
Function    FindFileInProjectPaths(FileName   : WideString) : WideString;
Function    SupportsParameter       (Name : WideString) : Boolean;
Function    GetParameterValue       (Name : WideString) : WideString;
Procedure   Export_CoreGenParameters(Parameters : PChar);


Function    GetChildCore          (Index      : Integer   ) : INexusCore;
Function    GetChildCoreDesignator(Index      : Integer   ) : WideString;
Function    GetChildCoreCount                               : Integer;
Function    GetCrossViewProcessId                           : Integer;
Procedure   SetCrossViewProcessId(ProcessID : THandle);
Procedure   CrossProbe;
Function    GetChildProgramMemoryCore : INexusCore;
Procedure   SetEmbeddedProjectPath(AProjectPath : WideString);
Procedure   UpdateHexFileState;
Function    GetChildProgramMemoryCoreCount : Integer;
```

Example

```
Function DeviceDesignator(AIndex : Integer) : String;
Var
    NexusWorkBench     : INexusWorkbench;
    DeviceLink         : IDeviceLink;
    NexusCore          : INexusCore;
    SoftDeviceCOunt    : Intger;
Begin
    NexusWorkBench := GetNexusWorkbench;
    If NexusWorkBench.GetSoftDeviceCount > 0 Then
        SoftDeviceCount := NexusWorkBench.GetSoftDeviceCount
                                                     Else Exit;

    For i := 0 To SoftDeviceCount -1 Do
```

```
    Begin
        DeviceLink     := NexusWorkBench.GetSoftDevice(i);
        NexusCore      := DeviceLink.GetNexusCore;
        ShowMessage(NexusCore.GetComponentDesignator);
End;
```

INexusDevice interface

INexusDevice's Methods

```
Function     JTagIndex                                 : Integer;

Function     JTagChannel                               : IJTagChannel;

Procedure    SetJTagChannel(AJTagChannel : IJTagChannel);

Procedure    SetJTagIndex  (AJTagIndex   : Integer     );

Function     NexusDriver : INexusDriver;

Function     ResetDevice                               : LongBool;

Function     GetDeviceState                            : TDeviceState;

Function     ProgramDevice(BitFilename: WideString)    : Integer;

Function     ReadUserCode                              : LongWord;

Function     FlowStage_Build                           : IProcessFlow;

Function     FlowStage_Download                        : IProcessFlow;

Function     CoreGen_OutputExtension                   : WideString;

Function     GetUniqueId                               : WideString;

Function     GetHexFilePath                            : WideString;

Function     StatusString                              : WideString;
```

```
Function    BitmapHandle                          : THandle;

Procedure   ResetNexusRegisters;

Procedure   SingleStepProcessor;

Procedure   RunSingleStepInSubstitutionRegister;

Procedure   ResetProcessor;

Procedure   PauseProcessor;

Procedure   ContinueProcessor;

Procedure   RunSteps(StepCount: Integer);

Function    ProcessorIsPaused  : LongBool;

Function    ProcessorIsReset   : LongBool;

Function    ProcessorIsRunning : LongBool;

Function    SoftBreakpointCount    : Integer;

Function    ProcessorRegisterCount : Integer;

Function    MemorySpaceCount       : Integer;

Function    GetMemorySpace       (Index   : Integer       ) : IMemorySpace;

Function    GetProcessorRegister (Index   : Integer       ) :
IProcessorRegister;
Function    GetProcessorRegister_PC                         :
IProcessorRegister;
Function    GetProcessorRegister_SP                         :
IProcessorRegister;
Function    GetMemorySpace_Program                          : IMemorySpace;

Function    GetSoftBreakpoint     (MemorySpace : Integer;
```

```
                                    Address     : TMemoryAddress ) :
ISoftBreakpoint;
Function    AddSoftBreakpoint      (MemorySpace : Integer;
                                    Address     : TMemoryAddress ) :
ISoftBreakpoint;
Function    RemoveSoftBreakpoint   (MemorySpace : Integer;
                                    Address     : TMemoryAddress ) :
LongBool;
Procedure   ClearAllSoftBreakpoints;

Function    ReadNexusRegisterAtAddress (Address       : Byte;
                                         BitCount      : Integer       ) :
LongWord;
Procedure   WriteNexusRegisterAtAddress(Address       : Byte;
                                         BitCount      : Integer;
                                         Data          : LongWord);

Function    ReadStringNexusRegisterAtAddress (Address     : Byte;
                                               BitCount    : Integer ) :
WideString;
Procedure   WriteStringNexusRegisterAtAddress(Address     : Byte;
                                               Data        : WideString);

Function    CreateViewer : Boolean;

Function    GetViewer    : IInstrumentView;

Procedure   ShowAboutDialog;

Function    Supports    (Action    : PChar     ) : LongBool;

Function    LittleEndian : LongBool;

Procedure   TemporarySuspend;

Procedure   ContinueFromTemporarySuspend;

Function    RunDiagnostic : LongBool;
```

```
Function    VendorToolsDescriptor : WideString;

Function    VendorToolsPresent    : LongBool;

Function    VendorToolsUpToDate   : LongBool;
```

INexusDriver interface

INexusDriver's Methods

```
{Top Level}
Function    GetNexusDevice : INexusDevice;
Function    SetNexusDevice(ANexusDevice : INexusDevice) : LongBool;
Function    Finalize                                    : LongBool;
Function    GetBitmapHandle                             : THandle;
Function    SaveProcessorState                          : LongBool;
Function    RestoreProcessorState                       : LongBool;
Procedure   CreateCustomViewer;
Function    GetCustomViewer : IInstrumentView;
Function    Supports                   (Action            : PChar) :
LongBool;
Procedure   ShowAboutDialog;
Function    VendorToolsDescriptor : WideString;
Function    VendorToolsPresent    : LongBool;
Function    VendorToolsUpToDate   : LongBool;

{Processor Registers}
Function    SetProcessorRegisterValue(RegisterIndex : Integer;  Value :
TRegisterValue ) : LongBool;
Function    GetProcessorRegisterValue(RegisterIndex : Integer ) : LongWord;
Function    GetProcessorRegisterWidth(RegisterIndex  Integer  ) : LongWord;
Function    GetProcessorRegisterName (RegisterIndex  Integer  ) :
WideString;
Function    GetProcessorRegisterShortName (RegisterIndex : Integer) :
WideString;
Function    GetProcessorRegisterCount   : Integer;
Function    GetRegisterIndex_PC         : Integer;
Function    GetRegisterIndex_SP         : Integer;
```

```
{Software Breakpoints}
Function  AddSoftBreakpoint(MemorySpace      : Integer;
                              Address            : TMemoryAddress) :
LongWord;
Function  RemoveSoftBreakpoint(MemorySpace      : Integer;
                               Address          : TMemoryAddress;
                               Replaced         : TMemoryElement): LongBool;

{Memory Spaces}
Function    GetMemorySpaceWidth          (MemorySpaceIndex : Integer) :
LongWord;
Function    GetMemorySpaceLength         (MemorySpaceIndex : Integer) :
Int64;
Function    GetMemorySpaceKind           (MemorySpaceIndex : Integer) :
TMemoryKind;
Function    GetMemorySpaceBytesPerUnit  (MemorySpaceIndex : Integer) :
LongWord;
Function    GetMemorySpaceName           (MemorySpaceIndex : Integer) :
WideString;
Function    GetMemorySpaceShortName      (MemorySpaceIndex : Integer) :
WideString;
Function    GetMemorySpaceCount      : Integer'
Function    GetMemorySpace_Program   : Integer;
Function    MemoryRead  (MemorySpaceIndex : Integer;
                          Buffer           : PMemoryArray;
                          Address          : LongWord;
                          Length           : Integer             ) :
LongBool;
Function    MemoryWrite (MemorySpaceIndex : Integer;
                          Buffer           : PMemoryArray;
                          Address          : TMemoryAddress;
                          Length           : Integer             ) :
LongBool;
Function    RunDiagnostic : LongBool;

{Physical Device Functions}
Function    ResetDevice                          : LongBool;
```

```
Function    GetDeviceState                          : TDeviceState;
Function    ProgramDevice(FileName : WideString): Integer;
Function    ReadUserCode                            : LongWord;

{Process Flow Functions}
Function    FlowStage_Build                         : IProcessFlow;
Function    FlowStage_Download                      : IProcessFlow;
Function    CoreGen_OutputExtension                 : WideString;

{Processor Control Functions}
Procedure   ResetNexusRegisters;
Procedure   SingleStepProcessor;
Procedure   RunSingleStepInSubstitutionRegister;
Procedure   ResetProcessor;
Procedure   PauseProcessor;
Procedure   ContinueProcessor;
Procedure   RunSteps(StepCount: Integer);
Function    ProcessorIsPaused  : LongBool;
Function    ProcessorIsReset   : LongBool;
Function    ProcessorIsRunning : LongBool;
```

Example

```
Procedure ResetProcessor;
Var
    NexusWorkBench    : INexusWorkbench;
    NexusDevice       : INexusDevice;
    DeviceLink        : IDeviceLink;
Begin
    NexusWorkBench := GetNexusWorkbench;
    DeviceLink := NexusWorkBench.GetCurrentHardDevice;
    NexusDevice := DeviceLink.GetNexusDevice;
    NexusDevice.ResetDevice;
End;
```

INexusNotification

INexusNotification's Properties

```
Property Code : Integer Read GetCode;
```

Nexus Notification types

```
TNexusNotification = Class(TNotificationWithCode, INexusNotification);
```

Nexus Notification codes

```
cNexusDeviceStatusesChanged     = 0;
cNexusHardDeviceChainChanged    = 1;
cNexusSoftDeviceChainChanged    = 2;
cNexusCurrentHardDeviceChanged  = 3;
cNexusCurrentSoftDeviceChanged  = 4;
cNexusProjectCoreTreeChanged    = 5;
cNexusProcessFlowStatusChanged  = 6;
cNexusConnectionStatusChanged   = 7;
cNexusCurrentBoardDeviceChanged = 8;
cNexusBoardDeviceChainChanged   = 9;
cNexusProjectListChanged        = 10;
cNexusOnIdle                    = 11;
cNexusBeginUpdate               = 12;
cNexusEndUpdate                 = 13;
```

INexusWorkbench

Overview

The Nexus Workbench is the big picture of the FPGA development process. There are several stages within the Nexar development system. To obtain the interface of the INexusWorkbench, simply invoke the GetNexusWorkbench function in your script.

In the **Devices** view of the Nexar application, there are three distinct chains, the Nanoboards, the Hard Devices and the Soft Devices.

The Nanoboard represents the hardware integrated development board, the hard devices represent the FPGA devices (as a daughter board plugged on a Nanoboard for example). Each physical device in the Hard Devices region of the Device View has a process flow.

In a FPGA device, you can have a number of cores, the processor core such as a 8051 processor and instrument cores such as logic analyzers all embedded in a FPGA device.

INexusWorkBench's Methods

```
Function  GetIsConnected                                     : LongBool;
```

```
Function  GetNexusCoreFromUniqueId (AUniqueId      : WideString) :
INexusCore;
Function  GetCoreFromDesignator    (Designator     : WideString;
                                    Project        : IProject   ) :
INexusCore;
Function  GetSoftDeviceFromDeviceId(AnId           : WideString) :
IDeviceLink;
Function  GetHardDeviceCount                       : Integer;

Function  GetSoftDeviceCount                       : Integer;

Function  GetBoardDeviceCount                      : Integer;


Function  GetCurrentHardDevice                     : IDeviceLink;

Procedure SetCurrentHardDevice     (ADevice        : IDeviceLink);

Function  GetCurrentSoftDevice                     : IDeviceLink;

Procedure SetCurrentSoftDevice     (ASoftDevice    : IDeviceLink);

Function  GetCurrentBoardDevice                    : IDeviceLink;

Procedure SetCurrentBoardDevice    (ADevice        : IDeviceLink);

Procedure SetCurrentCore           (ACore          : INexusCore);


Function  GetProjectLinkCount                      : Integer;

Function  GetPCBProjectLink        (aIndex         : Integer) :
IPCBProjectLink;
Function  ProcessFlowRunner                        : IProcessFlowRunner;


Function  AddHardDeviceByName        (aDeviceName    : WideString) :
IDeviceLink;
Function  RemoveHardDevice           (aDeviceLink    : IDeviceLink) :
Boolean;
```

```
Procedure SynthesizeCoresForProject  (aProject       : IAbstractVHDLProject;
                                      aConfiguration : WideString);


Function  GetPoll                                   : Boolean;
Procedure SetPoll                          (aValue : Boolean);
Function  GetPollInterval                           : Integer;
Procedure SetPollInterval                  (aValue : Integer);
Function  GetLive                                   : Boolean;
Procedure SetLive                          (aValue : Boolean);
Function  GetGoLiveAtStartup                        : Boolean;
Procedure SetGoLiveAtStartup               (aValue : Boolean);
Function  GetIgnoreSoftwareInHardFlow               : Boolean;
Procedure SetIgnoreSoftwareInHardFlow(aValue : Boolean);
Function  GetIgnoreFPGASourcesInFlow                : Boolean;
Procedure SetIgnoreFPGASourcesInFlow (aValue : Boolean);
Function  GetIgnoreVendorToolsVersion               : Boolean;
Procedure SetIgnoreVendorToolsVersion(aValue : Boolean);
Function  GetShowResultsSummary                     : Boolean;
Procedure SetShowResultsSummary            (aValue : Boolean);
```

INexusWorkBench properties

```
Property HardDevices [aIndex : Integer] : IDeviceLink Read GetHardDevice;
Property SoftDevices [aIndex : Integer] : IDeviceLink Read GetSoftDevice;
Property BoardDevices[aIndex : Integer] : IDeviceLink Read GetBoardDevice;
Property ProjectLinks    [aIndex : Integer] : IProjectLink     Read
GetProjectLink;
Property PCBProjectLinks [aIndex : Integer] : IPCBProjectLink  Read
GetPCBProjectLink;
```

IPCBProjectLink interface

Overview

The FPGA to PCB project linkage which uses the FPGA workspace map to manage linkage and perform updates between linked FPGA and PCB projects.

IPCBProjectLink's Methods

```
Function  GetProject                                                    : IProject;

Function  GetProjectFullPath                                            :
WideString;
Function  DocumentKind                                                  :
WideString;
Function  GetSchComponentLinkCount                  : Integer;

Function  GetSchComponentLink       (Index          : Integer)          :
ISchComponentLink;
Function  ContainsProjectLink       (AProjectLink : IProjectLink) : Boolean;

Procedure CrossProbe;
```

IPinMapping interface

IPinMapping's Methods

```
Function    GetPinName    : WideString;
Function    GetPinNumber  : WideString;
Function    GetScanPin    : IScanPin;
```

IProcessorRegister

IProcessorRegister Methods

```
Function    GetNexusDevice : INexusDevice;
Procedure   Import_FromCore;
Procedure   Export_ToCore;
Function    GetIndex       : Integer;
Function    GetName        : WideString;
Function    GetShortName   : WideString;
Function    GetWidth       : Integer;
Function    GetValue       : TRegisterValue;
Function    GetByte0       : Byte;   {LSB}
Function    GetByte1       : Byte;
Function    GetByte2       : Byte;
Function    GetByte3       : Byte;   {MSB}
Procedure   SetValue(NewValue : TRegisterValue);
```

IProjectLink

IProjectLink's Methods

```
Function  GetProject                                              : IProject;

Function  GetFPGAProject                                          :
IFPGAProject;
Function  GetProjectFullPath                                      : WideString;

Function  DocumentKind                                            : WideString;

Function  ContainsEmbeddedProject (AProjectPath : WideString)     : Boolean;

Procedure CrossProbe;
Function  GetCoreFromDesignator  (Designator  : WideString)       : INexusCore;

Function  GetNexusCoreCount_All                                   : Integer;

Function  GetNexusCoreCount_Instrument                            : Integer;
```

IProjectLink Properties

```
Property NexusCores_All       [aIndex : Integer] : INexusCore Read
GetNexusCore_All;
Property NexusCores_Instrument[aIndex : Integer] : INexusCore Read
GetNexusCore_Instrument;
```

IProcessFlow interface

Overview

A process flow denotes an existing physical device in the Hard Devices region of the Devices View. A process flow is composed of four stages - compile, synthesize, build and program.

An existing process flow has an available project - configuration combination. A configuration should contain a set of constraint files that targets a particular device.

IProcessFlow's Methods

```
Function    StageName                                             : WideString;

Function    StageCaption                                          : WideString;

Function    StageDescription                                      : WideString;
```

```
Function    InputDescription                                    : WideString;

Function    OutputDescription                                   : WideString;

Function    CanRun                                              : LongBool;

Function    IsCategory                                          : LongBool;

Function    Run           (ARunner : IProcessFlowRunner)        :
TFlowRunResult;
Function    RunIfOutOfDate(ARunner : IProcessFlowRunner)        :
TFlowRunResult;
Function    RunAll        (ARunner : IProcessFlowRunner)        :
TFlowRunResult;
Function    RunWholeStage (ARunner : IProcessFlowRunner)        :
TFlowRunResult;
Procedure   Stop;

Function    UpdateFlowState                                     : LongBool;

Function    FlowState                                           :
TProcessFlowState;
Function    IsRunning                                           : LongBool;

Function    RunningStage                                        :
IProcessFlow;
Function    LastRunResult                                       :
TFlowRunResult
Procedure   ClearLastRunResult;

Procedure   ProjectChanged   (aProjectPath : WideString    );

Procedure   SetBasePath      (aPath        : WideString    );

Function    GetBasePath : WideString;
Function    CanEditProperties                                   : LongBool;

Procedure   EditProperties;
```

```
Function    ReportCount                                           : Integer;

Function    ReportPath       (aIndex       : Integer       )    : WideString;

Function    GetParentStage                                        :
IProcessFlow;
Procedure   SetParentStage   (aStage       : IProcessFlow  );

Function    GetChildStageCount                                    : Integer;

Function    GetChildStage    (aIndex       : Integer       )    :
IProcessFlow;
Procedure   ClearChildren;

Procedure   AddChildStage    (aStage       : IProcessFlow  );

Function    GetDependenciesCount                                  : Integer;

Function    GetDependency    (aIndex       : Integer       )    :
IProcessFlow;
Procedure   ClearDependencies;

Procedure   AddDependency    (aStage       : IProcessFlow  );
```

IProcessFlowRunner interface

IProcessFlowRunner's Methods

```
Procedure   FlowStageStarting (AStage                              :
IProcessFlow);
Procedure   AddSummaryLine    (ASection, AKey, AValue, AReportPath :
WideString);
Procedure   AddError          (AType, AFullErrorText               :
WideString);
Procedure   RunStarted;

Procedure   RunEnded;

Function    FlowRunning                                            :
LongBool;
```

IRegisterAssociation interface

IRegisterAssocation's Methods

```
Function     GetRegisterName            : WideString;
Function     GetRegisterLength          : Integer;
Function     GetInstructionOpCodeCount : Integer;
Function     GetInstructionOpCode(Index : Integer) : IInstructionOpCode;
```

IScanPin interface

IScanPin's Methods

```
Function     GetDirection                         : TPinElectrical;
Function     GetPinName                           : WideString;
Function     GetPinNumber                         : WideString;
Function     GetKind                              : TScanPinKind;
Function     GetPinMapping                        : IPinMapping;
Function     GetBoundaryCellCount                 : Integer;
Function     GetBoundaryCell(Index : Integer) : IBoundaryCell;
Function     GetCurrentValue        : Integer;
```

ISchComponentLink interface

ISchComponentLink's Methods

```
Function     GetComponentDesignator    : WideString;
Function     GetLibraryReference       : WideString;
Function     GetComponentDocumentPath : WideString;
Function     GetSubProjectPath         : WideString;
Function     GetDescription            : WideString;
Function     GetProjectPath            : WideString;
Function     GetProjectLink            : IProjectLink;
Procedure    CrossProbe;
Procedure    SetSubProjectPath (AProjectPath : WideString);
```

ISoftBreakpoint interface

ISoftBreakpoint's Methods

```
Procedure    Import_FromCore;
Procedure    Export_ToCore;
Function     GetAddress      : TMemoryAddress;
```

```
Function    GetReplaced    : TMemoryElement;
Function    GetNexusDevice : INexusDevice;
Procedure   SetAddress (Value : TMemoryAddress);
Procedure   SetReplaced(Value : TMemoryElement);
Function    Enable        : LongBool;
Function    Disable       : LongBool;
```

Nexus enumerated types

The enumerated types are used for many of the Nexar interfaces methods which are covered in this section. For example the INexusCore interface has a Function GetCoreKind : TNexusCoreKind;. You can use this Enumerated Types section to check what the range is for the TNexusCoreKind type.

See also

Nexus Control Bits

Nexus Instruction Registers

Nexus Memory Access Bits

TBitValue

TCellFunction

TCellKind

TDeviceIOStandardDriveStrength

TDeviceIOStandardType

TDevicePinType

TDeviceState

TDisableResult

TEdgePolarity

TFlowRunResult

TMemoryElement

TMemoryKind

TNexusAction

TNexusActionTarget

TNexusBreakPointKind

TNexusCoreKind

TNexusNotification

TProcessFlowState

TScanClockHaltMode

TScanPinKind

TTapState

TTargetBoardKind

Nexus Control Bits

```
Nexus_Control_StepCounterBreakpointEnable = Bit_D7;
Nexus_Control_ExternalAccess              = Bit_D6;
Nexus_Control_PeripheralClockEnable       = Bit_D6;
```

```
Nexus_Control_Reset                        = Bit_D5;
Nexus_Control_DebugEnable                  = Bit_D4;
Nexus_Control_DebugAcknowledge             = Bit_D3;
Nexus_Control_DebugRequest                 = Bit_D2;
Nexus_Control_DebugStep                    = Bit_D1;
Nexus_Control_DebuggerProgramSelect        = Bit_D0;
```

Nexus Instruction Registers

```
JTAG_ExTest       = $00;
JTAG_IdCode       = $01;
JTAG_Reset        = $02;
JTAG_Memac        = $0A;
JTAG_NexusEnable = $0B;
JTAG_Bypass       = $0F;
```

Nexus Memory Access Bits

```
Nexus_MemAccess_MemoryRead   = Bit_D2;
Nexus_MemAccess_MemoryWrite = Bit_D1;
Nexus_MemAccess_Data         = 0;
Nexus_MemAccess_Program      = Bit_D0;
Nexus_MemAccess_ReadData     = Nexus_MemAccess_MemoryRead  Or
Nexus_MemAccess_Data;
Nexus_MemAccess_ReadProgram = Nexus_MemAccess_MemoryRead  Or
Nexus_MemAccess_Program;
Nexus_MemAccess_WriteData    = Nexus_MemAccess_MemoryWrite Or
Nexus_MemAccess_Data;
Nexus_MemAccess_WriteProgram= Nexus_MemAccess_MemoryWrite Or
Nexus_MemAccess_Program;
Nexus_MemAccess_Inactive     = 0;
```

TBitValue

```
TBitValue = (eBit_Undefined,eBit_0,eBit_1,eBit_X);
```

TCellFunction

```
TCellFunction =
(
  eCellFunction_Undefined,
  eCellFunction_Input,
```

```
  eCellFunction_Clock,
  eCellFunction_Output2,
  eCellFunction_Output3,
  eCellFunction_Control,
  eCellFunction_ControlR,
  eCellFunction_Internal,
  eCellFunction_BiDir,
  eCellFunction_ObserveOnly
);
```

TCellKind

```
TCellKind =
(
  eCellKind_Undefined,
  eCellKind_BC_0,
  eCellKind_BC_1,
  eCellKind_BC_2,
  eCellKind_BC_3,
  eCellKind_BC_4,
  eCellKind_BC_5,
  eCellKind_BC_6,
  eCellKind_BC_7,
  eCellKind_BC_8,
  eCellKind_BC_9,
  eCellKind_BC_10
);
```

TDeviceIOStandardDriveStrength

```
TDeviceIOStandardDriveStrength = (e2m, e4m, e6m, e8m, e12m, e16m, e24m);
```

TDeviceIOStandardSlewType

```
TDeviceIOStandardSlewType = (eSlow, eFast);
```

TDeviceIOStandardType

```
TDeviceIOStandardType =
(eLVTTL15,
 eLVTTL18,
```

```
eLVTTL25,
eLVTTL33,
eLVCMOS15,
eLVCMOS18,
eLVCMOS25,
eLVCMOS33,
eLVCMOS5,
ePCI33_3,
ePCI33_5,
ePCI66,
ePCIX_3,
eCOMPACTPCI_3,
eSSTL3I,
eSSTL3II,
eSSTL2I,
eSSTL2II,
eSSTL18I,
eSSTL18II,
eGTL,
eGTLP,
eHSTLI,
eHSTLII,
eHSTLIII,
eHSTLIV,
eHSTLI_18,
eHSTLII_18,
eHSTLIII_18,
eHSTLIV_18,
eCTT,
eAGP1x,
eAGP2x,
eTTL,
eLVCMOS12,
eGTL_DCI,
eGTLP_DCI,
eHSTLI_DCI,
```

```
eHSTLII_DCI,
eHSTLIII_DCI,
eHSTLIV_DCI,
eHSTLI_18_DCI,
eHSTLII_18_DCI,
eHSTLIII_18_DCI,
eHSTLIV_18_DCI,
eDHSTLI,
eDHSTLII,
eHTT,
ePCML,
eSSTL18I_DCI,
eSSTL18II_DCI,
eSSTL2I_DCI,
eSSTL2II_DCI,
eSSTL3I_DCI,
eSSTL3II_DCI,
eLVCMOS15_DCI,
eLVCMOS18_DCI,
eLVCMOS25_DCI,
eLVCMOS33_DCI,
eLVCMOS15_DCI_DV2,
eLVCMOS18_DCI_DV2,
eLVCMOS25_DCI_DV2,
eLVCMOS33_DCI_DV2,
eLVDS,
eLVPECL,
eDSSTL2,
eBLVDS25,
eLVPECL25,
eRSDS25,
eLVDS33,
eLVDS25_DCI,
eLVDS33_DCI,
eLVDSEXT25,
eLVDSEXT33,
```

```
 eLVDSEXT25_DCI,
 eLVDSEXT33_DCI,
 eLDT,
 eULVDS25,
 eLDT_DT,
 eLVDS_DT,
 eLVDSEXT25_DT,
 eULVDS25_DT
 );
```

TDevicePinType

```
TDevicePinType = (eIOPin, eVREFPin, eCLKPin, eSpecialPin);
```

TDeviceState

```
TDeviceState = (eDeviceState_Unknown,
                eDeviceState_Reset,
                eDeviceState_Programmed,
                eDeviceState_Programmed_ReadProtected,
                eDeviceState_Programmed_WriteProtected,
                eDeviceState_Programmed_ReadWriteProtected);
```

TDisableResult

```
TDisableResult =
(
  eDisableResult_Undefined,
  eDisableResult_HiZ,
  eDisableResult_Weak0,
  eDisableResult_Weak1,
  eDisableResult_Pull0,
  eDisableResult_Pull1,
  eDisableResult_Keeper
);
```

TEdgePolarity

```
TEdgePolarity = (eEdgeRising,eEdgeFalling);
```

TFlowRunResult

```
TFlowRunResult    = (eFlowRun_DidNotRun, eFlowRun_Cancelled,
eFlowRun_Failure, eFlowRun_Success);
```

TMemoryElement

```
TMemoryElement = LongWord;
```

TMemoryKind

```
TMemoryKind = (
eMemoryKind_Program,
eMemoryKind_Data
);
```

TNexusAction

```
TNexusAction = (eNexusAction_None,
    eNexusAction_ProcessorPause,
    eNexusAction_ProcessorContinue,
    eNexusAction_ProcessorReset,
    eNexusAction_ProcessorSingleStep,
    eNexusAction_ProgramCompile,
    eNexusAction_ProgramDownload,
    eNexusAction_ProgramDebug,
    eNexusAction_ShowAboutDialog,
    eNexusAction_DeviceReset,
    eNexusAction_DeviceDownload,
    eNexusAction_ShowViewer,
    eNexusAction_ChooseFileAndDownload,
    eNexusAction_ProcessorMenu
);
```

TNexusActionTarget

```
TNexusActionTarget =(    eNexusActionTarget_None,
    eNexusActionTarget_SoftDevice,
    eNexusActionTarget_HardDevice,
    eNexusActionTarget_SoftChain,
    eNexusActionTarget_HardChain,
    eNexusActionTarget_BoardDevice,
```

```
    eNexusActionTarget_BoardChain
);
```

TNexusBreakPointKind

```
TNexusBreakpointKind =
(
    eBreakpointKind_Program_FetchAnyOpcode,
    eBreakpointKind_Program_FetchSpecificOpcode,
    eBreakpointKind_DataAtAddress_AnyValue,
    eBreakpointKind_DataAtAddress_SpecificValue
);
```

TNexusCoreKind

```
TNexusCoreKind = (eNexusCoreKind_None,
                  eNexusCoreKind_Processor_OCDS,
                  eNexusCoreKind_Processor_Standard,
                  eNexusCoreKind_Memory_Program,
                  eNexusCoreKind_Memory_Data,
                  eNexusCoreKind_Instrument,
                  eNexusCoreKind_ClockManager);
```

TNexusNotification

```
TNexusNotification = Class(TNotificationWithCode, INexusNotification);
```

TProcessFlowState

```
TProcessFlowState = (eFlowState_UpToDate, eFlowState_OutOfDate,
eFlowState_Missing, eFlowState_None);
```

TRegisterValue

TRegisterValue = LongWord;

TScanClockHaltMode

```
TScanClockHaltMode =
(eScanClockHaltMode_None,eScanClockHaltMode_Low,eScanClockHaltMode_Both);
```

TScanPinKind

```
TScanPinKind = (eScanPinKind_Normal,
                eScanPinKind_TRST,
                eScanPinKind_TCK,
```

```
                eScanPinKind_TDO,
                eScanPinKind_TDI,
                eScanPinKind_TMS,
                eScanPinKind_Power,
                eScanPinKind_Ground);
```

TTapState

```
TTapState =
(
    TapState_Undefined,
    TEST_LOGIC_RESET,
    RUN_TEST_IDLE,
    SELECT_DR,
    CAPTURE_DR,
    SHIFT_DR,
    EXIT1_DR,
    PAUSE_DR,
    EXIT2_DR,
    UPDATE_DR,
    SELECT_IR,
    CAPTURE_IR,
    SHIFT_IR,
    EXIT1_IR,
    PAUSE_IR,
    EXIT2_IR,
    UPDATE_IR
);
```

TTargetBoardKind

```
TTargetBoardKind= (eTargetBoardKind_Unknown,
                   eTargetBoardKind_Xilinx,
                   eTargetBoardKind_Altera,
                   eTargetBoardKind_NanoBoard_NB1_Rev1,
                   eTargetBoardKind_NanoBoard_NB1_Rev2,
                   eTargetBoardKind_BurchBoard);
```

Nexar Constants

Bit Constants

```
Bit31 = $80000000;
Bit30 = $40000000;
Bit29 = $20000000;
Bit28 = $10000000;
Bit27 = $08000000;
Bit26 = $04000000;
Bit25 = $02000000;
Bit24 = $01000000;
Bit23 = $00800000;
Bit22 = $00400000;
Bit21 = $00200000;
Bit20 = $00100000;
Bit19 = $00080000;
Bit18 = $00040000;
Bit17 = $00020000;
Bit16 = $00010000;
Bit15 = $00008000;
Bit14 = $00004000;
Bit13 = $00002000;
Bit12 = $00001000;
Bit11 = $00000800;
Bit10 = $00000400;
Bit9  = $00000200;
Bit8  = $00000100;
Bit7  = $00000080;
Bit6  = $00000040;
Bit5  = $00000020;
Bit4  = $00000010;
Bit3  = $00000008;
Bit2  = $00000004;
Bit1  = $00000002;
Bit0  = $00000001;
```

Nexus functions

```
Function GetNexusWorkbench : INexusWorkbench;

Function GetNexusCoreKindFromParameters    (Parameters : PChar) :
TNexusCoreKind;
Function GetNexusCoreKindFromString        (S : TDynamicString) :
TNexusCoreKind;

Function GetJTagParallelPort  : IJtagParallelPort;
Function GetDeviceManager     : IDeviceManager;
Function GetCoreGenerator     : ICoreGenerator;

Function GetNexusActionFromParameters      (Parameters : PChar) :
TNexusAction;
Function GetNexusActionTargetFromParameters(Parameters : PChar) :
TNexusActionTarget;

Function DeviceIsNanoBoardController(NexusDevice : INexusDevice) : Boolean;
Function IdCodeIsNanoBoardController(IdCode      : LongWord    ) : Boolean;

Function GetMaskedIdCode(Const IdCode : LongWord) : LongWord;

Procedure InitializeMemoryArray    (Var MemoryArray     : TMemoryArray);

Function AddressesEqual            (Addr1, Addr2 : TMemoryAddress) :
Boolean;
Function ElementsEqual             (Element1, Element2 : TMemoryElement) :
Boolean;
Function  GetByteFromElement       (Index : Integer; AElement :
TMemoryElement) : Byte;
Procedure SetByteOnElement         (Index : Integer; Var AElement :
TMemoryElement; AByte : Byte;

Function  DecrementAddress         (Addr : TMemoryAddress; ByValue :
Cardinal) : TMemoryAddress;
Function  IncrementAddress         (Addr : TMemoryAddress; ByValue :
Cardinal) : TMemoryAddress;
```

```
FUnction IsDifferentialIOStandard(Const AStandardName : TDynamicString) :
Boolean;

//Overloaded functions
Function CompareIdCodes(Const IdCodeA : Integer;        Const IdCodeB :
Integer) : Boolean;
Function CompareIdCodes(Const IdCodeA : TDynamicString; Const IdCodeB :
Integer) : Boolean;
```

PCB API Reference

The PCB Application Programming Interface reference covers interfaces for PCB objects such as PCB documents and PCB design objects in the PCB Object Model.

What are interfaces?

Each method in the interface is implemented in the corresponding class. Interfaces are declared like classes but cannot be directly instantiated and do not have their own method definitions. Each interface, a class supports is actually a list of pointers to methods. Therefore, each time a method call is made to an interface, the interface actually diverts that call to one of it's pointers to a method, thus giving the object that really implements it, the chance to act.

The PCB interfaces exist as long there are associated existing objects in memory, thus when writing a script, you have the responsibility of checking whether the interface you wish to query exists or not before you proceed to invoke the interface's methods.

To obtain the PCB interface that points to the PCB editor object, invoke the **PCBServer** function in your script which returns you the **IPCB_ServerInterface** interface. This object interface obtains the PCB editor server object and then you can extract data from PCB objects and invoke PCB object's methods.

For example

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

Main PCB interfaces

- 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 DXP.

Script Examples

There are PCB script examples in the **\Examples\Scripts\PCB** folder which demonstrate the use of PCB interfaces.

See also

PCB Interfaces Overview

PCB Interfaces

PCB Functions

Client API Reference

Integrated Library API Reference

Nexar API

Schematic API Reference

Work Space Manager API Reference

Server Process Reference

Using PCB Interfaces

In this section,the PCB Object Model is facilitated by the PCB Editor in the DXP application. The PCB design objects and methods are available to use in your scripts in all script languages that DXP supports. The PCB design objects are wrapped by their corresponding PCB interfaces that make it possible to manipulate the associated data.

Basically an interface is simply a list of methods that a class declares that it implements. That is, each method in the interface is implemented in the corresponding class. Interfaces are declared like classes but cannot be directly instantiated and do not have their own method definitions. The PCB interfaces exist as long there are associated existing objects in memory, thus when writing code, you have the responsibility of checking whether the interface you wish to query exists or not before you proceed to invoke the interface's methods by checking that this interface doesn't have a **Nil** value or using the **Assigned** function with the interface object in question, such as **If Not Assigned(Board)** Then statement.

When you need to work with PCB design objects in DXP, 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 DXP, you would invoke the **GetCurrentPCBBoard** method from the **IPCB_ServerInterface** interface object.

Few examples below demonstrate the use of PCB interfaces.

Getting the currently open PCB document in DXP example

```
Board := PCBServer.GetCurrentPCBBoard;
If Board = Nil then Exit;
TheFilename := Board.FileName;
```

Fetching pads from a PCB document example

```
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;
```

```
    // Set up an object iterator to look for Pad objects only
    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;
```

See also

There are PCB script examples in the **\Examples\Scripts\Delphiscript Scripts\PCB** folder which demonstrate the use of PCB interfaces and how to fetch information from a PCB document.

Main PCB Interfaces

To use PCB interfaces, you need to obtain the **IPCB_ServerInterface** interface by invoking the **PCBServer** function in a script. The **IPCB_ServerInterface** interface is the gateway to fetching other PCB objects on an open PCB document in DXP.

Main PCB Object Interfaces

- The **IPCB_Primitive** interface is a generic interface used for all PCB design object interfaces.
- The **IPCB_Board** interface represents an existing PCB document in DXP.
- The **IPCB_ServerInterface** interface wraps the PCB server object loaded in DXP.

When you need to work with PCB design objects in DXP, 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 this **IPCB_ServerInterface** interface.

For example to get an access to the current PCB document open in DXP, you would invoke the **GetCurrentPCBBoard** method from the **IPCB_ServerInterface** interface.

GetCurrentPCBBoard Example

```
TheServer := PCBServer;
If TheServer = Nil Then Exit;

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

TheFileName := TheBoard.GetState_FileName;
```

Notes

The **Client** function returns the **IClient** interface representing the executable module of the DXP application and from this interface you can obtain the **IServerModule** which represents a server module loaded in DXP. This **IServerModule** interface enables you to retrieve information about its associated document views and panel views for the specified loaded server in DXP.

```
Var
    ServerModule : IServerModule;
Begin
    If Client = Nil Then Exit;
    ServerModule := Client.ServerModuleByName('PCB');
    ShowMessage('Doc Count = ' + IntToStr(ServerModule.DocumentCount));
End;
```

See also

Client and Server interfaces

IClient interface

IServerModule interface

IPCB_Primitive interface

IPCB_Board interface

IPCB_ServerInterface interface

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 method is a procedure or function is invoked by its interface.
- 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

- 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

```
//Set the Selected value
PCBComponent.Selected := True;

//Get the Selected value
ASelected             := PCBComponent.Selected;
```

Inheritance of PCB interfaces

IPCB_Primitive Interface

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

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.

IPCB_Group interface

A group object interface is a composite container type. This container stores child objects that is, primitives. A footprint in a PCB library, a board outline, polygon, component, coordinate and a dimension on a PCB document are group objects.

For example the X,Y coordinates of the **IPCB_Group** interface usually represents the reference coordinates of the group object such as the component.

IPCB_Rectangular interface

Board Array (embedded board) objects, Text objects and fill objects have rectangular coordinates. These objects are inherited from **IPCB_RectangularPrimitive** interface.

IPCB_AbstractIterator interface and its descendant iterator interfaces.

PCB design objects are accessed by the **IPCB_BoardIterator**

Child objects of group objects are accessed by the **IPCB_GroupIterator**

Footprints of a PCB library are accessed by the **IPCB_LIbraryIterator**

Child objects of a footprint are accessed by the **IPCB_GroupIterator**.

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. See IPCB_LibComponent interface for details.

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. See Client API online reference for details.

Opening a text document, you pass in the 'Text' string along with the full file name string in the **OpenDocument** method. 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.

Opening a text document using Client.OpenDocument method

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

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 in the script. For example, creating a text document, you pass in the 'Text' string to this function.

CreateNewDocumentFromDocumentKind example

```
Var
    Document : IServerDocument;
    Kind     : TDynamicString;
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    : TDynamicString;
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',Path);
        Client.ShowDocument(Doc);
    End;
    // do what you want with the new document.
End;
```

Checking the type of PCB documents in DXP

You can use the **GetCurrentPCBBoard** function from the **PCBServer** object and with the IPCB_Board interface object 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 UpperCase(Client.CurrentView.OwnerDocument.Kind) <> 'PCBLIB' Then Exit;
```

This code snippet uses the **Client.CurrentView.OwnerDocument.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 DXP, it prompts you to save this document. This is facilitated by setting the **IServerDocument.Modified** to true.

Document's Modified property 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;
```

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

```
Procedure ReDrawCurrentBoard(Dummy : Integer = 0);
Var
    P : String;
Begin
    P := 'Action=Redraw';
    MessageRouter_SendCommandToModule('PCB:Zoom', P, 255,
GetState_CurrentEditorWindow);
End;
```

PCB Measurement Units

The PCB editor supports two measurement base units, imperial (mils) and metric (mm). By default the design database base unit deals with mils (thousandths of an inch).

Internal to the PCB design database, the coordinates of all PCB objects are in Internal Units. The internal unit is 1/ 10 000 of a mil or 1 /10,000,000 inch. Note that 1 mil = 10,000 internal units.

Therefore the PCB design objects' dimensions or coordinates are measured in Coord or Internal Unit coordinate values.

There are functions that convert from mils or mm values to a Coord value. See an example that converts from Mils unit values to Coord values.

Examples

```
Via.X         := MilsToCoord(1000);
Via.Y         := MilsToCoord(1000);
Pad.HoleSize  := MilsToCoord(20);
```

Notes

- 1 mil = 10000 internal units
- 1 inch = 1000 mils
- 1 inch = 2.54 cm
- 1 inch = 25.4 mm and 1 cm = 10 mm

Converting Measurement Units

To convert from one measurement unit to another unit, use the following PCB functions:

```
Function  RealToMils      (C : TReal)        : TReal;
Function  RealToMMs       (C : TReal)        : TReal;
Function  CoordToMils     (C : TCoord)       : TReal;
Function  CoordToMMs      (C : TCoord)       : TReal;
Function  MilsToCoord     (M : TReal)        : TCoord;
Function  MMsToCoord      (M : TReal)        : TCoord;
Function  MilsToRealCoord(M : TReal)         : TReal;
Function  MMsToRealCoord (M : TReal)         : TReal;

Function  MetricString    (Var S            : TString;
                               DefaultUnits : TUnit) : Boolean;
Function  ImperialString (Var S             : TString;
                               DefaultUnits  : TUnit) : Boolean;

Procedure StringToCoordUnit(    S : TString;
                            Var C : TCoord;
                            Var U : TUnit);

Procedure StringToRealUnit(    S : TString;
                           Var R : TReal;
                           Var U : TUnit);

Function  CoordUnitToString (C : TCoord;
                             U : TUnit)    : TString;
Function  RealUnitToString  (R : TReal;
                             U : TUnit)    : TString;
```

See also

TCoord type

TUnit type

TReal type

PCB functions

PCB Layers

The PCB Editor supports three types of PCB layers, and the type of layers are;

Electrical Layers

- 32 signal layers (Top, mid1-mid30 and bottom layers)
- 16 internal plane layers

Mechanical Layers

- 16 mechanical layers for defining the board outline, include dimensions, fabrication details and any other mechanical details.

Special Layers

- Top and bottom silkscreen layers
- Solder and paste mask layers
- Drill layers
- Keep out layer
- Multi layer
- Connection layer
- Grid layers
- Hole layers.

Layer Is Used and Layer is Displayed properties

It is possible to control the visibility of PCB layers from a script. You can use the **LayerIsDisplayed** property from the **IPCB_Board** interface to control the visibility of the specified layer for the PCB document. The **LayerIsUsed** property is used to determine if there are objects on the specified layer.

Therefore you have the ability to toggle the **LayerIsDisplayed** and **LayerIsUsed** properties from your script. An example below shows how to enable the visibility for the Top Overlay and BottomOverlay layers.

Example

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

```
    Board.LayerIsDisplayed[eTopOverLay]    := True;
    Board.LayerIsDisplayed[eBottomOverLay] := True;
End;
```

See also

IPCB_Board

IPCB_LayerObject

IPCB_DielectricObject

IPCB_DrillLayerPair

IPCB_InternalPlane

IPCB_MechanicalLayerPair

Using the Layer Stack

The layer stack for a PCB document only deals with copper based layers such as signal and internal plane layers. Each layer in the layer stack can have dielectric information and layer pairs can be specified. However there is a **LayerObject** property in the **IPCB_LayerStack** interface which allows you to access any PCB layer for the PCB board.

Iterating copper layers within the Layer Stack

To query for existing copper layers (signal layers and internal players) within the layer stack, you can use the **FirstLayer** and **NextLayer** properties of the **IPCB_LayerStack** interface to iterate for such layers.

Query the PCB Layer Stack Example

```
Procedure QueryTheLayerStack;
Var
    PCBBoard      : IPCB_Board;
    TheLayerStack : IPCB_LayerStack;
    i             : Integer;
    LayerObj      : IPCB_LayerObject;
    LS            : String;
Begin
    PCBBoard := PCBServer.GetCurrentPCBBoard;
    If PCBBoard = Nil Then Exit;

    // Note that the Layer stack only stores existing copper based layers.
    TheLayerStack := PCBBoard.LayerStack;
    If TheLayerStack = Nil Then Exit;
    LS       := '';
```

```
    LayerObj := TheLayerStack.FirstLayer;
    Repeat
        LS       := LS + Layer2String(LayerObj.LayerID) + #13#10;
        LayerObj := TheLayerStack.NextLayer(LayerObj);
    Until LayerObj = Nil;

    ShowInfo('The Layer Stack has :'#13#10 + LS);
End;
```

Iterating for any PCB layer of a PCB document

To have access to any layer of the PCB document, you can use the **LayerObject** property of the **IPCB_LayerStack** interface. Although the **IPCB_LayerStack** interface basically deals with copper based layers that are used in the layer stack, this Layer Stack interface can be used to look for other PCB layers that are not in the layer stack.

- The **LayerObject** property from this layer stack interface obtains any PCB layer whether it is a keep out layer, top signal layer or a mechanical 16 layer..

Here is the example code that iterates through all 16 mechanical layers. It illustrates five different cases depending on **MechanicalLayerEnabled**, **IsDisplayed** and **UsedByPrim** properties. Note the **IsDisplayed** property needs a **IPCB_Board** parameter.

Note that if a mechanical layer is not enabled (check out the Mechanical Layers section in the Board Layers and Colors dialog) then this mechanical layer cannot be displayed nor have any objects on it. However if you programmatically add an object to a disabled mechanical layer or internal plane layer, it is enabled automatically.

Checking for PCB Mechanical Layers 1 - 16 Example

```
Var
    Board   : IPCB_Board;
    Layer   : TLayer;
    LS      : IPCB_LayerStack;
    LObject : IPCB_LayerObject;
    S       : String;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;

    LS := Board.LayerStack;
    If LS = Nil Then Exit;
    S := '';
```

```
    For Layer := eMechanical1 to eMechanical16 Do
    Begin
        LObject := LS.LayerObject[Layer];
        // If a mechanical layer is not enabled (as per the Board Layers and
Colors dialog) then
        // this layer cannot be displayed nor have any objects on it.
        If Not (LObject.MechanicalLayerEnabled) Then
            S := S + LObject.Name + ' is NOT enabled (thus it cannot be
displayed nor have any objects on it).' + #13
        Else
        Begin
           If (LObject.IsDisplayed[Board] = True) and (LObject.UsedByPrims)
Then
               S := S + LObject.Name + ' is displayed and there are objects
on it.' + #13;
           If (LObject.IsDisplayed[Board] = True) and Not
(LObject.UsedByPrims) Then
               S := S+ LObject.Name + ' is displayed and there are NO
objects on it.' + #13;
           If (LObject.IsDisplayed[Board] = False) and (LObject.UsedByPrims)
Then
               S := S + LObject.Name + ' is NOT displayed and there are
objects on it.' + #13;
           If (LObject.IsDisplayed[Board] = False) and Not
(LObject.UsedByPrims) Then
              S := S + LObject.Name + ' is NOT displayed and there are NO
objects on it.' + #13;
        End;
        // The IsInLayerStack property checks whether the layer is part of
the layer stack or not.
        // If Not LObject.IsInLayerStack
        //     ShowMessage(LObject.Name + 'is not in layer stack!');
    End;
    ShowMessage(S);
End;
```

See also

IPCB_LayerStack

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 which is like a pointer. 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 board outline, component, dimension, coordinate, polygon or a net object and each group object is composed of primitives. A group object is a composite-container object having child objects within or more simply it has its own small database that stores primitives. A component is a group object for example and thus is composed of primitives such as arcs and tracks.

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);
    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 **IPCB_ServerInterface** (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 CreateAVia script in the **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

Looking for PCB Objects

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 application programming interface of the DXP application.

The main function of an iterator is to traverse through the database to fetch certain PCB objects. An iterator traverses the database inside the PCB editor from the external server looking for similar objects. The PCB editor automatically selects which flat or spatial database system to use depending on which iteration method is used.

There are four types of iterators:

- Spatial iterators
- Board 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 footprint within a PCB library.

Board Iterator example

```
Var
    Board     : IPCB_Board;
    Pad       : IPCB_Primitive;
```

```
    Iterator : IPCB_BoardIterator;
Begin
    // Retrieve the current PCB document
    Board     := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;

    // Create an iterator to fetch pad objects
    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
        // do what you want with the fetched pad
        Pad := Iterator.NextPCBObject;
    End;
    Board.BoardIterator_Destroy(Iterator);
```

See also

See the Count_Pads script in the **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

Group Iterator example

```
Procedure CountTracks;
Var
    Track             : IPCB_Primitive;
    TrackIterator     : IPCB_GroupIterator;
    Component         : IPCB_Component;
    ComponentIterator : IPCB_BoardIterator;
    TrackCount        : Integer;
    ComponentCount    : Integer;
Begin
    TrackCount     := 0;
    ComponentCount := 0;
```

```
    If PCBServer.GetCurrentPCBBoard = Nil Then Exit;
    ComponentIterator := PCBServer.GetCurrentPCBBoard.BoardIterator_Create;
    ComponentIterator.AddFilter_ObjectSet(MkSet(eComponentObject));
    Component := ComponentIterator.FirstPCBObject;
    While (Component <> Nil) Do
    Begin
        TrackIterator := Component.GroupIterator_Create;
        TrackIterator.AddFilter_ObjectSet(MkSet(eTrackObject));
        TrackIterator.AddFilter_LayerSet(MkSet(eTopOverlay));
        Track := TrackIterator.FirstPCBObject;
        While (Track <> Nil) Do
        Begin
            Inc(TrackCount);
            Track := TrackIterator.NextPCBObject;
        End;
        ShowInfo('This component ' + Component.SourceDesignator  + ' has ' +
 IntToStr(TrackCount)  + ' tracks.');
        TrackCount := 0;
        Component.GroupIterator_Destroy(TrackIterator);
        Component := ComponentIterator.NextPCBObject;
        Inc(ComponentCount);
        If (ComponentCount > 5) Then Break;
    End;
    PCBServer.GetCurrentPCBBoard.BoardIterator_Destroy(ComponentIterator);
End;
```

See also

IPCB_BoardIterator

IPCB_LibraryIterator

IPCB_SpatialIterator

IPCB_GroupIterator

Creating/deleting PCB objects & updating the Undo system

When PCB objects are created and placed on a current PCB document or existing PCB objects removed from a current PCB document in DXP, the Undo system needs to be notified in one way or other.

For example, the simple CreateAVia script example (in \Examples\Scripts\DelphiScript Scripts\PCB folder) does not refresh the Undo system in the PCB editor.

There are two ways to update the Undo system; refreshing the system as one big undo when multiple objects have been added/deleted to/from the PCB document OR refreshing the system each time when each object has been added/deleted to/from the PCB document.

One big Undo

A/ The sequence is as follows for a one big undo operation when adding multiple PCB objects;

• Invoke the **PCBServer.PreProcess** method which initializes the robots in the PCB server once

• For each new object added, invoke the **PCB.SendMessageToRobots** method with the **PCBM_BoardRegisteration** message

• Invoke the **PCBServer.PostProcess** method which cleans up the robots in the PCB server once

Multiple little Undos

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

1 Invoke the **PCBServer.PreProcess** method which Initializes the Robots in the PCB server

2 Add a new object

3 Invoke the **SendMessageRobots** method with the **PCBM_BoardRegisteration message**

4 Invoke the **PCBServer.PostProcess** method which cleans up the robots in the PCB server

Repeat Steps 1-4 for each PCB object creation.

Adding objects and refreshing the Undo system in PCB editor

```
Var
    Board : IPCB_Board;
    Fill1 : IPCB_Fill1;
    Fill2 : IPCB_Fill2;
Begin
    CreateNewDocumentFromDocumentKind('PCB');
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;

    PCBServer.PreProcess;
    Fill1 := PCBServer.PCBObjectFactory(eFillObject, eNoDimension,
eCreate_Default);
    Fill1.X1Location := MilsToCoord(4000);
    Fill1.Y1Location := MilsToCoord(4000);
    Fill1.X2Location := MilsToCoord(4400);
    Fill1.Y2Location := MilsToCoord(4400);
    Fill1.Rotation   := 0;
    Fill1.Layer      := eTopLayer;
```

```
    Board.AddPCBObject(Fill1);

    // Notify the PCB robots a PCB object has been registered.
    PCBServer.SendMessageToRobots(
      Board.I_ObjectAddress,
      c_Broadcast,
      PCBM_BoardRegisteration,
      Fill1.I_ObjectAddress);

    Fill2 := PCBServer.PCBObjectFactory(eFillObject, eNoDimension,
eCreate_Default);
    Fill2.X1Location := MilsToCoord(5000);
    Fill2.Y1Location := MilsToCoord(3000);
    Fill2.X2Location := MilsToCoord(5500);
    Fill2.Y2Location := MilsToCoord(4000);
    Fill2.Rotation   := 45;
    Fill2.Layer      := eTopLayer;
    Board.AddPCBObject(Fill2);
    // notify the robots pcb object has been registered.
    PCBServer.SendMessageToRobots(
      Board.I_ObjectAddress,
      c_Broadcast,
      PCBM_BoardRegisteration,
      Fill2.I_ObjectAddress);

    // Clean up PCB robots
    PCBServer.PostProcess;
End;
```

See also

See the CreatePCBObjects script in the **\Examples\Scripts\DelphiScript\PCB** folder.

See the Undo script in the **\Examples\Scripts\DelphiScript\PCB** folder.

Removal of objects example

```
Procedure RemoveTracksOnTopLayer;
var
    CurrentPCBBoard : IPCB_Board;
```

```
    Iterator        : IPCB_BoardIterator;
    Track           : IPCB_Track;
    OldTrack        : IPCB_Track;
Begin
    CurrentPCBBoard := PCBServer.GetCurrentPCBBoard;
    If CurrentPCBBoard = Nil Then Exit;
    Iterator := CurrentPCBBoard.BoardIterator_Create;
    If Iterator = Nil Then Exit;
    Iterator.AddFilter_ObjectSet(MkSet(eTrackObject));
    Iterator.AddFilter_LayerSet(MkSet(eTopLayer));

    PCBServer.PreProcess;
    Try
        Track := Iterator.FirstPCBObject;
        While Track <> Nil Do
        Begin
            OldTrack := Track;
            Track := Iterator.NextPCBObject;
            CurrentPCBBoard.RemovePCBObject(OldTrack);
            PCBServer.SendMessageToRobots(
                CurrentPCBBoard.I_ObjectAddress,
                c_BroadCast,
                PCBM_BoardRegisteration,
                OldTrack.I_ObjectAddress);
        End;
    Finally
        CurrentPCBBoard.BoardIterator_Destroy(Iterator);
    End;
    PCBServer.PostProcess;
    Client.SendMessage('PCB:Zoom', 'Action=Redraw', 255,
Client.CurrentView);
End;
```

See also

See the DeletePCBObjects script in the **\Examples\Scripts\DelphiScript\PCB** folder

Modifying PCB objects and updating the Undo system

To modify PCB objects on a current PCB document, you will need to invoke certain PCB interface 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 sequence is as follows

- Invoke the **PCBServer.PreProcess** method to Initialize the robots in the PCB server
- Invoke the **SendMessageToRobots** method with a **PCBM_BeginModify** parameter
- Modify the PCB object
- Invoke the **SendMessageToRobots** method with a **PCBM_EndModify** parameter
- Invoke the **PCBServer.PostProcess** method to clean up the robots in the PCB server

Changing PCB object's attributes example

```
Var
    Board : IPCB_Board;
    Fill  : IPCB_Fill;
{.............................................................................
...}
{.............................................................................
...}
Procedure CreateObject(Dummy : Integer = 0);
Begin
    PCBServer.PreProcess;
    Fill := PCBServer.PCBObjectFactory(eFillObject, eNoDimension,
eCreate_Default);
    Fill.X1Location := MilsToCoord(4000);
    Fill.Y1Location := MilsToCoord(4000);
    Fill.X2Location := MilsToCoord(4400);
    Fill.Y2Location := MilsToCoord(4400);
    Fill.Rotation   := 0;
    Fill.Layer      := eTopLayer;

    // Adds the Fill object into the PCB document
    Board.AddPCBObject(Fill);
    PCBServer.PostProcess;
End;
{.............................................................................
.....}
```

```
{..............................................................................
.....}
Procedure ModifyObject(Dummy : Integer = 0);
Begin
    PCBServer.PreProcess;

    //Notify PCB that the fill object is going to be changed.
    PCBServer.SendMessageToRobots(
        Fill.I_ObjectAddress,
        c_Broadcast,
        PCBM_BeginModify ,
        c_NoEventData);

    Fill.Layer := eBottomLayer;

    //Notify PCB that the fill object has been changed.
    PCBServer.SendMessageToRobots(
        Fill.I_ObjectAddress,
        c_Broadcast,
        PCBM_EndModify ,
        c_NoEventData);

    PCBServer.PostProcess;
End;
{..............................................................................
....}
{..............................................................................
....}
Procedure RemoveObject(Dummy : Integer = 0);
Begin
    PCBServer.PreProcess;
    //Remove the fill object.
    Board.RemovePCBObject(Fill);
    PCBServer.PostProcess;
End;
{..............................................................................
....}
```

```
{.........................................................................
....}
Procedure CreateModifyRemoveAObject;
Begin
    CreateNewDocumentFromDocumentKind('PCB');
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;

    ShowInfo('Creating an object');
    CreateObject;
    Client.SendMessage('PCB:Zoom', 'Action=Redraw', 255,
Client.CurrentView);

    ShowInfo('Modifying this object');
    ModifyObject;
    Client.SendMessage('PCB:Zoom', 'Action=Redraw', 255,
Client.CurrentView);

    ShowInfo('Undoing the modification');
    RemoveObject;
    Client.SendMessage('PCB:Zoom', 'Action=Redraw', 255,
Client.CurrentView);
End;
```

Notes

When you change the properties of a PCB object on a PCB document, you need to employ the **PCBServer.SendMessageToRobots** method 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** interface.

See also

See the ModifyPCBObjects example in the **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

See the RotateAComponent Script in the **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

PCB Interactive feedback using the mouse

To find whether PCB design objects on the PCB/Library document are selected or not by the user, you can check the Selected property of a PCB design object interface. All design objects are inherited from the IPCB_Primitive interface which has the Selected property.

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 GetObejctAtCursor except that if there are objects occupying the same region on the PCB document. 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

```
Function  GetObjectAtCursor(ObjectSet       : TObjectSet;
                            LayerSet        : TLayerSet;
                            StatusBarText   : TString)  : IPCB_Primitive;

Function  GetObjectAtXYAskUserIfAmbiguous(HitX,
                                          HitY            : TCoord;
                                          ObjectSet       : TObjectSet;
                                          LayerSet        : TLayerSet;
                                          Action          : TEditingAction)
: IPCB_Primitive;

Function  ChooseRectangleByCorners(Prompt1         : TString;
                                   Prompt2         : TString;
                                   Var X1, Y1,
                                       X2, Y2      : TCoord)   : Boolean;

Function  ChooseLocation(Var X1, Y1       : TCoord;
                         Prompt           : TString)  : Boolean
```

ChooseRectangleByCorners Example

```
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 is a routine that queries PCB internal
    document data within the 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

IPCB_Board interface

SpatialIterator script in **\Scripts\Examples\DelphiScript Scripts\PCB** folder.

ShowPadProperties script in **\Scripts\Examples\DelphiScript Scripts\PCB** folder.

PCB Interfaces

PCB Object Model

An interface is just a means of access to an object in memory. To have access to the PCB server and massage 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.

A simplified PCB Interfaces hierarchy

IPCB_Primitive

    IPCB_Arc

    IPCB_Group

        IPCB_Net

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 document.

See also

IPCB_ServerInterface

IPCB_BoardOutline

IPCB_Board

IPCB_LayerStack

IPCB_LayerObject

IPCB_InternalPlane

IPCB_DrillLayerPair

IPCB_MechanicalLayerPairs

IPCB_SystemOptions

IPCB_InteractiveRoutingOptions

IPCB_Arc

IPCB_Pad

IPCB_Via

IPCB_Track

IPCB_Connection

IPCB_Embedded

IPCB_Violation

IPCB_Text

IPCB_Fill

IPCB_Coordinate

IPCB_Dimension

IPCB_Component

IPCB_Polygon

IPCB_Net

IPCB_LibComponent

IPCB_ServerInterface

Overview

When you need to work with PCB design objects in DXP, the starting point is to invoke the **PCBServer** function which returns the **IPCB_ServerInterface** interface. You can extract the all other derived PCB interfaces that are exposed in the I**PCB_ServerInterface** interface.

Note that these **IServerModule** interfaces represent loaded servers in DXP. The DXP application manages single instances of different server modules. Each server can have multiple server document kinds, for example the PCB server supports two server document kinds – PCB and PCBLIB design documents. A loaded server in DXP typically hosts documents and each document in turn hosts a document view and panel views. Thus a PCB server also has the **IServerModule** interface along with the **IPCB_ServerInterface** interface.

Notes

- To get an access to the current PCB document open in DXP, you would invoke the **GetCurrentPCBBoard** method from the **IPCB_ServerInterface** interface object to obtain the **IPCB_Board** interface.
- The factory methods produce specialized objects. For example the **PCBObjectFactory** method is invoked to produce a new PCB object. You will need to add this object in a PCB board. The **TObjectCreationKind** type denotes how the attributes of a new PCB object is set (either from software default settings or from global settings as defined in the Preferences dialog within PCB).
- The **SendMessageToRobots**, **PreProcess and PostProcess** methods are used when you need to keep the Undo system and other sub systems of the PCB editor in synchronization, when you are adding, deleting or modifying objects to/from the PCB document.

IPCB_ServerInterface methods

```
PCBObjectFactory
PCBClassFactory
PCBClassFactoryByClassMember
PCBRuleFactory
PCBLibCompFactory
DestroyPCBObject
DestroyPCBLibComp

GetPCBBoardByPath
GetCurrentPCBBoard

GetCurrentComponent
ObjectSupports

PreProcess
PostProcess
SendMessageToRobots
```

IPCB_ServerInterface properties

```
InteractiveRoutingOptions
SystemOptions
```

See also

Creating/Deleting PCB objects and updating the Undo system

Modifying PCB objects and updating the Undo system

TObjectId enumerated values

TDimensionKind enumerated values

TObjectCreationMode enumerated values

IPCB_ObjectClass interface

IPCB_Rule interface

IPCB_LibComponent interface

IPCB_Primitive interface

IPCB_Board interface

IPCB_SystemOptions interface

IPCB_InteractiveRoutingOptions interface

PCB Scripts from **\Examples\Scripts\Delphiscript\PCB** folder.

IPCB_ServerInterface methods

DestroyPCBLibObject method

(IPCB_ServerInterface interface)

Syntax

```
Procedure DestroyPCBLibComp  (Var APCBLibComp      : IPCB_LibComponent);
```

Description

This procedure destroys a footprint within a library but it is not eliminated from the computer's memory. A library is composed of footprints as pages and each footprint is represented by the **IPCB_LibComponent** interface.

See also

IPCB_ServerInterface interface

PCBDestroyObject method

(IPCB_ServerInterface interface)

Syntax

```
Procedure DestroyPCBObject   (Var APCBObject       : IPCB_Primitive);
```

Description

This procedure destroys a PCB object from the PCB document. It is removed but not eliminated from computer memory. For instance, the Undo system can bring this object back.

Example

```
var
    CurrentPCBBoard : IPCB_Board;
    Iterator        : IPCB_BoardIterator;
    Track           : IPCB_Track;
    OldTrack        : IPCB_Track;
Begin
    CurrentPCBBoard := PCBServer.GetCurrentPCBBoard;
    If CurrentPCBBoard = Nil Then Exit;

    Iterator := CurrentPCBBoard.BoardIterator_Create;
    If Iterator = Nil Then Exit;
    Iterator.AddFilter_ObjectSet(MkSet(eTrackObject));
    Iterator.AddFilter_LayerSet(MkSet(eTopLayer));
    PCBServer.PreProcess;
```

```
    Try
        Track := Iterator.FirstPCBObject;
        While Track <> Nil Do
        Begin
            OldTrack := Track;
            Track := Iterator.NextPCBObject;
            CurrentPCBBoard.RemovePCBObject(OldTrack);
            PCBServer.SendMessageToRobots(CurrentPCBBoard.I_ObjectAddress,
                                          c_BroadCast,
                                          PCBM_BoardRegisteration,
                                          OldTrack.I_ObjectAddress);
        End;
    Finally
        CurrentPCBBoard.BoardIterator_Destroy(Iterator);
    End;
    PCBServer.PostProcess;

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

See also

IPCB_ServerInterface interface

GetCurrentComponent method

(IPCB_ServerInterface interface)

Syntax

```
Function  GetCurrentComponent(Const Board : IPCB_Board) : TPCBString;
```

Description

This function returns the current footprint's name for the focussed and opened library in DXP. This can be used when a PCB library is being iterated for the current footprint.

Example

```
// For each page of library is a footprint
FootprintIterator := CurrentLib.LibraryIterator_Create;
```

```
FootprintIterator.SetState_FilterAll;

Try
   Footprint := FootprintIterator.FirstPCBObject; // IPCB_LibComponent
   While Footprint <> Nil Do
   Begin
       S := S + ' Current Footprint : ' +
            PCBServer.GetCurrentComponent(CurrentLib) + #13 + #13;

       S := S + Footprint.Name;
       Footprint := FootprintIterator.NextPCBObject;
    End;
 Finally
        CurrentLib.LibraryIterator_Destroy(FootprintIterator);
End;
ShowMessage(S);
```

See also

IPCB_ServerInterface interface

TPCBString type

LibraryIterator example in \DelphiScript Scripts\PCB\ folder.

GetCurrentPCBBoard method

(IPCB_ServerInterface interface)

Syntax

```
Function  GetCurrentPCBBoard : IPCB_Board;
```

Description

This function returns you the **IPCB_Board** interface which represents the PCB document OR the PCB Library document. The **IPCB_Board** interface has a **IsLibrary** function which determines which type the document is; the PCB or PCBLib document.

Example

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

```
    If Not Board.IsLibrary Then
    Begin
        showMessage('This is not a PCB library document.');
        Exit;
    End;
End;
```

See also

IPCB_ServerInterface interface

GetPCBBoardByPath method

(IPCB_ServerInterface interface)

Syntax

```
Function GetPCBBoardByPath (APath : TPCBString) : IPCB_Board;
```

Description

This function returns you the IPCB_Board interface representing the PCB document or the PCB Lib document only if the path (APath parameter) represents this document.

See also

IPCB_ServerInterface interface

ObjectSupports method

(IPCB_ServerInterface interface)

Syntax

```
Function ObjectSupports(Const Instance : TObject; Const IID : TGUID; Out
Intf) : Boolean;
```

Description

This function checks if the object in question is in fact one of the PCB interfaces.

See also

IPCB_ServerInterface interface

PCBClassObjectFactory method

(IPCB_ServerInterface interface)

Syntax

```
Function  PCBClassFactory(Const AClassKind : TObjectId) : IPCB_ObjectClass;
```

Description

This function produces an object represented by the **IPCB_ObjectClass** interface. An Object class is a Design Rules Class that can store members which represent a group of design objects targetted by the design rules system in the PCB editor.

See also

IPCB_ServerInterface interface

PCBClassObjectFactoryByClassMember method

PCBClassObjectFactoryByClassMember method

(IPCB_ServerInterface interface)

Syntax

```
Function  PCBClassFactoryByClassMember (Const AClassKind : TClassMemberKind)
: IPCB_ObjectClass;
```

Description

This function produces an object represented by the IPCB_ObjectClass interface. An Object class is a Design Rules Class that can store members which represent a group of design objects targetted by the design rules system in the PCB editor.

See also

IPCB_ServerInterface interface

PCBClassObjectFactory method

PCBLibCompFactory method

(IPCB_ServerInterface interface)

Syntax

```
Function  PCBLibCompFactory(Const APattern : TPCBString;
                            Const APath    : TPCBString) :
IPCB_LibComponent;
```

Description

This function produces an object which is represented by the **IPCB_LibComponent** interface. A footprint in a library is also represented by the **IPCB_LibComponent** interface.

See also

IPCB_ServerInterface interface

PCBObjectFactory method

(IPCB_Board interface)

Syntax

```
Function PCBObjectFactory(Const AObjectId      : TObjectId;
                          Const ADimensionKind : TDimensionKind =
eNoDimension;
                          Const ACreationMode  : TObjectCreationMode =

                          eCreate_Default) : IPCB_Primitive;
```

Description

This function produces a PCB design object which is represented by the IPCB_Primitive interface. The IPCB_Primitive interface is the ancestor interface for all PCB design objects in DXP.

The TObjectID value determines which object you wish to produce.

The TDimensionKind value determines which dimension object you wish to produce. By default it is eNoDimension.

The TObjectCreationMode type determines which default values are used - from the PCB Preferences dialog or default values used internally from the PCB Editor.

Example

```
Var
    Board : IPCB_Board;
    Via   : IPCB_Via;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;
    // 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 the new Via object in the board object
    Board.AddPCBObject(Via);

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

```
End;
```

See also

IPCB_ServerInterface interface

PCBRuleFactory method

(IPCB_ServerInterface interface)

Syntax

```
Function  PCBRuleFactory(Const ARuleKind : TRuleKind) : IPCB_Rule;
```

Description

This function produces a design rule object which is represented by the **IPCB_Rule** interface.

See also

IPCB_ServerInterface interface

PostProcess method

(IPCB_ServerInterface interface)

Syntax

```
Procedure PostProcess;
```

Description

This procedure cleans up the robots process in the PCB editor, after a **PreProcess** method and **SendMessageToRobots** messages have been invoked. This also stops the robots from listening to any more PCB messages.

Example

```
PCBServer.PreProcess;

//Notify PCB that the fill object is going to be changed.
PCBServer.SendMessageToRobots(
        Fill.I_ObjectAddress,
        c_Broadcast,
        PCBM_BeginModify ,
        c_NoEventData);

Fill.Layer := eBottomLayer;

//Notify PCB that the fill object has been changed.
```

```
PCBServer.SendMessageToRobots(
        Fill.I_ObjectAddress,
        c_Broadcast,
        PCBM_EndModify ,
        c_NoEventData);

PCBServer.PostProcess;
```

See also

IPCB_ServerInterface interface

PreProcess method

SendMessageToRobots method

Preprocess method

(IPCB_ServerInterface interface)

Syntax

```
Procedure PreProcess;
```

Description

This procedure initializes the PCB robots in the PCB editor so that the robots can listen to any PCB messages being broadcasted.

Example

```
PCBServer.PreProcess;

//Notify PCB that the fill object is going to be changed.
PCBServer.SendMessageToRobots(
        Fill.I_ObjectAddress,
        c_Broadcast,
        PCBM_BeginModify ,
        c_NoEventData);

Fill.Layer := eBottomLayer;

//Notify PCB that the fill object has been changed.
PCBServer.SendMessageToRobots(
        Fill.I_ObjectAddress,
        c_Broadcast,
```

```
        PCBM_EndModify ,
        c_NoEventData);

PCBServer.PostProcess;
```

See also

IPCB_ServerInterface interface

PostProcess method

SendMessageToRobots method

SendMessageToRobots method

(IPCB_ServerInterface interface)

Syntax

```
Procedure SendMessageToRobots(Source, Destination : Pointer; MessageID :
Word; MessageData : Pointer);
```

Description

The **SendMessageToRobots** method sends a specific Message with the Source and Designation parameters into the PCB editor where the PCB robots are listening. It is necessary to invoke the **PreProcess** method first, and to invoke the **PostProcess** method after the **SendMessageToRobots** methods.

Parameters

- The **Source** parameter represents the PCB object. You need to pass in the address of this object, thus the **I_ObjectAddress** method of a PCB Object Interface returns the address.
- The **Destination** parameter normally has the **c_Broadcast** constant which denotes that the message is being broadcasted into the PCB editor.
- The **MessageId** parameter represents one of the PCB message constants. See PCB Messages section for more details.
- The **MessageData** parameter can be one of the following values - **c_NoEventData** when a PCB object is being modified, or when this object is being registered into the PCB editor, and you need to pass in the address of this object, thus the **I_ObjectAddress** method of a PCB Object Interface need to be invoked to return the address.

Notes

The PCB Messages are messages that are broadcasted into the PCB Editor server by the **SendMessageToRobots** method. There are different types of messages that describe a specific action within the PCB server.

Example 1 - SendMessageToRobots with BeginModify and EndModify calls

```
//Initialize robots in PCB
PCBServer.PreProcess;
```

```
//Notify PCB that the fill object is going to be changed.
PCBServer.SendMessageToRobots(
        Fill.I_ObjectAddress,
        c_Broadcast,
        PCBM_BeginModify ,
        c_NoEventData);

Fill.Layer := eBottomLayer;

//Notify PCB that the fill object has been changed.
PCBServer.SendMessageToRobots(
        Fill.I_ObjectAddress,
        c_Broadcast,
        PCBM_EndModify ,
        c_NoEventData);

// Clean up robots in PCB
PCBServer.PostProcess;
```

Example 2 - SendMessageToRobots with BoardRegistration call

```
//Initialize robots in PCB
PCBServer.PreProcess;

//Create a text object;
TextObj := PCBServer.PCBObjectFactory(eTextObject, eNoDimension,
eCreate_Default);

// notify the event manager that the pcb object is going to be modified
PCBServer.SendMessageToRobots(TextObj.I_ObjectAddress ,c_Broadcast,
PCBM_BeginModify , c_NoEventData);

TextObj.XLocation := Sheet.SheetX + MilsToCoord(100);
TextObj.YLocation := Sheet.SheetY + MilsToCoord(100);
TextObj.Layer     := eTopOverlay;
TextObj.Text      := 'Text1';
TextObj.Size      := MilsToCoord(90);   // sets the height of the text.
```

```
Board.AddPCBObject(TextObj);

// notify the event manager that the pcb object has been modified
PCBServer.SendMessageToRobots(TextObj.I_ObjectAddress, c_Broadcast,
PCBM_EndModify          , c_NoEventData);

// notify that the pcb object has been registered in PCB.
PCBServer.SendMessageToRobots(Board.I_ObjectAddress, c_Broadcast,
PCBM_BoardRegisteration, TextObj.I_ObjectAddress);

// Clean up robots in PCB
PCBServer.PostProcess;
```

See also

IPCB_ServerInterface interface

PostProcess method

SendMessageToRobots method

PCB Message Constants

IPCB_ServerInterface properties

InteractiveRoutingOptions property

(IPCB_ServerInterface interface)

Syntax

```
Property  InteractiveRoutingOptions  : IPCB_InteractiveRoutingOptions Read
GetState_InteractiveRoutingOptions;
```

Description

This property returns you the **IPCB_InteractiveRoutingOptions** interface which represents the interactive routing options in the PCB editor.

See also

IPCB_ServerInterface interface

IPCB_InteractiveRoutingOptions interface

SystemOptions property

(IPCB_ServerInterface interface)

Syntax

```
Property  SystemOptions : IPCB_SystemOptions Read GetState_SystemOptions;
```

Description

The property returns you the **IPCB_SystemOptions** interface. This interface is represented by the System Options in the PCB editor.

See also

IPCB_ServerInterface interface

IPCB_SystemOptions interface

CanFastCrossSelect_Receive method

(IPCB_Board interface)

Syntax

```
Property  CanFastCrossSelect_Receive : Boolean Read
GetState_CanFastCrossSelect_Receive Write
SetState_CanFastCrossSelect_Receive;
```

See also

IPCB_ServerInterface interface

IPCB_SystemOptions interface

CanFastCrossSelect_Emit method

(IPCB_Board interface)

Syntax

```
Property  CanFastCrossSelect_Emit    : Boolean                        Read
GetState_CanFastCrossSelect_Emit    Write SetState_CanFastCrossSelect_Emit;
```

See also

IPCB_ServerInterface interface

IPCB_SystemOptions interface

IPCB_Board interface

Overview

The **IPCB_Board** interface encapsulates an opened PCB document in DXP and from this board interface object, you can add, delete PCB design objects, find out which layers are used and so on.

The **IPCB_Board** interface has iterative methods and interactive feedback methods. Basically you can retrieve an object interface for the PCB design object on the PCB that was clicked on. You can also retrieve the coordinates based on the mouse click on the PCB and also you can conduct defined searches on a PCB document with the parameters you have set up for the iterator. Refer to the Iterators section for more details.

Notes

- Check if the PCB server exists and if there is a PCB document before you do any PCB API calls. For example

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

- Some properties are only read only, meaning you can only retrieve data from property but not modify the data.
- To create a new object and add to the board object, firstly invoke the **PCBObjectFactory** from the **IPCB_ServerInterface** interface and then invoke the **AddPCBObject** method from a **IPCB_Board** interface.
- To look for objects on a PCB document, use one of the following iterators; Board Iterator, Group Iterator, Spatial iterator or a library iterator.
- Interactive feedback from the board can be done with the following methods: **GetObjectAtCursor**, **GetObjectAtXYAskUserIfAmbiguous**, **ChooseRectangleByCorners** and **ChooseLocation** functions.

IPCB_Board methods

```
IsLibrary
WindowBoundingRectangle
LayerPositionInSet

BoardIterator_Create
BoardIterator_Destroy
LibraryIterator_Create
LibraryIterator_Destroy
SpatialIterator_Create
SpatialIterator_Destroy
```

IPCB_Board properties

```
PCBWindow
FileName
XOrigin
YOrigin
XCursor
YCursor
DisplayUnit
CurrentLayer
LayerStack
LayerColor
SnapGridUnit
```

```
AddPCBObject
RemovePCBObject

GetPrimitiveCount
ConnectivelyValidateNets
ViewManager_Graphically
InvalidatePrimitive
GetPcbComponentByRefDes

ShowPCBObject
HidePCBObject
InvertPCBObject

CreateBoardOutline
UpdateBoardOutline

GetObjectAtCursor
GetObjectAtXYAskUserIfAmbiguous
ChooseRectangleByCorners
ChooseLocation

FindDominantRuleForObject
FindDominantRuleForObjectPair

AnalyzeNet
CleanNet
GetState_SplitPlaneNets
ClearUndoRedo
NewUndo
EndUndo
DoUndo
DoRedo
```

```
BigVisibleGridUnit
VisibleGridUnit
BigVisibleGridSize
VisibleGridSize
SnapGridSize
SnapGridSizeX
SnapGridSizeY
TrackGridSize
ViaGridSize
ComponentGridSize
ComponentGridSizeX
ComponentGridSizeY
DrawDotGrid
OutputOptions
ECOOptions
GerberOptions
PrinterOptions
PlacerOptions
LayerIsDisplayed
LayerIsUsed
InternalPlaneNetName
InternalPlane1NetName
InternalPlane2NetName
InternalPlane3NetName
InternalPlane4NetName
DrillLayerPairsCount
LayerPair
MechanicalPairs
BoardOutline
AutomaticSplitPlanes
PCBSheet
```

See also

TNetName value

TLayer enumerated values

IPCB_LayerStack interface

IPCB_OutputOptions interface

IPCB_ECOOptions interface

IPCB_GerberOptions interface

IPCB_PrinterOptions interface

IPCB_AdvancedPlacerOptions interface

QueryUsedLayers script in **\Examples\Scripts\DelphiScript\PCB** folder

SpatialIterator script in **\Examples\Scripts\DelphiScript\PCB** folder

IPCB_Board interface methods

AddObject method

(IPCB_Board interface)

Syntax

```
Procedure AddPCBObject(PCBObject : IPCB_Primitive);
```

Description

The **AddPCBObject** method adds a new Design Object into the PCB document created by a PCBObjectFactory method.

Example

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

    // 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 the new Via object in the board object (representing the current
PCB document)
```

```
    Board.AddPCBObject(Via);
End;
```

See also

IPCB_Board interface

AnalyzeNet method

(IPCB_Board interface)

Syntax

```
Procedure AnalyzeNet(Const ANet : IPCB_Net);
```

Description

This procedure analyzes a supplied net object in the form of IPCB_Net interface.

See also

IPCB_Board interface

BoardIterator_Create method

(IPCB_Board interface)

Syntax

```
Function  BoardIterator_Create : IPCB_BoardIterator;
```

Description

The **BoardIterator_Create** method creates a board iterator which is used to search for design objects on the PCB document. After the search has been conducted, invoke the **BoardIterator_Destroy** method to destroy the board iterator object.

Example

```
    // 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);
```

See also

IPCB_Board interface

BoardIterator_Destroy method

(IPCB_Board interface)

Syntax

```
Procedure BoardIterator_Destroy(Var AIterator : IPCB_BoardIterator);
```

Description

The BoardIterator_Destroy method destroys the board iterator object after it has been used to conduct a search on the PCB document for specified board objects.

Example

```
    // 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);
```

See also

IPCB_Board interface

BoardIterator_Create method.

ChooseLocation method

(IPCB_Board interface)

Syntax

```
Function  ChooseLocation(Var X1, Y1 : TCoord;
                             Prompt : TPCBString): Boolean;
```

Description

The function returns you the X1 and Y1 coordinates after you are prompted with a string on the status bar of the DXP and clicking anywhere on the PCB document.

Example

```
Function ChooseComponent : IPCB_Component;
Var
    Board : IPCB_Board;
    x,y   : TCoord;
Begin
    Pcbserver.PreProcess;
    Try
        Board := PCBServer.GetCurrentPCBBoard;
        If Not Assigned(Board) Then
        Begin
            ShowMessage('The Current Document is not a Protel PCB
Document.');
            Exit;
        End;

        Board.ChooseLocation(x,y, 'Choose Component');
        Result := Board.GetObjectAtXYAskUserIfAmbiguous(x,y
,MkSet(eComponentObject), AllLayers, eEditAction_Select);
        If Assigned(Result) Then
        Begin
            ShowMessage(Result.name.text);
            Result.Selected := True;
            //Do Something with the component
        End;
    Finally
        Pcbserver.PostProcess;
```

```
    End;
End;
```

See also

IPCB_Board interface

GetObjectAtXYAskUserIfAmbiguous method

ChooseRectangleByCorners method

(IPCB_Board interface)

Syntax

```
Function  ChooseRectangleByCorners(Prompt1    : TPCBString;
                                   Prompt2    : TPCBString;
                                   Var X1, Y1,
                                       X2, Y2 : TCoord) : Boolean;
```

Description

The **ChooseRectangleByCorners** method prompts you twice to choose the two sets of coordinates that define a boundary rectangle on the PCB document. The method returns you the X1,Y1, X2, Y2 values that can be used for calculations or for the spatial iterator for example.

Example

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

If Not (Board.ChooseRectangleByCorners( 'Choose first corner',
                                         'Choose final corner',
                                          x1,y1,x2,y2)) Then Exit;

// The coordinates from the ChooseRectangleByCorners method
// can be used for a spatial iterator for example
```

See also

IPCB_Board interface

IPCB_SpatialIterator

ChooseLocation method

CleanNet method

(IPCB_Board interface)

Syntax

```
Procedure CleanNet(Const ANet : IPCB_Net);
```

Description

The CleanNet procedure cleans up the supplied net represented by the IPCB_Net parameter. It cleans up by re-organizing and re-arranging the net topology.

See also

IPCB_Board interface

ClearUndoRedo method

(IPCB_Board interface)

Syntax

```
Procedure ClearUndoRedo;
```

Description

This clears out the UndoRedo facility in the PCB editor.

See also

IPCB_Board interface

ConnectivelyValidateNets method

(IPCB_Board interface)

Syntax

```
Procedure ConnectivelyValidateNets;
```

Description

This procedure validates the connectivitiy of nets on the PCB document.

See also

IPCB_Board interface

CreateBoardOutline method

(IPCB_Board interface)

Syntax

```
Function  CreateBoardOutline : IPCB_BoardOutline;
```

Description

The function creates a board outline represented by the **IPCB_BoardOutline** interface.

See also

IPCB_Board interface

IPCB_BoardOutline interface

DoRedo method

(IPCB_Board interface)

Syntax

```
Procedure DoRedo;
```

Description

This procedure invokes the Redo facility in the PCB editor.

See also

IPCB_Board interface

DoUndo method

(IPCB_Board interface)

Syntax

```
Procedure DoUndo;
```

Description

This procedure invokes the Undo facility in the PCB editor.

See also

IPCB_Board interface

EndUndo method

(IPCB_Board interface)

Syntax

```
Procedure EndUndo;
```

Description

This procedure ends the Undo process in the PCB editor.

See also

IPCB_Board interface

FindDominantRuleForObject method

(IPCB_Board interface)

Syntax

```
Function  FindDominantRuleForObject(APrimitive : IPCB_Primitive;
                                    ARuleKind  : TRuleKInd) : IPCB_Rule;
```

Description

This function returns the dominant specified rule for the primitive which is targetted by this rule.

See also

IPCB_Board interface

FindDominantRuleForObjectPair method

(IPCB_Board interface)

Syntax

```
Function  FindDominantRuleForObjectPair(APrimitive1,
                                        APrimitive2 : IPCB_Primitive;
                                        ARuleKind   : TRuleKInd) :
IPCB_Rule;
```

Description

This function returns the dominant specified binary rule for the two primitives which are targetted by this rule.

See also

IPCB_Board interface

GetObjectAtCursor method

(IPCB_Board interface)

Syntax

```
Function  GetObjectAtCursor(ObjectSet     : TObjectSet;
                            LayerSet      : TLayerSet;
                            StatusBarText : TPCBString) : IPCB_Primitive;
```

Description

This function returns the design object that is within the mouse's clicked coordinates on the PCB document.

Parameters

The ObjectSet parameter specifies which object types can be returned.

The LayerSet parameter specifies the objects on which layers that can be returned.

The StatusbarText parameter specifies the text on the status bar of the DXP application when the function is invoked.

See also

IPCB_Board interface

GetObjectAtXYAskUserIfAmbiguous method

(IPCB_Board interface)

Syntax

```
Function GetObjectAtXYAskUserIfAmbiguous(HitX,
                                         HitY      : TCoord;
                                         ObjectSet : TObjectSet;
                                         LayerSet  : TLayerSet;
                                         Action    : TEditingAction) :
IPCB_Primitive;
```

Description

The function returns the design object with the following parameters.

Parameters

The HitX parameter specifies the X coordinate value.

The HitY parameter specifies the Y coordinate value.

The ObjectSet parameter specifies which object types can be returned.

The LayerSet parameter specifies the objects on which layers that can be returned.

The Action parameter specifies what is happening when this method is invoked.

Example

Example

```
Function ChooseComponent : IPCB_Component;
Var
    Board : IPCB_Board;
    x,y   : TCoord;
Begin
    Pcbserver.PreProcess;
    Try
        Board := PCBServer.GetCurrentPCBBoard;
        If Not Assigned(Board) Then
        Begin
```

```
            ShowMessage('The Current Document is not a Protel PCB
Document.');
            Exit;
        End;

        Board.ChooseLocation(x,y, 'Choose Component');
        Result := Board.GetObjectAtXYAskUserIfAmbiguous(x,y
,MkSet(eComponentObject), AllLayers, eEditAction_Select);
        If Assigned(Result) Then
        Begin
            ShowMessage(Result.name.text);
            Result.Selected := True;
            //Do Something with the component
        End;
    Finally
        Pcbserver.PostProcess;
    End;
End;
```

See also

IPCB_ServerInterface interface

IPCB_Board interface

TObjectSet type

TLayerSet type

TEditingAction type

GetPcbComponentByRefDes method

(IPCB_Board interface)

Syntax

```
Function  GetPcbComponentByRefDes(Value : TString) : IPCB_Component;
```

Description

This function returns the component by its valid reference designator.

See also

IPCB_Board interface

GetPrimitiveCount method

(IPCB_Board interface)

Syntax

```
Function  GetPrimitiveCount(AObjSet  : TObjectSet;
                            LayerSet : TLayerSet;
                            AMethod  : TIterationMethod)  :Integer;
```

Description

The function returns the number of primitives which is dependent on the parameters supplied - the object kinds to look for, which layers to look for and how the search is conducted.

Parameters

The ObjectSet parameter specifies which object types can be returned.

The LayerSet parameter specifies the objects on which layers that can be returned.

The AMethod parameter specifies how the search is conducted.

See also

IPCB_Board interface

TObjectSet type

TLayerSet type

TIterationMethod type

GetState_SplitPlaneNets method

(IPCB_Board interface)

Syntax

```
Procedure GetState_SplitPlaneNets(NetsList  : TStringList);
```

Description

This procedure retrieves the list of nets for split planes on the PCB document in a TStringList container.

See also

IPCB_Board interface

HidePCBObject method

(IPCB_Board interface)

Syntax

```
Procedure HidePCBObject(Const PCBObject : IPCB_Primitive);
```

Description

This method hides the specified object on the PCB document from view.

See also

IPCB_Board interface

InvertPCBObject method

ShowPCBObject method

InvertPCBObject method

(IPCB_Board interface)

Syntax

```
Procedure InvertPCBObject(Const PCBObject : IPCB_Primitive);
```

Description

This method inverts the colors of the specified object on the PCB document.

See also

IPCB_Board interface

ShowPCBObject method

HidePCBObject method

IsLibrary method

(IPCB_Board interface)

Syntax

```
Function  IsLibrary : Boolean;
```

Description

This function returns a true value if the PCB document is a library otherwise a false value is returned.

Example

```
Var
    CurrentLib : IPCB_Board;
Begin
    CurrentLib := PCBServer.GetCurrentPCBBoard;
    If CurrentLib = Nil Then
    Begin
        ShowMessage('This is not a PCB document');
        Exit;
    End;
```

```
    // Check if CurrentLib is a library otherwise exit.
    If Not CurrentLib.IsLibrary Then
    Begin
        showMessage('This is not a PCB library document.');
        Exit;
    End;
```

See also

IPCB_Board interface

LayerPositionInSet method

(IPCB_Board interface)

Syntax

```
Function  LayerPositionInSet(ALayerSet : TLayerSet;
                             ALayerObj : IPCB_LayerObject)  : Integer;
```

Description

This function returns a positive value with 1 being the first layer and a higher number being the lower layer in the list. This function is useful for checking low and high layers of a layer pair.

Example

```
Begin
    PCBBoard := PCBServer.GetCurrentPCBBoard;
    If PCBBoard = Nil Then Exit;

    LayerPairs := TStringList.Create;
    For i := 0 To PCBBoard.DrillLayerPairsCount - 1 Do
    Begin
        PCBLayerPair := PCBBoard.LayerPair[i];
        LowLayerObj  :=
PCBBoard.LayerStack.LayerObject[PCBLayerPair.LowLayer];
        HighLayerObj :=
PCBBoard.LayerStack.LayerObject[PCBLayerPair.HighLayer];
        LowPos       := PCBBoard.LayerPositionInSet(SignalLayers +
InternalPlanes,
                                                    LowLayerObj);
        HighPos      := PCBBoard.LayerPositionInSet(SignalLayers +
InternalPlanes,
```

```
                                              HighLayerObj);
        If LowPos <= HighPos Then
            LayerPairs.Add(LowLayerObj .Name + ' - ' + HighLayerObj.Name)
        Else
            LayerPairs.Add(HighLayerObj.Name + ' - ' + LowLayerObj .Name);
    End;

    // Format the layer pairs data string and display it.
    LS := '';
    For i := 0 to LayerPairs.Count - 1 Do
        LS := LS + LayerPairs[i] + #13#10;
    ShowInfo('Layer Pairs:'#13#10 + LS);
    LayerPairs.Free;
End;
```

See also

IPCB_Board interface

IPCB_DrillLayerPair interface

LibraryIterator_Create method

(IPCB_Board interface)

Syntax

```
Function  LibraryIterator_Create : IPCB_LibraryIterator;
```

Description

This function creates a library iterator object which is used to conduct a search in a PCB library document for footprints. When the search is done, invoke the **LibraryIterator_Destroy** method.

Notes

A footprint is represented by a **IPCB_LibComponent** interface.

Example

```
// For each page of library is a footprint
FootprintIterator := CurrentLib.LibraryIterator_Create;
FootprintIterator.SetState_FilterAll;

// Within each page, fetch primitives of the footprint
// A footprint is a IPCB_LibComponent inherited from
// IPCB_Group which is a container object storing primitives.
```

```
Footprint := FootprintIterator.FirstPCBObject;
While Footprint <> Nil Do
Begin
    S := S + Footprint.Name;
    // do what you want with the footprint.
    Footprint := FootprintIterator.NextPCBObject;
End;
CurrentLib.LibraryIterator_Destroy(FootprintIterator);
```

See also

IPCB_Board interface

IPCB_LibComponent interface

IPCB_GroupIterator interface

LibraryIterator_Destroy method

(IPCB_Board interface)

Syntax

```
Procedure LibraryIterator_Destroy(Var AIterator : IPCB_LibraryIterator);
```

Description

This method destroys the Library iterator object after a search has been conducted in a library. A library consists of individual footprints and each footprint is represented by a IPCB_LibComponent interface.

Example

```
// For each page of library is a footprint
FootprintIterator := CurrentLib.LibraryIterator_Create;
FootprintIterator.SetState_FilterAll;

// Within each page, fetch primitives of the footprint
// A footprint is a IPCB_LibComponent inherited from
// IPCB_Group which is a container object storing primitives.
Footprint := FootprintIterator.FirstPCBObject;
While Footprint <> Nil Do
Begin
    S := S + Footprint.Name;
    // do what you want with the footprint.
    Footprint := FootprintIterator.NextPCBObject;
```

```
End;
CurrentLib.LibraryIterator_Destroy(FootprintIterator);
```

See also

IPCB_Board interface

IPCB_LibComponent interface

IPCB_GroupIterator interface

NewUndo method

(IPCB_Board interface)

Syntax

```
Procedure NewUndo;
```

Description

This procedure creates a new undo process in the PCB editor.

See also

IPCB_Board interface

RemoveObject method

(IPCB_Board interface)

Syntax

```
Procedure RemovePCBObject(PCBObject : IPCB_Primitive);
```

Description

This method removes the PCB object from the PCB board but it is not completely destroyed, which means it can be undone.

Example

```
Try
    Track := Iterator.FirstPCBObject;
    While Track <> Nil Do
    Begin
        OldTrack := Track;
        Track := Iterator.NextPCBObject;
        CurrentPCBBoard.RemovePCBObject(OldTrack);
        PCBServer.SendMessageToRobots(CurrentPCBBoard.I_ObjectAddress,
                                      c_BroadCast,
                                      PCBM_BoardRegisteration,
```

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

See also

IPCB_Board interface

ShowPCBObject method

(IPCB_Board interface)

Syntax

```
Procedure ShowPCBObject(Const PCBObject : IPCB_Primitive);
```

Description

This procedure makes the specified hidden PCB object visible.

See also

IPCB_Board interface

InvertPCBObject method

HidePCBObject method

SpatialIterator_Create method

(IPCB_Board interface)

Syntax

```
Function  SpatialIterator_Create : IPCB_SpatialIterator;
```

Description

This method creates a spatial iterator which conducts a search within defined boundary on a PCB document.

Example

```
    Iterator := Board.SpatialIterator_Create;

    (* Top/Bottom Layers and Arc/Track objects defined
       for the Spatial iterator constraints *)
    ASetOfLayers  := MkSet(eTopLayer,eBottomLayer);
    ASetOfObjects := MkSet(eArcObject,eTrackObject);
```

```
Iterator.AddFilter_ObjectSet(ASetOfObjects);
Iterator.AddFilter_LayerSet(ASetOfLayers);
Iterator.AddFilter_Area(X1,Y1,X2,Y2);

(* Iterate for tracks and arcs on bottom/top layers *)
PCBObject := Iterator.FirstPCBObject;
While PCBObject <> 0 Do
Begin
     PCBObject.Selected := True;
     PCBObject := Iterator.NextPCBObject;
End;
Board.SpatialIterator_Destroy(Iterator);
```

See also

IPCB_Board interface

SpatialIterator_Destroy method

SpatialIterator_Destroy method

(IPCB_Board interface)

Syntax

```
Procedure SpatialIterator_Destroy(Var AIterator : IPCB_SpatialIterator);
```

Description

This method destroys the spatial iterator object after it has finished conducting a search within a defined boundary on the PCB document.

Example

```
Iterator := Board.SpatialIterator_Create;

(* Top/Bottom Layers and Arc/Track objects defined
   for the Spatial iterator constraints *)
ASetOfLayers  := MkSet(eTopLayer,eBottomLayer);
ASetOfObjects := MkSet(eArcObject,eTrackObject);

Iterator.AddFilter_ObjectSet(ASetOfObjects);
Iterator.AddFilter_LayerSet(ASetOfLayers);
Iterator.AddFilter_Area(X1,Y1,X2,Y2);
```

```
(* Iterate for tracks and arcs on bottom/top layers *)
PCBObject := Iterator.FirstPCBObject;
While PCBObject <> 0 Do
Begin
     PCBObject.Selected := True;
     PCBObject := Iterator.NextPCBObject;
End;
Board.SpatialIterator_Destroy(Iterator);
```

See also

IPCB_Board interface

SpatialIterator_Create method

UpdateBoardOutline method

(IPCB_Board interface)

Syntax

```
Procedure UpdateBoardOutline;
```

Description

This method refreshes the Board outline.

See also

IPCB_Board interface

ViewManager_GraphicallyInvalidatePrimitive method

(IPCB_Board interface)

Syntax

```
Procedure ViewManager_GraphicallyInvalidatePrimitive(PCBObject :
IPCB_Primitive);
```

Description

This procedure forces a repaint of the design object on the PCB document.

See also

IPCB_Board interface

WindowBoundingRectangle method

(IPCB_Board interface)

Syntax

```
Function  WindowBoundingRectangle : TCoordRect;
```

Description

This function returns the coordinates of the bounds of a PCB window.

See also

IPCB_Board interface

IPCB_Board interface properties

AutomaticSplitPlanes property

(IPCB_Board interface)

Syntax

```
Property  AutomaticSplitPlanes : Boolean Read GetState_AutomaticSplitPlanes
 Write SetState_AutomaticSplitPlanes;
```

Description

The AutomaticSplitPlanes property returns you the boolean value whether the split planes are done automatically or not. This property is implemented by its GetState_AutomaticSplitPlanes and SetState_AutomaticSplitPlanes methods.

See also

IPCB_Board interface

BigVisibleGridSize property

(IPCB_Board interface)

Syntax

```
BigVisibleGridSize : TReal Read GetState_BigVisibleGridSize    Write
SetState_BigVisibleGridSize;
```

Description

This property retrieves or sets the Big Visible Grid Size in TReal type. This Grid Size is used for reference purposes and there are two visible grids.

See also

IPCB_Board interface

VisibleGridSize property

BigVisibleGridUnit property

(IPCB_Board interface)

Syntax

```
Property  BigVisibleGridUnit : TUnit Read GetState_BigVisibleGridUnit
   Write SetState_BigVisibleGridUnit;
```

Description

This property retrieves or sets the big visible grid's measurement units in Imperial or Metric units. There are two visible grids to use for reference purposes.

See also

IPCB_Board interface

VisibleGridUnit property

TUnit type

BoardOutline property

(IPCB_Board interface)

Syntax

```
Property  BoardOutline : IPCB_BoardOutline Read GetState_BoardOutline;
```

Description

The Board Outline represents the board outline which encompasses a board design on a PCB document. The board outline is represented by the **IPCB_BoardOutline** interface and inherited from the **IPCB_Polygon** interface because the Board Outline is composed of vertices (tracks and arcs only).

Example

```
Var
    PCB_Board : IPCB_Board;
    BR        : TCoordRect;
    NewUnit   : TUnit;
Begin
    PCB_Board := PCBServer.GetCurrentPCBBoard;
    If PCB_Board = Nil Then Exit;
    If PCB_Board.IsLibrary Then Exit;

    PCB_Board.BoardOutline.Invalidate;
    PCB_Board.BoardOutline.Rebuild;
    PCB_Board.BoardOutline.Validate;
    BR := PCB_Board.BoardOutline.BoundingRectangle;
```

```
    // do something else

End;
```

See also

IPCB_Board interface

IPCB_BoardOutline interface

ComponentGridSize property

(IPCB_Board interface)

Syntax

```
Property  ComponentGridSize : TDouble Read GetState_ComponentGridSize
    Write SetState_ComponentGridSize;
```

Description

This property represents the component grid size for components to be accurately placed on. This component grid size sets the X and Y values simultaneously. If you wish to define different X and Y grid sizes, then use the ComponentGridSizeX and ComponentGridSizeY properties.

See also

IPCB_Board interface

ComponentGridSizeX property

ComponentGridSizeY property

TDouble type

ComponentGridSizeX

(IPCB_Board interface)

Syntax

```
Property  ComponentGridSizeX                : TDouble
                   Read GetState_ComponentGridSizeX    Write
SetState_ComponentGridSizeX;
```

Description

This property represents the component grid size for components to be accurately placed on. To define different X and Y grid sizes, use the **ComponentGridSizeX** and **ComponentGridSizeY** properties, otherwise to set the same values for the component grid sizes X and Y simultaneously.

See also

IPCB_Board interface

ComponentGridSize

ComponentGridSizeY

ComponentGridSizeY property

(IPCB_Board interface)

Syntax

```
Property  ComponentGridSizeY : TDouble Read GetState_ComponentGridSizeY
Write SetState_ComponentGridSizeY;
```

Description

This property represents the component grid size for components to be accurately placed on. To define different X and Y grid sizes, use the **ComponentGridSizeX** and **ComponentGridSizeY** properties, otherwise to set the same values for the component grid sizes X and Y simultaneously.

See also

IPCB_Board interface

CurrentLayer property

(IPCB_Board interface)

Syntax

```
Property  CurrentLayer : TLayer Read GetState_CurrentLayer;
```

See also

IPCB_Board interface

DisplayUnit property

(IPCB_Board interface)

Syntax

```
Property DisplayUnit : TUnit Read GetState_DisplayUnit Write
SetState_DisplayUnit;
```

Description

This property retrieves or sets the measurement units for the PCB document display purposes in Imperial or Metric units.

Example

```
Var
    Board : IPCB_Board;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
```

```
    If Board = Nil Then Exit;

    ShowMessage(
        'Board Handle  = '   + IntToStr    (Board.I_ObjectAddress)      +
#13 +
        'Window Handle = '   + IntToStr    (Board.PCBWindow)            +
#13 +
        'Board Filename ='   +              Board.FileName              +
#13 +
        'Origin X = '        + IntToStr    (Board.XOrigin)              +
#13 +
        'Origin Y = '        + IntToStr    (Board.YOrigin)              +
#13 +
        'Board Units = '     + UnitToString(Board.DisplayUnit)          +
#13 +
        'Current layer = '   + Layer2String(Board.CurrentLayer)         +
#13);
End;
```

See also

IPCB_Board interface

DrawDotGrid property

(IPCB_Board interface)

Syntax

```
Property  DrawDotGrid : Boolean Read GetState_DrawDotGrid Write
SetState_DrawDotGrid;
```

Description

This property denotes whether the grid has dotted or continuous lines.

See also

IPCB_Board interface

DrillLayersPairsCount property

(IPCB_Board interface)

Syntax

```
Property  DrillLayerPairsCount : Integer Read GetState_DrillLayerPairsCount;
```

Description

This property returns the number of drill layer pairs for the board. A drill layer pair is represented by the **IPCB_DrillLayerPair** interface.

Example

```
Var
    PCBBoard      : IPCB_Board;
    i             : Integer;
    LayerPairs    : TStringList;
    PCBLayerPair : IPCB_DrillLayerPair;
    LowLayerObj   : IPCB_LayerObject;
    HighLayerObj : IPCB_LayerObject;

    LowPos        : Integer;
    HighPos       : Integer;
    LS            : String;
Begin
    PCBBoard := PCBServer.GetCurrentPCBBoard;
    If PCBBoard = Nil Then Exit;

    For i := 0 To PCBBoard.DrillLayerPairsCount - 1 Do
    Begin
        PCBLayerPair := PCBBoard.LayerPair[i];
        LowLayerObj  :=
PCBBoard.LayerStack.LayerObject[PCBLayerPair.LowLayer];
        HighLayerObj :=
PCBBoard.LayerStack.LayerObject[PCBLayerPair.HighLayer];

        // do what you want with the LowLayerObj and HighLayerObj objects
    End;
End;
```

See also

IPCB_Board interface

LayerPair property

IPCB_DrillLayerPair interface

FileName property

(IPCB_Board interface)

Syntax

```
Property  FileName : TPCBString Read GetState_FileName;
```

Description

The FileName property denotes the filename of the PCB document that the **IPCB_Board** interface is associated with. The Filename property is read only, which means you can retrieve the filename string only.

Example

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

    ShowMessage(
        'Board Handle   = '    + IntToStr    (Board.I_ObjectAddress)       +
#13 +
        'Window Handle = '    + IntToStr    (Board.PCBWindow)             +
#13 +
        'Board Filename ='    +              Board.FileName               +
#13 +
        'Origin X = '         + IntToStr    (Board.XOrigin)               +
#13 +
        'Origin Y = '         + IntToStr    (Board.YOrigin)               +
#13 +
        'Board Units = '      + UnitToString(Board.DisplayUnit)           +
#13 +
        'Current layer = '    + Layer2String(Board.CurrentLayer)          +
#13);
End;
```

See also

IPCB_Board interface

InternalPlane1NetName property

(IPCB_Board interface)

Syntax

```
Property  InternalPlane1NetName : TPCBString Read
GetState_InternalPlane1NetName Write SetState_InternalPlane1NetName;
```

See also

IPCB_Board interface

InternalPlane2NetName property

(IPCB_Board interface)

Syntax

```
Property  InternalPlane2NetName             : TPCBString
                Read GetState_InternalPlane2NetName Write
SetState_InternalPlane2NetName;
```

See also

IPCB_Board interface

InternalPlane3NetName property

(IPCB_Board interface)

Syntax

```
Property  InternalPlane3NetName             : TPCBString
                Read GetState_InternalPlane3NetName Write
SetState_InternalPlane3NetName;
```

See also

IPCB_Board interface

InternalPlane4NetName

(IPCB_Board interface)

Syntax

```
Property  InternalPlane4NetName : TPCBString Read
GetState_InternalPlane4NetName Write SetState_InternalPlane4NetName;
```

See also

IPCB_Board interface

InternalPlaneNetName property

(IPCB_Board interface)

Syntax

```
Property  InternalPlaneNetName [L : TLayer] : TPCBString Read
GetState_InternalPlaneNetName  Write SetState_InternalPlaneNetName;
```

Description

This property returns or sets the net name for the internal plane in question.

See also

IPCB_Board interface

TLayer type

LayerColor property

(IPCB_Board interface)

Syntax

```
Property  LayerColor [L : TLayer] : TColorRef Read GetState_LayerColor;
```

Description

This property returns the layer color of TColorRef type. This type is defined in the Windows.pas which is part of the Borland Delphi Run-Time Library.

See also

IPCB_Board interface

TColorRef type

LayerIsDisplayed property

(IPCB_Board interface)

Syntax

```
Property  LayerIsDisplayed     [L : TLayer] : Boolean Read
GetState_LayerIsDisplayed      Write SetState_LayerIsDisplayed;
```

Description

The **LayerIsDisplayed** property controls the display of layers for the PCB document. You can fetch or set the

Example

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

// Check for each signal layer for used/display setting
For Layer := eTopLayer to eMultiLayer Do
```

```
    If PCBBoard.LayerIsUsed[Layer] Then
        If PCBBoard.LayerIsDisplayed[Layer] Then
            \\ do something
```

See also

IPCB_Board interface

LayerIsUsed property

(IPCB_Board interface)

Syntax

```
Property  LayerIsUsed [L : TLayer] : Boolean Read GetState_LayerIsUsed Write
SetState_LayerIsUsed;
```

Description

This property retrieves or sets the boolean value for whether the layer is used by primitives or not. Normally when a layer has primitives (design objects) on it, the layer is used.

Example

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

// Check for each signal layer for used/display setting
For Layer := eTopLayer to eMultiLayer Do
    If PCBBoard.LayerIsUsed[Layer] Then
        If PCBBoard.LayerIsDisplayed[Layer] Then
            \\ do something
```

See also

IPCB_Board interface

LayerPair property

(IPCB_Board interface)

Syntax

```
Property  LayerPair [I : Integer] : IPCB_DrillLayerPair        Read
GetState_LayerPair;
```

Description

This property returns you the layer pair associated with the IPCB_DrillLayerPair interface. A drill layer pair has two drill layers.

Example

```
Var
    PCBBoard     : IPCB_Board;
    i            : Integer;
    LayerPairs   : TStringList;
    PCBLayerPair : IPCB_DrillLayerPair;
    LowLayerObj  : IPCB_LayerObject;
    HighLayerObj : IPCB_LayerObject;
    LowPos       : Integer;
    HighPos      : Integer;
    LS           : String;
Begin
    PCBBoard := PCBServer.GetCurrentPCBBoard;
    If PCBBoard = Nil Then Exit;

    // Show the Current Layer for the PCB document.
    ShowInfo('Current Layer: ' + Layer2String(PCBBoard.CurrentLayer));

    LayerPairs := TStringList.Create;
    For i := 0 To PCBBoard.DrillLayerPairsCount - 1 Do
    Begin
        PCBLayerPair := PCBBoard.LayerPair[i];
        LowLayerObj  :=
PCBBoard.LayerStack.LayerObject[PCBLayerPair.LowLayer];
        HighLayerObj :=
PCBBoard.LayerStack.LayerObject[PCBLayerPair.HighLayer];

        LowPos       := PCBBoard.LayerPositionInSet(SignalLayers +
InternalPlanes, LowLayerObj);
        HighPos      := PCBBoard.LayerPositionInSet(SignalLayers +
InternalPlanes, HighLayerObj);
        If LowPos <= HighPos Then
            LayerPairs.Add(LowLayerObj .Name + ' - ' + HighLayerObj.Name)
        Else
            LayerPairs.Add(HighLayerObj.Name + ' - ' + LowLayerObj .Name);
    End;
```

```
   // Display layer pairs.
    LS := '';
    For i := 0 to LayerPairs.Count - 1 Do
        LS := LS + LayerPairs[i] + #13#10;

    ShowInfo('Layer Pairs:'#13#10 + LS);
    LayerPairs.Free;
End;
```

See also

IPCB_Board interface

LayerStack property

(IPCB_Board interface)

Syntax

```
Property  LayerStack : IPCB_LayerStack Read GetState_LayerStack;
```

Description

The layer stack property fetches the **IPCB_LayerStack** interface for the current PCB document. The Layer stack only stores copper layers (signal and internal planes).

Example

```
Var
    PCBBoard      : IPCB_Board;
    TheLayerStack : IPCB_LayerStack;
    i             : Integer;
    LayerObj      : IPCB_LayerObject;
    LS            : String;
Begin
    PCBBoard := PCBServer.GetCurrentPCBBoard;
    If PCBBoard = Nil Then Exit;

    // Note that the Layer stack only stores existing copper based layers.
    // But you can use the LayerObject property to fetch all layers.
    TheLayerStack := PCBBoard.LayerStack;
    If TheLayerStack = Nil Then Exit;
    LS       := '';
    LayerObj := TheLayerStack.FirstLayer;
```

```
    Repeat
        LS       := LS + Layer2String(LayerObj.LayerID) + #13#10;
        LayerObj := TheLayerStack.NextLayer(LayerObj);
    Until LayerObj = Nil;
    ShowInfo('The Layer Stack has :'#13#10 + LS);
End;
```

See also

IPCB_LayerStack interface

IPCB_LayerObject interface

IPCB_Board interface

MechanicalPairs property

(IPCB_Board interface)

Syntax

```
Property  MechanicalPairs : IPCB_MechanicalLayerPairs  Read
GetState_MechanicalPairs;
```

Description

There are 16 general purpose mechanical layers for defining the board layout, placing dimensions on, including fabrication details on, or any other mechanical details the design requires.

The purpose of the **IPCB_MechanicalLayerPairs** Interface is to provide which Mechanical layers are paired to one another.

When a component incorporates objects on one or more Mechanical layers which have been paired, the Layer property of those objects changes when the Layer property of the component is toggled (between the Top and Bottom layers), just like objects on the non-Mechanical layers which have always been paired to one another, along with the Top and Bottom (copper) layers, the Top and Bottom Overlay layers, the Top and Bottom Paste Mask layers, and the Top and Bottom Solder Mask layers.

See also

IPCB_Board interface

IPCB_MechanicalPairs interface

PCBSheet property

(IPCB_Board interface)

Syntax

```
Property PCBSheet : IPCB_Sheet Read GetState_PCBSheet;
```

Description

This property returns the IPCB_Sheet interface which is represented by the sheet workspace. A sheet encapsulates the sheet borders, the fabrication and assembly information, and the board outline.

See also

IPCB_Board interface

IPCB_Sheet interface

PCBWindow property

(IPCB_Board interface)

Syntax

```
Property  PCBWindow : HWND Read GetState_Window;
```

Description

This property returns the raw Windows handle for a window handle of a PCB document in DXP.

See also

IPCB_Board interface

SnapGridSizeX

(IPCB_Board interface)

Syntax

```
Property  SnapGridSizeX : TDouble Read GetState_SnapGridSizeX Write
SetState_SnapGridSizeX;
```

Description

This property retrieves or sets the Snap Grid size X value. To set both X and Y values simultaneously for the Snap Grid, use the **SnapGridSize** property.

See also

IPCB_Board interface

SnapGridSizeY property

SnapGridSize property

SnapGridSizeY property

(IPCB_Board interface)

Syntax

```
Property  SnapGridSizeY : TDouble Read GetState_SnapGridSizeY Write
SetState_SnapGridSizeY;
```

Description

This property retrieves or sets the Snap Grid size Y value. To set both X and Y values simultaneously for the Snap Grid, use the **SnapGridSize** property.

See also

IPCB_Board interface

SnapGridSizeX property

SnapGridSize property

SnapGridSize property

(IPCB_Board interface)

Syntax

```
Property  SnapGridSize                        : TDouble
                   Read GetState_SnapGridSize          Write
SetState_SnapGridSize;
```

Description

The SnapGridSize property sets the X and Y values for the Snap Grid simultaneously. If you want to have different X and Y values for this snap grid, use the SnapGridSizeX and SnapGridSizeY properties.

See also

IPCB_Board interface

SnapGridSizeX property

SnapGridSizeY property

SnapGridUnit property

(IPCB_Board interface)

Syntax

```
Property  SnapGridUnit : TUnit Read GetState_SnapGridUnit Write
SetState_SnapGridUnit;
```

Description

The SnapGridUnit property retrieves or sets the measurement unit for the Snap Grid Unit. It can be in Imperial or Metric units.

See also

IPCB_Board interface

TUnit type

TrackGridSize property

(IPCB_Board interface)

Syntax

```
Property  TrackGridSize : TDouble  Read GetState_TrackGridSize Write
SetState_TrackGridSize;
```

Description

This property retrieves or sets the track grid size in both X and Y directions simultaneously.

See also

IPCB_Board interface

ViaGridSize property

ViaGridSize property

(IPCB_Board interface)

Syntax

```
Property ViaGridSize : TDouble Read GetState_ViaGridSize  Write
SetState_ViaGridSize;
```

Description

This property retrieves or sets the via grid size in both X and Y directions simultaneously.

See also

IPCB_Board interface

TrackGridSize property

VisibleGridSize property

(IPCB_Board interface)

Syntax

```
Property  VisibleGridSize : TReal Read GetState_VisibleGridSize Write
SetState_VisibleGridSize;
```

Description

This property retrieves or sets the Visible Grid Size in TReal type. This Grid Size is used for reference purposes and there are two visible grids.

See also

IPCB_Board interface

BigVisibleGridSize property

VisibleGridUnit property

(IPCB_Board interface)

Syntax

```
Property  VisibleGridUnit : TUnit Read GetState_VisibleGridUnit Write
SetState_VisibleGridUnit;
```

Description

This property retrieves or sets the big visible grid's measurement units in Imperial or Metric units. There are two visible grids to use for reference purposes.

See also

IPCB_Board interface

BigVisibleGridUnit interface

TUnit type

XOrigin property

(IPCB_Board interface)

Syntax

```
Property  XOrigin : TCoord  Read GetState_XOrigin Write SetState_XOrigin;
```

Description

This property sets or retrieves the X coordinate of the absolute origin of the board. Only if an Origin marker has been set on the PCB document then the X,Y origin coordinates will be valid. Otherwise 0,0 is returned if the Origin marker doesn't exist.

See also

IPCB_Board interface

XCursor property

(IPCB_Board interface)

Syntax

```
Property  XCursor : TCoord Read GetState_XCursor  Write SetState_XCursor;
```

Description

This property retrieves or sets the x coordinate of the cursor of the latest mouse click on the PCB document.

See also

IPCB_Board interface

YCursor property

(IPCB_Board interface)

Syntax

```
Property  YCursor : TCoord Read GetState_YCursor Write SetState_YCursor;
```

Description

This property retrieves or sets the Y coordinate of the cursor of the latest mouse click on the PCB document.

See also

IPCB_Board interface

YOrigin property

(IPCB_Board interface)

Syntax

```
Property  YOrigin : TCoord Read GetState_YOrigin Write SetState_YOrigin;
```

Description

This property sets or retrieves the Y coordinate of the absolute origin of the board. Only if an Origin marker has been set on the PCB document then the X,Y origin coordinates will be valid. Otherwise 0,0 is returned if the Origin marker doesn't exist.

See also

IPCB_Board interface

ECOOptions property

(IPCB_Board interface)

Syntax

```
Property  ECOOptions : IPCB_ECOOptions Read GetState_ECOOptions;
```

Description

This property returns you the IPCB_ECOOptions interface which represents the Options for the Engineering Order Change facility in the PCB editor.

See also

IPCB_Board interface

IPCB_ECOOptions interface

GerberOptions property

(IPCB_Board interface)

Syntax

```
Property  GerberOptions : IPCB_GerberOptions Read GetState_GerberOptions;
```

Description

This property returns you the IPCB_GerberOptions interface which represents the Options for the Gerbers facility in the PCB editor.

See also

IPCB_Board interface

IPCB_GerberOptions interface

PlacerOptions property

(IPCB_Board interface)

Syntax

```
Property  PlacerOptions : IPCB_AdvancedPlacerOptions Read
GetState_PlacerOptions;
```

Description

This property returns you the IPCB_PlacerOptions interface which represents the Options for the Placement facility in the PCB editor.

See also

IPCB_Board interface

IPCB_PlacerOptions interface

PrinterOptions property

(IPCB_Board interface)

Syntax

```
Property  PrinterOptions : IPCB_PrinterOptions Read GetState_PrinterOptions;
```

Description

This property returns you the IPCB_PrinterOptions interface which represents the Options for the Printer setup facility in the PCB editor.

See also

IPCB_Board interface

IPCB_PrinterOptions interface

OutputOptions property

(IPCB_Board interface)

Syntax

```
Property  OutputOptions : IPCB_OutputOptions Read GetState_OutputOptions;
```

Description

This property returns you the IPCB_OutputOptions interface which represents the Options for the Output facility in the PCB editor.

See also

IPCB_Board interface

IPCB_OutputOptions interface

IPCB_Sheet interface

Overview

The **IPCB_Sheet** interface represents the background workspace for the PCB document and can include fabrication and assembly documentation as well as the board outline.

Notes

- The sheet behind the PCB can be shown or not.
- The coordinates of the PCB sheet can be defined programmatically.

IPCB_Sheet methods	IPCB_Sheet properties
I_ObjectAddress	`SheetX`
	`SheetY`
	`SheetWidth`
	`SheetHeight`
	`ShowSheet`
	`LockSheet`

See also

IPCB_Board

IPCB_Sheet interface methods

I_ObjectAddress method

(IPCB_AbstractIterator, IPCB_BoardIterator, IPCB_SpatialIterator, IPCB_GroupIterator, IPCB_Sheet)

Syntax

```
Function I_ObjectAddress : TPCBObjectHandle;
```

Description

The **I_ObjectAddress** property retrieves the pointer to the iterator object. This property is useful for situations where you need to have references to objects (not to object interfaces) and store them in a TList container.

See also

IPCB_Sheet interface

IPCB_Sheet interface properties

SheetHeight property

(IPCB_Board interface)

Syntax

```
Property  SheetHeight : TCoord Read GetState_SheetHeight Write
SetState_SheetHeight;
```

See also

IPCB_Sheet interface

SheetWidth property

(IPCB_Sheet interface)

Syntax

```
Property  SheetWidth : TCoord Read GetState_SheetWidth Write
SetState_SheetWidth;
```

See also

IPCB_Sheet interface

SheetX property

(IPCB_Sheet interface)

Syntax

```
Property  SheetX : TCoord Read GetState_SheetX Write SetState_SheetX;
```

See also

IPCB_Sheet interface

SheetY property

(IPCB_Sheet interface)

Syntax

```
Property  SheetY : TCoord Read GetState_SheetY Write SetState_SheetY;
```

See also

IPCB_Sheet interface

ShowSheet method

(IPCB_Sheet interface)

Syntax

```
Property  ShowSheet : Boolean Read GetState_ShowSheet Write
SetState_ShowSheet;
```

Description

This property retrieves or sets the boolean value. The Sheet property represents the bounds where a board outline and assembly / fabrication details are included within.

Example

```
Function UnitToString(U : TUnit) : TPCBString;
Begin
    Result := '';
    Case U of
       eImperial : Result := 'Imperial (mil)';
       eMetric   : Result := 'Metric (mm)';
    End;
End;
{...........................................................................
...}
{...........................................................................
...}
Function BoolToString(B : Boolean) : TPCBString;
Begin
    Result := 'False';
    If B Then Result := True;
End;
{...........................................................................
...}
{...........................................................................
...}
Procedure Query_Board;
```

```
Var
    Board         : IPCB_Board;
    LibraryExists : TPCBString;
    AShowSheet    : TPCBString;
    ALockSheet    : TPCBString;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;
    LibraryExists := BoolToString(Board.IsLibrary);
    AShowSheet    := BoolToString(Board.PCBSheet.ShowSheet);
    ALockSheet    := BoolToString(Board.PCBSheet.LockSheet);
    ShowMessage(
        'Board Handle  = '   + IntToStr    (Board.I_ObjectAddress)      +
#13 +
        'Window Handle = '   + IntToStr    (Board.PCBWindow)            +
#13 +
        'Board Filename ='   +              Board.FileName              +
#13 +
        'Is a Library = '    +              LibraryExists               +
#13 +
        'Origin X = '        + IntToStr    (Board.XOrigin)              +
#13 +
        'Origin Y = '        + IntToStr    (Board.YOrigin)              +
#13 +
        'Board Units = '     + UnitToString(Board.DisplayUnit)          +
#13 +
        'Current layer = '   + Layer2String(Board.CurrentLayer)         +
#13 +
        'Sheet.X = '         + IntToStr    (Board.PCBSheet.SheetX)      +
#13 +
        'Sheet.Y = '         + IntToStr    (Board.PCBSheet.SheetY)      +
#13 +
        'Sheet.Height = '    + IntToStr    (Board.PCBSheet.SheetHeight) +
#13 +
        'Sheet.Width = '     + IntToStr    (Board.PCBSheet.SheetWidth)
        'Sheet is shown = '  +              AShowSheet                  +
#13 +
        'Sheet is locked = ' +              ALockSheet
    );
```

```
End;
```

See also

IPCB_Sheet interface

LockSheet method

(IPCB_Sheet interface)

Syntax

```
Property  LockSheet : Boolean Read GetState_LockSheet Write
SetState_LockSheet;
```

See also

IPCB_Sheet interface

IPCB_LayerStack interface

Overview

The **IPCB_LayerStack** interface represents the layer stack for the current PCB document. This Layer Stack interface is a property within in the **IPCB_Board** interface.

Strictly speaking, the **IPCB_LayerStack** interface represents the layer stack and therefore only has copper based layers such as top, mid1-30, bottom layers and internal planes. However you can use the **LayerObject** property with the **IPCB_Board** parameter passed in to obtain any PCB layer for the PCB document.

Iterating copper layers within the Layer Stack

To query for existing copper layers (signal layers and internal players) within the layer stack, you can use the **FirstLayer** and **NextLayer** properties of the **IPCB_LayerStack** interface to iterate for such layers.

Notes

- Each layer can be represented as a IPCB_LayerObject, IPCB_InternalPlane, IPCB_DrillLayerPair or IPCB_MechanicalLayerPairs interfaces.
- A layer can have dielectric properties which is represented by a **IPCB_DielectricObject** interface.
- To have access to other layers of the PCB document, use the **LayerObject** property of the **IPCB_LayerStack** interface.

IPCB_LayerStack methods	IPCB_LayerStack properties
FirstLayer	Board
NextLayer	LayerObject
PreviousLayer	DielectricTop
LastLayer	DielectricBottom

```
InsertLayer                          ShowDielectricTop
LastInternalPlane                    ShowDielectricBottom
FirstAvailableSignalLayer
FirstAvailableInternalPlane
SignalLayerCount
```

See also

Using PCB Layers

Using the PCB Layer Stack

IPCB_LayerObject interface

IPCB_InternalPlane interface

IPCB_Board interface

IPCB_DielectricObject interface

QueryLayerStack and QueryMechLayers script in the **\Examples\Scripts\Delphiscript\PCB** folder

IPCB_LayerStack interface methods

FirstLayer method

(IPCB_LayerStack interface)

Syntax

```
Function  FirstLayer : IPCB_LayerObject;
```

Description

The Firstlayer property fetches the first layer stored in the layer stack for the PCB document. To fetch the next layer in the layer stack, invoke the NextLayer property. Notice that the layer stack only stores signal and internal (copper based) layers.

Example

```
Var
    PCBBoard       : IPCB_Board;
    TheLayerStack : IPCB_LayerStack;
    i              : Integer;
    LayerObj       : IPCB_LayerObject;
    LS             : String;
Begin
    PCBBoard := PCBServer.GetCurrentPCBBoard;
    If PCBBoard = Nil Then Exit;
```

```
    TheLayerStack := PCBBoard.LayerStack;
    If TheLayerStack = Nil Then Exit;
    LS        := '';
    LayerObj := TheLayerStack.FirstLayer;
    Repeat
        LS        := LS + Layer2String(LayerObj.LayerID) + #13#10;
        LayerObj := TheLayerStack.NextLayer(LayerObj);
    Until LayerObj = Nil;
    ShowInfo('The Layer Stack has :'#13#10 + LS);
End;
```

See also

IPCB_LayerStack interface

FirstAvailableInternalPlane method

(IPCB_LayerStack interface)

Syntax

```
Function FirstAvailableInternalPlane : IPCB_InternalPlane;
```

See also

IPCB_LayerStack interface

FirstAvailableSignalLayer method

(IPCB_LayerStack interface)

Syntax

```
Function  FirstAvailableSignalLayer : IPCB_LayerObject;
```

Description

This function retrieves the first available signal layer from the layer stack. A layer stack only stores copper based layers such as signal and internal plane layers.

See also

IPCB_LayerStack interface

IPCB_LayerObject interface

InsertLayer method

(IPCB_LayerStack interface)

Syntax

```
Procedure InsertLayer(L : TLayer);
```

See also

IPCB_LayerStack interface

LastInternalPlane method

(IPCB_LayerStack interface)

Syntax

```
Function LastInternalPlane : IPCB_InternalPlane;
```

Description

This function retrieves the last internal plane from the layer stack if it exists. If there is no internal planes in the layer stack, the function will return a Nil value.

See also

IPCB_LayerStack interface

IPCB_InternalPlane interface

LastLayer property

(IPCB_LayerStack interface)

Syntax

```
Function LastLayer : IPCB_LayerObject;
```

See also

IPCB_LayerStack interface

NextLayer property

(IPCB_LayerStack interface)

Syntax

```
Function  NextLayer(L : IPCB_LayerObject) : IPCB_LayerObject;
```

Description

The Nextlayer property fetches the next layer stored in the layer stack for the PCB document after the FirstLayer property has been invoked. Notice that the layer stack only stores signal and internal (copper based) layers.

Example

```
Var
    PCBBoard       : IPCB_Board;
    TheLayerStack : IPCB_LayerStack;
    i              : Integer;
    LayerObj       : IPCB_LayerObject;
```

```
    LS              : String;
Begin
    PCBBoard := PCBServer.GetCurrentPCBBoard;
    If PCBBoard = Nil Then Exit;

    // Note that the Layer stack only stores existing copper based layers.
    TheLayerStack := PCBBoard.LayerStack;
    If TheLayerStack = Nil Then Exit;
    LS        := '';
    LayerObj := TheLayerStack.FirstLayer;
    Repeat
        LS        := LS + Layer2String(LayerObj.LayerID) + #13#10;
        LayerObj := TheLayerStack.NextLayer(LayerObj);
    Until LayerObj = Nil;

    ShowInfo('The Layer Stack has :'#13#10 + LS);
End;
```

See also

IPCB_LayerStack interface

PreviousLayer method

(IPCB_LayerStack interface)

See also

IPCB_LayerStack interface

SignalLayerCount method

(IPCB_LayerStack interface)

Syntax

```
Function SignalLayerCount : Integer;
```

Description

This function returns the number of signal layers in the layer stack for the PCB document.

See also

IPCB_LayerStack interface

IPCB_LayerStack interface properties

Board property

(IPCB_LayerStack interface)

Syntax

```
Property Board : IPCB_Board Read GetState_Board;
```

Description

This property returns the PCB document that is represented by the **IPCB_Board** interface, that the layer stack is associated with.

See also

IPCB_LayerStack interface

IPCB_Board interface

DielectricBottom property

(IPCB_Board interface)

Syntax

```
Property  DielectricBottom : IPCB_DielectricObject Read
GetState_DielectricBottom;
```

Description

This property returns the **IPCB_DielectricObject** interface associated with the dielectric information for the bottom layer of the layer stack.

See also

IPCB_DielectricObject interface

DielectricTop property

(IPCB_Board interface)

Syntax

```
Property  DielectricTop : IPCB_DielectricObject Read GetState_DielectricTop;
```

Description

This property returns the **IPCB_DielectricObject** interface associated with the dielectric information for the top layer of the layer stack.

See also

IPCB_DielectricObject interface

LayerObject property

(IPCB_LayerStack interface)

Syntax

```
Property  LayerObject [L : TLayer] : IPCB_LayerObject  Read
GetState_LayerObject;
```

Description

The LayerObject property retrieves the layer object interface for the specified layer, L of TLayer type. It is a read only property.

Example

```
Var
    PCBBoard      : IPCB_Board;
    TheLayerStack : IPCB_LayerStack;
    i             : Integer;
    LayerObj      : IPCB_LayerObject;
    LS            : String;
Begin
    PCBBoard := PCBServer.GetCurrentPCBBoard;
    If PCBBoard = Nil Then Exit;

    TheLayerStack := PCBBoard.LayerStack;
    If TheLayerStack = Nil Then Exit;
    LS        := '';
    LayerObj := TheLayerStack.FirstLayer;
    Repeat
        LS        := LS + Layer2String(LayerObj.LayerID) + #13#10;
        LayerObj  := TheLayerStack.NextLayer(LayerObj);
    Until LayerObj = Nil;
    ShowInfo('The Layer Stack has :'#13#10 + LS);
End;
```

See also

IPCB_LayerStack interface

IPCB_LayerObject interface

TLayer type

ShowDielectricBottom property

(IPCB_LayerStack interface)

Syntax

```
Property  ShowDielectricBottom            : Boolean           Read
GetState_ShowBotDielectric    Write SetState_ShowBotDielectric;
    End;
```

Description

This property enables or disables the dielectric layer for the bottom layer.

See also

IPCB_LayerStack interface

ShowDielectricTop property

(IPCB_LayerStack interface)

Syntax

```
Property ShowDielectricTop : Boolean Read GetState_ShowTopDielectric Write
SetState_ShowTopDielectric;
```

Description

This property enables or disables the dielectric layer for the top layer.

See also

IPCB_LayerStack interface

IPCB_LayerObject interface

Overview

The **IPCB_LayerObject** interface represents a layer used in a PCB document. Each layer has properties such as layer id, name, used by primitives and whether it is displayed for example. This interface is a property in the **IPCB_LayerStack** interface.

The layer stack for a PCB document only deals with copper based layers such as signal and internal plane layers. Each layer in the layer stack can have dielectric information and layer pairs can be specified. However there is a **LayerObject** property in the **IPCB_LayerStack** interface which allows you to access any PCB layer for the PCB board.

Iterating for any PCB layer of a PCB document

Although the **IPCB_LayerStack** interface basically deals with copper based layers that are used in the layer stack, this Layer Stack interface can be used to look for other PCB layers that are not in the layer stack. The **LayerObject** property from this layer stack interface obtains any PCB layer whether it is a keep out layer, top signal layer or a mechanical 16 layer.

Methods

```
Function  I_ObjectAddress : TPCBObjectHandle;
Function  IsInLayerStack  : Boolean;
```

Properties

```
Property LayerStack                      : IPCB_LayerStack
Property LayerID                         : TLayer
Property Name                            : TPCBString
Property CopperThickness                 : TCoord
Property Dielectric                      : IPCB_DielectricObject
Property UsedByPrims                     : Boolean
Property IsDisplayed[Board : IPCB_Board] : Boolean
Property PreviousLayer                   : TLayer
Property NextLayer                       : TLayer
Property MechanicalLayerEnabled          : Boolean
```

Example

```
Var
    Board   : IPCB_Board;
    Layer   : TLayer;
    LS      : IPCB_LayerStack;
    LObject : IPCB_LayerObject;
    S       : TPCBString;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;
    LS := Board.LayerStack;
    If LS = Nil Then Exit;
    S := '';

    Layer := eTopLayer;
    LObject := LS.LayerObject[Layer];

    // Check if layer is populated, displayed or not.
    If (LObject.IsDisplayed[Board] = True) and (LObject.UsedByPrims) Then
        ShowInfo(LObject.Name + ' is displayed and there are objects on
it.');
```

```
    If (LObject.IsDisplayed[Board] = True) and Not (LObject.UsedByPrims)
Then
        ShowInfo(LObject.Name + ' is displayed and there are NO objects on
it.');
    If (LObject.IsDisplayed[Board] = False) and (LObject.UsedByPrims) Then
        ShowInfo(LObject.Name + ' is NOT displayed and there are objects on
it.');
    If (LObject.IsDisplayed[Board] = False) and Not (LObject.UsedByPrims)
Then
        ShowInfo(LObject.Name + ' is NOT displayed and there are NO objects
on it.');
    ShowMessage(S);
```

See also

TLayer enumerated values

TCoord value

IPCB_DielectricObject interface

IPCB_LayerStack interface

IPCB_DielectricObject

Overview

The **IPCB_DielectricObject** interface represents the dielectric properties for the specified PCB layer.

Notes

The **IPCB_DielectricObject** interface is a standalone interface.

Properties

```
Property  DielectricMaterial : TPCBString
Property  DielectricType     : TDielectricType
Property  DielectricConstant : TReal
Property  DielectricHeight   : TCoord
```

Example

```
Function ConvertDielectricTypeTOString (DT : TDielectricType): String;
Begin
    Result := 'Unknown Type';
    Case DT Of
        eNoDielectric     : Result := 'No Dielectric';
        eCore             : Result := 'Core';
```

```
        ePrePreg         : Result := 'PrePreg';
        eSurfaceMaterial : Result := 'Surface Material';
    End;
End;
{.......................................................................
...}
{.......................................................................
...}
Function GetLayerInfo(Board : IPCB_Board; Var LayerID : TLayer) : String;
Var
    LayerObj : IPCB_LayerObject;
Begin
    LayerObj := Board.LayerStack.LayerObject[LayerId];
    Result := Layer2String(LayerID) + ', ' + LayerObj.Name + ', ' +
              'Copper' + ', ' + FloatToStr(LayerObj.CopperThickness / 10000)
+ ', ';
    If LayerObj.Dielectric.DielectricType <> eNoDielectric Then
    Begin
       Result := Result +
ConvertDielectricTypeTOString(LayerObj.Dielectric.DielectricType) + ', ' +
                 LayerObj.Dielectric.DielectricMaterial +  ', ' +
FloatToStr(LayerObj.Dielectric.DielectricHeight / 10000) + ', ' +
                 FloatToStr(LayerObj.Dielectric.DielectricConstant);
    End;
    LayerObj := Board.LayerStack.NextLayer(LayerObj);

    If LayerObj <> Nil Then
        LayerID := LayerObj.LayerID
    Else
        LayerID := eNoLayer;
End;
{.......................................................................
...}
{.......................................................................
...}
Procedure FetchLayersInformation;
Var
    Board : IPCB_Board;
```

```
    Str   : String;
    Layer : TLayer;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;

    Str := 'Layer, Name, Material, Cu Thickness,  Dielectric Material, type,
constant, height ' + #13#10;
    Layer := MinLayer;
    Repeat
       Str := Str + GetLayerInfo(Board, Layer) + #13#10;
    Until Layer = eNoLayer;

    // Do what you want with the Str string.
End;
```

See also

IPCB_LayerStack interface

LayerReport script in the **\Examples\Scripts\DelphiScript\PCB** folder.

IPCB_DrillLayerPair

Overview

The **IPCB_DrillLayerPair** interface represents the paired drill layer for the layer stack up for the PCB document.

Notes

- The **IPCB_DrillLayerPair** interface is a standalone interface
- The **IPCB_DrillLayerPair** interface is a **DrillLayerPair** property from the **IPCB_Board** interface

Methods

```
Function  I_ObjectAddress        : TPCBObjectHandle;
Function  GetState_Description   : TPCBString;
Function  IsSimilarTo(ADLP : IPCB_DrillLayerPair) : Boolean;
Procedure OrderLayers;
```

Properties

```
Property LowLayer          : TLayer
Property HighLayer         : TLayer
Property StartLayer        : IPCB_LayerObject
```

```
Property StopLayer         : IPCB_LayerObject
Property Board             : IPCB_Board
Property PlotDrillDrawing : Boolean
Property PlotDrillGuide    : Boolean
```

Example

```
Var
    PCBBoard     : IPCB_Board;
    i            : Integer;
    LayerPairs   : TStringList;
    PCBLayerPair : IPCB_DrillLayerPair;
    LowLayerObj  : IPCB_LayerObject;
    HighLayerObj : IPCB_LayerObject;
    LowPos       : Integer;
    HighPos      : Integer;
    LS           : String;
Begin
    PCBBoard := PCBServer.GetCurrentPCBBoard;
    If PCBBoard = Nil Then Exit;

    // Show the current layer
    ShowInfo('Current Layer: ' + Layer2String(PCBBoard.CurrentLayer));

    LayerPairs := TStringList.Create;
    For i := 0 To PCBBoard.DrillLayerPairsCount - 1 Do
    Begin
        PCBLayerPair := PCBBoard.LayerPair[i];
        LowLayerObj  :=
PCBBoard.LayerStack.LayerObject[PCBLayerPair.LowLayer];
        HighLayerObj :=
PCBBoard.LayerStack.LayerObject[PCBLayerPair.HighLayer];
        LowPos       := PCBBoard.LayerPositionInSet(SignalLayers +
InternalPlanes, LowLayerObj);
        HighPos      := PCBBoard.LayerPositionInSet(SignalLayers +
InternalPlanes, HighLayerObj);
        If LowPos <= HighPos Then
            LayerPairs.Add(LowLayerObj .Name + ' - ' + HighLayerObj.Name)
        Else
```

```
            LayerPairs.Add(HighLayerObj.Name + ' - ' + LowLayerObj .Name);
    End;

    //Display layer pairs.
    LS := '';
    For i := 0 to LayerPairs.Count - 1 Do
        LS := LS + LayerPairs[i] + #13#10;
    ShowInfo('Layer Pairs:'#13#10 + LS);
    LayerPairs.Free;
End;
```

See also

TLayer enumerated values

TCoord value

IPCB_LayerObject interface

IPCB_Board interface

IPCB_InternalPlane

Overview

This **IPCB_InternalPlane** interface represents an existing internal plane used on a PCB document. 16 internal planes are supported, and a net can be assigned to each of these layers or share a power plane between a number of nets by splitting the it into two or more isolated areas. Pad and via connections to power planes are controlled by the Plane design rules.

Properties

```
Property PullBackDistance          : TCoord
Property NetName                   : TPCBString
Property FirstPreviousSignalLayer : TLayer   //Read only
Property FirstNextSignalLayer      : TLayer   //Read only
```

See also

TLayer enumerated values

TCoord value

TNetName value

IPCB_LayerStack interface

IPCB_MechanicalLayerPairs

Overview

There are 16 general purpose mechanical layers for defining the board layout, placing dimensions on, including fabrication details on, or any other mechanical details the design requires.

The purpose of the **IPCB_MechanicalLayerPairs** Interface is to provide which Mechanical layers are paired to one another. Pairing of Mechanical layers has been provided to users since the DXP version of Protel.

When a component incorporates objects on one or more Mechanical layers which have been paired, the Layer property of those objects changes when the Layer property of the component is toggled (between the Top and Bottom layers), just like objects on the non-Mechanical layers which have always been paired to one another, to wit the Top and Bottom (copper) layers, the Top and Bottom Overlay layers, the Top and Bottom Paste Mask layers, and the Top and Bottom Solder Mask layers.

Notes

- The **IPCB_MechanicalLayerPairs** interface is a MechanicalPairs property of the **IPCB_Board** interface.
- Invoke the **Count** method to obtain the number of mechanical layer pairs for the existing PCB document. Indexed mechanical layer pairs which is a **LayerPair[]** property can be returned. This property returns a **TMechanicalLayerPair** record of two PCB layers.

Methods

```
Procedure Clear;
Function  Count                            : Integer;
Function  AddPair    (Layer1,
                      Layer2 : TLayer)     : Integer;
Function  RemovePair (Layer1,
                      Layer2 : TLayer)     : Boolean;
Function  PairDefined(Layer1,
                      Layer2 : TLayer)      : Boolean;
Function  LayerUsed   (Layer   : TLayer)    : Boolean;
Function  FlipLayer(Var L   : TLayer)       : Boolean;

Procedure Import_FromParameters(Params : PChar);
Procedure Export_ToParameters  (Params : PChar);
```

Properties

```
LayerPair [I : Integer] : TMechanicalLayerPair
```

Example

```
Var
```

```
    Board   : IPCB_Board;
    Layer   : TLayer;
    LS      : IPCB_LayerStack;
    LObject : IPCB_LayerObject;
    S       : TPCBString;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;
    LS := Board.LayerStack;
    If LS = Nil Then Exit;
    S := '';
    For Layer := eMechanical1 to eMechanical16 Do
    Begin
        LObject := LS.LayerObject[Layer];
        // If a mechanical layer is not enabled (as per the Board Layers and
        //   Colors dialog) then this layer cannot be displayed nor have any
objects on it.
        If Not (LObject.MechanicalLayerEnabled) Then
            S := S + LObject.Name + ' is NOT enabled (thus it cannot be
displayed nor have any objects on it).' + #13
        Else
        Begin
           If (LObject.IsDisplayed[Board] = True) and (LObject.UsedByPrims)
Then
               S := S + LObject.Name + ' is displayed and there are objects
on it.' + #13;
           If (LObject.IsDisplayed[Board] = True) and Not
(LObject.UsedByPrims) Then
               S := S+ LObject.Name + ' is displayed and there are NO
objects on it.' + #13;
           If (LObject.IsDisplayed[Board] = False) and (LObject.UsedByPrims)
Then
                S := S + LObject.Name + ' is NOT displayed and there are
objects on it.' + #13;
           If (LObject.IsDisplayed[Board] = False) and Not
(LObject.UsedByPrims) Then
              S := S + LObject.Name + ' is NOT displayed and there are NO
objects on it.' + #13;
```

```
        End;
    End;
    ShowMessage(S);
End;
```

See also

TLayer enumerated values

TMechanicalLayerPair values

IPCB_LayerStack interface

IPCB_AbstractOptions

Overview

The object associated with the IPCB_AbstractOptions interface cannot be instantiated. This interface is the base interface for other options related interfaces such as SystemOptions and InteractiveRoutingOptions through IPCB_ServerInterface. These option objects are global objects created by the PCB Server.

The other OutputOptions, ECOOptions, GerberOptions, PrinterOptions and PlacerOptions interfaces are referenced through IPCB_Board interface.

Notes

- Ancestor interface for ECO Options, Output Options, Gerber Options, Printer Options, Advanced Placer Options, SystemOptions, Design Rule Checker Options, SpecctraRouter Options and Interactive Routing options interfaces.

Methods

```
Procedure Import_FromParameters          (DisplayUnit : TUnit;
                                          Parameters  : PChar);
Procedure Export_ToParameters            (Parameters  : PChar);
Procedure Import_FromParameters_Version4 (DisplayUnit : TUnit;
                                          Parameters  : PChar);
Procedure Export_ToParameters_Version4   (Parameters  : PChar);
Procedure Import_FromParameters_Version3 (DisplayUnit : TUnit;
                                          Parameters  : PChar);
Procedure Export_ToParameters_Version3   (Parameters  : PChar);
Function  I_ObjectAddress : TPCBObjectHandle;
```

Properties

```
OptionsObjectID : TOptionsObjectId
```

See also

IPCB_ECOOptions interface

IPCB_OutputOptions interface

IPCB_GerberOptions interface

IPCB_PrinterOptions interface

IPCB_AdvancedPlacerOptions interface

IPCB_SystemOptions interface

IPCB_DesignRuleCheckerOptions interface

IPCB_SpecctraRouterOptions interface

IPCB_InteractiveRoutingOptions interface

IPCB_AdvancedPlacerOptions

Overview

The IPCB_AdvancedPlacerOptions interface represents the options for the placement application.

Notes

- Derived from IPCB_AbstractOptions interface

IPCB_ Properties

```
Property PlaceLargeClear        : TCoord
Property PlaceSmallClear        : TCoord
Property PlaceUseRotation       : Boolean
Property PlaceUseLayerSwap      : Boolean
Property PlaceByPassNet1        : TPCBString
Property PlaceByPassNet2        : TPCBString
Property PlaceUseAdvancedPlace  : Boolean
Property PlaceUseGrouping       : Boolean
```

See also

IPCB_AbstractOptions interface

IPCB_DesignRuleCheckerOptions

Overview

The IPCB_DesignRuleCheckerOptions interface deals with the DRC options.

Notes

- Derived from IPCB_AbstractOptions interface

IPCB_DesignRuleCheckerOptions Methods

```
Procedure Export_ToParameters_GeneralOptions         (Parameters : PChar);
Procedure Export_ToParameters_RulesToCheck           (Parameters : PChar);
Procedure Export_ToParameters_RulesToCheck_Version3  (Parameters : PChar);
Procedure Import_FromParameters_GeneralOptions       (Parameters : PChar);
Procedure Import_FromParameters_RulesToCheck         (Parameters : PChar);
```

IPCB_DesignRuleCheckerOptions Properties

```
Property OnLineRuleSetToCheck            : TRuleSet
Property DoMakeDRCFile                   : Boolean
Property DoMakeDRCErrorList              : Boolean
```

```
Property DoSubNetDetails                  : Boolean
Property RuleSetToCheck                   : TRuleSet
Property ReportFilename                   : TPCBString
Property ExternalNetListFileName          : TPCBString
Property CheckExternalNetList             : Boolean
Property MaxViolationCount                : Integer
Property InternalPlaneWarnings            : Boolean
Property VerifyShortingCopper             : Boolean
```

See also

IPCB_AbstractOptions interface

IPCB_ECOOptions

Overview

The IPCB_ECOOptions represents an existing Engineering Change Order options object in a PCB document.

Notes

- Derived from IPCB_AbstractOptions interface

IPCB_ECCOptions Properties

```
Property ECOIsActive : Boolean
Property ECOFileName : TString
```

See also

IPCB_AbstractOptions interface

IPCB_GerberOptions

Overview

The tolerance range used when matching apertures for each item in the plots. If no exact match for an item is available in the current aperture list, the software checks to see if a larger aperture exists within this tolerance range and uses it instead.

If no suitable aperture exists within the tolerance range, the software will attempt to "paint" with a larger aperture to create the required shape. This requires that a suitable larger aperture is available, and that this aperture can be used for "painting".

Note: Match tolerances are normally only used when you are targeting a vector photoplotter, which require a fixed, or supplied aperture file. They will not be required if the apertures have been created from the PCB. If match tolerances are not required they should be left at the default of 0.005 mil.

Notes

- Derived from IPCB_AbstractOptions interface

Properties

```
Property SortOutput           : Boolean
Property UseSoftwareArcs      : Boolean
Property CenterPhotoPlots     : Boolean
Property EmbedApertures       : Boolean
Property Panelize             : Boolean
Property G54                  : Boolean
Property PlusTol              : TCoord
Property MinusTol             : TCoord
Property FilmSizeX            : TCoord
Property FilmSizeY            : TCoord
Property BorderSize           : TCoord
Property AptTable             : TPCBString
Property MaxAperSize          : TCoord
Property ReliefShapesAllowed : Boolean
Property PadsFlashOnly        : Boolean
Property GerberUnits          : Integer
Property GerberDecs           : Integer
```

See also

IPCB_AbstractOptions interface

IPCB_InteractiveRoutingOptions

Overview

The IPCB_InteractiveRoutingOptions interface represents the options for the interactive routing module in the PCB editor.

Notes

- Derived from IPCB_AbstractOptions interface

Methods

```
Procedure Export_ToParameters_GeneralOptions(Parameters : PChar);
Procedure Export_ToParameters_LayerOptions  (Parameters : PChar);
Procedure Export_ToParameters_LayerOptions_Version3(Parameters : PChar);
```

Properties

```
PlaceTrackMode     : TPlaceTrackMode
OldTrackDrawLayer  : TLayer
TrackArcX          : TCoord
TrackArcY          : TCoord
TrackArcRadius     : TCoord
TrackArcAngle1     : TCoord
TrackArcAngle2     : TCoord
OldTrackArcX       : TCoord
OldTrackArcY       : TCoord
OldTrackArcRadius  : TCoord
OldTrackArcAngle1  : TCoord
OldTrackArcAngle2  : TCoord
OldTrackDrawSize   : TCoord
OldMidx            : TCoord
OldMidy            : TCoord
OldCx              : TCoord
OldCy              : TCoord
EndLineX           : TCoord
EndLineY           : TCoord
Midx               : TCoord
MidY               : TCoord
StartX             : TCoord
StartY             : TCoord
Beginx             : TCoord
Beginy             : TCoord
```

See also

IPCB_AbstractOptions interface

IPCB_MechanicalLayerPairs

Overview

There are 16 general purpose mechanical layers for defining the board layout, placing dimensions on, including fabrication details on, or any other mechanical details the design requires.

The purpose of the **IPCB_MechanicalLayerPairs** Interface is to provide which Mechanical layers are paired to one another. Pairing of Mechanical layers has been provided to users since the DXP version of Protel.

When a component incorporates objects on one or more Mechanical layers which have been paired, the Layer property of those objects changes when the Layer property of the component is toggled (between the Top and Bottom layers), just like objects on the non-Mechanical layers which have always been paired to one another, to wit the Top and Bottom (copper) layers, the Top and Bottom Overlay layers, the Top and Bottom Paste Mask layers, and the Top and Bottom Solder Mask layers.

Notes

- The **IPCB_MechanicalLayerPairs** interface is a MechanicalPairs property of the **IPCB_Board** interface.
- Invoke the **Count** method to obtain the number of mechanical layer pairs for the existing PCB document. Indexed mechanical layer pairs which is a **LayerPair[]** property can be returned. This property returns a **TMechanicalLayerPair** record of two PCB layers.

Methods

```
Procedure Clear;
Function  Count                            : Integer;
Function  AddPair    (Layer1,
                      Layer2 : TLayer)    : Integer;
Function  RemovePair (Layer1,
                      Layer2 : TLayer)    : Boolean;
Function  PairDefined(Layer1,
                      Layer2 : TLayer)     : Boolean;
Function  LayerUsed   (Layer  : TLayer)    : Boolean;
Function  FlipLayer(Var L  : TLayer)       : Boolean;

Procedure Import_FromParameters(Params : PChar);
Procedure Export_ToParameters  (Params : PChar);
```

Properties

```
LayerPair [I : Integer] : TMechanicalLayerPair
```

Example

```
Var
    Board   : IPCB_Board;
    Layer   : TLayer;
    LS      : IPCB_LayerStack;
    LObject : IPCB_LayerObject;
    S       : TPCBString;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
```

```
    If Board = Nil Then Exit;
    LS := Board.LayerStack;
    If LS = Nil Then Exit;
    S := '';
    For Layer := eMechanical1 to eMechanical16 Do
    Begin
        LObject := LS.LayerObject[Layer];
        // If a mechanical layer is not enabled (as per the Board Layers and
        //   Colors dialog) then this layer cannot be displayed nor have any
objects on it.
        If Not (LObject.MechanicalLayerEnabled) Then
            S := S + LObject.Name + ' is NOT enabled (thus it cannot be
displayed nor have any objects on it).' + #13
        Else
        Begin
           If (LObject.IsDisplayed[Board] = True) and (LObject.UsedByPrims)
Then
               S := S + LObject.Name + ' is displayed and there are objects
on it.' + #13;
           If (LObject.IsDisplayed[Board] = True) and Not
(LObject.UsedByPrims) Then
               S := S+ LObject.Name + ' is displayed and there are NO
objects on it.' + #13;
           If (LObject.IsDisplayed[Board] = False) and (LObject.UsedByPrims)
Then
                S := S + LObject.Name + ' is NOT displayed and there are
objects on it.' + #13;
           If (LObject.IsDisplayed[Board] = False) and Not
(LObject.UsedByPrims) Then
              S := S + LObject.Name + ' is NOT displayed and there are NO
objects on it.' + #13;
        End;
    End;
    ShowMessage(S);
End;
```

See also

TLayer enumerated values

TMechanicalLayerPair values

IPCB_LayerStack interface

IPCB_OutputOptions

Overview

The IPCB_OutputOptions interface represents the options for the generation of PCB output such as including mechanical layers in plots etc.

Notes

- Derived from IPCB_AbstractOptions interface

Methods

```
Procedure Import_FromParameters_GeneralOptions (DisplayUnit : TUnit;
                                                       Parameters  : PChar);
Procedure Import_FromParameters_LayerOptions   (Parameters  : PChar);
Procedure Import_FromParameters_LayerOptions_Version3 (Parameters  : PChar);
Procedure Export_ToParameters_GeneralOptions   (Parameters  : PChar);
Procedure Export_ToParameters_LayerOptions     (Parameters  : PChar);
Procedure Export_ToParameters_LayerOptions_Version3   (Parameters  : PChar);
```

Properties

```
Property DrillGuideHoleSize               : TCoord
Property DrillDrawSymbolSize              : TCoord
Property DrillSymbolKind                  : TDrillS
Property MultiLayerOnPadMaster            : Boolean
Property TopLayerOnPadMaster              : Boolean
Property BottomLayerOnPadMaster           : Boolean
Property IncludeViasInSolderMask          : Boolean
Property IncludeMech1WithAllPlots         : Boolean
Property IncludeMech2WithAllPlots         : Boolean
Property IncludeMech3WithAllPlots         : Boolean
Property IncludeMech4WithAllPlots         : Boolean
Property IncludeMech5WithAllPlots         : Boolean
Property IncludeMech6WithAllPlots         : Boolean
Property IncludeMech7WithAllPlots         : Boolean
Property IncludeMech8WithAllPlots         : Boolean
Property IncludeMech9WithAllPlots         : Boolean
Property IncludeMech10WithAllPlots        : Boolean
Property IncludeMech11WithAllPlots        : Boolean
```

```
Property IncludeMech12WithAllPlots       : Boolean
Property IncludeMech13WithAllPlots       : Boolean
Property IncludeMech14WithAllPlots       : Boolean
Property IncludeMech15WithAllPlots       : Boolean
Property IncludeMech16WithAllPlots       : Boolean
Property IncludeUnconnectedPads          : Boolean
Property PlotLayer [PL : TPlotLayer]     : Boolean
Property FlipLayer [PL : TPlotLayer]     : Boolean
```

See also

IPCB_AbstractOptions interface

IPCB_PinSwapOptions

Overview

The **IPCB_PinSwapOptions** interface represents the Pin Swapper functionality in PCB. It is used to swap pins of a large PCB component effortlessly.

Notes

- Derived from IPCB_AbstractOptions interface

Methods

```
Function  GetState_Quiet                         : Boolean;
Procedure SetState_Quiet(Value : Boolean);
Function  GetState_IgnoreNetsCount               : Integer;
Function  GetState_IgnoreNetIndexOf(Value : TString) : Integer;
Procedure ClearIgnoreNets;
Procedure AddIgnoreNet               (Value : TString);
Function  GetState_IgnoreNet         (Value : Integer): TString;
Function  GetState_IgnoreNetClassesCount : Integer;
Function  GetState_IgnoreNetClassIndexOf(Value : TString) : Integer;
Procedure ClearIgnoreNetClasses;
Procedure AddIgnoreNetClass          (Value : TString);
Function  GetState_IgnoreNetClass    (Value : Integer) : TString;
Function  GetState_IgnoreComponentsCount            : Integer;
Procedure ClearIgnoreComponents;
Procedure AddIgnoreComponent        (Value : TString);
Function  GetState_IgnoreComponent(Value : Integer)   : TString;
Function  GetState_CrossoverRatio                     : Integer;
```

```
Procedure SetState_CrossoverRatio        (Value : Integer);
Function  GetState_IgnoreComponentIndexOf(Value : TString) : Integer;
```

See also

IPCB_AbstractOptions interface

IPCB_PrinterOptions

Overview

Notes

- Derived from IPCB_AbstractOptions interface

Methods

```
Procedure Import_FromParameters_GeneralOptions        (DisplayUnit : TUnit;
                                                        Parameters  : PChar);
Procedure Import_FromParameters_LayerOptions          (Parameters  : PChar);
Procedure Import_FromParameters_LayerOptions_Version3 (Parameters  : PChar);
Procedure Export_ToParameters_GeneralOptions          (Parameters  : PChar);
Procedure Export_ToParameters_LayerOptions            (Parameters  : PChar);
Procedure Export_ToParameters_LayerOptions_Version3   (Parameters  : PChar);
```

Properties

```
Property Device           : TPCBString
Property Driver           : TPCBString
Property OutPut           : TPCBString
Property OutputDriverType : TOutputDriverType
Property ShowHoles        : Boolean
Property ScaleToFitPage   : Boolean
Property UsePrinterFonts  : Boolean
Property UseSoftwareArcs  : Boolean
Property BatchType        : TPrinterBatch
Property CompositeType    : TPrinterComposite
Property cBorderSize      : TCoord
Property Scale            : TGeometry
Property XCorrect         : TGeometry
Property YCorrect         : TGeometry
Property PlotMode [OId : TObjectId]          : TDrawMode
```

```
Property PlotPadNets       : Boolean
Property PlotPadNumbers    : Boolean
Property PlotterScale      : TGeometry
Property PlotterXCorrect   : TGeometry
Property PlotterYCorrect   : TGeometry
Property PlotterXOffset    : TCoord
Property PlotterYOffset    : TCoord
Property PlotterShowHoles : Boolean
Property PlotterUseSoftwareArcs : Boolean
Property PlotterWaitBetweenSheets : Boolean
Property PlotterOutputPort                    : TOutputPort
Property PlotterLanguage                      : TPlotterLanguage
Property PlotterPens [PId : Integer]          : TPlotterPen
Property CompositePlotMonoLayers  [L : TLayer] : TColor
Property CompositePlotColorLayers [L : TLayer] : TColor
Property CompositePlotLayers      [L : TLayer] : Boolean
Property CompositePlotPens        [L : TLayer] : Integer
```

See also

IPCB_AbstractOptions interface

IPCB_SpecctraRouterOptions

Overview

The IPCB_SpecctraRouterOptions interface represents the options for the Specctra Router application.

Notes

- Derived from IPCB_AbstractOptions interface

Properties

```
Property Setback      [I : Integer]      : TCoord
Property DoSetback    [I : Integer]      : Boolean
Property DoBus                           : Boolean
Property BusDiagonal                     : Boolean
Property DoQuit                          : Boolean
Property WireGrid                        : TReal
Property ViaGrid                         : TReal
Property DoSeedVias                      : Boolean
Property NoConflicts                     : Boolean
```

```
Property AdvancedDo                              : Boolean
Property ReorderNets                             : Boolean
Property ProtectPreRoutes                        : Boolean
Property SeedViaLimit                            : TCoord
Property RoutePasses                             : Integer
Property CleanPasses                             : Integer
Property FilterPasses                            : Integer
Property LayerCost    [L : TLayer]               : TCCTCost
Property LayerWWCost  [L : TLayer]               : TCCTCost
Property WwCost                                  : TCCTCost
Property CrossCost                               : TCCTCost
Property ViaCost                                 : TCCTCost
Property OffGridCost                             : TCCTCost
Property OffCenterCost                           : TCCTCost
Property SideExitCost                            : TCCTCost
Property SqueezeCost                             : TCCTCost
Property LayerTax     [L : TLayer]               : TCCTTax
Property LayerWWTax   [L : TLayer]               : TCCTTax
Property WwTax                                   : TCCTTax
Property CrossTax                                : TCCTTax
Property ViaTax                                  : TCCTTax
Property OffGridTax                              : TCCTTax
Property OffCenterTax                            : TCCTTax
Property SideExitTax                             : TCCTTax
Property SqueezeTax                              : TCCTTax
Property DoCritic                                : Boolean
Property DoMiter                                 : Boolean
Property DoRecorner                              : Boolean
Property DoFanout                                : Boolean
Property FoPower                                 : Boolean
Property FoSignal                                : Boolean
Property FoIn                                    : Boolean
Property FoOut                                   : Boolean
Property FoVias                                  : Boolean
Property FoPads                                  : Boolean
Property FoPasses                                : Integer
```

```
Property ForceVias                        : Boolean
Property DoSpread                         : Boolean
Property SortKind                         : TCCTSort
Property SortDir                          : TCCTSortDir
Property Adv10                            : Boolean
Property Dfm10                            : Boolean
Property Hyb10                            : Boolean
Property SpVersion                        : Integer
Property MinimizePads                     : Boolean
```

See also

IPCB_AbstractOptions interface

IPCB_SystemOptions

Overview

The IPCB_SystemOptions interface points to the global system options in the PCB Editor server. To obtain this interface, call the PCBServer.SystemOptions and assign it to a variable of IPCB_SystemOptions interface type.

Notes

- Derived from IPCB_AbstractOptions interface

Methods

```
Procedure Import_FromIniFile;
Procedure Export_ToIniFile;
Procedure AddComponentMapping (Value : TComponentTypeMapping);
```

Properties

```
{DisplayOptions}
Property UndoRedoStackSize                    : Integer
Property SingleLayerMode                      : Boolean
Property LockPreRoutes                        : Boolean
Property DrawMode   [OId : TObjectID]         : TDrawMode
Property FromTosDisplayMode                   : TFromToDisplayMode
Property PadTypesDisplayMode                  : TFromToDisplayMode
Property DraftTrackThreshold                  : TCoord
Property CleanRedraw                          : Boolean
Property ShowInvisibleObjects                 : Boolean
Property DisplaySpecialStrings                : Boolean
```

```
Property RedrawLayerOnToggle                : Boolean
Property UseCurrentForMultiLayer            : Boolean
Property UseNetColorForHighlight            : Boolean
Property HighlightFull                      : Boolean
Property ShowAllPrimitivesInHighlightedNets : Boolean
Property UseTransparent                     : Boolean
Property UseDithered                        : Boolean
Property ShowPadNets                        : Boolean
Property ShowPadNumbers                     : Boolean
Property ShowTestPoints                     : Boolean
Property ShowViaNets                        : Boolean
Property ShowStatusInfo                     : Boolean
Property ShowStatusInterval                 : Integer
Property BoardCursorType                    : TGraphicsCursor
Property TextToRectSize                     : Integer
Property AutoPan                            : Boolean
Property LayerDrawingOrder [I : Integer]    : TLayer

{PlaceArray Options}
Property RepeatRotateItem                   : Boolean
Property RepeatCircular                     : Boolean
Property RepeatDegrees                      : TGeometry
Property RepeatX                            : TGeometry
Property RepeatY                            : TGeometry
Property RepeatXUnit                        : TUnit
Property RepeatYUnit                        : TUnit
Property RepeatCountDefault                 : Integer
Property RepeatInc                          : TPCBString

{Com Port Options}
Property Com1Parameters     : TSerialParameters
Property Com2Parameters     : TSerialParameters
Property Com3Parameters     : TSerialParameters
Property Com4Parameters     : TSerialParameters

{Netlist load options}
```

```
Property CheckPatterns           : Boolean
Property CheckComments           : Boolean
Property NetlistReportFile       : Boolean
Property NetlistReportDialog     : Boolean
Property DeleteUnconnectedComps : Boolean
Property DeleteUnconnectedPrims : Boolean

{Misc System Options}
Property GlobalEditIncludeArcsWithTracks : Boolean
Property ValidateOnLoad                  : Boolean
Property SaveDefs                        : Boolean
Property DoOnlineDRC                     : Boolean
Property LoopRemoval                     : Boolean
Property UseSmartTrackEnds               : Boolean
Property DeleteDeadEnds                  : Boolean
Property QuestionDelete                  : Boolean
Property QuestionGlobalChange            : Boolean
Property QuestionDrag                    : Boolean
Property NearestComponent                : Boolean
Property RemoveDuplicatesOnOutput        : Boolean
Property DuplicateDesignatorsAllowed     : Boolean
Property AutoVia                         : Boolean
Property SnapToCentre                    : Boolean
Property ReportsCSV                      : Boolean
Property ClickClearsSelection            : Boolean
Property HoldShiftToSelectObjectId [OId : TObjectID] : Boolean
Property MustHoldShiftToSelect           : Boolean
Property DoubleClickRunsInspector        : Boolean
Property DefaultPrimsPermanent           : Boolean
Property DragMode                        : TPcbDragMode
Property RotationStep                    : TAngle
Property OnlySelectVisible               : Boolean
Property PlaceShoveDepth                 : Integer
Property LayerColors[L : TLayer]         : TColor
Property AutoPanMode                     : TAutoPanMode
Property AutoPanSmallStep                : Integer
```

```
Property AutoPanLargeStep                 : Integer
Property AutoPanUnit                      : TAutoPanUnit
Property AutoPanSpeed                     : Integer
Property InteractiveRouteMode             : TInteractiveRouteMode
Property PolygonThreshold                 : Integer
Property PolygonRepour                    : TPolygonRepourMode
Property PlowThroughPolygons              : Boolean
Property ProtectLockedPrimitives          : Boolean
Property ConfirmSelectionMemoryClear      : Boolean
Property ComponentMoveKind                : TComponentMoveKind
Property SameNamePadstackReplacementMode : TSameNamePadstackReplacementMode
Property PadstackUpdateFromGlobalsOnLoad : TSameNamePadstackReplacementMode
Property PlaneDrawMode           : TPlaneDrawMode
Property BoardAreaColor          : TColor
Property BoardLineColor          : TColor
Property SheetAreaColor          : TColor
Property SheetLineColor          : TColor
Property WorkspaceColor1         : TColor
Property WorkspaceColor2         : TColor
```

Example

```
Var
    PCBSystemOptions : IPCB_SystemOptions;
Begin
    PCBSystemOptions := PCBServer.SystemOptions;
    If PCBSystemOptions = Nil Then Exit;
    If PcbSystemOptions.BoardCursorType = eCurShapeCross90 Then
        PcbSystemOptions.BoardCursorType := eCurShapeBigCross
    Else If PcbSystemOptions.BoardCursorType = eCurShapeBigCross Then
        PcbSystemOptions.BoardCursorType := eCurShapeCross45
    Else
        PcbSystemOptions.BoardCursorType := eCurShapeCross90;
End.
```

See also

IPCB_AbstractOptions interface

TPCBDragMode enumerated values

TGraphicsCursor enumerated values

TComponentTypeMapping enumerated values

TComponentMoveKind enumerated values

TPolygonRepourMode enumerated values

TSameNamePadstackReplacementMode enumerated values

TPlaneDrawMode enumerated values

TAutoPanUnit enumerated values

TAutoPanMode enumerated values

TInteractiveRouteMode enumerated values

PCB Iterators

An iterator conducts a search through a PCB document's design database to fetch PCB design objects. With an iterator, you can control which objects on which layers and within specified regions.

There are four different types of iterators; Board Iterator, Library Iterator, Spatial Iterator and Group Iterator. The board iterator is for conducting searches on a PCB document, the library iterator on library documents, spatial iterators conducting searches within a restricted boundary on a document and the group iterator conducting searches for primitives within a group object such as tracks and arcs within a component object.

The scripting system's Delphi Script doesn't support sets, therefore to pass in a set of layers or a set of objects, you need to use the **MkSet** function to create a pseudo set of objects or layers for the **AddFilter_ObjectSet** or **AddFilterLayerSet** procedures.

For example

```
BoardIterator.AddFilter_ObjectSet(MkSet(eTrackObject,eFillObject));
```

See also

IPCB_AbstractIterator interface

IPCB_BoardIterator interface

IPCB_LibraryIterator interface

IPCB_SpatialIterator interface

IPCB_GroupIterator interface

IPCB_AbstractIterator

Overview

An abstract iterator object interface which is the ancestor interface for a board, spatial, group and library Iterators. An iterator object iterates through a design database to fetch specified objects within a specified region on a specified layer if necessary.

Notes

- To specify the object set or the layer set, you need to use the **MkSet** function to create a set of objects. Delphiscript language does not support Object Pascal's sets.

Methods

```
Function   I_ObjectAddress       : TPCBObjectHandle;
Function   FirstPCBObject        : IPCB_Primitive;
Function   NextPCBObject         : IPCB_Primitive


Procedure SetState_FilterAll;
```

```
Procedure AddFilter_ObjectSet (AObjectSet  : TObjectSet);
Procedure AddFilter_LayerSet  (ALayerSet   : TLayerSet);
Procedure AddFilter_Area      (X1,
                               Y1,
                               X2,
                               Y2 : TCoord);
Procedure AddFilter_AllLayers;
```

See also

IPCB_BoardIterator interface

IPCB_LibraryIterator interface

IPCB_SpatialIterator interface

IPCB_Primitive interface

TObjectSet

TObjectId enumerated values

TLayerSet

TLayer enumerated values

MkSet function

IPCB_BoardIterator

Overview

The **IPCB_BoardIterator** iterates through a PCB document to fetch PCB design objects on this PCB.

With the iterator, you can control which objects on which layers and within specified regions with the **AddFilter_ObjectSet**, **AddFilter_LayerSet** and **AddFilter_Area** methods to be fetched.

The **AddFilter_method** controls how design objects are fetched. The **TIterationMethod** type has three different values; eProcessAll, eProcessFree, eProcessComponents.

Notes

- Delphiscript doesn't support sets, therefore to pass in a set of layers or a set of objects, you need to use the **MkSet** function to create a pseudo set of objects or layers for the **AddFilter_ObjectSet** or **AddFilterLayerSet** procedures. For example **BoardIterator.AddFilter_ObjectSet(MkSet(eTrackObject,eFillObject));**

Methods

```
Function   I_ObjectAddress   : TPCBObjectHandle;

Function   FirstPCBObject    : IPCB_Primitive;
Function   NextPCBObject     : IPCB_Primitive
```

```
Procedure SetState_FilterAll;

Procedure AddFilter_ObjectSet (AObjectSet  : TObjectSet);
Procedure AddFilter_LayerSet  (ALayerSet   : TLayerSet);
Procedure AddFilter_Area      (X1,
                               Y1,
                               X2,
                               Y2          : TCoord);

Procedure AddFilter_AllLayers;
Procedure AddFilter_Method (AMethod : TIterationMethod);
```

Example

```
Var
    BoardHandle : IPCB_Board;
    Pad         : IPCB_Primitive;
    Iterator    : IPCB_BoardIterator;
    PadNumber   : Integer;
Begin
    // Retrieve the current board
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;

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

    PadNumber       := 0;
    // 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));
```

See also

IPCB_BoardIterator interface

IPCB_LibraryIterator interface

IPCB_SpatialIterator interface

IPCB_Primitive interface

TObjectSet

TObjectId enumerated values

TIterationMethod enumerated values

TLayerSet

TLayer enumerated values

MkSet function

Scripts in the **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

IPCB_LibraryIterator

Overview

The IPCB_LibraryIterator object interface Iterates through a loaded PCB library in DXP to fetch PCB footprints and its primitives. The footprints of a PCB library is equivalent to "pages" of a library. The library iterator basically retrieves the footprints and to retrieve the child objects of each footprint, you need to employ the group iterator.

Notes

- IPCB_LibraryIterator has only methods inherited from the IPCB_AbstractIterator interface and is reproduced here for reference.
- A library is represented by the IPCB_Board interface.
- A PCB footprint (as a page of the library) is represented by its IPCB_LibComponent interface which is inherited from the IPCB_Group object interface.
- A PCB footprint is composed of child objects such as pads and tracks. Therefore the footprint has its own IPCB_GroupIterator to fetch its own child objects.
- Delphiscript doesnt support sets, therefore to pass in a set of layers or a set of objects, you need to use the **MkSet** function to create a pseudo set of objects or layers for the AddFilter_ObjectSet or AddFilterLayerSet procedures.
- For example LibraryIterator.AddFilter_ObjectSet(**MkSet**(eTrackObject,eFillObject));

Methods

```
Function  I_ObjectAddress   : TPCBObjectHandle;

Function  FirstPCBObject   : IPCB_Primitive;
Function  NextPCBObject    : IPCB_Primitive

Procedure AddFilter_ObjectSet (AObjectSet  : TObjectSet);
Procedure AddFilter_LayerSet  (ALayerSet   : TLayerSet);
Procedure AddFilter_Area      (X1,
                               Y1,
                               X2,
                               Y2 : TCoord);
Procedure AddFilter_AllLayers;

Procedure SetState_FilterAll;
```

Example

```
Procedure LookInsideFootprints;
Var
    CurrentLib         : IPCB_Board;
    AObject            : IPCB_Primitive;
    FootprintIterator : IPCB_LibraryIterator;
    Iterator           : IPCB_GroupIterator;
    Footprint          : IPCB_LibComponent;
    FirstTime          : Boolean;
    NoOfPrims          : Integer;
    S                  : TString;
Begin
    CurrentLib := PCBServer.GetCurrentPCBBoard;
    If CurrentLib = Nil Then
    Begin
        ShowMessage('This is not a PCB document');
        Exit;
    End;
    // Check if CurrentLib is a library otherwise exit.
    If Not CurrentLib.IsLibrary Then
    Begin
```

```
        showMessage('This is not a PCB library document.');
        Exit;
    End;
    // For each page of library is a footprint
    FootprintIterator := CurrentLib.LibraryIterator_Create;
    FootprintIterator.SetState_FilterAll;
    S          := '';
    FirstTime := True;
    Try
        // Within each page, fetch primitives of the footprint
        // A footprint is a IPCB_LibComponent inherited from
        // IPCB_Group which is a container object storing primitives.
        Footprint := FootprintIterator.FirstPCBObject; // IPCB_LibComponent
        While Footprint <> Nil Do
        Begin
           If FirstTime Then
           Begin
              S := S + ExtractFileName(Footprint.Board.FileName) + #13;
              S := S + ' Current Footprint : ' +
                   PCBServer.GetCurrentComponent(CurrentLib)+ #13 + #13;
           End;

           S := S + Footprint.Name;

           Iterator := Footprint.GroupIterator_Create;
           Iterator.SetState_FilterAll;
           // Counts number of prims for each Footprint as a
IPCB_LibComponent
           // Note that the IPCB_LibComponent has a GetPrimitiveCount method
           NoOfPrims := 0;
           AObject := Iterator.FirstPCBObject;
           While (AObject <> Nil) Do
           Begin
               // counts child objects or primitives
               // for each footprint.
               Inc(NoOfPrims);
```

```
                // do what you want with the AObject.
                AObject := Iterator.NextPCBObject;
            End;
            S := S + ' has ' + IntToStr(NoOfPrims) + ' Primitives.' + #13;
            FirstTime := False;
            Footprint.GroupIterator_Destroy(Iterator);
            Footprint := FootprintIterator.NextPCBObject;
        End;
    Finally
        CurrentLib.LibraryIterator_Destroy(FootprintIterator);
    End;
    ShowMessage(S);
End;
```

See also

IPCB_BoardIterator interface

IPCB_SpatialIterator interface

IPCB_GroupIterator interface

IPCB_Primitive interface

TObjectSet

TObjectId enumerated values

TLayerSet

TLayer enumerated values

MkSet function

LibraryIterator example from **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

IPCB_SpatialIterator

Overview

The IPCB_SpatialIterator interface iterates through a defined region on the loaded PCB document in DXP to fetch PCB design objects.

You will need to specify the object set, the layer set and the area for the spatial iterator to conduct its search within a defined boundary. The following methods are AddFilter_ObjectSet, AddFilter_LayerSet and AddFilter_Area.

Notes

- **IPCB_SpatialIterator** has only methods inherited from the **IPCB_AbstractIterator** interface and is reproduced here for reference.

- Delphiscript doesn't support sets, therefore to pass in a set of layers or a set of objects, you need to use the **MkSet** function to create a pseudo set of objects or layers for the AddFilter_ObjectSet or AddFilterLayerSet procedures. For example SpatialIterator.AddFilter_ObjectSet(**MkSet**(eTrackObject,eFillObject));

Methods (inherited from IPCB_AbstractIterator)

```
Function  I_ObjectAddress  : TPCBObjectHandle;

Function  FirstPCBObject  : IPCB_Primitive;
Function  NextPCBObject   : IPCB_Primitive

Procedure AddFilter_ObjectSet  (AObjectSet  : TObjectSet);
Procedure AddFilter_LayerSet   (ALayerSet   : TLayerSet);
Procedure AddFilter_Area       (X1,
                                Y1,
                                X2,
                                Y2 : TCoord);

Procedure AddFilter_AllLayers;

Procedure SetState_FilterAll;
```

Example

```
    (* Top/Bottom Layers and Arc/Track objects defined *)
    (* for the Spatial iterator constraints *)
    ASetOfLayers  := MkSet(eTopLayer,eBottomLayer);
    ASetOfObjects := MkSet(eArcObject,eTrackObject);

    Iterator := Board.SpatialIterator_Create;
    Iterator.AddFilter_ObjectSet(ASetOfObjects);
    Iterator.AddFilter_LayerSet(ASetOfLayers);
    Iterator.AddFilter_Area(X1,Y1,X2,Y2);

    (* Iterate for tracks and arcs on bottom/top layers *)
    PCBObject := Iterator.FirstPCBObject;
    While PCBObject <> 0 Do
    Begin
         PCBObject.Selected := True;
```

```
        PCBObject := Iterator.NextPCBObject;
    End;
    Board.SpatialIterator_Destroy(Iterator);
```

See also

IPCB_BoardIterator interface

IPCB_LibraryIterator interface

IPCB_GroupIterator interface.

IPCB_Primitive interface

TObjectSet

TObjectId enumerated values

TLayerSet

TLayer enumerated values

MkSet function

Spatial iterator script in **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

IPCB_GroupIterator

Overview

The **IPCB_GroupIterator** interface deals with group objects such as board layouts, polygons, components, footprints in a PCB library, coordinates and dimensions that have child objects within.

When you need to fetch child objects of a group object such as tracks and arcs of a footprint in a PCB library, you need to create a Group Iterator for that group object.

The sequence is basically as follows;

- Set up a board iterator to fetch design objects from the PCB/Library document
- For each design object that is a group object (such as polygons and components), setup a group iterator and fetch child objects for that group object.
- Destroy the group iterator when finished iterating child objects for that group object
- Destroy the board/library iterator when finished iterating

Notes

- IPCB_GroupIterator has methods inherited from the IPCB_AbstractIterator interface and is reproduced here for reference.
- Delphiscript does not support sets, therefore to pass in a set of layers or a set of objects, you need to use the MkSet function to create a pseudo set of objects or layers for the AddFilter_ObjectSet or AddFilterLayerSet procedures.
- For example LibraryIterator.AddFilter_ObjectSet(**MkSet**(eTrackObject,eFillObject));

Methods

```
Function  I_ObjectAddress    : TPCBObjectHandle;
```

```
Function  FirstPCBObject     : IPCB_Primitive;
Function  NextPCBObject      : IPCB_Primitive

Procedure AddFilter_ObjectSet  (AObjectSet  : TObjectSet);
Procedure AddFilter_LayerSet   (ALayerSet   : TLayerSet);
Procedure AddFilter_Area       (X1,
                                Y1,
                                X2,
                                Y2 : TCoord);

Procedure AddFilter_AllLayers;

Procedure SetState_FilterAll;
```

Example

```
Procedure CountTracks;
Var
    Track             : IPCB_Track;
    ChildIterator     : IPCB_GroupIterator;
    Component         : IPCB_Component;
    ComponentIterator : IPCB_BoardIterator;
    TrackCount        : Integer;
Begin
    TrackCount     := 0;
    If PCBServer.GetCurrentPCBBoard = Nil Then Exit;

    // Create a board iterator to fetch a component.
    ComponentIteratorHandle :=
PCBServer.GetCurrentPCBBoard.BoardIterator_Create;
    ComponentIteratorHandle.AddFilter_ObjectSet(MkSet(eComponentObject));

    If Component <> Nil Then
    Begin
        // Create an iterator from the component to fetch
        // its child objects.
        ChildIterator := Component.GroupIterator_Create;
        ChildIterator.AddFilter_ObjectSet(MkSet(eTrackObject));
```

```
        ChildIterator.AddFilter_LayerSet(MkSet(eTopOverlay));
        Track := ChildIterator.FirstPCBObject;
        While (Track <> Nil) Do
        Begin
            Inc(TrackCount);
            Track := ChildIterator.NextPCBObject;
        End;

        ShowInfo('This component ' + Component.SourceDesignator  +
                 ' has ' +  IntToStr(TrackCount)  + ' tracks.');
        // When finished iterating component's child objects,
        // destroy the component's        group iterator.
        Component.GroupIterator_Destroy(TrackIterator);
    End;
    // when finished iterating on PCB document, destroy the board iterator.
    PCBServer.GetCurrentPCBBoard.BoardIterator_Destroy(ComponentIterator);
End;
```

See also

IPCB_BoardIterator interface

IPCB_LibraryIterator interface

IPCB_SpatialIterator interface

IPCB_Primitive interface

TObjectSet

TObjectId enumerated values

TLayerSet

TLayer enumerated values

MkSet function in Delphiscript Reference

LibraryIterator script example in **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

CountTracksInComponent script example in **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

PCB Design Objects

A PCB design object on a PCB document is represented by its interface. An interface represents an existing object in memory and its properties and methods can be invoked.

A PCB design object is basically a primitive or a group object. A primitive can be a track or an arc object. A group object is an object that is composed of child objects. For example a board outline or a component is a group object.

Since many design objects are descended from ancestor interfaces and thus the ancestor methods and properties are also available to use.

For example the **IPCB_Text** interface is inherited from an immediate **IPCB_RectangularPrimitive** interface and in turn inherited from the **IPCB_Primitive** interface. If you check the **IPCB_Text** entry in this online help you will see the following information.

The IPCB_Text Interface hierarchy is as follows:

- **IPCB_Primitive**
  - **IPCB_RectangularPrimitive**
    - **IPCB_Text**

and so on.

This PCB Design Objects section is broken up into several categories- Primitives, Group Objects and Rectangular Objects.

- Primitives include arcs, embedded objects, fills, fromtos, pads, nets, tracks, vias, violations, object classes and connections.
- Group objects include board outlines, coordinates, components, polygons, library components (footprints), nets and dimensions (including different dimension types).
- Rectangular objects include text objects, embedded board arrays and fills.

See also

PCB Documents

PCB Objects

Creating/Deleting objects and updating the Undo system

Modifying PCB objects and updating the Undo system

IPCB_Arc

IPCB_ObjectClass

IPCB_Pad

IPCB_Via

IPCB_Track

IPCB_Embedded

IPCB_Violation

IPCB_Text

IPCB_Fill

IPCB_Coordinate

IPCB_Dimension

IPCB_Component

IPCB_Polygon

IPCB_Net

IPCB_LibComponent

IPCB_EmbeddedBoard

IPCB_Primitive Interface

Overview

The **IPCB_Primitive** interface hierarchy is as follows.

IPCB_Primitive methods	IPCB_Primitive properties
GetState_Board	Board
GetState_ObjectId	ObjectId
GetState_Layer	Layer
GetState_Selected	Index
SetState_Selected	Selected
GetState_IsPreRoute	IsPreRoute
SetState_IsPreRoute	InSelectionMemory [I
GetState_InSelectionMemory	PadCacheRobotFlag
SetState_InSelectionMemory	Enabled
GetState_PadCacheRobotFlag	Enabled_Direct
SetState_PadCacheRobotFlag	Enabled_vNet
GetState_Enabled	Enabled_vPolygon
SetState_Enabled	Enabled_vComponent
GetState_Enabled_Direct	Enabled_vCoordinate
SetState_Enabled_Direct	Enabled_vDimension
GetState_Enabled_vNet	Used
SetState_Enabled_vNet	DRCError
GetState_Enabled_vPolygon	MiscFlag1
SetState_Enabled_vPolygon	MiscFlag2
GetState_Enabled_vComponent	MiscFlag3
SetState_Enabled_vComponent	EnableDraw
GetState_Enabled_vCoordinate	Moveable

SetState_Enabled_vCoordinate
GetState_Enabled_vDimension
SetState_Enabled_vDimension
GetState_Used
SetState_Used
GetState_DRCError
SetState_DRCError
GetState_MiscFlag1
SetState_MiscFlag1
GetState_MiscFlag2
SetState_MiscFlag2
GetState_MiscFlag3
SetState_MiscFlag3
GetState_EnableDraw
SetState_EnableDraw
GetState_Moveable
SetState_Moveable
GetState_UserRouted
SetState_UserRouted
GetState_TearDrop
SetState_TearDrop
GetState_IsTenting
SetState_IsTenting
GetState_IsTenting_Top
SetState_IsTenting_Top
GetState_IsTenting_Bottom
SetState_IsTenting_Bottom
GetState_IsTestPoint_Top
SetState_IsTestPoint_Top
GetState_IsTestPoint_Bottom
SetState_IsTestPoint_Bottom
GetState_IsKeepout
SetState_IsKeepout
GetState_AllowGlobalEdit
SetState_AllowGlobalEdit
GetState_PolygonOutline

UserRouted
TearDrop
IsTenting
IsTenting_Top
IsTenting_Bottom
IsTestpoint_Top
IsTestpoint_Bottom
IsKeepout
AllowGlobalEdit
PolygonOutline
InBoard
InPolygon
InComponent
InNet
InCoordinate
InDimension
IsElectricalPrim
ObjectIDString
Identifier
Descriptor
Detail
PowerPlaneConnectStyle
ReliefConductorWidth
ReliefEntries
ReliefAirGap
PasteMaskExpansion
SolderMaskExpansion
PowerPlaneClearance
PowerPlaneReliefExpansion
Net
Component
Polygon
Coordinate
Dimension
ViewableObjectID

SetState_PolygonOutline
GetState_InBoard
SetState_InBoard
GetState_InPolygon
SetState_InPolygon
GetState_InComponent
SetState_InComponent
GetState_InNet
SetState_InNet
GetState_InCoordinate
SetState_InCoordinate
GetState_InDimension
SetState_InDimension
GetState_IsElectricalPrim
SetState_Board
SetState_Layer
GetState_ObjectIDString
GetState_Identifier
GetState_DescriptorString
GetState_DetailString
GetState_Index
SetState_Index
GetState_PowerPlaneConnectStyle
GetState_ReliefConductorWidth
GetState_ReliefEntries
GetState_ReliefAirGap
GetState_PasteMaskExpansion
GetState_SolderMaskExpansion
GetState_PowerPlaneClearance
GetState_PowerPlaneReliefExpansion
GetState_Net
GetState_Component
GetState_Polygon
GetState_Coordinate
GetState_Dimension
GetState_ViewableObjectID

SetState_Net
SetState_Component
SetState_Polygon
SetState_Coordinate
SetState_Dimension
I_ObjectAddress
BoundingRectangle
BoundingRectangleForSelection
BoundingRectangleForPainting
IsHidden
IsFreePrimitive
IsSaveable
AddPCBObject
RemovePCBObject
MoveByXY
MoveToXY
RotateBy
FlipXY
Mirror
SwapLayerPairs
GraphicallyInvalidate

IPCB_Primitive Interface methods

AddPCBObject method

(IPCB_Primitive interface)

Syntax

```
Procedure AddPCBObject;
```

Description

This method adds a child PCB object within this object.

See also

IPCB_Primitive interface

BoundingRectangle method

(IPCB_Primitive interface)

Syntax

```
Function BoundingRectangle : TCoordRect;
```

Description

This method obtains the coordinates (of TCoordRect type) for the bounding rectangle of the PCB design object.

See also

IPCB_Primitive interface

BoundingRectangleForPainting method

(IPCB_Primitive interface)

Syntax

```
Function BoundingRectangleForPainting : TCoordRect;
```

Description

This method obtains the coordinates (of TCoordRect type) for the bounding rectangle of the PCB design object for re-painting on the PCB document.

See also

IPCB_Primitive interface

BoundingRectangleForSelection method

(IPCB_Primitive interface)

Syntax

```
Function BoundingRectangleForSelection : TCoordRect;
```

Description

This method obtains the coordinates (of TCoordRect type) for the bounding rectangle of the PCB design object for selection purposes on the PCB document.

See also

IPCB_Primitive interface

FlipXY method

(IPCB_Primitive interface)

Syntax

```
Procedure FlipXY (Axis : TCoord;MirrOp : TMirrorOperation);
```

Description

This method flips the object across the specified axis of TCoord and how it is mirrored according to the TMirrorOperation type.

See also

IPCB_Primitive interface

TMirrorOperation type

TCoord type

GetState_AllowGlobalEdit method

(IPCB_Primitive interface)

Syntax

```
Function GetState_AllowGlobalEdit : Boolean;
```

See also

IPCB_Primitive interface

GetState_Board method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Board : IPCB_Board;
```

See also

IPCB_Primitive interface

GetState_Component method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Component : IPCB_Component;
```

See also

IPCB_Primitive interface

GetState_Coordinate method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Coordinate : IPCB_Coordinate;
```

See also

IPCB_Primitive interface

GetState_DescriptorString method

(IPCB_Primitive interface)

Syntax

```
Function GetState_DescriptorString : TPCBString;
```

See also

IPCB_Primitive interface

GetState_DetailString method

(IPCB_Primitive interface)

Syntax

```
Function GetState_DetailString : TPCBString;
```

See also

IPCB_Primitive interface

GetState_Dimension method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Dimension : IPCB_Dimension;
```

See also

IPCB_Primitive interface

GetState_DRCError method

(IPCB_Primitive interface)

Syntax

```
Function GetState_DRCError : Boolean;
```

See also

IPCB_Primitive interface

GetState_Enabled method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Enabled : Boolean;
```

See also

IPCB_Primitive interface

GetState_Enabled_Direct method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Enabled_Direct : Boolean;
```

See also

IPCB_Primitive interface

GetState_Enabled_vComponent method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Enabled_vComponent : Boolean;
```

See also

IPCB_Primitive interface

GetState_Enabled_vCoordinate method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Enabled_vCoordinate : Boolean;
```

See also

IPCB_Primitive interface

GetState_Enabled_vDimension method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Enabled_vDimension : Boolean;
```

See also

IPCB_Primitive interface

GetState_Enabled_vNet method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Enabled_vNet : Boolean;
```

See also

IPCB_Primitive interface

GetState_Enabled_vPolygon method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Enabled_vPolygon : Boolean;
```

See also

IPCB_Primitive interface

GetState_EnableDraw method

(IPCB_Primitive interface)

Syntax

```
Function GetState_EnableDraw : Boolean;
```

See also

IPCB_Primitive interface

GetState_Identifier method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Identifier : TPCBString;
```

See also

IPCB_Primitive interface

GetState_InBoard method

(IPCB_Primitive interface)

Syntax

```
Function GetState_InBoard : Boolean;
```

See also

IPCB_Primitive interface

GetState_InComponent method

(IPCB_Primitive interface)

Syntax

```
Function GetState_InComponent : Boolean;
```

See also

IPCB_Primitive interface

GetState_InCoordinate method

(IPCB_Primitive interface)

Syntax

```
Function GetState_InCoordinate : Boolean;
```

See also

IPCB_Primitive interface

GetState_Index method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Index : Word;
```

See also

IPCB_Primitive interface

GetState_InDimension method

(IPCB_Primitive interface)

Syntax

```
Function GetState_InDimension : Boolean;
```

See also

IPCB_Primitive interface

GetState_InNet method

(IPCB_Primitive interface)

Syntax

```
Function GetState_InNet : Boolean;
```

See also

IPCB_Primitive interface

GetState_InPolygon method

(IPCB_Primitive interface)

Syntax

```
Function GetState_InPolygon : Boolean;
```

See also

IPCB_Primitive interface

GetState_InSelectionMemory method

(IPCB_Primitive interface)

Syntax

```
Function GetState_InSelectionMemory (Index : Integer) : Boolean;
```

See also

IPCB_Primitive interface

GetState_IsElectricalPrim method

(IPCB_Primitive interface)

Syntax

```
Function GetState_IsElectricalPrim : Boolean;
```

See also

IPCB_Primitive interface

GetState_IsKeepout method

(IPCB_Primitive interface)

Syntax

```
Function GetState_IsKeepout : Boolean;
```

See also

IPCB_Primitive interface

GetState_IsPreRoute method

(IPCB_Primitive interface)

Syntax

```
Function GetState_IsPreRoute : Boolean;
```

See also

IPCB_Primitive interface

GetState_IsTenting method

(IPCB_Primitive interface)

Syntax

```
Function GetState_IsTenting : Boolean;
```

See also

IPCB_Primitive interface

GetState_IsTenting_Bottom method

(IPCB_Primitive interface)

Syntax

```
Function GetState_IsTenting_Bottom : Boolean;
```

See also

IPCB_Primitive interface

GetState_IsTenting_Top method

(IPCB_Primitive interface)

Syntax

```
Function GetState_IsTenting_Top : Boolean;
```

See also

IPCB_Primitive interface

GetState_IsTestPoint_Bottom method

(IPCB_Primitive interface)

Syntax

```
Function GetState_IsTestPoint_Bottom : Boolean;
```

See also

IPCB_Primitive interface

GetState_IsTestPoint_Top method

(IPCB_Primitive interface)

Syntax

```
Function GetState_IsTestPoint_Top : Boolean;
```

See also

IPCB_Primitive interface

GetState_Layer method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Layer : TLayer;
```

See also

IPCB_Primitive interface

GetState_MiscFlag1 method

(IPCB_Primitive interface)

Syntax

```
Function GetState_MiscFlag1 : Boolean;
```

See also

IPCB_Primitive interface

GetState_MiscFlag2 method

(IPCB_Primitive interface)

Syntax

```
Function GetState_MiscFlag2 : Boolean;
```

See also

IPCB_Primitive interface

GetState_MiscFlag3 method

(IPCB_Primitive interface)

Syntax

```
Function GetState_MiscFlag3 : Boolean;
```

See also

IPCB_Primitive interface

GetState_Moveable method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Moveable : Boolean;
```

See also

IPCB_Primitive interface

GetState_Net method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Net : IPCB_Net;
```

See also

IPCB_Primitive interface

GetState_ObjectId method

(IPCB_Primitive interface)

Syntax

```
Function GetState_ObjectId : TObjectId;
```

See also

IPCB_Primitive interface

GetState_ObjectIDString method

(IPCB_Primitive interface)

Syntax

```
Function GetState_ObjectIDString : TPCBString;
```

See also

IPCB_Primitive interface

GetState_PadCacheRobotFlag method

(IPCB_Primitive interface)

Syntax

```
Function GetState_PadCacheRobotFlag : Boolean;
```

See also

IPCB_Primitive interface

GetState_PasteMaskExpansion method

(IPCB_Primitive interface)

Syntax

```
Function GetState_PasteMaskExpansion : TCoord;
```

See also

IPCB_Primitive interface

GetState_Polygon method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Polygon : IPCB_Polygon;
```

See also

IPCB_Primitive interface

GetState_PolygonOutline method

(IPCB_Primitive interface)

Syntax

```
Function GetState_PolygonOutline : Boolean;
```

See also

IPCB_Primitive interface

GetState_PowerPlaneClearance method

(IPCB_Primitive interface)

Syntax

```
Function GetState_PowerPlaneClearance : TCoord;
```

See also

IPCB_Primitive interface

GetState_PowerPlaneConnectStyle method

(IPCB_Primitive interface)

Syntax

```
Function GetState_PowerPlaneConnectStyle : TPlaneConnectStyle;
```

See also

IPCB_Primitive interface

GetState_PowerPlaneReliefExpansion method

(IPCB_Primitive interface)

Syntax

```
Function GetState_PowerPlaneReliefExpansion : TCoord;
```

See also

IPCB_Primitive interface

GetState_ReliefAirGap method

(IPCB_Primitive interface)

Syntax

```
Function GetState_ReliefAirGap : TCoord;
```

See also

IPCB_Primitive interface

GetState_ReliefConductorWidth method

(IPCB_Primitive interface)

Syntax

```
Function GetState_ReliefConductorWidth : TCoord;
```

See also

IPCB_Primitive interface

GetState_ReliefEntries method

(IPCB_Primitive interface)

Syntax

```
Function GetState_ReliefEntries : Integer;
```

See also

IPCB_Primitive interface

GetState_Selected method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Selected : Boolean;
```

See also

IPCB_Primitive interface

GetState_SolderMaskExpansion method

(IPCB_Primitive interface)

Syntax

```
Function GetState_SolderMaskExpansion : TCoord;
```

See also

IPCB_Primitive interface

GetState_TearDrop method

(IPCB_Primitive interface)

Syntax

```
Function GetState_TearDrop : Boolean;
```

See also

IPCB_Primitive interface

GetState_Used method

(IPCB_Primitive interface)

Syntax

```
Function GetState_Used : Boolean;
```

See also

IPCB_Primitive interface

GetState_UserRouted method

(IPCB_Primitive interface)

Syntax

```
Function GetState_UserRouted : Boolean;
```

See also

IPCB_Primitive interface

SetState_AllowGlobalEdit method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_AllowGlobalEdit (Value : Boolean);
```

See also

IPCB_Primitive interface

GetState_ViewableObjectID method

(IPCB_Primitive interface)

Syntax

```
Function GetState_ViewableObjectID : TViewableObjectID;
```

See also

IPCB_Primitive interface

GraphicallyInvalidate method

(IPCB_Primitive interface)

Syntax

```
Procedure GraphicallyInvalidate;
```

Description

This procedure renders the PCB design object invalid, so a forced paint operation takes place to refresh the graphical contents of this object.

See also

IPCB_Primitive interface

IsElectricalPrim property

(IPCB_Primitive interface)

Syntax

```
Property IsElectricalPrim : Boolean Read GetState_IsElectricalPrim;
```

Description

This property returns a boolean value whether this object has an electrical property or not.

See also

IPCB_Primitive interface

IsFreePrimitive method

(IPCB_Primitive interface)

Syntax

```
Function IsFreePrimitive : Boolean;
```

Description

This function returns a boolean value determining whether the object is a free primitive (as a standalone object) or not.

See also

IPCB_Primitive interface

IsHidden method

(IPCB_Primitive interface)

Syntax

```
Function IsHidden : Boolean;
```

See also

IPCB_Primitive interface

IsKeepout property

(IPCB_Primitive interface)

Syntax

```
Property IsKeepout : Boolean Read GetState_IsKeepout Write
SetState_IsKeepout;
```

See also

IPCB_Primitive interface

IsPreRoute property

(IPCB_Primitive interface)

Syntax

```
Property IsPreRoute : Boolean Read GetState_IsPreRoute Write
SetState_IsPreRoute;
```

See also

IPCB_Primitive interface

IsSaveable method

(IPCB_Primitive interface)

Syntax

```
Function IsSaveable (AVer : TAdvPCBFileFormatVersion) : Boolean;
```

See also

IPCB_Primitive interface

I_ObjectAddress method

(IPCB_Primitive interface)

Syntax

```
Function I_ObjectAddress : TPCBObjectHandle;
```

Description

The I_ObjectAddress function returns the pointer value or the handle of the PCB object. Normally you will use this function when using the PCB robots and sending PCB messages.

Example

```
//Notify PCB that the fill object has been changed.
```

```
PCBServer.SendMessageToRobots(
        Fill.I_ObjectAddress,
        c_Broadcast,
        PCBM_EndModify ,
        c_NoEventData);
```

See also

IPCB_Primitive interface

SendMessageToRobots method

MoveByXY method

(IPCB_Primitive interface)

Syntax

```
Procedure MoveByXY (AX, AY : TCoord);
```

See also

IPCB_Primitive interface

MoveToXY method

(IPCB_Primitive interface)

Syntax

```
Procedure MoveToXY (AX, AY : TCoord);
```

See also

IPCB_Primitive interface

RemovePCBObject method

(IPCB_Primitive interface)

Syntax

```
Procedure RemovePCBObject;
```

Description

This procedure removes a child object associated with this PCB design object. Normally a group object is composed of child objects and you provoke this method to remove child objects.

See also

IPCB_Primitive interface

AddPCBObject method

IPCB_Group interface

SetState_Board method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Board (ABoard : IPCB_Board);
```

See also

IPCB_Primitive interface

SetState_Component method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Component (Value : IPCB_Component);
```

See also

IPCB_Primitive interface

SetState_Coordinate method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Coordinate (Value : IPCB_Coordinate);
```

See also

IPCB_Primitive interface

SetState_Dimension method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Dimension (Value : IPCB_Dimension);
```

See also

IPCB_Primitive interface

SetState_DRCError method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_DRCError (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Enabled method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Enabled (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Enabled_Direct method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Enabled_Direct (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Enabled_vComponent method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Enabled_vComponent (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Enabled_vCoordinate method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Enabled_vCoordinate(Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Enabled_vDimension method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Enabled_vDimension (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Enabled_vNet method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Enabled_vNet (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Enabled_vPolygon method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Enabled_vPolygon (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_EnableDraw method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_EnableDraw (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_InBoard method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_InBoard (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_InComponent method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_InComponent (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_InCoordinate method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_InCoordinate (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Index method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Index (AIndex : Word);
```

See also

IPCB_Primitive interface

SetState_InDimension method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_InDimension (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_InNet method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_InNet (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_InPolygon method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_InPolygon (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_InSelectionMemory method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_InSelectionMemory (Index : Integer;Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_IsKeepout method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_IsKeepout (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_IsPreRoute method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_IsPreRoute (B : Boolean);
```

See also

IPCB_Primitive interface

SetState_IsTenting method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_IsTenting (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_IsTenting_Bottom method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_IsTenting_Bottom (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_IsTenting_Top method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_IsTenting_Top (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_IsTestPoint_Bottom method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_IsTestPoint_Bottom (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_IsTestPoint_Top method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_IsTestPoint_Top (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Layer method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Layer (ALayer : TLayer);
```

See also

IPCB_Primitive interface

SetState_MiscFlag1 method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_MiscFlag1 (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_MiscFlag2 method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_MiscFlag2 (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_MiscFlag3 method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_MiscFlag3 (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Moveable method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Moveable (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Net method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Net (Value : IPCB_Net);
```

See also

IPCB_Primitive interface

SetState_PadCacheRobotFlag method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_PadCacheRobotFlag (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Polygon method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Polygon (Value : IPCB_Polygon);
```

See also

IPCB_Primitive interface

SetState_PolygonOutline method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_PolygonOutline (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Selected method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Selected (B : Boolean);
```

See also

IPCB_Primitive interface

SetState_TearDrop method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_TearDrop (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_Used method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_Used (Value : Boolean);
```

See also

IPCB_Primitive interface

SetState_UserRouted method

(IPCB_Primitive interface)

Syntax

```
Procedure SetState_UserRouted (Value : Boolean);
```

See also

IPCB_Primitive interface

SwapLayerPairs method

(IPCB_Primitive interface)

Syntax

```
Procedure SwapLayerPairs;
```

See also

IPCB_Primitive interface

IPCB_Primitive Interface Properties

AllowGlobalEdit property

(IPCB_Primitive interface)

Syntax

```
Property AllowGlobalEdit : Boolean Read GetState_AllowGlobalEdit Write
SetState_AllowGlobalEdit;
```

See also

IPCB_Primitive interface

Board property

(IPCB_Primitive interface)

Syntax

```
Property Board : IPCB_Board Read GetState_Board Write SetState_Board;
```

Description

The Board property returns the IPCB_Board interface that this object is associated with.

See also

IPCB_Primitive interface

Component property

(IPCB_Primitive interface)

Syntax

```
Property Component : IPCB_Component Read GetState_Component Write
SetState_Component;
```

Description

This Component property returns the IPCB_Component interface representing the component that this object is associated with (part of).

See also

IPCB_Primitive interface

Coordinate property

(IPCB_Primitive interface)

Syntax

```
Property Coordinate : IPCB_Coordinate Read GetState_Coordinate Write
SetState_Coordinate;
```

Description

This Coordinate property returns the IPCB_Coordinate interface representing the coordinate that this object is associated with (part of).

See also

IPCB_Primitive interface

Descriptor property

(IPCB_Primitive interface)

Syntax

```
Property Descriptor : TPCBString Read GetState_DescriptorString;
```

Description

This Descriptor property returns a string of TPCBString type associated with the PCB design object.

See also

IPCB_Primitive interface

TPCBString type

Detail property

(IPCB_Primitive interface)

Syntax

```
Property Detail : TPCBString Read GetState_DetailString;
```

See also

IPCB_Primitive interface

Dimension property

(IPCB_Primitive interface)

Syntax

```
Property Dimension : IPCB_Dimension Read GetState_Dimension Write
SetState_Dimension;
```

See also

IPCB_Primitive interface

DRCError property

(IPCB_Primitive interface)

Syntax

```
Property DRCError : Boolean Read GetState_DRCError Write SetState_DRCError;
```

See also

IPCB_Primitive interface

Enabled property

(IPCB_Primitive interface)

Syntax

```
Property Enabled : Boolean Read GetState_Enabled Write SetState_Enabled;
```

IPCB_Primitive interface

EnableDraw property

(IPCB_Primitive interface)

Syntax

```
Property EnableDraw : Boolean Read GetState_EnableDraw Write
SetState_EnableDraw;
```

See also

IPCB_Primitive interface

Enabled_Direct property

(IPCB_Primitive interface)

Syntax

```
Property Enabled_Direct : Boolean Read GetState_Enabled_Direct Write
SetState_Enabled_Direct;
```

See also

IPCB_Primitive interface

Enabled_vComponent property

(IPCB_Primitive interface)

Syntax

```
Property Enabled_vComponent : Boolean Read GetState_Enabled_vComponent Write
SetState_Enabled_vComponent;
```

See also

IPCB_Primitive interface

Enabled_vCoordinate property

(IPCB_Primitive interface)

Syntax

```
Property Enabled_vCoordinate : Boolean Read GetState_Enabled_vCoordinate
Write SetState_Enabled_vCoordinate;
```

See also

IPCB_Primitive interface

Enabled_vDimension property

(IPCB_Primitive interface)

Syntax

```
Property Enabled_vDimension : Boolean Read GetState_Enabled_vDimension Write
SetState_Enabled_vDimension;
```

See also

IPCB_Primitive interface

Enabled_vNet property

(IPCB_Primitive interface)

Syntax

```
Property Enabled_vNet : Boolean Read GetState_Enabled_vNet Write
SetState_Enabled_vNet;
```

See also

IPCB_Primitive interface

Enabled_vPolygon property

(IPCB_Primitive interface)

Syntax

```
Property Enabled_vPolygon : Boolean Read GetState_Enabled_vPolygon Write
SetState_Enabled_vPolygon;
```

See also

IPCB_Primitive interface

TearDrop property

(IPCB_Primitive interface)

Syntax

```
Property TearDrop : Boolean Read GetState_TearDrop Write SetState_TearDrop;
```

See also

IPCB_Primitive interface

Identifier property

(IPCB_Primitive interface)

Syntax

```
Property Identifier : TPCBString Read GetState_Identifier;
```

See also

IPCB_Primitive interface

InBoard property

(IPCB_Primitive interface)

Syntax

```
Property InBoard : Boolean Read GetState_InBoard Write SetState_InBoard;
```

See also

IPCB_Primitive interface

InComponent property

(IPCB_Primitive interface)

Syntax

```
Property InComponent : Boolean Read GetState_InComponent Write
SetState_InComponent;
```

Description

This property tests whether the PCB design object is in a Component object or not.

See also

IPCB_Primitive interface

InCoordinate property

(IPCB_Primitive interface)

Syntax

```
Property InCoordinate : Boolean Read GetState_InCoordinate Write
SetState_InCoordinate;
```

Description

This property tests whether the PCB design object is in a Coordinate object or not.

See also

IPCB_Primitive interface

Index property

(IPCB_Primitive interface)

Syntax

```
Property Index : Word Read GetState_Index Write SetState_Index;
```

See also

IPCB_Primitive interface

InDimension property

(IPCB_Primitive interface)

Syntax

```
Property InDimension : Boolean Read GetState_InDimension Write
SetState_InDimension;
```

Description

This property tests whether the PCB design object is in a Dimension object or not.

See also

IPCB_Primitive interface

InNet property

(IPCB_Primitive interface)

Syntax

```
Property InNet : Boolean Read GetState_InNet Write SetState_InNet;
```

Description

This property tests whether the PCB design object is part of a Net object or not.

See also

IPCB_Primitive interface

InPolygon property

(IPCB_Primitive interface)

Syntax

```
Property InPolygon : Boolean Read GetState_InPolygon Write
SetState_InPolygon;
```

Description

This property tests whether the PCB design object is in a Polygon object or not.

See also

IPCB_Primitive interface

InSelectionMemory property

(IPCB_Primitive interface)

Syntax

```
Property InSelectionMemory [I : Integer] : Boolean Read
GetState_InSelectionMemory Write SetState_InSelectionMemory;
```

See also

IPCB_Primitive interface

IsTenting property

(IPCB_Primitive interface)

Syntax

```
Property IsTenting : Boolean Read GetState_IsTenting Write
SetState_IsTenting;
```

See also

IPCB_Primitive interface

IsTenting_Bottom property

(IPCB_Primitive interface)

Syntax

```
Property IsTenting_Bottom : Boolean Read GetState_IsTenting_Bottom Write
SetState_IsTenting_Bottom;
```

See also

IPCB_Primitive interface

IsTenting_Top property

(IPCB_Primitive interface)

Syntax

```
Property IsTenting_Top : Boolean Read GetState_IsTenting_Top Write
SetState_IsTenting_Top;
```

See also

IPCB_Primitive interface

IsTestpoint_Bottom property

(IPCB_Primitive interface)

Syntax

```
Property IsTestpoint_Bottom : Boolean Read GetState_IsTestpoint_Bottom Write
SetState_IsTestpoint_Bottom;
```

See also

IPCB_Primitive interface

IsTestpoint_Top property

(IPCB_Primitive interface)

Syntax

```
Property IsTestpoint_Top : Boolean Read GetState_IsTestpoint_Top Write
SetState_IsTestpoint_Top;
```

See also

IPCB_Primitive interface

Layer property

(IPCB_Primitive interface)

Syntax

```
Property Layer : TLayer Read GetState_Layer Write SetState_Layer;
```

Description

The Layer property returns the layer type the PCB design object is on.

See also

IPCB_Primitive interface

TLayer type

Mirror method

(IPCB_Primitive interface)

Syntax

```
Procedure Mirror (Axis : TCoord;MirrOp : TMirrorOperation);
```

See also

IPCB_Primitive interface

MiscFlag1 property

(IPCB_Primitive interface)

Syntax

```
Property MiscFlag1 : Boolean Read GetState_MiscFlag1 Write
SetState_MiscFlag1;
```

See also

IPCB_Primitive interface

MiscFlag2 property

(IPCB_Primitive interface)

Syntax

```
Property MiscFlag2 : Boolean Read GetState_MiscFlag2 Write
SetState_MiscFlag2;
```

See also

IPCB_Primitive interface

MiscFlag3 property

(IPCB_Primitive interface)

Syntax

```
Property MiscFlag3 : Boolean Read GetState_MiscFlag3 Write
SetState_MiscFlag3;
```

See also

IPCB_Primitive interface

Moveable property

(IPCB_Primitive interface)

Syntax

```
Property Moveable : Boolean Read GetState_Moveable Write SetState_Moveable;
```

See also

IPCB_Primitive interface

Net property

(IPCB_Primitive interface)

Syntax

```
Property Net : IPCB_Net Read GetState_Net Write SetState_Net;
```

See also

IPCB_Primitive interface

IPCB_Net interface

ObjectId property

(IPCB_Primitive interface)

Syntax

```
Property ObjectId : TObjectId Read GetState_ObjectId;
```

Description

This property returns a ObjectID of TObjectId type denoting the type of object.

See also

IPCB_Primitive interface

TObjectID type

ObjectIDString property

(IPCB_Primitive interface)

Syntax

```
Property ObjectIDString : TPCBString Read GetState_ObjectIDString;
```

Description

This property returns the Object ID string denoting what type of PCB Design object.

See also

IPCB_Primitive interface

TPCBString type

PadCacheRobotFlag property

(IPCB_Primitive interface)

Syntax

```
Property PadCacheRobotFlag : Boolean Read GetState_PadCacheRobotFlag Write
SetState_PadCacheRobotFlag;
```

See also

IPCB_Primitive interface

PasteMaskExpansion property

(IPCB_Primitive interface)

Syntax

```
Property PasteMaskExpansion : TCoord Read GetState_PasteMaskExpansion;
```

Description

The Paste Mask Expansion Rule specifies the amount of radial expansion or radial contraction of each pad site. The Expansion property sets or returns the radial expansion or contraction value (a negative value denotes contraction).

Note that, a Paste Mask expansion property for a pad object is currently relevant just for pads on top and bottom copper layers. Vias do not have a paste mask layer. Paste mask layers are used to design stencils which will selectively place solder paste on a blank PCB. Solder paste is only placed on pads where component leads are to be soldered to them. Vias normally don't have anything soldered onto them.

See also

IPCB_Primitive interface

Polygon property

(IPCB_Primitive interface)

Syntax

```
Property Polygon : IPCB_Polygon Read GetState_Polygon Write
SetState_Polygon;
```

Description

This Polygone property returns the IPCB_Polygon interface representing the polygon that this object is associated with (part of).

See also

IPCB_Primitive interface

PolygonOutline property

(IPCB_Primitive interface)

Syntax

```
Property PolygonOutline : Boolean Read GetState_PolygonOutline Write
SetState_PolygonOutline;
```

See also

IPCB_Primitive interface

PowerPlaneClearance property

(IPCB_Primitive interface)

Syntax

```
Property PowerPlaneClearance : TCoord Read GetState_PowerPlaneClearance;
```

Description

The power plane clearance property determines the clearance of the power plane with a value of TCoord type.

See also

IPCB_Primitive interface

PowerPlaneConnectStyle property

(IPCB_Primitive interface)

Syntax

```
Property PowerPlaneConnectStyle : TPlaneConnectStyle Read
GetState_PowerPlaneConnectStyle;
```

Description

This power plane connect style rule specifies the style of the connection from a component pin to a power plane. There are two connection types - direct connections (the pin to solid copper) or thermal relief connection.

See also

IPCB_Primitive interface

TPlaneConnectStyle type

PowerPlaneReliefExpansion property

(IPCB_Primitive interface)

Syntax

```
Property PowerPlaneReliefExpansion : TCoord Read
GetState_PowerPlaneReliefExpansion;
```

See also

IPCB_Primitive interface

ReliefAirGap property

(IPCB_Primitive interface)

Syntax

```
Property ReliefAirGap : TCoord Read GetState_ReliefAirGap;
```

See also

IPCB_Primitive interface

ReliefConductorWidth property

(IPCB_Primitive interface)

Syntax

```
Property ReliefConductorWidth : TCoord Read GetState_ReliefConductorWidth;
```

See also

IPCB_Primitive interface

ReliefEntries property

(IPCB_Primitive interface)

Syntax

```
Property ReliefEntries : Integer Read GetState_ReliefEntries;
```

See also

IPCB_Primitive interface

RotateBy method

(IPCB_Primitive interface)

Syntax

```
Procedure RotateBy (Angle : TAngle);
```

See also

IPCB_Primitive interface

Selected property

(IPCB_Primitive interface)

Syntax

```
Property Selected : Boolean Read GetState_Selected Write SetState_Selected;
```

Description

This property denotes whether this PCB design object is selected or not. When a object is selected, it can be copied or moved on the PCB document.

See also

IPCB_Primitive interface

SolderMaskExpansion property

(IPCB_Primitive interface)

Syntax

```
Property SolderMaskExpansion : TCoord Read GetState_SolderMaskExpansion;
```

Description

The solder mask expansion rule defines the shape that is created on the solder mask layer at each pad and via site. This shape is expanded or contracted radially by the specified amount.

Note that, a Paste Mask expansion property for a pad object is currently relevant just for pads on top and bottom copper layers. Vias do not have a paste mask layer. Paste mask layers are used to design stencils which will selectively place solder paste on a blank PCB. Solder paste is only placed on pads where component leads are to be soldered to them. Vias normally don't have anything soldered onto them. Vias do not have a paste mask layer.

See also

IPCB_Primitive interface

Used property

(IPCB_Primitive interface)

Syntax

```
Property Used : Boolean Read GetState_Used Write SetState_Used;
```

See also

IPCB_Primitive interface

UserRouted property

(IPCB_Primitive interface)

Syntax

```
Property UserRouted : Boolean Read GetState_UserRouted Write
SetState_UserRouted;
```

See also

IPCB_Primitive interface

ViewableObjectID property

(IPCB_Primitive interface)

Syntax

```
Property ViewableObjectID : TViewableObjectID Read
GetState_ViewableObjectID;
```

See also

IPCB_Primitive interface

IPCB_Arc interface

Overview

Arcs are circular track segments with a definable width and can be placed on any layer. Arcs can have re-sizeable angles. You can set the angles to 0 and 360 respectively to obtain a circle object. Arcs have a variety of uses in the PCB design layout. For example, arcs can be used to outline component shapes. Arcs can also be placed on a signal layer and be electrically connected to tracks.

The **IPCB_Arc** hierarchy;

- **IPCB_Primitive**
    - **IPCB_Arc**

IPCB_Arc methods	IPCB_Arc properties
`RotateAroundXY`	`XCenter`
`GetState_StrictHitTest`	`YCenter`
	`Radius`
	`LineWidth`
	`StartAngle`
	`EndAngle`
	`StartX`
	`StartY`
	`EndX`
	`EndY`

Example

```
Var
    Board     : IPCB_Board;
    WorkSpace : IWorkSpace;
    Arc       : IPCB_Arc;
Begin
    // Create a new PCB documen
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('PCB');

    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil then exit;
```

```
    Arc := PCBServer.PCBObjectFactory(eArcObject, eNoDimension,
eCreate_Default);
    Arc.XCenter    := MilsToCoord(Board.XOrigin + 1800);
    Arc.YCenter    := MilsToCoord(Board.YOrigin + 1800);
    Arc.Radius     := MilsToCoord(200);
    Arc.LineWidth  := MilsToCoord(50);
    Arc.StartAngle := 0;
    Arc.EndAngle   := 270;
    Arc.Layer      := eBottomLayer;

    // Add the new arc object to the PCB database.
    Board.AddPCBObject(Arc);

    // Repaint the PCB Worksheet
    ResetParameters;
    AddStringParameter('Action', 'All');
    RunProcess('PCB:Zoom');
End;
```

See also

IPCB_Primitive interface

PCB Design Objects

IPCB_Arc interface methods

RotateAroundXY method

(IPCB_Arc interface)

Syntax

```
Procedure RotateAroundXY(AX,
                         AY        : TCoord;
                         Angle     : TAngle);
```

Description

This method provides the arc the ability to be rotated at a specified AX,AY and Angle values.

See also

IPCB_Arc interface

TCoord type

GetState_StrictHitTest method

(IPCB_Arc interface)

Syntax

```
Function  GetState_StrictHitTest(HitX, HitY : TCoord) : Boolean;
```

Description

This function tests if the arc object in question has been touched by the mouse click and returns a boolean value.

See also

IPCB_Arc interface

TCoord type

IPCB_Arc interface Properties

EndAngle property

(IPCB_Arc interface)

Syntax

```
Property EndAngle : TAngle Read GetState_EndAngle;
```

Description

This read only property returns the final angle of the arc object. The StartAngle property represents the first angle of the same arc object.

See also

IPCB_Arc interface

TAngle type

EndX property

(IPCB_Arc interface)

Syntax

```
Property EndX : TCoord Read GetState_EndX;
```

Description

This read only property returns the final X location of the arc object.

See also

IPCB_Arc interface

TCoord type

EndY property

(IPCB_Arc interface)

Syntax

```
Property EndY : TCoord Read GetState_EndY;
```

Description

This read only property returns the final Y location of the arc object as a TCoord value.

See also

IPCB_Arc interface

TCoord type

LineWidth property

(IPCB_Arc interface)

Syntax

```
Property  LineWidth                                : TCoord            Read
GetState_LineWidth            Write SetState_LineWidth;
```

Description

The **linewidth** property represents the line thickness of the arc.

See also

IPCB_Arc interface

TCoord value

Radius property

(IPCB_Radius interface)

Syntax

```
Property Radius : TCoord Read GetState_Radius Write SetState_Radius;
```

Description

The Radius property represents the radius of the arc in TCoord type.

See also

IPCB_Arc interface

TCoord type

StartAngle property

(IPCB_Arc interface)

Syntax

```
Property  StartAngle : TAngle Read GetState_StartAngle  Write
SetState_StartAngle;
```

Description

The StartAngle property represents the initial angle of the arc object. The EndAngle property represents the final angle of the same arc object.

See also

IPCB_Arc interface

TAngle type

StartX property

(IPCB_Arc interface)

Syntax

```
Property StartX  : TCoord Read GetState_StartX;
```

Description

The StartX property represents the starting X location of the arc object in TCoord type. The EndX property represents the final X location of the same arc object.

See also

IPCB_Arc interface

TCoord type

StartX property

(IPCB_Arc interface)

Syntax

```
Property StartY  TCoord Read GetState_StartY;
```

Description

The StartY property represents the starting Y location of the arc object in TCoord type. The EndY property represents the final Y location of the same arc object.

See also

IPCB_Arc interface

TCoord type

XCenter property

(IPCB_Arc interface)

Syntax

```
Property  XCenter : TCoord Read GetState_CenterX Write SetState_CenterX;
```

Description

The XCenter property represents the X location of the center of the arc object in TCoord type.

See also

IPCB_Arc interface

TCoord type

YCenter property

(IPCB_Arc interface)

Syntax

```
Property  YCenter : TCoord Read GetState_CenterY Write SetState_CenterY;
```

Description

The YCenter property represents the Y location of the center of the arc object in TCoord type.

See also

IPCB_Arc interface

TCoord type

IPCB_Connection interface

Overview

The **IPCB_Connection** interface represents a connection between two nodes on a PCB document. The two nodes can be on two different layers and the connection style can be a connected line or a broken specially marked connection.

The IPCB_Connection hierarchy:

- **IPCB_Primitive**
  - **IPCB_Connection**

IPCB_Connection methods	IPCB_Connection properties
`IsRedundant`	`X1`
`RotateAroundXY`	`Y1`
	`X2`
	`Y2`
	`Layer1`
	`Layer2`
	`Mode`

See also

IPCB_Primitive interface

TLayer enumerated values

TConnectionMode enumerated values

PCB Design Objects

IPCB_Embedded interface

Overview

An IPCB_Embedded interface represents an embedded object in a PCB document. An embedded object is not a visible object and cannot be manipulated by normal means in DXP. An embedded object can be used to store information which gets saved in the PCB document file when this file is saved. Each embedded object is identified by its Name property and the Description property can be used to store information.

The IPCB_Embedded hierarchy;

- **IPCB_Primitive**
  - **IPCB_Embedded**

IPCB_Embedded methods

IPCB_Embedded properties

Name

Description

Example

```
Procedure RetrieveEmbeddedObjects;
Var
    Board    : IPCB_Board;
    Iterator : IPCB_BoardIterator;
    Embd     : IPCB_Embedded;
Begin
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then
    Begin
        ShowWarning('This document is not a PCB document!');
        Exit;
    End;
    Iterator := PCBServer.GetCurrentPCBBoard.BoardIterator_Create;
    Iterator.AddFilter_ObjectSet(MkSet(eEmbeddedObject));
    Iterator.AddFilter_LayerSet (AllLayers);
    Iterator.AddFilter_Method   (eProcessAll);
    Embd   := Iterator.FirstPCBObject;
    While Embd <> Nil Do
    Begin
        ShowInfo('The Embedded object''s name field : ' + Embd.Name + #13#10
+
                 'description field : '                + Embd.Description);
        Embd := Iterator.NextPCBObject;
    End;
    PCBServer.GetCurrentPCBBoard.BoardIterator_Destroy(Iterator);
End;
```

See also

IPCB_Primitive interface

PCB Design Objects

EmbeddedObjects example in **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

IPCB_FromTo interface

Overview

The IPCB_FromTo interface represents a FromTo object on a PCB document, as a node to a node (a pin to a pin for example) and has a NetName property.

The IPCB_FromTo hierarchy:

- **IPCB_Primitive**
  - **IPCB_FromTo**

IPCB_FromTo methods	IPCB_FromTo properties
`GetNet`	`FromPad`
`GetFromPad`	`ToPad`
`GetToPad`	`NetName`
`GetState_RoutedLength`	

See also

IPCB_Primitive interface

IPCB_Pad interface

IPCB_Net interface

PCB Design Objects

IPCB_Pad interface

Overview

Pad objects are hole connectors for components and for connection to signal tracks. Pads can be either multilayered or single layered.

Pad shapes include circular, rectangular, rounded rectangular or octagonal with X, Y sizes definable from 1 to 10000mils.

Hole size can range from 0 (SMD) to 1000mils. Pads can be identified with a designator up to four characters long. On a multilayer pad, the Top layer, Mid layer and Bottom layer pad shape and size can be independently assigned to define a pad stack. Note that the surface mount components and edge connectors have single layer pads on the Top and/or Bottom layers.

Protel DXP supports a Full Stack Pad mode for ultimate control over the padstack. This allows different sizes and shapes on all signal layers. Protel DXP also supports three types of pad definitions: Simple, Top-Mid-Bottom and Full Stack.

Pads and vias can be selectively tented on the top or bottom side.

Note that, a Paste Mask expansion property for a pad object is currently relevant just for pads on top and bottom copper layers. Vias do not have a paste mask layer. Paste mask layers are used to design stencils which will selectively place solder paste on a blank PCB. Solder paste is only placed on pads where component leads are to be soldered to them. Vias normally don't have anything soldered onto them.

The IPCB_Pad hierarchy;

- **IPCB_Primitive**
  - **IPCB_Pad**

IPCB_Pad methods

```
BoundingRectangleOnLayer
RotateAroundXY
IsPadStack
IsSurfaceMount
PlaneConnectionStyleForLayer
```

IPCB_Pad properties

```
X
Y
PinDescriptor
IsConnectedToPlane
Mode
XSizeOnLayer
YSizeOnLayer
ShapeOnLayer
XStackSizeOnLayer
YStackSizeOnLayer
StackShapeonLayer
TopXSize
TopYSize
MidXSize
MidYSize
BotXSize
BotYSize
TopShape
MidShape
BotShape
HoleSize
Rotation
Name
Width
SwapID_Pad
SwapID_Gate
SwappedPadName
GateID
Cache
```

```
WidthOnLayer
```

Example

This example creates a new pad object and its associated new pad cache and places it on the current PCB document.

```
Procedure PlaceAPCBPad;
Var
    Board         : IPCB_Board;
    WorkSpace     : IWorkSpace;

    Pad           : IPCB_Pad;
    Padcache      : TPadCache;
    TopLayerWidth : TCoord;
Begin
    //Create a new PCB document
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('PCB');

    If PCBServer = Nil Then Exit;
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil then exit;

    // Create a Pad object
    Pad := PCBServer.PCBObjectFactory(ePadObject, eNoDimension,
eCreate_Default);
    Pad.SetState_XLocation  := MilsToCoord(3000);
    Pad.SetState_YLocation  := MilsToCoord(3000);

    // Setup a pad cache which has common values
    Padcache := Pad.GetState_Cache;
    Padcache.ReliefAirGap              := MilsToCoord(11);
    Padcache.PowerPlaneReliefExpansion := MilsToCoord(11);
    Padcache.PowerPlaneClearance       := MilsToCoord(11);
    Padcache.ReliefConductorWidth      := MilsToCoord(11);
    Padcache.SolderMaskExpansion       := MilsToCoord(11);
    Padcache.SolderMaskExpansionValid  := eCacheManual;
```

```
    Padcache.PasteMaskExpansion        := MilsToCoord(11);
    Padcache.PasteMaskExpansionValid   := eCacheManual;

    // Assign a new pad cache to the pad
    Pad.SetState_Cache := Padcache;
    TopLayerWidth      := Pad.GetState_WidthOnLayer(eBottomLayer);
    Board.AddPCBObject(Pad);

    // Refresh PCB document
    ResetParameters;
    AddStringParameter('Action', 'All');
    RunProcess('PCB:Zoom');
End;
```

See also

IPCB_Primitive interface

IPCB_Via interface

TPadName value

TPadCache value

TPadSwapName value

TShape enumerated values

TAngle value

PCB Design Objects

Script examples in **\Examples\Scripts\DelphiScript Scripts\PCB** folder

IPCB_Pad interface methods

BoundingRectangleOnLayer method

(IPCB_Pad interface)

Syntax

```
Function BoundingRectangleOnLayer (ALayer : TLayer) : TCoordRect;
```

Description

Example

See also

IPCB_Pad interface

RotateAroundXY method

(IPCB_Board interface)

Syntax

```
Procedure RotateAroundXY(AX, AY : TCoord;
                         Angle  : TAngle);
```

Description

This method provides the pad the ability to be rotated at a specified AX,AY values. If the AX and AY values are the same as the X,Y reference points of the pad, then the RotateAroundXY method and Rotation property is equivalent.

See also

IPCB_Pad interface

Rotation property

IsPadStack property

(IPCB_Pad interface)

Syntax

```
Function IsPadStack : Boolean;
```

Description

The IsPadStack function returns a boolean value indicating the pad is a pad stack with various shapes and sizes on different layers or just a simple pad object.

See also

IPCB_Pad interface

IsSurfaceMount method

(IPCB_Board interface)

Syntax

```
Function  IsSurfaceMount : Boolean;
```

Description

By definition a pad is a surface mount pad if the holesize of the pad is zero.

See also

IPCB_Pad interface

PlaneConnectionStyleForLayer method

(IPCB_Pad interface)

Syntax

```
Function  PlaneConnectionStyleForLayer(ALayer : TLayer) :
TPlaneConnectionStyle;
```

See also

IPCB_Pad interface

IsConnectedToPlane property.

IPCB_Pad interface Properties

X property

(IPCB_Pad interface)

Syntax

```
Property  X : TCoord Read GetState_XLocation Write SetState_XLocation;
```

Description

The X property represents the current X position of the center of the pad relative to the current origin.

The values can be in either mils or millimeters. To specify the units when typing a number, add the mil or mm suffix to the value.

The X property is implemented by its two GetState_XLocation and SetState_XLocation methods.

See also

IPCB_Pad interface

TCoord type

Y property

(IPCB_Pad interface)

Syntax

```
Property  Y : TCoord Read GetState_XLocation Write SetState_XLocation;
```

Description

The Y property represents the current Y position of the center of the pad relative to the current origin.

The values can be in either mils or millimeters. To specify the units when typing a number, add the mil or mm suffix to the value.

The Y property is implemented by its two GetState_YLocation and SetState_YLocation methods.

See also

IPCB_Pad interface

TCoord type

PinDescriptor property

(IPCB_Pad interface)

Syntax

```
Property  PinDescriptor : TPCBString   Read GetState_PinDescriptorString;
```

Description

This property represents the pin descriptor string.

See also

IPCB_Pad interface

IsConnectedToPlane property

(IPCB_Pad interface)

Syntax

```
Property  IsConnectedToPlane[L : TLayer]  : Boolean      Read
GetState_IsConnectedToPlane  Write SetState_IsConnectedToPlane;
```

Description

This property represents a pad whether it is connected to a specified layer or not by its boolean value.

See also

IPCB_Pad interface

Mode property

(IPCB_Pad interface)

Syntax

```
Property  Mode : TPadMode Read GetState_Mode Write SetState_Mode;
```

Description

The Mode property represents the type of pad stack up the pad is built as. As a simple pad, as a pad with a top-middle-bottom pad stack or as a complex stack up.

See also

IPCB_Pad interface

TPadMode type

XSizeOnLayer property

(IPCB_Pad interface)

Syntax

```
Property  XSizeOnLayer[L : TLayer]        : TCoord       Read
GetState_XSizeOnLayer;
```

Description

A read only property that returns the X Size of the pad on the specified layer.

See also

IPCB_Pad interface

TCoord type

YSizeOnLayer property

(IPCB_Pad interface)

Syntax

```
Property  YSizeOnLayer[L : TLayer]        : TCoord       Read
GetState_YSizeOnLayer;
```

Description

A read only property that returns the Y Size of the pad on the specified layer.

See also

IPCB_Pad interface

TCoord type

ShapeOnLayer property

(IPCB_Pad interface)

Syntax

```
Property  ShapeOnLayer[L : TLayer]        : TShape       Read
GetState_ShapeOnLayer;
```

Description

A read only property that returns the shape of the pad on a specified layer. Applies to local stack up pads.

Pad shapes include circular, rectangular, rounded rectangular or octagonal with X, Y sizes definable from 1 to 10000mils.

See also

IPCB_Pad interface

TShape type

XStackSizeOnLayer property

(IPCB_Pad interface)

Syntax

```
Property  XStackSizeOnLayer[L : TLayer]   : TCoord       Read
GetState_XStackSizeOnLayer   Write SetState_XStackSizeOnLayer;
```

Description

This property represents the X Size of the stack up pad on the specified layer.

See also

IPCB_Pad interface

TCoord type

YStackSizeOnLayer property

(IPCB_Pad interface)

Syntax

```
Property  YStackSizeOnLayer[L : TLayer]   : TCoord       Read
GetState_YStackSizeOnLayer   Write SetState_YStackSizeOnLayer;
```

Description

This property represents the Y Size of the stack up pad on the specified layer.

See also

IPCB_Pad interface

TCoord type

StackShapeOnLayer property

(IPCB_Pad interface)

Syntax

```
Property  StackShapeOnLayer[L : TLayer]   : TShape       Read
GetState_StackShapeOnLayer   Write SetState_StackShapeOnLayer;
```

Description

This property represents the Shape of the stack up pad on the specified layer.

See also

IPCB_Pad interface

TopXSize property

(IPCB_Pad interface)

Syntax

```
Property TopXSize : TCoord Read GetState_TopXSize Write SetState_TopXSize;
```

Description

The property represents the current size of the top layer pads when using a top-middle-bottom pad stack. The X and Y sizes set the size of the pads in the horizontal (X) direction and vertical (Y) direction respectively. Range for both values is 1mil to 10000mil. The X and Y size can be set independently to define asymmetric pad shapes.

See also

IPCB_Pad interface

TCoord type

TopYSize property

(IPCB_Pad interface)

Syntax

```
Property TopYSize : TCoord Read GetState_TopYSize Write SetState_TopYSize;
```

Description

The property represents the current size of the top layer pads when using a top-middle-bottom pad stack. The X and Y sizes set the size of the pads in the horizontal (X) direction and vertical (Y) direction respectively. Range for both values is 1mil to 10000mil. The X and Y size can be set independently to define asymmetric pad shapes.

See also

IPCB_Pad interface

TCoord type

MidXSize property

(IPCB_Pad interface)

Syntax

```
Property MidXSize : TCoord Read GetState_MidXSize Write SetState_MidXSize;
```

Description

The property represents the current size of the middle layer pads when using a top-middle-bottom pad stack. The X and Y sizes set the size of the pads in the horizontal (X) direction and vertical (Y) direction respectively. Range for both values is 1mil to 10000mil. The X and Y size can be set independently to define asymmetric pad shapes.

See also

IPCB_Pad interface

TCoord type

MidYSize property

(IPCB_Pad interface)

Syntax

```
Property  MidYSize : TCoord Read GetState_MidYSize Write SetState_MidYSize;
```

Description

The property represents the current size of the middle layer pads when using a top-middle-bottom pad stack. The X and Y sizes set the size of the pads in the horizontal (X) direction and vertical (Y) direction respectively. Range for both values is 1mil to 10000mil. The X and Y size can be set independently to define asymmetric pad shapes.

See also

IPCB_Pad interface

TCoord type

BotXSize property

(IPCB_Board interface)

Syntax

```
Property  BotXSize: TCoord Read GetState_BotXSize Write SetState_BotXSize;
```

Description

The property represents the current size of the bottom layer pads when using a top-middle-bottom pad stack. The X and Y sizes set the size of the pads in the horizontal (X) direction and vertical (Y) direction respectively. Range for both values is 1mil to 10000mil. The X and Y size can be set independently to define asymmetric pad shapes.

See also

IPCB_Pad interface

TCoord type

BotYSize property

(IPCB_Pad interface)

Syntax

```
Property BotYSize : TCoord Read GetState_BotXSize Write SetState_BotYSize;
```

Description

The property represents the current size of the bottom layer pads when using a top-middle-bottom pad stack. The X and Y sizes set the size of the pads in the horizontal (X) direction and vertical (Y) direction respectively. Range for both values is 1mil to 10000mil. The X and Y size can be set independently to define asymmetric pad shapes.

See also

IPCB_Pad interface

TCoord type

TopShape property

(IPCB_Board interface)

Syntax

```
Property  TopShape : TShape Read GetState_TopShape Write SetState_TopShape;
```

Description

The property represents the current shape of the top layer pads when using a top-middle-bottom pad stack. Basic pad shapes include Rounded, Rectangular, and Octagonal.

To change the pad shape, use one of the following shape types from the TShape type. The X and Y size can be set independently to define asymmetric pad shapes with X, Y sizes definable from 1 to 10000mils.

See also

IPCB_Pad interface

TShape type

MidShape property

(IPCB_Pad interface)

Syntax

```
Property  MidShape : TShape Read GetState_MidShape Write SetState_MidShape;
```

Description

The property represents the current shape of the middle layer pads when using a top-middle-bottom pad stack. Basic pad shapes include Rounded, Rectangular, and Octagonal.

To change the pad shape, use one of the following shape types from the TShape type. The X and Y size can be set independently to define asymmetric pad shapes with X, Y sizes definable from 1 to 10000mils.

See also

IPCB_Pad interface

TShape type

BotShape property

(IPCB_Pad interface)

Syntax

```
Property  BotShape : TShape Read GetState_BotShape Write SetState_BotShape;
```

Description

The property represents the current shape of the top layer pads when using a top-middle-bottom pad stack. Basic pad shapes include Rounded, Rectangular, and Octagonal.

To change the pad shape, use one of the following shape types from the TShape type. The X and Y size can be set independently to define asymmetric pad shapes with X, Y sizes definable from 1 to 10000mils.

See also

IPCB_Pad interface

TShape type

HoleSize property

(IPCB_Pad interface)

Syntax

```
Property  HoleSize : TCoord Read GetState_HoleSize Write SetState_HoleSize;
```

Description

The property represents the current hole size for the pad. The value specifies the diameter of the hole, in mils or mm, to be drilled in the pad during fabrication. For SMD pads or edge connectors this should be set to zero. The hole size can be set from 0 to 1000 mils, and can be set larger than the pad to define (copper free) mechanical holes.

Set the value to change the pad hole size. Values can be entered in either mils or millimeters using MilsToCoord or MMToCoord functions.

See also

IPCB_Pad interface

TCoord type

Rotation property

(IPCB_Pad interface)

Syntax

```
Property Rotation : TAngle Read GetState_Rotation Write SetState_Rotation;
```

Description

This property represents the current pad rotation in degrees. Edit the value to change the rotation of the pad. The value represents an anti-clockwise rotation in degrees. Minimum angular resolution is 0.001 degrees.

Note you can use the RotateAroundXY method instead to rotate the pad around the specified X,Y values on the PCB document rather than the pad's X and Y locations.

See also

IPCB_Pad interface

RotateAroundXY method

TAngle type

Name property

(IPCB_Pad interface)

Syntax

```
Property Name : TPCBString   Read GetState_Name Write SetState_Name;
```

See also

IPCB_Pad interface

Width property

(IPCB_Pad interface)

Syntax

```
Property Width [L : TLayer] : TCoord Read GetState_WidthOnLayer;
```

See also

IPCB_Pad interface

SwapID_Pad property

(IPCB_Pad interface)

Syntax

```
Property  SwapID_Pad : TPCBString   Read GetState_SwapID_Pad          Write
SetState_SwapID_Pad;
```

See also

IPCB_Pad interface

SwapID_Gate property

(IPCB_Pad interface)

Syntax

```
Property  SwapID_Gate : TPCBString   Read GetState_SwapID_Gate Write
SetState_SwapID_Gate;
```

See also

IPCB_Pad interface

SwappedPadName property

(IPCB_Pad interface)

Syntax

```
Property  SwappedPadName : TPCBString   Read GetState_SwappedPadName
     Write SetState_SwappedPadName;
```

See also

IPCB_Pad interface

GateId property

(IPCB_Pad interface)

Syntax

```
Property GateID : Integer Read GetState_GateId Write SetState_GateID;
```

See also

IPCB_Pad interface

Cache property

(IPCB_Pad interface)

Syntax

```
Property  Cache : TPadCache    Read GetState_Cache Write SetState_Cache;
```

Description

The cache property deals with design rules associated with pad objects such as plane relief expansion and power plane clearance values.

Example

```
Procedure PlaceAPCBPad;
Var
    Board         : IPCB_Board;
    WorkSpace     : IWorkSpace;

    Pad           : IPCB_Pad;
```

```
    Padcache      : TPadCache;
Begin
    //Create a new PCB document
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('PCB');

    If PCBServer = Nil Then Exit;
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil then exit;

    // Create a Pad object
    Pad := PCBServer.PCBObjectFactory(ePadObject, eNoDimension,
eCreate_Default);
    Pad.X  := MilsToCoord(3000);
    Pad.Y  := MilsToCoord(3000);

    // Setup a pad cache which has common values
    Padcache := Pad.GetState_Cache;
    Padcache.ReliefAirGap                 := MilsToCoord(11);
    Padcache.PowerPlaneReliefExpansion := MilsToCoord(11);
    Padcache.PowerPlaneClearance        := MilsToCoord(11);
    Padcache.ReliefConductorWidth       := MilsToCoord(11);
    Padcache.SolderMaskExpansion        := MilsToCoord(11);
    Padcache.SolderMaskExpansionValid  := eCacheManual;
    Padcache.PasteMaskExpansion         := MilsToCoord(11);
    Padcache.PasteMaskExpansionValid   := eCacheManual;

    // Assign a new pad cache to the pad
    Pad.SetState_Cache := Padcache;
    Board.AddPCBObject(Pad);

    // Refresh PCB document
    ResetParameters;
    AddStringParameter('Action', 'All');
    RunProcess('PCB:Zoom');
```

```
End;
```

See also

IPCB_Pad interface

TPadCache record

WidthOnLayer property

(IPCB_Pad interface)

Syntax

```
Property  WidthOnLayer[L : TLayer]        : TCoord       Read
GetState_WidthOnLayer;
```

Description

This WidthOnLayer property returns the larger value of the X and Y values associated with the pad object. The larger of the two values is referred to as the width on the specified layer.

See also

IPCB_Pad interface

IPCB_ObjectClass interface

Overview

A class is defined as a group or set of objects, identified by its unique class name. The PCB editor in the Design Explorer supports Net Classes, Component Classes and From-To Classes. An object can belong to more than one class. You can create classes (or groups) of objects. Classes of Components, Nets and From-Tos can be created, and multiple membership is permitted. Classes are used to quickly identify a group of objects. For example, you could create a class of components called Surface Mount.

When you set up a paste mask expansion rule for the surface mount components, you simply set the rule scope to Component Class and select the Surface Mount class. Or you may have a set of nets, such as the power nets, which have different clearance requirements from the signal nets. You can create a Net Class which includes all these nets, and then use the Net Class scope when you define the clearance design rule for these nets.

Notes

- The SuperClass property denotes whether or not the interface contains all members of a particular kind. If this field is set to true, the members of the TPCBObjectClass object cannot be edited.
- An ObjectClass object can be created from the PCBObjectFactory method from the IPCB_ServerInterface interface.

The IPCB_ObjectClass hierarchy;

- **IPCB_Primitive**
  - **IPCB_ObjectClass**

IPCB_ObjectClass methods	IPCB_ObjectClass properties
`AddMemberByName`	`MemberKind`
`AddMember`	`Name`
`RemoveMember`	`SuperClass`
`RemoveAllMembers`	`MemberName[I : Integer]`
`IsMember`	
`IsLayerMember`	
`AddLayerMember`	
`RemoveLayerMember`	
`IsValidObjectKind`	

See also

IPCB_Primitive interface

IPCB_ServerInterface interface

TClassMemberKind enumerated values

IPCB_Track interface

Overview

Tracks can be placed on any layer and their widths can range from 0.001 to 10000 mils wide. Tracks are used to create polygon planes and are also used in coordinates, dimensions and components.

Tracks that carry either signals or power supply can be placed on:

• Top (component side) signal layer.

• Any of the thirty mid signal layers.

• Bottom (solder side) signal layer.

Non-electrical tracks can also be placed on:

• Any of the silk screen overlays (normally used for component package outlines).

• Any of the sixteen internal plane layers (used as voids in these solid copper planes).

• The keep out layer to define the board perimeter for autorouting and auto component placement

• Any of the sixteen mechanical layers for mechanical details.

• Solder or paste mask layers for any special openings required in the masks

The IPCB_Track hierarchy;

- **IPCB_Primitive**
  - **IPCB_Track**

IPCB_Track methods	IPCB_Track properties
	`X1`
	`Y1`
	`X2`
	`Y2`
	`Width`

Example

```
Var
    Board     : IPCB_Board;
    WorkSpace : IWorkSpace;
    Track     : IPCB_Track;
Begin
    //Create a new PCB document
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
```

```
    Workspace.DM_CreateNewDocument('PCB');

    // Check if the new PCB document exists.
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil then exit;

    // Create a Track object
    Track             := PCBServer.PCBObjectFactory(eTrackObject,
eNoDimension, eCreate_Default);
    Track.X1          := MilsToCoord(X1);
    Track.Y1          := MilsToCoord(Y1);
    Track.X2          := MilsToCoord(X2);
    Track.Y2          := MilsToCoord(Y2);
    Track.Layer       := Layer;
    Track.Width       := MilsToCoord(Width);

    // Add the new track into the PCB document
    Board.AddPCBObject(Track);

    // Refresh the PCB document.
    ResetParameters;
    AddStringParameter('Action', 'All');
    RunProcess('PCB:Zoom');
End;
```

See also

IPCB_Primitive interface

PCB Design Objects

TCoord type

CountTracksInComponent example in **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

IPCB_Track interface Properties

X1 property

(IPCB_Track interface)

Syntax

```
Property X1 : TCoord Read GetState_X1 Write SetState_X1;
```

Description

This property represents the X1 or the initial X value of the track in TCoord type.

See also

IPCB_Track interface

TCoord type

Y1 property

(IPCB_Track interface)

Syntax

```
Property Y1 : TCoord Read GetState_Y1 Write SetState_Y1;
```

Description

This property represents the Y1 or the initial Y value of the track in TCoord type.

See also

IPCB_Track interface

TCoord type

X2 property

(IPCB_Track interface)

Syntax

```
Property X2 : TCoord Read GetState_X2 Write SetState_X2;
```

Description

This property represents the X2 or the final X value of the track in TCoord type.

See also

IPCB_Track interface

TCoord type

Y2 property

(IPCB_Track interface)

Syntax

```
Property Y2 : TCoord Read GetState_Y2 Write SetState_Y2;
```

Description

This property represents the Y2 or the final Y value of the track in TCoord type.

See also

IPCB_Track interface

TCoord type

Width property

(IPCB_Track interface)

Syntax

```
Property Width : TCoord Read GetState_Width Write SetState_Width;
```

Description

This property represents the track width in TCoord type.

See also

IPCB_Track interface

TCoord type

IPCB_Via interface

Overview

When tracks from two layers need to be connected, vias are placed to carry a signal from one layer to the other. Vias are like round pads, which are drilled and usually through-plated when the board is fabricated. Vias can be multi-layered, blind or buried.

A multi-layer via passes through the board from the Top layer to the Bottom layer and allows connections to all other signal layers.

A blind via connects from the surface of the board to an internal layer, a buried via connects from one internal layer to another internal layer. In DXP, Vias, including blind and buried, can connect to internal planes.

Vias do not have a paste mask layer.

The IPCB_Via hierarchy;

- **IPCB_Primitive**
  - **IPCB_Via**

IPCB_Via methods

PlaneConnectionStyleForLayer

RotateAroundXY

IntersectLayer

IPCB_Via properties

X

Y

IsConnectedToPlane

LowLayer

HighLayer

StartLayer

StopLayer

HoleSize

Size

SizeOnLayer

ShapeOnLayer

Cache

Example

```
Var
    Board     : IPCB_Board;
    WorkSpace : IWorkSpace;
    Via       : IPCB_Via;
    ViaCache  : TPadCache;
Begin
    // Create a new PCB document
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('PCB');

    // Check if the new PCB document exists or not.
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil then exit;

    // Create a Via object
    Via          := PCBServer.PCBObjectFactory(eViaObject, eNoDimension,
eCreate_Default);
    Via.X        := MilsToCoord(2000);
    Via.Y        := MilsToCoord(2000);
    Via.Size     := MilsToCoord(50);
```

```
    Via.HoleSize  := MilsToCoord(20);
    Via.LowLayer  := eTopLayer;
    Via.HighLayer := eBottomLayer;

    // Setup a pad cache
    Viacache := Via.GetState_Cache;
    Viacache.ReliefAirGap := MilsToCoord(11);
    Viacache.PowerPlaneReliefExpansion := MilsToCoord(11);
    Viacache.PowerPlaneClearance       := MilsToCoord(11);
    Viacache.ReliefConductorWidth      := MilsToCoord(11);
    Viacache.SolderMaskExpansion       := MilsToCoord(11);
    Viacache.SolderMaskExpansionValid  := eCacheManual;
    Viacache.PasteMaskExpansion        := MilsToCoord(11);
    Viacache.PasteMaskExpansionValid   := eCacheManual;

    // Assign the new Via cache to the via
    Via.SetState_Cache := Viacache;
    Board.AddPCBObject(Via);

    // Refresh PCB document.
    ResetParameters;
    AddStringParameter('Action', 'All');
    RunProcess('PCB:Zoom');
End;
```

See also

IPCB_Primitive interface

IPCB_Pad interface

TLayer enumerated values

TPlaneConnectionStyle enumerated values

TCoord value

TAngle value

TPadCache values

IPCB_LayerObject interface

PCB Design Objects

CreateAVia example in **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

IPCB_Violation interface

Overview

A Violation object captures the rule that has been violated between two PCB objects that are affected by a binary design rule or a PCB object affected by a unary design rule detected in the PCB editor, with the description of the violation and the type of rule used.

The IPCB_Violation hierarchy;

- **IPCB_Primitive**
  - **IPCB_Violation**

IPCB_Violation methods

`IsRedundant`

`GetState_ShortDescriptorString`

IPCB_Violation properties

`Name`

`Rule`

`Primitive1`

`Primitive2`

`Description`

See also

IPCB_Primitive interface

PCB Design Objects

Violations script in **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

PCB Group Objects

A PCB design object on a PCB document is represented by its interface. An interface represents an existing object in memory and its properties and methods can be invoked. A PCB design object is basically a primitive or a group object. A primitive can be a track or an arc object. A group object is an object that is composed of child objects. For example a board outline or a component is a group object.

The **IPCB_BoardOutline, IPCB_Coordinate, IPCB_Dimension, IPCB_Polygon, IPCB_Net and IPCB_LibComponent** are group object interfaces.

Since many design objects are descended from ancestor interfaces and thus the ancestor methods and properties are also available to use.

For example the **IPCB_BoardOutline** interface is inherited from an immediate **IPCB_Group** interface and in turn inherited from the **IPCB_Primitive** interface. If you check the **IPCB_BoardOutline** entry in this online help you will see the following information;

The **IPCB_BoardOutline** Interface hierarchy is as follows:

- **IPCB_Primitive**
  - **IPCB_Group**
    - **IPCB_BoardOutline**

See also

PCB Documents

PCB Objects

IPCB_Group

IPCB_BoardOutline

IPCB_Coordinate

IPCB_Component

IPCB_Dimension

IPCB_LibComponent

IPCB_Polygon

IPCB_Net

IPCB_Group interface

Overview

The **IPCB_Group** interface is an immediate ancestor for **IPCB_Net**, **IPCB_LibComponent**, **IPCB_Polygon**, **IPCB_Coordinate**, **IPCB_Dimension** and its descendant interfaces.

The **IPCB_Group** interface is a composite object interface which means it can store objects. Thus a group object is an object composed of primitives such as arcs, tracks and fills. For example a polygon consists of child tracks and arcs. A footprint in a PCB library consists of child objects such as arcs, pads and tracks.

The **IPCB_Group** interface hierarchy is as follows:

- **IPCB_Primitive**
  - **IPCB_Group**

Notes

- To fetch objects of a group object, you employ the Group Iterator with the GroupIterator_Create and GroupIterator_Destory methods.
- To add or remove child objects from a group object, you employ the AddPCBObject or the RemovePCBObject methods.
- To fetch the reference coordinates of a group object, the X,Y properties define the reference point.

IPCB_Group methods	IPCB_Group properties
`FreePrimitives`	`X`
`GetPrimitiveAt`	`Y`
`GetPrimitiveCount`	`PrimitiveLock`
`SetState_XSizeYSize`	`LayerUsed`
`FastSetState_XSizeYSize`	
`SetState_LayersUsedArray`	
`GroupIterator_Create`	
`GroupIterator_Destroy`	
`AddPCBObject`	
`RemovePCBObject`	

See also

IPCB_Primitive interface

IPCB_Net interface

IPCB_LibComponent interface

IPCB_Polygon interface

IPCB_Coordinate interface

IPCB_Dimension interface

IPCB_GroupIterator interface

PCB Design Objects

IPCB_Group interface methods

AddPCBObject method

(IPCB_Group interface)

Syntax

```
Procedure AddPCBObject(PCBObject : IPCB_Primitive);
```

Description

The AddPCBObject method adds a new child object into the group object such as a board outline object or a component object. A group object is made up of child objects called primitives.

See also

IPCB_Group interface

FastSetState_XSizeYSize method

(IPCB_Group interface)

Syntax

```
Function FastSetState_XSizeYSize : Boolean;
```

See also

IPCB_Group interface

FreePrimitives method

(IPCB_Group interface)

Syntax

```
Procedure FreePrimitives;
```

See also

IPCB_Group

GetPrimitiveAt method

(IPCB_Group interface)

Syntax

```
Function  GetPrimitiveAt(I        : Integer;
                         ObjectId : TObjectId) : IPCB_Primitive;
```

See also

IPCB_Group interface

GetPrimitiveCount method

(IPCB_Group interface)

Syntax

```
Function  GetPrimitiveCount (ObjectSet  : TObjectSet) : Integer;
```

Description

The function returns the number of child objects (such as arcs and tracks) associated with the group object such as the component.

See also

IPCB_Group

GroupIterator_Create

(IPCB_Group interface)

Syntax

```
Function GroupIterator_Create : IPCB_GroupIterator;
```

Description

The GroupIterator_Create method creates a group iterator for the group object, so that the child objects can be searched from within the group object. This group iterator searches for child objects of a group object, such as a component, footprint, polygon, dimension, board layout and so on.

Example

```
Var
    Track                   : IPCB_Primitive;
    TrackIteratorHandle     : IPCB_GroupIterator;
    Component               : IPCB_Component;
    ComponentIteratorHandle : IPCB_BoardIterator;
    TrackCount              : Integer;
    ComponentCount          : Integer;
Begin
    TrackCount     := 0;
    ComponentCount := 0;
    If PCBServer.GetCurrentPCBBoard = Nil Then Exit;

    ComponentIteratorHandle :=
PCBServer.GetCurrentPCBBoard.BoardIterator_Create;
    ComponentIteratorHandle.AddFilter_ObjectSet(MkSet(eComponentObject));
    ComponentIteratorHandle.AddFilter_LayerSet(AllLayers);
    ComponentIteratorHandle.AddFilter_Method(eProcessAll);
    Component := ComponentIteratorHandle.FirstPCBObject;

    While (Component <> Nil) Do
    Begin
        TrackIteratorHandle := Component.GroupIterator_Create;
        TrackIteratorHandle.AddFilter_ObjectSet(MkSet(eTrackObject));
        TrackIteratorHandle.AddFilter_LayerSet(MkSet(eTopOverlay));
        Track := TrackIteratorHandle.FirstPCBObject;
        While (Track <> Nil) Do
        Begin
            Inc(TrackCount);
            Track := TrackIteratorHandle.NextPCBObject;
        End;
        ShowInfo('This component ' + Component.SourceDesignator  + ' has ' +
 IntToStr(TrackCount)  + ' tracks.');
```

```
        TrackCount := 0;
        Component.GroupIterator_Destroy(TrackIteratorHandle);
        Component := ComponentIteratorHandle.NextPCBObject;
        Inc(ComponentCount);
        If (ComponentCount > 5) Then Break;
    End;
    PCBServer.GetCurrentPCBBoard.BoardIterator_Destroy(ComponentIteratorHand
le);
End;
```

See also

IPCB_Group interface

IPCB_GroupIterator interface

GroupIterator_Destroy method

GroupIterator_Destroy

(IPCB_Group interface)

Syntax

```
Procedure GroupIterator_Destroy(Var AIterator : IPCB_GroupIterator);
```

Description

This function destroys the group iterator object after it has been used in a search for group objects.

Example

```
While (Component <> Nil) Do
Begin
    TrackIteratorHandle := Component.GroupIterator_Create;
    TrackIteratorHandle.AddFilter_ObjectSet(MkSet(eTrackObject));
    TrackIteratorHandle.AddFilter_LayerSet(MkSet(eTopOverlay));
    Track := TrackIteratorHandle.FirstPCBObject;
    While (Track <> Nil) Do
    Begin
        Track := TrackIteratorHandle.NextPCBObject;
    End;
End;
Component.GroupIterator_Destroy(TrackIteratorHandle);
```

See also

IPCB_Group interface

IPCB_GroupIterator interface

GroupIterator_Create method

RemovePCBObject method

(IPCB_Group interface)

Syntax

```
Procedure RemovePCBObject(PCBObject : IPCB_Primitive);
```

Description

The procedure removes a child object thats associated with the group object, for example removing an arc object from the polygon object.

See also

IPCB_Group interface

SetState_LayersUsedArray method

(IPCB_Group interface)

Syntax

```
Procedure SetState_LayersUsedArray;
```

See also

IPCB_Group interface

SetState_XSizeYSize method

(IPCB_Group interface)

Syntax

```
Function  SetState_XSizeYSize : Boolean;
```

See also

IPCB_Group interface

Properties

LayerIsUsed property

(IPCB_Group interface)

Syntax

```
Property  LayerUsed [L : TLayer] : Boolean Read GetState_LayerUsed Write
SetState_LayerUsed;
```

Description

The property represents the layer that is being used by the group object.

See also

IPCB_Group

PrimitiveLock property

(IPCB_Group interface)

Syntax

```
Property PrimitiveLock : Boolean Read GetState_PrimitiveLock Write
SetState_PrimitiveLock;
```

Description

The PrimitiveLock property denotes whether the primitives of the group object can be edited individually or not. Normally all the child objects or primitives of a group can only be accessed as a group object.

See also

IPCB_Group

X property

(IPCB_Group interface)

Syntax

```
Property X : TCoord Read GetState_XLocation Write SetState_XLocation;
```

Description

The X property defines the reference point of the group object.

See also

IPCB_Group interface

Y property

(IPCB_Group interface)

Syntax

```
Property Y : TCoord Read GetState_YLocation Write SetState_YLocation;
```

Description

The Y property defines the reference point of the group object.

See also

IPCB_Group interface

IPCB_BoardOutline

Overview

The board outline object represents the board shape which defines the extents or boundary of the board in the PCB Editor. A board outline object is essentially a closed polygon and is inherited from the **IPCB_Polygon** interface.

The PCB Editor uses the board outline shape to determine the extents of the power planes for plane edge pull back, used when splitting power planes and for calculating the board edge when design data is exported to other tools such as the 3D viewer tool.

A board outline is a group object therefore it is composed of pull back primitives namely tracks and arcs as the vertices for the closed polygon of the board outline. Although the board outline object interface is inherited from the **IPCB_Polygon** interface, you cannot use layer, net assignment and repour polygon behaviours for a board outline).

The **IPCB_BoardOutline** interface hierarchy is as follows;

- **IPCB_Primitive**
  - **IPCB_Group**
    - **IPCB_BoardOutline**

Notes

- The IPCB_BoardOutline interface is inherited from IPCB_Polygon interface and in turn from IPCB_Group interface.
- To iterate the board outline for the pullback primitives, you create and use a group iterator because the borad outline is a group object which in turn is composed of child objects.
- The IPCB_BoardOutline interface is a BoardOutline property from the IPCB_Board interface.

IPCB_Group methods	IPCB_Group properties
`FreePrimitives`	`X`
`GetPrimitiveAt`	`Y`
`GetPrimitiveCount`	`PrimitiveLock`
`SetState_XSizeYSize`	`LayerUsed`
`FastSetState_XSizeYSize`	
`SetState_LayersUsedArray`	
`GroupIterator_Create`	
`GroupIterator_Destroy`	
`AddPCBObject`	
`RemovePCBObject`	

IPCB_BoardOutline methods

GetState_HitPrimitive

Rebuild

Validate

Invalidate

InvalidatePlane

IPCB_BoardOutline properties

Example

```
Procedure Query_Board_Outline;
Var
    PCB_Board : IPCB_Board;
    BR        : TCoordRect;
    NewUnit   : TUnit;
Begin

    PCB_Board := PCBServer.GetCurrentPCBBoard;
    If PCB_Board = Nil Then Exit;
    If PCB_Board.IsLibrary Then Exit;

    PCB_Board.BoardOutline.Invalidate;
    PCB_Board.BoardOutline.Rebuild;
    PCB_Board.BoardOutline.Validate;

    // The BoundingRectangle method is defined in IPCB_Primitive interface
    BR := PCB_Board.BoardOutline.BoundingRectangle;
    If PCB_Board.DisplayUnit = eImperial Then NewUnit := eMetric
                                         Else NewUnit := eImperial;
    ShowMessage(
        'Board Outline Width  : ' +
        CoordUnitToString(BR.right - BR.left,
                          PCB_Board.DisplayUnit) + #13 +
        'Board Outline Height : ' +
        CoordUnitToString(BR.top - BR.bottom,
                          PCB_Board.DisplayUnit));
End;
```

See also

PCB Design Objects

PCB_Primitive interface

IPCB_Group interface

IPCB_Polygon interface

IPCB_GroupIterator interface

PCB_Outline script in \Examples\Scripts\Delphiscript Scripts\PCB folder.

BoardOutlineDetails script in \Examples\Scripts\Delphiscript Scripts\PCB folder.

IPCB_Coordinate

Overview

Coordinate markers are used to indicate the coordinates of specific points in a PCB workspace. A coordinate marker consists of a point marker and the X and Y coordinates of the position

The IPCB_Coordinate interface hierarchy is as follows;

- **IPCB_Primitive**
  - **IPCB_Group**
    - **IPCB_Coordinate**

IPCB_Group methods	IPCB_Group properties
`FreePrimitives`	`X`
`GetPrimitiveAt`	`Y`
`GetPrimitiveCount`	`PrimitiveLock`
`SetState_XSizeYSize`	`LayerUsed`
`FastSetState_XSizeYSize`	
`SetState_LayersUsedArray`	
`GroupIterator_Create`	
`GroupIterator_Destroy`	
`AddPCBObject`	
`RemovePCBObject`	

IPCB_Coordinate methods

`SetState_xSizeySize`

`RotateAroundXY`

`Text`

`Track1`

`Track2`

`GetState_StrictHitTest`

IPCB_Coordinate properties

`Size`

`LineWidth`

`TextHeight`

`TextWidth`

`TextFont`

`Style`

`Rotation`

See also

PCB Design Objects

IPCB_Primitive interface

IPCB_Group interface

IPCB_GroupIterator interface

IPCB_Component

Overview

Components are defined by footprints, which are stored in a PCB library (or part of an integrated library). Note, a footprint can be linked to a schematic component.

When a footprint is placed in the workspace, it is assigned a designator (and optional comment). It is then referred to as a component. A component is composed of primitives (normally tracks, arcs, and pads).

Components are defined by footprints, which are stored in a PCB library. When a footprint is placed in the workspace, it is assigned a designator (and optional comment). It is then referred to as a component with the defined reference. The origin in the library editor defines the reference point of a footprint.

The **IPCB_Component** interface hierarchy is as follows:

- **IPCB_Primitive**
  - **IPCB_Group**
    - **IPCB_Component**

Notes

- The reference point of a component is set by the X,Y fields inherited from **IPCB_Group** interface. You can obtain the bounding rectangle of the component and calculate the mid point X and Y values to enable rotation about the center of the component if desired.
- The rotation property of a component is set according to the reference point of a component, therefore the **Rotation** property and the **RotateAroundXY** method are equivalent only if you use the X,Y parameters for the **RotateAroundXY** method that are the same as the reference point of the component.
- A component is a group object and therefore is composed of child objects such as arcs and tracks. You use a group iterator to fetch the child objects for that component.
- A component type is determined by its TComponentKind type defined in WorkSpace Manager API.

IPCB_Group methods	IPCB_Group properties
`FreePrimitives`	`X`
`GetPrimitiveAt`	`Y`
`GetPrimitiveCount`	`PrimitiveLock`
`SetState_XSizeYSize`	`LayerUsed`
`FastSetState_XSizeYSize`	
`SetState_LayersUsedArray`	
`GroupIterator_Create`	
`GroupIterator_Destroy`	
`AddPCBObject`	

RemovePCBObject

IPCB_Component methods

ChangeNameAutoposition
ChangeCommentAutoposition
SetState_xSizeySize
RotateAroundXY
FlipComponent
Rebuild
Getstate_PadByName
LoadCompFromLibrary
AutoPosition_NameComment

IPCB_Component properties

ChannelOffset
ComponentKind
Name
Comment
Pattern
NameOn
CommentOn
GroupNum
UnionIndex
Rotation
Height
NameAutoPosition
CommentAutoPosition
SourceDesignator
SourceUniqueId
SourceHierarchicalPath
SourceFootprintLibrary
SourceComponentLibrary
SourceLibReference
SourceDescription
FootprintDescription
DefaultPCB3DModel

See also

PCB Design Objects
IPCB_Primitive interface
IPCB_Group interface
IPCB_GroupIterator interface
IPCB_Text interface
TComponentKind enumerated values
TTextAutoposition enumerated values

IPCB_Component methods

ChangeNameAutoposition method

(IPCB_Component interface)

Syntax

```
Function ChangeNameAutoposition(Value : TTextAutoposition) : Boolean;
```

Description

Text can be positioned automatically by selecting a TTextAutoposition type. It will maintain this position as the component is moved/rotated. When Manual is selected the text maintains its position relative to the component, regardless of the rotation. When the text is moved with the cursor the Autoposition option is switched to Manual.

Invoke the **Autoposition_NameComment** method after changing the autoposition.

See also

IPCB_Component interface

TTextAutoposition type

AutoPosition_NameComment method

ChangeCommentAutoposition method

(IPCB_Component interface)

Syntax

```
Function  ChangeCommentAutoposition(Value : TTextAutoposition) : Boolean;
```

Description

Text can be positioned automatically by selecting a TTextAutoposition type. It will maintain this position as the component is moved/rotated. When Manual is selected the text maintains its position relative to the component, regardless of the rotation. When the text is moved with the cursor the Autoposition option is switched to Manual.

Invoke the **Autoposition_NameComment** method after changing the autoposition.

See also

IPCB_Component interface

TTextAutoposition type

AutoPosition_NameComment method

GetState_PadByName method

(IPCB_Component interface)

Syntax

```
Function Getstate_PadByName(S : TPCBString) : IPCB_Primitive;
```

Description

The function retrieves the pad object associated with the component if the name matches the pad's designator.

See also

IPCB_Component interface

SetState_xSizeySize

(IPCB_Component interface)

Syntax

```
Function SetState_xSizeySize : Boolean;
```

See also

IPCB_Component interface

RotateAroundXY method

(IPCB_Component interface)

Syntax

```
Procedure RotateAroundXY(AX,
                         AY    : TCoord;
                         Angle : TAngle);
```

Description

This method provides the component the ability to be rotated at a specified AX,AY values. If the AX and AY values are the same as the X,Y reference points of the pad, then the **RotateAroundXY** method and **Rotation** property is equivalent.

See also

IPCB_Component interface

FlipComponent method

(IPCB_Component interface)

Syntax

```
Procedure FlipComponent;
```

Description

The procedure flips the component from one layer to other, ie from top layer to bottom layer and all its associated text objects such as the designator are flipped as well (including the reversal of the text themselves).

See also

IPCB_Component interface

Rebuild method

(IPCB_Component interface)

Syntax

```
Procedure Rebuild;
```

Description

The Rebuild procedure rebuilds the component and refreshes the graphical display of this component as well. This procedure can be invoked after the component has been adjusted in terms of its rotation, removal or addition of new primitives for example.

See also

IPCB_Component interface

LoadCompFromLibrary method

(IPCB_Board interface)

Syntax

```
Function  LoadCompFromLibrary : Boolean;
```

Description

Invoke this function to check if the component has been loaded from a library.

See also

IPCB_Component interface

AutoPosition_NameComment method

(IPCB_Component interface)

Syntax

```
Procedure AutoPosition_NameComment;
```

Description

Invoke this position after the Name or/and Comment autoposition types have been changed.

See also

IPCB_Component interface

ChangeNameAutoposition method

ChangeCommentAutoposition method

IPCB_Component Properties

ChannelOffset property

(IPCB_Component interface)

Syntax

```
Property  ChannelOffset : TChannelOffset Read GetState_ChannelOffset Write
SetState_ChannelOffset;
```

Description

When a design is first transferred from schematic to PCB, each component on a schematic sheet is given a unique channel offset. The offset is used in a multi-channel design when the Copy Room Format command is executed, identifying the instantiation of the component in each room. If a component is not part of a channel (room) it is given a value of -1.

See also

IPCB_Component interface

ComponentKind property

(IPCB_Component interface)

Syntax

```
Property ComponentKind : TComponentKind Read GetState_ComponentKind Write
SetState_ComponentKind;
```

Description

The property represents the component kind which is a **TComponentKind** type from the WorkSpace Manager API.

eComponentKind_Standard

These components possess standard electrical properties, are always synchronized and are the type most commonly used on a board.

eComponentKind_Standard_NoBOM

These components possess standard electrical properties, and are synchronized BUT are not included in any BOM file produced from the file.

eComponentKind_Mechanical

These components do not have electrical properties and will appear in the BOM. They are synchronized if the same components exist on both the Schematic and PCB documents. An example is a heatsink.

eComponentKind_Graphical

These components are not used during synchronization or checked for electrical errors. These components are used, for example, when adding company logos to documents.

eComponentKind_TieNet_BOM

These components short two or more different nets for routing and these components will appear in the BOM and are maintained during synchronization.

eComponentKind_TieNet_NoBOM

These components short two or more different nets for routing and these components will NOT appear in the BOM and are maintained during synchronization. Note: Enable the Verify Shorting Copper option in the Design Rule Check dialog to verify that there is no unconnected copper in the component (Tools » Design Rule Check).

See also

IPCB_Component interface

TComponentKind type

Name property

(IPCB_Component interface)

Syntax

```
Property  Name : IPCB_Text Read GetState_Name;
```

Description

This property represents the name or the designator of the component and it represents the name object associated with the component. This name object is encapsulated as a IPCB_Text object. You need to check if this name object exists before extracting or setting new data to the name object.

See also

IPCB_Component interface

IPCB_Text interface

Comment property

(IPCB_Component interface)

Syntax

```
Property Comment : IPCB_Text Read GetState_Comment;
```

Description

The Comment property represents the comment object associated with the component. This comment object is encapsulated as a IPCB_Text object. You need to check if this comment object exists before extracting or setting new data to the comment object.

See also

IPCB_Component interface

IPCB_Text interface

Pattern property

(IPCB_Component interface)

Syntax

```
Property  Pattern : TPCBString Read GetState_Pattern Write SetState_Pattern;
```

Description

This property represents the name of the footprint. The footprint is the graphical representation of a PCB component and is used to display it on the PCB, and usually contains component outline and connection pads along with an unique designator. Footprints are stored in PCB library files or Integrated libraries, which can be edited using the PCB Library Editor to create new footprints or edit existing ones.

See also

IPCB_Component interface

NameOn property

(IPCB_Component interface)

Syntax

```
Property  NameOn                                     : Boolean              Read
GetState_NameOn                 Write SetState_NameOn;
```

Description

This property denotes whether the name object is visible or not.

See also

IPCB_Component interface

CommentOn property

(IPCB_Component interface)

Syntax

```
Property  CommentOn                                  : Boolean              Read
GetState_CommentOn              Write SetState_CommentOn;
```

Description

This property denotes if the comment object is visible or not.

See also

IPCB_Component interface

GroupNum property

(IPCB_Component interface)

Syntax

```
Property  GroupNum                                    : Integer              Read
GetState_GroupNum                Write SetState_GroupNum;
```

See also

IPCB_Component interface

UnionIndex property

(IPCB_Component interface)

Syntax

```
Property UnionIndex :Integer Read GetState_UnionIndex Write
SetState_UnionIndex;
```

See also

IPCB_Component interface

Rotation property

(IPCB_Component interface)

Syntax

```
Property  Rotation : TAngle Read GetState_Rotation Write SetState_Rotation;
```

See also

IPCB_Component interface

Height property

(IPCB_Component interface)

Syntax

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

Description

This property represents the height value of the component object.

See also

IPCB_Component interface

PCB_Component_Heights script from the \Examples\Scripts\DelphiScript Scripts\PCB folder.

NameAutoPosition property

(IPCB_Component interface)

Syntax

```
Property NameAutoPosition : TTextAutoposition   Read GetState_NameAutoPos
Write SetState_NameAutoPos;
```

See also

IPCB_Component interface

CommentAutoPosition property

(IPCB_Component interface)

Syntax

```
Property  CommentAutoPosition               : TTextAutoposition   Read
GetState_CommentAutoPos         Write SetState_CommentAutoPos;
```

See also

IPCB_Component interface

SourceDesignator property

(IPCB_Component interface)

Syntax

```
Property  SourceDesignator                  : TPCBString          Read
GetState_SourceDesignator       Write SetState_SourceDesignator;
```

Description

This property represents the current designator of the schematic component that the PCB footprint is linked to. This property will be blank if the PCB component is not linked to a Schematic component.

See also

IPCB_Component interface

SourceUniqueID

(IPCB_Component interface)

Syntax

```
Property  SourceUniqueId                    : TPCBString          Read
GetState_SourceUniqueId         Write SetState_SourceUniqueId;
```

Description

Unique Identifiers (UIDs) are used to match each schematic component to the corresponding PCB component. When a schematic design is transferred to a PCB, the UID of the schematic component is appended to the UIDs of each of the sheet symbols in the project hierarchy above the component. Note you will need to run the Component Links dialog to update the link if the corresponding source component UID has changed.

Note, you can use the WorkSpace Manager interface's DM_GenerateUniqueID method to obtain an UniqueID value and assign it to the SourceUniqueId property.

See also

IPCB_Component interface

DM_GenerateUniqueID method

SourceHierarchicalPath property

(IPCB_Component interface)

Syntax

```
Property  SourceHierarchicalPath           : TPCBString         Read
GetState_SourceHierarchicalPath Write SetState_SourceHierarchicalPath;
```

Description

This field uniquely identifies the source reference path to the PCB component. The path can be multi-level depending on whether it is a multi channel (sheet symbols) or a normal design (schematic sheets including a project/master sheet).

Note: When a schematic design is transferred to a PCB document, each PCB footprint is populated with schematic reference information and a schematic component can have PCB models, Signal integrity models etc linked into it.

See also

IPCB_Component interface

SourceFootprintLibrary

(IPCB_Component interface)

Syntax

```
Property  SourceFootprintLibrary           : TPCBString         Read
GetState_SourceFootprintLibrary Write SetState_SourceFootprintLibrary;
```

Description

This source library field denotes the integrated library where the schematic component is from. Note: When a schematic design is transferred to a PCB document, each PCB footprint is populated with schematic reference information and a schematic component can have PCB models, Signal integrity models etc linked into it.

See also

IPCB_Component interface

SourceComponentLibrary

(IPCB_Component interface)

Syntax

```
Property  SourceComponentLibrary            : TPCBString          Read
GetState_SourceComponentLibrary Write SetState_SourceComponentLibrary;
```

Description

This source library field denotes the name of the Schematic library or integrated library where the PCB footprint is linked to this schematic component. Note a PCB component is a placed PCB footprint with a defined designator value.

See also

IPCB_Component interface

SourceLibReference

(IPCB_Component interface)

Syntax

```
Property  SourceLibReference                : TPCBString          Read
GetState_SourceLibReference     Write SetState_SourceLibReference;
```

Description

The library reference field is the name of the schematic component from the schematic library / integrated library that the PCB footprint is linked to. A schematic component can have different implementation models linked to it such as a signal integrity, simulation and PCB models.

See also

IPCB_Component interface

SourceDescription

(IPCB_Component interface)

Syntax

```
Property  SourceDescription                 : TPCBString          Read
GetState_SourceDescription      Write SetState_SourceDescription;
```

Description

This property represents the description of the reference link of a schematic component from the PCB component.

See also

IPCB_Component interface

Footprint Property

(IPCB_Component interface)

Syntax

```
Property  FootprintDescription              : TPCBString        Read
GetState_FootprintDescription   Write SetState_FootprintDescription;
```

Description

This property represents the footprint description.

See also

IPCB_Component interface

DefaultPCB3DModel

(IPCB_Component interface)

Syntax

```
Property  DefaultPCB3DModel                 : TPCBString        Read
GetState_DefaultPCB3DModel      Write SetState_DefaultPCB3DModel;
```

Description

This property represents the default PCB 3D model string/name.

See also

IPCB_Component interface

IPCB_Polygon interface

Overview

Polygons are similar to area fills, except that they can fill irregular shaped areas of a board and can connect to a specified net as they are poured. By adjusting the grid and track size, a polygon plane can be either solid (copper) areas or a cross hatched lattice. Polygons can be poured on any layer, however if a polygon is placed on a non signal layer, it will not be poured around existing objects.

Polygons are group objects, therefore they have child objects such as tracks and arcs. You can use the **IPCB_GroupIterator** interface with the GroupIterator_Create and GroupIterator_Destroy methods from the **IPCB_Polygon** to fetch child objects.

The IPCB_Polygon interface hierarchy:

- **IPCB_Primitive**
  - **IPCB_Group**
    - **IPCB_Polygon**

IPCB_Group methods

FreePrimitives
GetPrimitiveAt
GetPrimitiveCount
SetState_XSizeYSize
FastSetState_XSizeYSize
SetState_LayersUsedArray
GroupIterator_Create
GroupIterator_Destroy
AddPCBObject
RemovePCBObject

IPCB_Group properties

X
Y
PrimitiveLock
LayerUsed

IPCB_Polygon methods

GetState_HitPrimitive
PrimitiveInsidePoly
Rebuild
SetState_XSizeYSize
SetState_CopperPourInvalid
SetState_CopperPourValid
GetState_CopperPourInvalid
GetState_InRepour
CopperPourValidate
AcceptsLayer
PointInPolygon

xBoundingRectangle
GetState_StrictHitTest
GrowPolyshape

IPCB_Polygon Properties

AreaSize
PolygonType
RemoveDead
UseOctagons
AvoidObsticles
PourOver
Grid
TrackSize
MinTrack
PointCount
Segments
PolyHatchStyle
BorderWidth
ExpandOutline
RemoveIslandsByArea
IslandAreaThreshold
RemoveNarrowNecks
ClipAcuteCorners
MitreCorners
DrawRemovedNecks
DrawRemovedIslands

```
DrawDeadCopper
ArcApproximation
```

Notes

- Polygons can be on internal planes. For example if there are multi layer pads on a PCB document, then all the internal planes are connected to these multi-layer pads as split planes and are called split plane polygons. Check the PolygonType property.
- The grid property denotes the grid which the tracks within a polygon are placed. Ideally this grid is a fraction of the component pin pitch, to allow the most effective placement of the polygon tracks.
- The segments property denotes the array of segments used to construct a polygon. Each segment consists of a record consisting of one group of points in X, Y coordinates as a line (**ePolySegmentline** type) or an arc, a radius and two angles (**ePolySegmentArc** type). Each segment record has a **Kind** field which denotes the type of segment it is.
- A segment of a polygon either as an arc or a track is encapsulated as a **TPolySegment** record as shown below.

```
TPolySegment = Record
    Kind      : TPolySegmentType;

    {Vertex}
    vx,vy      : TCoord;

    {Arc}
    cx,cy      : TCoord;
    Radius     : TCoord;
    Angle1     : TAngle;
    Angle2     : TAngle;
End;
```

Example

```
Procedure IteratePolygons;
Var
    Board       : IPCB_Board;
    Polygon     : IPCB_Polygon;
    Iterator    : IPCB_BoardIterator;
    PolygonRpt  : TStringList;
    FileName    : TPCBString;
    Document    : IServerDocument;
    PolyNo      : Integer;
```

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

    // Search for Polygons and for each polygon found
    // get its attributes and put them in a TStringList object
    // to be saved as a text file.
    Iterator        := Board.BoardIterator_Create;
    Iterator.AddFilter_ObjectSet(MkSet(ePolyObject));
    Iterator.AddFilter_LayerSet(AllLayers);
    Iterator.AddFilter_Method(eProcessAll);

    PolyNo     := 0;
    PolygonRpt := TStringList.Create;

    Polygon := Iterator.FirstPCBObject;
    While (Polygon <> Nil) Do
    Begin
        Inc(PolyNo);
        PolygonRpt.Add('Polygon No : '          + IntToStr(PolyNo));
        //Check if Net exists before getting the Name property.
        If Polygon.Net <> Nil Then
            PolygonRpt.Add(' Polygon Net : '     + Polygon.Net.Name);

        If Polygon.PolygonType = eSignalLayerPolygon Then
            PolygonRpt.Add(' Polygon type : '       + 'Polygon on Signal
Layer')
        Else
            PolygonRpt.Add(' Polygon type : '       + 'Split plane
polygon')

        PolygonRpt.Add(' Polygon BorderWidth : ' +
FloatToStr(Polygon.BorderWidth));
        PolygonRpt.Add('  Area size : '          +
FloatToStr(Polygon.AreaSize));
```

```
        // Segments of a polygon
        For I := 0 To Polygon.PointCount - 1 Do
        Begin
            If Polygon.Segments[I].Kind = ePolySegmentLine Then
            Begin
                PolygonRpt.Add(' Polygon Segment Line at X: ' +
IntToStr(Polygon.Segments[I].vx));
                PolygonRpt.Add(' Polygon Segment Line at Y: ' +
IntToStr(Polygon.Segments[I].vy));
            End
            Else
            Begin
                PolygonRpt.Add(' Polygon Segment Arc 1  : ' +
FloatToStr(Polygon.Segments[I].Angle1));
                PolygonRpt.Add(' Polygon Segment Arc 2  : ' +
FloatToStr(Polygon.Segments[I].Angle2));
                PolygonRpt.Add(' Polygon Segment Radius : ' +
FloatToStr(Polygon.Segments[I].Radius));
            End;
        End;
        PolygonRpt.Add('');
        Polygon := Iterator.NextPCBObject;
    End;
    Board.BoardIterator_Destroy(Iterator);

    // The TStringList contains Polygon data and is saved as
    // a text file.
    FileName := ChangeFileExt(Board.FileName,'.pol');
    PolygonRpt.SaveToFile(Filename);
    PolygonRpt.Free;

    // Display the Polygons report
    Document  := Client.OpenDocument('Text', FileName);
    If Document <> Nil Then
        Client.ShowDocument(Document);
End;
```

See also

PCB Design Objects

IPCB_Primitive interface

IPCB_Group interface

IPCB_GroupIterator interface

TPolygonType enumerated values

TPolySegment enumerated values

TPolyHatchStyle enumerated values

OutlinePerimeter example from the **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

IteratePolygons example from the **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

IPCB_Polygon interface methods and properties

ArcApproximation method

(IPCB_Polygon interface)

Syntax

```
Property    ArcApproximation    : TCoord   Read GetState_ArcApproximation
     Write SetState_ArcApproximation   ;
```

AreaSize property

(IPCB_Polygon interface)

Syntax

```
Property AreaSize : Extended Read GetState_AreaSize Write SetState_AreaSize;
```

Description

The AreaSize property returns the size of the polygon in Extended type. The GetState_AreaSize and SetState_AreaSize are methods for this property.

See also

IPCB_Polygon interface

AvoidObsticles property

(IPCB_Polygon interface)

Syntax

```
Property  AvoidObsticles : Boolean  Read GetState_AvoidObsticles Write
SetState_AvoidObsticles;
```

See also

IPCB_Polygon interface

BorderWidth method

(IPCB_Polygon interface)

Syntax

```
Property  BorderWidth : TCoord Read GetState_BorderWidth Write
SetState_BorderWidth;
```

ClipAcuteCorners method

(IPCB_Polygon interface)

Syntax

```
Property ClipAcuteCorners : Boolean  Read GetState_ClipAcuteCorners Write
SetState_ClipAcuteCorners;
```

DrawDeadCopper method

(IPCB_Polygon interface)

Syntax

```
Property DrawDeadCopper : Boolean  Read GetState_DrawDeadCopper Write
SetState_DrawDeadCopper;
```

DrawRemovedIslands method

(IPCB_Polygon interface)

Syntax

```
Property DrawRemovedIslands : Boolean Read GetState_DrawRemovedIslands Write
SetState_DrawRemovedIslands;
```

DrawRemovedNecks method

(IPCB_Polygon interface)

Syntax

```
Property DrawRemovedNecks : Boolean Read GetState_DrawRemovedNecks Write
SetState_DrawRemovedNecks;
```

ExpandOutline property

(IPCB_Polygon interface)

Syntax

```
Property ExpandOutline : Boolean  Read GetState_ExpandOutline Write
SetState_ExpandOutline;
```

Grid property

(IPCB_Polygon interface)

Syntax

```
Property Grid : TCoord Read GetState_Grid Write SetState_Grid;
```

IslandAreaThreshold property

(IPCB_Polygon interface)

Syntax

```
Property IslandAreaThreshold : Extended Read GetState_IslandAreaThreshold
  Write SetState_IslandAreaThreshold;
```

MinTrack property

(IPCB_Polygon interface)

Syntax

```
Property MinTrack : TCoord Read GetState_MinTrack Write SetState_MinTrack;
```

MitreCorners property

(IPCB_Polygon interface)

Syntax

```
Property MitreCorners : Boolean  Read GetState_MitreCorners Write
SetState_MitreCorners;
```

PointCount property

(IPCB_Polygon interface)

Syntax

```
Property PointCount : Integer Read GetState_PointCount Write
SetState_PointCount;
```

PolygonType property

(IPCB_Board interface)

Syntax

```
Property PolygonType : TPolygonType Read GetState_PolygonType Write
SetState_PolygonType;
```

Description

The PolygonType property defines what type the polygon is, whether it is a polygon on a signal layer, or a split plane polygon.

See also

IPCB_Polygon interface

TPolygonType type

PolyHatchStyle property

(IPCB_Polygon interface)

Syntax

```
Property PolyHatchStyle : TPolyHatchStyle Read GetState_PolyHatchStyle Write
SetState_PolyHatchStyle;
```

PourOver property

(IPCB_Polygon interface)

Syntax

```
Property PourOver : Boolean Read GetState_PourOver Write SetState_PourOver;
```

See also

IPCB_Polygon interface

RemoveDead property

(IPCB_Board interface)

Syntax

```
Property RemoveDead : Boolean Read GetState_RemoveDead Write
SetState_RemoveDead;
```

RemoveIslandsByArea property

(IPCB_Polygon interface)

Syntax

```
Property RemoveIslandsByArea : Boolean  Read GetState_RemoveIslandsByArea
Write SetState_RemoveIslandsByArea;
```

RemoveNarrowNecks property

(IPCB_Polygon interface)

Syntax

```
Property RemoveNarrowNecks : Boolean  Read GetState_RemoveNarrowNecks Write
SetState_RemoveNarrowNecks;
```

Segments property

(IPCB_Polygon interface)

Syntax

```
Property Segments [I : Integer] : TPolySegment Read GetState_Segments Write
SetState_Segments;
```

TrackSize property

(IPCB_Polygon interface)

Syntax

```
Property TrackSize : TCoord Read GetState_TrackSize Write
SetState_TrackSize;
```

UseOctagons property

(IPCB_Board interface)

Syntax

```
Property UseOctagons : Boolean Read GetState_UseOctagons Write
SetState_UseOctagons;
```

IPCB_Net

Overview

A net object can store net information from a PCB document. The net object contains information about the components used in the design, and the connectivity created in the design, stored in the form of nets. A net object is a list of pin to pin connections that are electrically connected in the design. The arrangement of the pin to pin connections is called the net topology.

The net objects are system generated objects, which means, you can retrieve the net names of PCB objects that have a net property on a PCB document.

By default the PCB editor arranges the pin to pin connections of each net to give the shortest overall connection length. To have control of the arrangement of the pin to pin connections in a net, the PCB editor allows the user to define a set of From-Tos.

The IPCB_Net interface hierarchy is as follows:

- **IPCB_Primitive**
  - **IPCB_Group**
    - **IPCB_Net**

Notes

- The ConnectsVisible property denotes the visibility of a net. If True, connections are visible.

IPCB_Group methods

```
FreePrimitives
GetPrimitiveAt
GetPrimitiveCount
SetState_XSizeYSize
FastSetState_XSizeYSize
SetState_LayersUsedArray
GroupIterator_Create
GroupIterator_Destroy
AddPCBObject
RemovePCBObject
```

IPCB_Group properties

```
X
Y
PrimitiveLock
LayerUsed
```

IPCB_Net methods

```
Rebuild
HideNetConnects
ShowNetConnects
ConnectivelyInValidate
CancelGroupWarehouseRegistration
RegisterWithGroupWarehouse
GetLogicalNet
```

IPCB_Net properties

```
Color
Name
ConnectsVisible
ConnectivelyInvalid
RoutedLength
ViaCount
PinCount
PadByName
PadByPinDescription
IsHighlighted
```

Example

```
Procedure IterateNetObjects;
Var
    Board        : IPCB_Board;
    Net          : IPCB_Net;
    Iterator     : IPCB_BoardIterator;
    LS           : TPCBString;
Begin
    // Retrieve the current board
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;
```

```
    // Create the iterator that will look for Net objects only
    Iterator        := Board.BoardIterator_Create;
    Iterator.AddFilter_ObjectSet(MkSet(eNetObject));
    Iterator.AddFilter_LayerSet(AllLayers);
    Iterator.AddFilter_Method(eProcessAll);
    // Search for Net objects and get their Net Name values
    LS := '';
    Net := Iterator.FirstPCBObject;
    While (Net <> Nil) Do
    Begin
        LS := LS + Net.Name + ', ';
        Net := Iterator.NextPCBObject;
    End;
    Board.BoardIterator_Destroy(Iterator);
    // Display the Net Names on a dialog.
    ShowInfo('Nets = ' + LS);
 End;
```

See also

PCB Design Objects

IPCB_Primitive interface

IPCB_Group interface

IPCB_GroupIterator interface

IterateNets script from the **\Examples\Scripts\DelphiScript Scripts\PCB** folder.

Dimension Objects

IPCB_OriginalDimension

Overview

The IPCB_OriginalDimension interface represents the dimensioning information on the current PCB layer. The dimension value is the distance between the start and end markers, measured in the default units. Note that the original dimension object has been superseded by a new set of dimension objects

Notes

- The IPCB_OriginalDimension interface hierarchy is as follows;
- IPCB_Primitive
  - IPCB_Group
    - IPCB_Dimension
      - IPCB_OriginalDimension

IPCB_OriginalDimension Methods

```
Function  Text    : IPCB_Text;
Function  Track1  : IPCB_Primitive;
Function  Track2  : IPCB_Primitive;
Function  Track3  : IPCB_Primitive;
Function  Track4  : IPCB_Primitive;
Function  Track5  : IPCB_Primitive;
Function  Track6  : IPCB_Primitive;
Function  Track7  : IPCB_Primitive;
Function  Track8  : IPCB_Primitive;
```

See also

IPCB_Dimension interface

PCB Design Objects

IPCB_Dimension

Overview

Dimension objects are used for dimensional details of a PCB board in either imperial or metric units and can be placed on any layer. To create an original Dimension objects, use the IPCB_OriginalDimension class which is used in P99SE and earlier versions.

Protel DXP introduced several new dimension styles - Linear, Angular, Radial, Leader, Datum, Baseline, Center, Linear Diameter and Radial Diameter objects

Notes

- The IPCB_Dimension interface is the ancestor interface for IPCB_OriginalDimension, IPCB_LinearDImension, IPCB_AngularDimension, IPCB_RadialDimension, IPCB_LeaderDimension, IPCB_DatumDimension, IPCB_BaselineDimension, IPCB_CenterDimension, IPCB_LinearDiameterDimension, IPCB_RadialDiameterDimension interfaces.
- The DimensionKind property determines the type a dimension object is.
- A dimension object especially a baseline or a leader dimension has multiple reference points. The references (a reference consists of a record of an object along with its x and y coordinate point, an anchor and is a start or end marker). A reference point is either the start or end marker and the length of two reference points is the dimensional length.

IPCB_Group methods

```
FreePrimitives
GetPrimitiveAt
GetPrimitiveCount
SetState_XSizeYSize
FastSetState_XSizeYSize
SetState_LayersUsedArray
GroupIterator_Create
GroupIterator_Destroy
AddPCBObject
RemovePCBObject
```

IPCB_Group properties

```
X
Y
PrimitiveLock
LayerUsed
```

IPCB_Dimension Methods

```
Procedure MoveTextByXY (AX,
                        AY    : TCoord);
Procedure MoveTextToXY (AX,
                        AY    : TCoord);
Procedure RotateAroundXY(AX,
                         AY    : TCoord;
                         Angle : TAngle);
Procedure References_Add(R : TDimensionReference);
Procedure References_Delete(Index : Integer);
Procedure References_DeleteLast;
Function  References_IndexOf(P     : IPCB_Primitive;
                             Index : Integer) : Integer;
Function  References_Validate : Boolean;
```

IPCB_Dimension Properties

```
DimensionKind       : TDimensionKind
TextX               : TCoord
TextY               : TCoord
X1Location          : TCoord
Y1Location          : TCoord
Size                : TCoord
LineWidth           : TCoord
TextHeight          : TCoord
TextWidth           : TCoord
TextFont            : TFontID
TextLineWidth       : TCoord
TextPosition        : TDimensionTextPosition
TextGap             : TCoord
TextFormat          : TPCBString
TextDimensionUnit   : TDimensionUnit
TextPrecision       : Integer
TextPrefix          : TPCBString
TextSuffix          : TPCBString
TextValue           : TReal
ArrowSize           : TCoord
ArrowLineWidth      : TCoord
ArrowLength         : TCoord
ArrowPosition       : TDimensionArrowPosition
ExtensionOffset     : TCoord
ExtensionLineWidth  : TCoord
ExtensionPickGap    : TCoord
Style               : TUnitStyle
References [I : Integer] : TDimensionReference
References_Count          : Integer // Read only
```

See also

IPCB_Primitive interface

TDimensionTextPosition enumerated values

TDimensionUnit enumerated values

TDimensionArrowPosition enumerated values

TDimensionReference enumerated values

TUnitStyle enumerated values

PCB Design Objects

IPCB_AngularDimension

Overview

The IPCB_AngularDimension object interface allows for the dimensioning of angular distances. There are four references (two reference points associated with two reference objects) which need to be defined and the dimension text is then placed. The references may be tracks, fills, or polygons.

Notes

The IPCB_AngularDimension interface hierarchy is as follows;

- IPCB_Primitive
  - IPCB_Group
    - IPCB_Dimension
      - IPCB_AngularDimension
- The Radius property denotes the radius size of the IPCB_AngularDimension object.
- The Sector property denotes which sector the IPCB_AngularDimension is using. Sector 1 is the angle between 0 – 90 degrees. 2 = 90 – 180 degrees. 3 = 180 =270 degrees. 4 = 270 = 360 or 0 degrees.

IPCB_AngularDimension Methods

```
Function  Text              : IPCB_Text;
Function  Arc1              : IPCB_Arc;
Function  Arc2              : IPCB_Arc;
Function  Arrow1_Track1     : IPCB_Track;
Function  Arrow1_Track2     : IPCB_Track;
Function  Arrow2_Track1     : IPCB_Track;
Function  Arrow2_Track2     : IPCB_Track;
Function  Extension1_Track  : IPCB_Track;
Function  Extension2_Track  : IPCB_Track;
```

IPCB_AngularDimension Properties

```
Property Radius  : TCoord
Property Sector  : Integer
```

See also

IPCB_Dimension interface

IPCB_Track interface

IPCB_Text interface

IPCB_Arc interface

PCB Design Objects

IPCB_BaselineDimension

Overview

The IPCB_BaselineDimension interface allows for the dimensioning of a linear distance of a collection of references, relative to a single reference. The first reference point is the base reference and all the subsequent points are relative to this base reference. The dimension value in each case is the distance between each reference point and the base reference measured in default units. The references may be objects (tracks, arcs, pads, vias, text, fills, polygons or components) or points in free space

Notes

The IPCB_BaselineDimension interface hierarchy is as follows;

- IPCB_Primitive
  - IPCB_Group
    - IPCB_Dimension
      - IPCB_BaselineDimension
- The angle property denotes the angle or rotation of the IPCB_BaselineDimension object with respect to the horizontal plane.
- Since a baseline dimension allows for the dimensioning of a linear distance over a collection of references, thus for each reference relative to the base reference, there is a text location. Use the TextLocationsCount field to obtain the number of dimension labels.

IPCB_Group methods	IPCB_Group properties
`FreePrimitives`	`X`
`GetPrimitiveAt`	`Y`
`GetPrimitiveCount`	`PrimitiveLock`
`SetState_XSizeYSize`	`LayerUsed`
`FastSetState_XSizeYSize`	
`SetState_LayersUsedArray`	
`GroupIterator_Create`	
`GroupIterator_Destroy`	
`AddPCBObject`	
`RemovePCBObject`	

IPCB_BaselineDimension Methods

```
Function  Text        : IPCB_Text;
```

```
Function  Texts         (I : Integer) : IPCB_Text;
Function  Arrow1_Track1(I : Integer) : IPCB_Track;
Function  Arrow1_Track2(I : Integer) : IPCB_Track;
Function  Arrow2_Track1(I : Integer) : IPCB_Track;
Function  Arrow2_Track2(I : Integer) : IPCB_Track;
Function  Line_Track1   (I : Integer) : IPCB_Track;
Function  Line_Track2   (I : Integer) : IPCB_Track;
Function  Extension1_Track (I : Integer) : IPCB_Track;
Function  Extension2_Track (I : Integer) : IPCB_Track;
Procedure TextLocations_Add    (Point : TCoordPoint);:
Procedure TextLocations_Delete(Index : Integer);
Procedure TextLocations_DeleteLast;
Procedure TextLocations_Clear;
```

IPCB_BaselineDimension Properties

```
Property Angle                          : TAngle
Property TextLocations [I : Integer] : TCoordPoint
Property TextLocationsCount             : Integer
```

See also

IPCB_Dimension interface

IPCB_Track interface

IPCB_Text interface

PCB Design Objects

IPCB_CenterDimension

Overview

The IPCB_CenterDimension object interface allows for the center of an arc or circle to be marked

Notes

The IPCB_CenterDimension interface hierarchy is as follows;

- IPCB_Primitive
  - IPCB_Group
    - IPCB_Dimension
      - IPCB_CenterDimension
- The angle property denotes the angle or rotation of the IPCB_CenterDimension object with respect to the horizontal plane.

IPCB_CenterDimension Methods

```
Function   Cross_Vertical_Track   : IPCB_Track;
Function   Cross_Horizontal_Track : IPCB_Track;
```

IPCB_CenterDimension Properties

```
Property Angle : TAngle
```

See also

IPCB_Dimension interface

IPCB_Track interface

PCB Design Objects

IPCB_DatumDimension

Overview

The IPCB_DatumDimension interface references the dimensioning of a linear distance of a collection of objects, relative to a single object. The dimension value is the distance between each reference object and the base object measured in the default units. The references may be tracks, arcs, pads, vias, text, fills, polygons or components.

Notes

The IPCB_DatumDimension interface hierarchy is as follows:

- IPCB_Primitive
  - IPCB_Group
    - IPCB_Dimension
      - IPCB_DatumDimension

IPCB_DatumDimension Methods

```
Function   Text                               : IPCB_Text;
Function   Texts           (I     : Integer) : IPCB_Text;
Function   Extension_Track (I     : Integer) : IPCB_Track;
```

IPCB_DatumDimension Properties

```
Property Angle : TAngle
```

See also

IPCB_Dimension interface

IPCB_Track interface

IPCB_Text interface

PCB Design Objects

IPCB_LeaderDimension

Overview

The IPCB_LeaderDimension object interface allows for the labeling of an object, point or area. There are three types of leader dimensions available which reflect the label text either being encapsulated by a circle or square or not at all. The pointer can also be an arrow or a dot which is size -definable.

Notes

The IPCB_LeaderDimension interface hierarchy is as follows:

- IPCB_Primitive
  - IPCB_Group
    - IPCB_Dimension
      - IPCB_LeaderDimension
- There are three types of leaders available: eNoShape = standard leader which means the dimension text is not enclosed at all. eRectangular the label text is encapsulated by a square, and eRounded – the dimension text is encapsulated by a circle.
- The Dot property denotes the dot symbol attached to the pointer of the leader dimension object as a dot or as an arrow.
- If the Dot field is enabled, then you can specify the size of the dot as a TCoord value.

IPCB_LeaderDimension Methods

```
Function  Text              : IPCB_Text;
Function  Dot_Arc           : IPCB_Arc;
Function  Circle_Arc        : IPCB_Arc;
Function  Arrow_Track1      : IPCB_Track;
Function  Arrow_Track2      : IPCB_Track;
Function  Square_Track1     : IPCB_Track;
Function  Square_Track2     : IPCB_Track;
Function  Square_Track3     : IPCB_Track;
Function  Square_Track4     : IPCB_Track;
Function  Line_Track (I : Integer) : IPCB_Track;
```

IPCB_LeaderDimension Properties

```
Property Shape     : TShape
Property Dot       : Boolean
Property DotSize   : TCoord
```

See also

IPCB_Dimension interface

IPCB_Track interface

IPCB_Text interface

IPCB_Arc interface

PCB Design Objects

IPCB_LinearDiameterDimension

Overview

The IPCB_LinearDimension interface references the dimensioning information on the current layer with respect to a linear distance. The dimension value is the distance between the start and end markers (reference points) measured in the default units. The references may be objects (tracks, arcs, pads, vias, text fills, polygons or components) or points in free space.

Notes

- The IPCB_LinearDiameterDimension interface hierarchy is as follows;
- IPCB_Primitive
  - IPCB_Group
    - **IPCB_Dimension**
      - **IPCB_LinearDiameterDimension**

Immediate ancestor IPCB_LinearDimension Methods

```
Function  Text                : IPCB_Text;
Function  Arrow1_Track1       : IPCB_Track;
Function  Arrow1_Track2       : IPCB_Track;
Function  Arrow2_Track1       : IPCB_Track;
Function  Arrow2_Track2       : IPCB_Track;
Function  Line_Track1         : IPCB_Track;
Function  Line_Track2         : IPCB_Track;
Function  Extension1_Track    : IPCB_Track;
Function  Extension2_Track    : IPCB_Track;
```

Immediate ancestor IPCB_LinearDimension Properties

```
Property Angle : TAngle
```

See also

IPCB_Dimension interface

IPCB_Track interface

PCB Design Objects

IPCB_LinearDimension

Overview

The IPCB_LinearDimension object interface places dimensioning information on the current layer with respect to a linear distance. The dimension value is the distance between the start and end markers (reference points) measured in the default units. The references may be objects (tracks, arcs, pads, vias, text fills, polygons or components) or points in free space.

IPCB_LinearDimension object interface has no introduced methods and properties, therefore refer to the IPCB_Dimension interface object entry for details.

Notes

- The IPCB_LinearDimension interface hierarchy is as follows;
- IPCB_Primitive
  - IPCB_Group
    - IPCB_Dimension
      - IPCB_LinearDimension
- The angle property denotes the angle or rotation of the TPCBLinearDimension object with respect to the horizontal plane.

IPCB_LinearDimension Methods

```
Function  Text              : IPCB_Text;
Function  Arrow1_Track1     : IPCB_Track;
Function  Arrow1_Track2     : IPCB_Track;
Function  Arrow2_Track1     : IPCB_Track;
Function  Arrow2_Track2     : IPCB_Track;
Function  Line_Track1       : IPCB_Track;
Function  Line_Track2       : IPCB_Track;
Function  Extension1_Track : IPCB_Track;
Function  Extension2_Track : IPCB_Track;
```

IPCB_LinearDimension Properties

```
Property Angle : TAngle
```

See also

IPCB_Dimension interface

PCB Design Objects

IPCB_RadialDimension

Overview

The IPCB_RadialDimension object interface allows for the dimensioning of a radius with respect to an arc or a circle. The dimension can be placed internally or externally on an arc or a circle.

Notes

- The IPCB_RadialDimension interface hierarchy is as follows;
- IPCB_Primitive
  - IPCB_Group
    - IPCB_Dimension
      - IPCB_RadialDimension
- This field shows the current angular step setting for the dimension. This is the rotation step used when placing the arrow portion of the dimension. Moving the arrow around the circle or arc during placement of the dimension, the number and position of possible places to anchor the dimension are determined by this angular step value.

IPCB_RadialDimension Methods

```
Function  Text         : IPCB_Text;
Function  Arrow_Track1 : IPCB_Track;
Function  Arrow_Track2 : IPCB_Track;
Function  Line1_Track  : IPCB_Track;
Function  Line2_Track  : IPCB_Track;
```

IPCB_RadialDimension Property

```
Property AngleStep : TAngle
```

See also

IPCB_Dimension interface

IPCB_Track interface

IPCB_Text interface

PCB Design Objects

IPCB_RadialDiameterDimension

Overview

The IPCB_RadialDiameterDimension interface references the dimensioning of an arc or circle with respect to the diameter, rather than the radius. The dimension can be placed either internally or externally with respect to the arc or circle

Notes

- The IPCB_RadialDiameterDimension interface hierarchy is as follows;

- IPCB_Primitive
  - IPCB_Group
    - IPCB_Dimension
      - IPCB_RadialDiameterDimension

IPCB_RadialDiameterDimension Methods

```
Function  Arrow2_Track1  : IPCB_Track;
Function  Arrow2_Track2  : IPCB_Track;
Function  Line3_Track    : IPCB_Track;
```

See also

IPCB_Dimension interface

IPCB_Track interface

PCB Design Objects

Rectangular Objects

IPCB_RectangularPrimitive

Overview

The **IPCB_RectangularPrimitive** interface is the ancestor interface for **IPCB_Fill** and **IPCB_Text** interfaces and contains the rectangular coordinates as well as the rotation property.

The **IPCB_RectangularPrimitive** interface hierarchy is as follows;

- **IPCB_Primitive**
  - **IPCB_RectangularPrimitive**

IPCB_RectangularPrimitive methods	IPCB_RectangularPrimitive properties
`RotateAroundXY`	`XLocation`
`IsRedundant`	`YLocation`
`SetState_XSizeYSize`	`X1Location`
	`Y1Location`
	`X2Location`
	`Y2Location`
	`Rotation`

See also

PCB Design Objects

IPCB_Primitive interface

IPCB_EmbeddedBoard

Overview

An **IPCB_EmbeddedBoard** interface represents a board array object in a PCB document. A board array object is a panel on the PCB document consisting of a multiple copy of the same PCB document specified by the DocumentPath field.

The multiple copy is specified by the RowCount and ColCount fields.

The spacing between the copies are specified by the RowSpacing and ColSpacing fields.

The IPCB_EmbeddedBoard hierarchy;

- **IPCB_RectangularPrimitive**
  - **IPCB_EmbeddedBoard**

IPCB_Embedded methods	IPCB_Embedded properties
	RowCount
	ColCount
	RowSpacing
	ColSpacing
	DocumentPath
	ChildBoard

See also

IPCB_RectangularPrimitive interface

PCB Design Objects

IPCB_Fill

Overview

The IPCB_Fill interface represents a PCB fill object on a PCB document.

Notes

- The IPCB_Fill interface hierarchy is as follows;
- **IPCB_Primitive**
  - **IPCB_RectangularPrimitive**
    - **IPCB_Fill**

IPCB_RectangularPrimitive methods	IPCB_RectangularPrimitive properties
RotateAroundXY	XLocation
IsRedundant	YLocation
SetState_XSizeYSize	X1Location
	Y1Location
	X2Location
	Y2Location
	Rotation

Example

```
Var
    WorkSpace : IWorkSpace;
    Board     : IPCB_Board;
```

```
    Fill      : IPCB_Fill;
Begin
    //Create a new PCB document
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('PCB');

    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil then exit;

    // Create a Fill object
    Fill             := PCBServer.PCBObjectFactory(eFillObject,
eNoDimension,eCreate_Default);
    Fill.X1Location  := MilsToCoord(2000);
    Fill.Y1Location  := MilsToCoord(2000);
    Fill.X2Location  := MilsToCoord(2500);
    Fill.Y2Location  := MilsToCoord(2500);
    Fill.Layer       := eBottomLayer;
    Fill.Rotation    := 45;
    // Add a new Fill into the PCB design database.
    Board.AddPCBObject(Fill);

    // Refresh the PCB document
    ResetParameters;
    AddStringParameter('Action', 'All');
    RunProcess('PCB:Zoom');
End;
```

See also

PCB Design Objects

IPCB_Primitive interface

IPCB_RectangularPrimitive interface

Undo script in \Examples\Scripts\PCB folder.

IPCB_Text

Overview

Text strings can be placed on any layer with any height. There are two classes of text strings: Free text strings and component text (designators and comments). Free text strings are standalone strings which could be used as descriptors or labels for any application on the workspace. There are two component text objects- designator attribute and comment attribute. Each component must have a unique designator and thus designators are not globally editable. The comment attribute is globally editable though.

The PCB editor includes special strings which are interpreted when output (printing, plotting or generating gerber files) is generated. For example, the string .PRINT_DATE will be replaced by the current date when output is generated.

Notes

The IPCB_Text Interface hierarchy is as follows:

- **IPCB_Primitive**
  - **IPCB_RectangularPrimitive**
    - **IPCB_Text**

- Text objects are not inherited from the **IPCB_group** interface, therefore fetching child objects within a text object is not possible.
- Text objects are rectangular primitives with rectangular coordinates properties and the rotation property.

IPCB_RectangularPrimitive methods

`RotateAroundXY`

`IsRedundant`

`SetState_XSizeYSize`

IPCB_RectangularPrimitive properties

`XLocation`

`YLocation`

`X1Location`

`Y1Location`

`X2Location`

`Y2Location`

`Rotation`

IPCB_Text methods

```
IsHidden
IsDesignator
IsComment
InAutoDimension
GetDesignatorDisplayString
RotationHandle
```

IPCB_Text properties

```
Size
FontID
Text
Width
MirrorFlag
UnderlyingString
```

Example

```
Var
    Board     : IPCB_Board;
    WorkSpace : IWorkSpace;
    TextObj   : IPCB_Text;
Begin
    //create a new pcb document
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('PCB');

    // Retrieve the reference to the new board
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil then exit;

    // Create a text object on a top overlay layer
    Board.LayerIsDisplayed[eTopOverLay] := True;
    TextObj := PCBServer.PCBObjectFactory(eTextObject, eNoDimension,
eCreate_Default);
    TextObj.XLocation := Board.XOrigin + MilsToCoord(4000);
    TextObj.YLocation := Board.YOrigin + MilsToCoord(2000);
    TextObj.Layer     := eTopOverlay;
    TextObj.Text      := 'Text Object';
    TextObj.Size      := MilsToCoord(90);   // sets the height of the text.
    Board.AddPCBObject(TextObj);
End;
```

See also

PCB Design Objects

IPCB_Primitive interface

IPCB_RectangularPrimitive interface

Place_Strings script example in **\Scripts\DelphiScript Scripts\PCB** folder.

PCB Rule Objects

The PCB editor incorporates a large set of design rules to help define compliance/constraints regarding the placement of PCB objects, routing methods, and netlists.

These rules include clearances, object geometry, impedance control, routing priority, routing topology and parallelism. Rule scope is the extent of each rule determined. The scope allows you to define the set of target objects that a particular instance of a rule is to be applied to.

IPCB_Rule

Overview

The IPCB_Rule interface object encapsulates an existing PCB design rule in an opened PCB document in DXP. Each design rule has its own Unique ID. To set the scope of a rule, unary or binary scope expressions are defined.

The PCB editor incorporates a large set of design rules to help define compliance/constraints regarding the placement of PCB objects, routing methods, and netlists. These rules include clearances, object geometry, impedance control, routing priority, routing topology and parallelism. Rule scope is the extent of each rule determined. The scope allows you to define the set of target objects that a particular instance of a rule is to be applied to.

IPCB_Rule Methods

```
Function  Priority                         : TRulePrecedence;
Function  ScopeKindIsValid  (AScopeKind   : TScopeKind)          : Boolean;
Function  Scope1Includes    (P            : IPCB_Primitive)      : Boolean;
Function  Scope2Includes    (P            : IPCB_Primitive)      : Boolean;
Function  NetScopeMatches   (P1,
                             P2           : IPCB_Primitive)      : Boolean;
Function  CheckBinaryScope  (P1,
                             P2           : IPCB_Primitive)      : Boolean;
Function  CheckUnaryScope   (P            : IPCB_Primitive)      : Boolean;
Function  GetState_DataSummaryString      : TPCBString;
Function  GetState_ShortDescriptorString : TPCBString;
Function  GetState_ScopeDescriptorString : TPCBString;
Function  ActualCheck                (P1,
                                      P2  : IPCB_Primitive)      :
IPCB_Violation;
```

IPCB_Rule Properties

```
Property Scope1Expression : TPCBString
Property Scope2Expression : TPCBString
```

```
Property RuleKind          : TRuleKind
Property NetScope          : TNetScope
Property LayerKind         : TRuleLayerKind
Property Comment           : TPCBString
Property Name              : TPCBString
Property DRCEnabled        : Boolean
Property UniqueId          : TPCBString       //Read only
```

Enumerated Types

TScopeKind

TNetScope

TRuleKind

TRuleLayerKind

IPCB_AcuteAngle rule

Overview

The IPCB_AcuteAngleRule interface specifies the minimum angle permitted at a track corner.

IPCB_AcuteAngle Properties

```
Minimum : TAngle
```

IPCB_BrokenNetRule rule

Overview

The IPCB_BrokenNetRule rule deals with broken nets in relation to polygons. Polygons that are affected by the broken net rules are highlighted or not.

IPCB_BrokenNetRule Properties

```
HighlightPolygons : Boolean
```

IPCB_ComponentClearanceConstraint rule

Overview

The Component Clearance Constraint PCB Design rule has available Check Mode setting:

Quick Check – uses a components' bounding rectangle to define its shape. The bounding rectangle is the smallest rectangle that encloses all the primitives that make up a component.

Multi Layer Check – also uses a component bounding rectangle, but considers through-hole component pads on a board with components on both sides, allowing surface mount components to be placed under a through-hole component.

Full Check – uses the exact shape that encloses all the primitives that make up each component. Use this option if the design includes a large number of circular or irregular shaped components.

IPCB_ComponentClearanceConstraint Properties

```
Property Gap                : TCoord
Property CollisionCheckMode : TComponentCollisionCheckMode
```

See also

TComponentCollisionCheckMode

IPCB_ComponentRotationsRule rule

Overview

The IPCB_ComponentRotationsRule specifies allowable component orientations. Multiple orientations are permitted, allowing the autoplacer to use any of the enabled orientations. The allowed component orientations are: 0,90,180, 270, or AllRotations. It is possible to have multiple settings, for example setting at 0 and 270 degrees rotations only.

IPCB_ComponentRotationsRule Properties

```
Property AllowedRotations : Integer
```

IPCB_ConfinementConstraint rule

Overview

The IPCB_ConfinementConstraint interface specifies a rectangular region in which a set of objects is either allowed, or not allowed. Use this function to define a region that a class of components must be placed in.

IPCB_ConfinementConstraint Methods

```
Procedure RotateAroundXY (AX,
                          AY    : TCoord;
                          Angle : TAngle);
```

IPCB_ConfinementConstraint Properties

```
Property X            : TCoord
Property Y            : TCoord
Property Kind         : TConfinementStyle
Property Layer        : TLayer
Property BoundingRect : TCoordRect
```

IPCB_ClearanceConstraint Rule

Overview

This interface defines the minimum clearance between any two primitive objects on a copper layer.

Important Notes

- The PrimitivesViolate function.
- The Gap property determines the gap size of the track segments.

IPCB_ClearanceConstraint Methods

```
Function  PrimitivesViolate(P1, P2  : IPCB_Primitive) : Boolean;
```

IPCB_ClearanceConstraint Properties

```
Property Gap  : TCoord
```

IPCB_DaisyChainStubLengthConstraint rule

Overview

The daisy chain stub length rule specifies the maximum permissible stub length for a net with a daisy chain topology.

Important Notes

Limit property for the stub length.

IPCB_DaisyChainStubLengthConstraint Properties

```
Property Limit : TCoord
```

IPCB_FanoutControlRule rule

Overview

The IPCB_FanoutControl rule determines how BGAs on a PCB document is going to be fanned in respect to vias placement for routing.

IPCB_FanoutControlRule Properties

```
Property FanoutStyle                     : TFanoutStyle
Property FanoutDirection                 : TFanoutDirection
Property BGAFanoutDirection              : TBGAFanoutDirection
Property BGAFanoutViaMode                : TBGAFanoutViaMode
Property ViaGrid                         : TCoord
```

IPCB_LayerPairsRule rule

Overview

The IPCB_LayerPairsRule interface deals with whether the layer pairs are going to be enforced or not on the current PCB document.

IPCB_LayerPairsRule Properties

```
Property EnforceLayerPairs : Boolean
```

IPCB_MatchedNetLengthsConstraint rule

Overview

The matched net lengths rule specifies the degree to which nets can have different lengths.

Important Notes

The 90 degree style is the most compact and the Rounded style is the least compact.

IPCB_MatchedNetLengthsConstraint Methods

```
Function  MatchLengthForFromTo(P1,P2 : IPCB_Primitive) : IPCB_Violation;
Function  MatchLengthForNet   (P1,P2 : IPCB_Primitive) : IPCB_Violation;
```

IPCB_MatchedNetLengthsConstraint Properties

```
Property Amplitude : TCoord
Property Gap       : TCoord
Property Style     : TLengthenerStyle
Property Tolerance : TCoord
```

IPCB_MaxMinHeightConstraint rule

Overview

The IPCB_MaxMinHeightConstraint rule deals with heights of components, and you can set the maximum, minimum and preferred height values for targeted components on a PCB document.

Important Notes

MaxHeight, MinHeight and PreferedHeight properties.

IPCB_MaxMinHeightConstraint Properties

```
Property MaxHeight      : TCoord
Property MinHeight      : TCoord
Property PreferedHeight : TCoord
```

IPCB_MaxMinHoleSizeConstraint rule

Overview

The IPCB_MaxMinHoleSizeContraint rule deals with the constraints of hole sizes on a PCB document.

IPCB_MaxMinHoleSizeConstraint Properties

```
Property AbsoluteValues : Boolean
Property MaxLimit       : TCoord
Property MinLimit       : TCoord
Property MaxPercent     : TReal
```

```
Property MinPercent     : TReal
```

IPCB_MaxMinWidthConstraint rule

Overview

This routing width constraint interface defines the minimum, favored and maximum width of tracks and arcs on copper layers.

IPCB_MaxMinWidth Properties

```
Property  MaxWidth    [Const L : TLayer]  : TCoord
Property  MinWidth    [Const L : TLayer]  : TCoord
Property  FavoredWidth[Const L : TLayer]  : TCoord
Property  ImpedanceDriven                 : Boolean
Property  MinImpedance                    : TDouble
Property  FavoredImpedance                : TDouble
Property  MaxImpedance                    : TDouble
```

IPCB_MaxMinLengthConstraint rule

Overview

This IPCB_MaxMinLengthConstraint rule defines the minimum and maximum lengths of a net.

IPCB_MaxMinLengthConstraint Properties

```
Property MaxLimit : TCoord
Property MinLimit : TCoord
```

IPCB_MinimumAnnularRing rule

Overview

The minimum annular ring rule determines the minimum size of an annular ring.

IPCB_MinimumAnnularRing Properties

```
Property Minimum : TCoord
```

IPCB_MaximumViaCountRule rule

Overview

The maximum via count rule specifies the maximum number of vias permitted on a PCB document.

Important Notes

Set or return the maximum number of vias for the Limit property

IPCB_MaximumViaCount Properties

```
Property Limit : Integer
```

IPCB_NetsToIgnoreRule rule

Overview

The Nets To Ignore rule determines which nets to ignore during Design Rule Check.

IPCB_NetsToIgnoreRule Methods

No new interface methods

IPCB_NetsToIgnoreRule Properties

No new interface properties

See also

IPCB_Rule interface

IPCB_ParallelSegmentConstraint rule

Overview

This rule specifies the distance two track segments can run in parallel, for a given separation. Note that this rule tests track segments, not collections of track segments. Apply multiple parallel segment constraints to a net to approximate crosstalk characteristics that vary as a function of length and gap.

Important Notes

The Gap and Limit properties concern the track segments.

IPCB_ParallelSegmentConstraint Properties

```
Property Gap   : TCoord
Property Limit : TCoord
```

IPCB_PasteMaskExpansionRule rule

Overview

The IPCB_PasteMaskExpansionRule function returns or sets values for a paste mask expansion rule object. The Paste Mask Expansion Rule specifies the amount of radial expansion or radial contraction of each pad site.

Important Notes

The Expansion property sets or returns the radial expansion or contraction value (a negative value denotes contraction).

IPCB_PasteMaskExpansionRule Properties

```
Property Expansion : TCoord
```

IPCB_PermittedLayersRule rule

Overview

The IPCB_PermittedLayersRule function returns or sets the permitted layers rule which specifies the layers components can be placed on during placement with the Cluster Placer. The Cluster Placer does not change the layer a component is on, you must set the component layer prior to running the placer.

IPCB_PermittedLayersRule Properties

```
Property PermittedLayers : TLayerSet
```

IPCB_PowerPlaneClearanceRule rule

Overview

The power plane clearance rule determines the clearance of the power plane with a value of TCoord type.

IPCB_PowerPlaneClearanceRule Properties

```
Property Clearance : TCoord
```

IPCB_PowerPlaneConnectStyleRule rule

Overview

This power plane connect style rule specifies the style of the connection from a component pin to a power plane. There are two connection types - direct connections (the pin to solid copper) or thermal relief connection.

Important Notes

The TPlaneConnectStyle type determines the connection style for a plane. If Thermal Relief connection is used, then the thermal relief conductor width, the relief expansion, the width of the air gap and the number of relief entries need to be determined. If direct connection style is used, then the previous parameters are not needed.

IPCB_PowerPlaneConnectStyleRule Properties

```
Property PlaneConnectStyle     : TPlaneConnectStyle
Property ReliefExpansion       : TCoord
Property ReliefConductorWidth : TCoord
Property ReliefEntries         : Integer
Property ReliefAirGap          : TCoord
```

IPCB_PolygonConnectStyleRule rule

Overview

The Polygon Connect Style Rule returns or sets the polygon connect style rule which specifies how the polygon is connected to the power plane.

Important Notes

• The TPlaneConnectStyle type specifies the polygon connect style rule which is relief connection to a polygon, or direct connection to a polygon from a component pin. That is, the type of connection from a component pin to the polygon.

• The relief conductor width property denotes the width of the conductor between two air gaps.

• The relief entries property specifies the number of relief entries (2 or 4) for the relief connection of the polygon connection. For other types of connection, this field is irrelevant.

• The PolygonReliefAngle type specifies the angle of relief connections in 45 or 90 degrees.

IPCB_PolygonConnectStyleRule Properties

```
Property ConnectStyle          : TPlaneConnectStyle
Property ReliefConductorWidth : TCoord
Property ReliefEntries         : Integer
Property PolygonReliefAngle    : TPolygonReliefAngle
```

IPCB_RoutingCornerStyleRule

Overview

This routing corners rule specifies the corner style to be used during autorouting a PCB document.

Important Notes

• The TCornerStyle type sets or returns the corner style which can be a 45 degree camfer or rounded using an arc.

• The minsetback and maxsetback properties specify the minimum and maximum distance from the corner location to the start of the corner chamfer or arc.

IPCB_RoutingCornerStyleRule Properties

```
Property Style        TCornerStyle
Property MinSetBack : TCoord
Property MaxSetBack : TCoord
```

IPCB_RoutingLayersRule rule

Overview

This routing layers rule specifies the preferred routing direction for layer to be used during autorouting.

Important Notes

N/A

IPCB_RoutingLayersRule Properties

```
Property RoutingLayers [L : TLayer] : Boolean
```

IPCB_RoutingPriorityRule rule

Overview

This routing priority rule function assigns a routing priority which is used to set the order of how the nets will be auto routed.

IPCB_RoutingPriorityRule Properties

```
Property RoutingPriority : Integer
```

IPCB_RoutingTopologyRule rule

Overview

This routing topology rule function specifies the topology of the net. The net compromises a pattern of the pin-to-pin connections. A topology is applied to a net for specific reasons, for example to minimise signal reflections, daisy chain topology is used.

Notes

The Topology property sets or returns the topology of the net. The following topologies can be applied: Shortest, Horizontal, Vertical, Daisy-Simple, Daisy-Mid Driven, Daisy-Balanced, or Star.

IPCB_RoutingTopologyRule Properties

```
Property Topology: TNetTopology
```

IPCB_RoutingViaStyleRule rule

Overview

This routing via style rule specifies the via object to be used during autorouting. Vias can be through-hole, Blind (from a surface layer to an inner layer) or Buried (between two inner layers).

Important Notes

The ViaStyle property sets or returns the via style. Vias can be thru-hole, blind (from a surface layer to an inner layer) or buried (between two inner layers).

IPCB_RoutingViaStyleRule Properties

```
Property MinHoleWidth      : TCoord
Property MaxHoleWidth      : TCoord
Property PreferedHoleWidth : TCoord
Property MinWidth          : TCoord
```

```
Property MaxWidth          : TCoord
Property PreferedWidth     : TCoord
Property ViaStyle          : TRouteVia
```

IPCB_RuleSupplyNets rule

Overview

This IPCB_RuleSupplyNets interface specifies the supply nets on the board. The signal integrity analyzer needs to know each supply net name and voltage.

IPCB_RuleSupplyNets Properties

```
Property Voltage : Double
```

IPCB_ShortCircuitConstraint rule

Overview

The short circuit constraint rule includes a constraint to test for short circuits between primitive objects on the copper layers. A short circuit exists when two objects that have different net names touch.

Notes

The Allowed property sets or returns the boolean value whether or not the short circuit constraint rule is allowed.

IPCB_ShortCircuitConstraint Properties

```
Property Allowed : Boolean
```

IPCB_SMDNeckDownConstraint rule

Overview

IPCB_SMDToPlaneConstraint Properties

```
Property Percent : TReal
```

IPCB_SMDToCornerConstraint rule

Overview

Important Notes

The Distance property determines the distance between the SMD and a corner.

IPCB_SMDToCornerConstraint Properties

```
Property Distance : TCoord
```

IPCB_SMDToPlaneConstraint rule

Overview

IPCB_SMDToPlaneConstraint Methods

```
Function  IsInternalPlaneNet(Net   : IPCB_Net; Board : IPCb_Board): Boolean;
```

IPCB_SMDToPlaneConstraint Properties

```
Property Distance : TCoord
```

IPCB_SolderMaskExpansionRule rule

Overview

The solder mask expansion rule defines the shape that is created on the solder mask layer at each pad and via site. This shape is expanded or contracted radially by the amount specified by this rule.

IPCB_SolderMaskExpansion Properties

```
Property Expansion : TCoord
```

IPCB_TestPointStyleRule rule

Overview

The Protel autorouter includes a testpoint generator, which can identify existing pads and vias as testpoints, as well as adding testpoint pads to nets which can not be accessed at existing pads and vias. Generally the testpoint types are used in bare board testing or are used for in-circuit testing.

IPCB_TestPointStyleRule Methods

Procedure DoDefaultStyleOrder;

IPCB_TestPointStyleRule Properties

```
Property TestpointUnderComponent                  : Boolean
Property MinSize                                  : TCoord
Property MaxSize                                  : TCoord
Property PreferedSize                             : TCoord
Property MinHoleSize                              : TCoord
Property MaxHoleSize                              : TCoord
Property PreferedHoleSize                         : TCoord
Property TestpointGrid                            : TCoord
Property OrderArray [I : Integer]                 : TTestPointStyle
Property AllowedSide                              : TTestpointAllowedSideSet
Property AllowedStyleSet                          : TTestPointStyleSet
```

```
Property Allowed [I : TTestPointStyle]          : Boolean
Property TestpointPriority[I : TTestPointStyle] : Integer
```

IPCB_TestPointUsage rule

Overview

Protel's autorouter includes a testpoint generator, which can identify existing pads and vias as testpoints, as well as adding testpoint pads to nets which can not be accessed at existing pads and vias. Generally the testpoint types are used in bare board testing or are used for in-circuit testing.

IPCB_TestPointUsage Properties

```
Property Valid               : TTestpointValid
Property AllowMultipleOnNet : Boolean
```

IPCB_UnConnectedPinRule rule

Overview

This interface deals with unconnected pins on a PCB document.

IPCB_UnConnectedPinRule Properties

No new properties.

See also

IPCB_Rule interface

IPCB_ViasUnderSMDConstraint rule

Overview

The Vias Under SMD constraint rule specifies if vias can be placed under SMD pads during autorouting.

IPCB_ViasUnderSMDConstraint Properties

```
Property Allowed : Boolean
```

Signal Integrity Design Rules

IPCB_SignalStimulus rule

Overview

The IPCB_SignalStimulus rule concerns with the definition of a signal for stimulus, such as the stimulus type, signal level, start, stop times and the period of the signal.

IPCB_SignalStimulus Methods

```
Procedure Export_ToStmFile           (AFilename : TString);
```

IPCB_SignalStimulus Properties

```
Property Kind       : TStimulusType
Property Level      : TSignalLevel
Property StartTime  : TReal
Property StopTime   : TReal
Property PeriodTime : TReal
```

IPCB_MaxOvershootFall rule

Overview

The IPCB_MaxOvershootFall interface specifies the maximum allowable overshoot (ringing below the base value) on the falling edge of the signal.

IPCB_MaxOvershootFall Properties

```
Property Maximum : TReal
```

IPCB_MaxOvershootRise rule

Overview

The IPCB_MaxOvershootRise interface specifies the maximum allowable overshoot (ringing above the base value) on the rising edge of the signal.

IPCB_MaxOvershootRise Properties

```
Property  Maximum : TReal
```

IPCB_MaxUndershootFall

Overview

The IPCB_MaxUndershootFall interface specifies the maximum allowable undershoot (ringing above the base value) on the falling edge of the signal.

IPCB_MaxUndershootFall Properties

```
Property  Maximum : TReal
```

IPCB_MaxUndershootRise rule

Overview

The IPCB_MaxUndershootRise function specifies the maximum allowable undershoot (ringing below the top value) on the rising edge of the signal.

IIPCB_MaxUndershootRise Properties

```
Property Maximum : TReal
```

IPCB_RuleMaxMinImpedance rule

Overview

The IPCB_RuleMaxMinImpedance interface returns or sets values for a MaxMin Impedance rule object depending on the query mode (eGetState or eSetState). This rule specifies the minimum and maximum net impedance allowed. Net impedance is a function of the conductor geometry and conductivity, the surrounding dielectric material (the board base material, multilayer insulation, solder mask, etc) and the physical geometry of the board (distance to other conductors in the z-plane). This function defines the minimum and maximum impedance values allowed for the signal integrity rule.

IPCB_RuleMaxMinImpedance Properties

```
Property Minimum : TReal
Property Maximum : TReal
```

IPCB_RuleMinSignalTopValue rule

Overview

The IPCB_RuleMinSignalTopValue function specifies the minimum allowable signal top value. The top value is the voltage that a signal settles into the minimum top state.

IPCB_RuleMinSignalTopValue Properties

```
Property Minimum : TReal
```

IPCB_RuleMaxSignalBaseValue rule

Overview

The IPCB_RuleMaxSignalBaseValue function specifies the maximum allowable base value. The base value is the voltage that a signal settles to in the low state.

IPCB_ RuleMaxSignalBaseValue Properties

```
Property Maximum : TReal
```

IPCB_RuleFlightTime_RisingEdge rule

Overview

The IPCB_RuleFlightTime_RisingEdge interface returns or sets values for the flight time of the rising edge of a signal. The flight time is the signal delay introduced by the interconnect structure. It is calculated as the time it takes to drive the actual input to the threshold voltage, less the time it would take to drive a reference load (connected directly to the output) to the threshold voltage.

IPCB_RuleFlightTime_RisingEdge Properties

```
Property MaximumFlightTime : TReal
```

IPCB_RuleFlightTime_FallingEdge rule

Overview

The IPCB_RuleFlightTime_FallingEdge interface returns or sets values for the flight time of the falling edge of a signal. The flight time is the signal delay introduced by the interconnect structure. It is calculated as the time it takes to drive the actual input to the threshold voltage, less the time it would take to drive a reference load (connected directly to the output) to the threshold voltage.

IPCB_RuleFlightTime_FallingEdge Properties

```
Property MaximumFlightTime : TReal
```

IPCB_RuleMaxSlopeRisingEdge rule

Overview

The IPCB_RuleMaxSlope_RisingEdge interface specifies the maximum allowable slope on the rising edge of the signal. The slope is the time it takes for a signal to rise from the threshold voltage to a valid high voltage.

IPCB_RuleMaxSlopeRisingEdge Properties

```
Property MaxSlope : TReal
```

IPCB_RuleMaxSlopeFallingEdge rule

Overview

The IPCB_RuleMaxSlope_FallingEdge interface specifies the maximum allowable slope on the falling edge of the signal. The slope is the time it takes for a signal to fall from the threshold voltage to a valid low voltage.

IPCB_RuleMaxSlopeFallingEdge Properties

```
Property MaxSlope : TReal
```

PCB Enumerated Types

The enumerated types are used for many of the schematic interfaces methods which are covered in this section. For example the I:CB_Board interface has a Property LayerIsUsed [L : TLayer] : Boolean property. You can use this Enumerated Types section to check what the range is for the TLayer type.

See also

PCB API Reference

PCB types table

TAngle	TNetScope
TAutoPanMode	TObjectId
TAutoPanUnit	TObjectCreationMode
TClassMemberKind	TObjectSet
TComponentStyle	TPadCache
TComponentMoveKind	TPadMode
TComponentTypeMapping	TPadName
TConnectionMode	TPadSwapName
TCornerStyle	TPCBDragMode
TDaisyChainStyle	TPlaneConnectStyle
TDielectricRecord	TPlaneConnectionStyle
TDimensionArrowPosition	TPlaneDrawMode
TDimensionTextPosition	TPolyHatchStyle
TDimensionKind	TPolygonType
TDimensionReference	TPolygonRepourMode
TDimensionType	TPolySegmentType
TDimensionUnit	TRuleKind
TGraphicsCursor	TRuleLayerKind
TIterationMethod	TScopeKind
TInteractiveRouteMode	TShape
TLayer	TSameNamePadstackReplacementMode
TLayerSet	TTextAlignment
TMechanicalLayerPair	TTextAutoposition
TNetName	TUnitStyle
TNetTopology	TUnit

TAngle

Double type.

TApertureUse

```
TApertureUse         = ( eNoApertureUse,
                         eMultiUse,
                         eDrawUse,
                         eFlashUse);
```

TAutoPanMode

```
TAutoPanMode         = ( eNoAutoPan
                         eReCentre
                         eFixedJump
                         eShiftAccellerator
                         eShiftDeccellerator
                         eBallistic
                         eAdaptive
                       );
```

TAutoPanUnit

```
TAutoPanUnit         = ( eAutoPanByMils
                         eAutoPanByPixels
                       );
```

TBaud

```
TBaud                = ( eBaud110                                   ,
                         eBaud150                                   ,
                         eBaud300                                   ,
                         eBaud600                                   ,
                         eBaud1200                                  ,
                         eBaud2400                                  ,
                         eBaud4800                                  ,
                         eBaud9600                                  ,
                         eBaud19200
                       );
```

TBGAFanoutDirection

```
TBGAFanoutDirection   = ( eBGAFanoutDirection_Out                    ,
                          eBGAFanoutDirection_NE                     ,
                          eBGAFanoutDirection_SE                     ,
                          eBGAFanoutDirection_SW                     ,
                          eBGAFanoutDirection_NW                     ,
                          eBGAFanoutDirection_In
                         );
```

TBGAFanoutViaMode

```
TBGAFanoutViaMode     = ( eBGAFanoutVia_Closest                      ,
                          eBGAFanoutVia_Centered
                        );
```

TCacheState

```
TCacheState           = ( eCacheInvalid,
                          eCacheValid,
                          eCacheManual);
```

TChangeScope

```
TChangeScope          = ( eChangeNone                                ,
                          eChangeThisItem                            ,
                          eChangeAllPrimitives                       ,
                          eChangeAllFreePrimitives                   ,
                          eChangeComponentDesignators                ,
                          eChangeComponentComments                   ,
                          eChangeLibraryAllComponents                ,
                          eChangeCancelled
                        );
```

TClassMemberKind

```
TClassMemberKind      = ( eClassMemberKind_Net                       ,
                          eClassMemberKind_Component                 ,
                          eClassMemberKind_FromTo                    ,
                          eClassMemberKind_Pad                       ,
                          eClassMemberKind_Layer                     ,
                          eClassMemberKind_DesignChannel
```

```
                        );
```

TComponentStyle

```
TComponentStyle      = ( eComponentStyle_Unknown                  ,
                         eComponentStyle_Small                    ,
                         eComponentStyle_SmallSMT                 ,
                         eComponentStyle_Edge                     ,
                         eComponentStyle_DIP                      ,
                         eComponentStyle_SIP                      ,
                         eComponentStyle_SMSIP                    ,
                         eComponentStyle_SMDIP                    ,
                         eComponentStyle_LCC                      ,
                         eComponentStyle_BGA                      ,
                         eComponentStyle_PGA
                       );
```

TComponentCollisionCheckMode

```
TComponentCollisionCheckMode
                     = ( eQuickCheck
                         eMultiLayerCheck
                         eFullCheck
                       );
```

TComponentMoveKind

```
TComponentMoveKind   = ( eNoComponentMoveNoAction
                         eJumpToComponent
                         eMoveComponentToCursor
                       );
```

TComponentType

```
TComponentType       = ( eBJT                                     ,
                         eCapactitor                              ,
                         eConnector                               ,
                         eDiode                                   ,
                         eIC                                      ,
                         eInductor                                ,
                         eResistor
```

```
                    );
```

TConfinementStyle

```
TConfinementStyle    = ( eConfineIn,
                         eConfineOut);
```

TConnectionMode

```
TConnectionMode      = ( eRatsNestConnection
                         eBrokenNetMarker
                       );
```

TCoord

```
TCoord = Integer;
```

Note

You can use MilsToCoord, MMsToCoord and CoordToMMs, CoordToMils functions to convert a value unit to a different value unit.

See also

PCB Functions

TCoordPoint

```
TCoordPoint = Record
        x,
        y : TCoord;
End;
```

TCoordRect

```
TCoordRect    = Record
    Case Integer of
       0 :(left,bottom,right,top : TCoord);
       1 :(x1,y1,x2,y2           : TCoord);
       2 :(Location1,Location2   : TPoint);
End;
```

Note TPoint is a Borland Delphi defined type in the Types.pas unit.

TCopyMode

```
TCopyMode = ( eFullCopy,
              eFieldCopy);
```

TCornerStyle

```
TCornerStyle          = ( eCornerStyle_90,
                          eCornerStyle_45,
                          eCornerStyle_Round);
```

TDaisyChainStyle

```
TDaisyChainStyle      = ( eDaisyChainLoad                            ,
                          eDaisyChainTerminator                      ,
                          eDaisyChainSource
                        );
```

TDataBits

```
TDataBits             = ( eDataBits5                                 ,
                          eDataBits6                                 ,
                          eDataBits7                                 ,
                          eDataBits8
                        );
```

TDielectricType

```
TDielectricType = (eNoDielectric,
                   eCore,
                   ePrePreg,
                   eSurfaceMaterial);
```

TDimensionArrowPosition

```
TDimensionArrowPosition = ( eInside,eOutside);
```

TDimensionTextPosition

```
TDimensionTextPosition
                      = ( eTextAuto                                  ,
                          eTextCenter                                ,
                          eTextTop                                   ,
                          eTextBottom                                ,
                          eTextRight                                 ,
                          eTextLeft                                  ,
                          eTextInsideRight                           ,
                          eTextInsideLeft                            ,
```

```
                         eTextUniDirectional                   ,
                         eTextManual
                       );
```

TDimensionKind

```
TDimensionKind         = ( eNoDimension            ,
                           eLinearDimension        ,
                           eAngularDimension       ,
                           eRadialDimension        ,
                           eLeaderDimension        ,
                           eDatumDimension         ,
                           eBaselineDimension      ,
                           eCenterDimension        ,
                           eOriginalDimension      ,
                           eLinearDiameterDimension,
                           eRadialDiameterDimension
                         );
```

TDimensionReference

```
 TDimensionReference              = Record
     Primitive                    : IPCB_Primitive;
     Point                        : TCoordPoint;
     Anchor                       : Integer;
 End;
```

TDimensionUnit

```
TDimensionUnit         = ( eMils                                 ,
                           eInches                               ,
                           eMillimeters                          ,
                           eCentimeters                          ,
                           eDegrees                              ,
                           eRadians                              ,
                           eAutomaticUnit
                         );
```

TDisplay

```
TDisplay               = ( eOverWrite
```

```
                         eHide
                         eShow
                         eInvert
                         eHighLight
                       );
```

TDrillSymbol

```
TDrillSymbol         = ( eSymbols                                   ,
                         eNumbers                                   ,
                         eLetters
                       ); {Used by gerber and Print/ Plot}
```

TDynamicString

```
TDynamicString = AnsiString;
```

TDrawMode (PCB)

```
TDrawMode            = ( eDrawFull,
                         eDrawDraft,
                         eDrawHidden);
```

TEditingAction

```
TEditingAction=(eEditAction_Focus,
                eEditAction_Move,
                eEditAction_Change,
                eEditAction_Delete,
                eEditAction_Select,
                eEditAction_NonGraphicalSelect,
                eEditAction_Measure,
                eEditAction_Dimension);
```

TFanoutDirection

```
TFanoutDirection     = ( eFanoutDirection_None                      ,
                         eFanoutDirection_InOnly                    ,
                         eFanoutDirection_OutOnly                   ,
                         eFanoutDirection_InThenOut                 ,
                         eFanoutDirection_OutThenIn                 ,
                         eFanoutDirection_Alternating
                       );
```

TFanoutStyle

```
TFanoutStyle          = ( eFanoutStyle_Auto                           ,
                            eFanoutStyle_Rows                          ,
                            eFanoutStyle_Staggered                     ,
                            eFanoutStyle_BGA                           ,
                            eFanoutStyle_UnderPads
                          );
```

TFromToDisplayMode

```
TFromToDisplayMode    = ( eFromToDisplayMode_Automatic                ,
                          eFromToDisplayMode_Hide                     ,
                          eFromToDisplayMode_Show
                        );
```

TGraphicsCursor

```
TGraphicsCursor = ( eCurShapeCross90,
                    eCurShapeBigCross,
                    eCurShapeCross45);
```

THandshaking

```
THandshaking          = ( eHandshakingNone                            ,
                          eHandshakingXonXOff                         ,
                          eHandshakingHardwire
                        );
```

TIterationMethod

```
TIterationMethod = ( eProcessAll, eProcessFree, eProcessComponent);
```

TEnabledRoutingLayers

```
TEnabledRoutingLayers = Array [eTopLayer..eBottomLayer] Of Boolean;
```

TInteractiveRouteMode

```
 TInteractiveRouteMode
                       = ( eIgnoreObstacle
                           eAvoidObstacle
                           ePushObstacle
                         );
```

TLayer

```
TLayer = (eNoLayer          ,
          eTopLayer         ,
          eMidLayer1        ,
          eMidLayer2        ,
          eMidLayer3        ,
          eMidLayer4        ,
          eMidLayer5        ,
          eMidLayer6        ,
          eMidLayer7        ,
          eMidLayer8        ,
          eMidLayer9        ,
          eMidLayer10       ,
          eMidLayer11       ,
          eMidLayer12       ,
          eMidLayer13       ,
          eMidLayer14       ,
          eMidLayer15       ,
          eMidLayer16       ,
          eMidLayer17       ,
          eMidLayer18       ,
          eMidLayer19       ,
          eMidLayer20       ,
          eMidLayer21       ,
          eMidLayer22       ,
          eMidLayer23       ,
          eMidLayer24       ,
          eMidLayer25       ,
          eMidLayer26       ,
          eMidLayer27       ,
          eMidLayer28       ,
          eMidLayer29       ,
          eMidLayer30       ,
          eBottomLayer      ,
          eTopOverlay       ,
          eBottomOverlay    ,
```

```
eTopPaste          ,
eBottomPaste       ,
eTopSolder         ,
eBottomSolder      ,
eInternalPlane1    ,
eInternalPlane2    ,
eInternalPlane3    ,
eInternalPlane4    ,
eInternalPlane5    ,
eInternalPlane6    ,
eInternalPlane7    ,
eInternalPlane8    ,
eInternalPlane9    ,
eInternalPlane10   ,
eInternalPlane11   ,
eInternalPlane12   ,
eInternalPlane13   ,
eInternalPlane14   ,
eInternalPlane15   ,
eInternalPlane16   ,
eDrillGuide        ,
eKeepOutLayer      ,
eMechanical1       ,
eMechanical2       ,
eMechanical3       ,
eMechanical4       ,
eMechanical5       ,
eMechanical6       ,
eMechanical7       ,
eMechanical8       ,
eMechanical9       ,
eMechanical10      ,
eMechanical11      ,
eMechanical12      ,
eMechanical13      ,
eMechanical14      ,
```

```
        eMechanical15     ,
        eMechanical16     ,
        eDrillDrawing     ,
        eMultiLayer       ,
        eConnectLayer     ,
        eBackGroundLayer  ,
        eDRCErrorLayer    ,
        eHighlightLayer   ,
        eGridColor1       ,
        eGridColor10      ,
        ePadHoleLayer     ,
        eViaHoleLayer
      );
```

TLayerSet

```
TLayerSet = Set of TLayer;
```

See also

TLayer

TLayerStackStyle

```
TLayerStackStyle     = ( eLayerStack_Pairs        ,
                         eLayerStacks_InsidePairs,
                         eLayerStackBuildup);
```

TLengthenerStyle

```
TLengthenerStyle     = ( eLengthenerStyle_90                        ,
                         eLengthenerStyle_45                        ,
                         eLengthenerStyle_Round
                       );
```

TLogicalDrawingMode

```
TLogicalDrawingMode  = ( eDisplaySolid                              ,
                         eDisplayHollow                             ,
                         eDisplaySelected                           ,
                         eDisplayDRC                                ,
                         eDisplayFocused                            ,
                         eDisplayMultiFocused
```

```
                    );
```

TMechanicalLayerPair

```
TMechanicalLayerPair         = Record
    Layer1                   : TLayer;
    Layer2                   : TLayer;
End;
```

TMirrorOperation

```
TMirrorOperation = (eHMirror,eVMirror);
```

TNetTopology

```
TNetTopology         = ( eNetTopology_Shortest                  ,
                         eNetTopology_Horizontal                ,
                         eNetTopology_Vertical                  ,
                         eNetTopology_DaisyChain_Simple         ,
                         eNetTopology_DaisyChain_MidDriven      ,
                         eNetTopology_DaisyChain_Balanced       ,
                         eNetTopology_Starburst
                       );
```

TNetScope

```
TNetScope            = ( eNetScope_DifferentNetsOnly            ,
                         eNetScope_SameNetOnly                  ,
                         eNetScope_AnyNet
                       );
```

TObjectId

```
TObjectId = ( eNoObject           ,
              eArcObject          ,
              ePadObject          ,
              eViaObject          ,
              eTrackObject        ,
              eTextObject         ,
              eFillObject         ,
              eConnectionObject   ,
              eNetObject          ,
              eComponentObject    ,
```

```
            ePolyObject        ,
            eRegionObject      ,
            eDimensionObject   ,
            eCoordinateObject  ,
            eClassObject       ,
            eRuleObject        ,
            eFromToObject      ,
            eViolationObject   ,
            eEmbeddedObject    ,
            eEmbeddedBoardObject,
            eTraceObject       ,
            eSpareViaObject    ,
            eBoardObject       ,
            eBoardOutlineObject
          );
```

TObjectCreationMode

```
TObjectCreationMode  = ( eCreate_Default,
                         eCreate_GlobalCopy
                       );
```

TObjectSet

```
TObjectSet = Set of TObjectId;
```

See also

TObjectId

TOptionsObjectId

```
 TOptionsObjectId    = ( eAbstractOptions                    ,
                         eOutputOptions                      ,
                         ePrinterOptions                     ,
                         eGerberOptions                      ,
                         eAdvancedPlacerOptions              ,
                         eDesignRuleCheckerOptions           ,
                         eSpecctraRouterOptions              ,
                         eAdvancedRouterOptions              ,
                         eEngineeringChangeOrderOptions      ,
                         eInteractiveRoutingOptions          ,
```

```
                       eSystemOptions                          ,
                       ePinSwapOptions
                      );
```

TOutputDriverType

```
TOutputDriverType    = ( eUnknownDriver                        ,
                         eProtelGerber                         ,
                         eProtelPlot_Composite                 ,
                         eProtelPlot_Final                     ,
                         eStandardDriver_Composite             ,
                         eStandardDriver_Final
                       );
```

TOutputPort

```
TOutputPort          = ( eOutputPortCom1                       ,
                         eOutputPortCom2                       ,
                         eOutputPortCom3                       ,
                         eOutputPortCom4                       ,
                         eOutputPortFile
                       );
```

TPadCache

```
TPadCache                            = Record
    PlaneConnectionStyle             : TPlaneConnectionStyle;
    ReliefConductorWidth             : TCoord;
    ReliefEntries                    : SmallInt;
    ReliefAirGap                     : TCoord;
    PowerPlaneReliefExpansion        : TCoord;
    PowerPlaneClearance              : TCoord;
    PasteMaskExpansion               : TCoord;
    SolderMaskExpansion              : TCoord;
    Planes                           : Word;
    PlaneConnectionStyleValid        : TCacheState;
    ReliefConductorWidthValid        : TCacheState;
    ReliefEntriesValid               : TCacheState;
    ReliefAirGapValid                : TCacheState;
    PowerPlaneReliefExpansionValid   : TCacheState;
```

```
    PasteMaskExpansionValid        : TCacheState;
    SolderMaskExpansionValid       : TCacheState;
    PowerPlaneClearanceValid       : TCacheState;
    PlanesValid                    : TCacheState;
End;
```

TPadMode

```
TPadMode = (ePadMode_Simple,
            ePadMode_LocalStack,
            ePadMode_ExternalStack);
```

TParity

```
TParity                 = ( eParityNone                                ,
                            eParityEven                                ,
                            eParityOdd                                 ,
                            eParityMark                                ,
                            eParitySpace
                          );
```

TPCBDragMode

```
TPcbDragMode            = ( eDragNone
                            eDragAllTracks
                            eDragConnectedTracks
                          );
```

TPCBObjectHandle

```
TPCBObjectHandle = Pointer;
```

TPCBString

```
TPCBString  = WideString;
```

TPlaceTrackMode

```
TPlaceTrackMode         = ( ePlaceTrackNone                            ,
                            ePlaceTrackAny                             ,
                            ePlaceTrack9090                            ,
                            ePlaceTrack4590                            ,
                            ePlaceTrack90Arc
                          );
```

TPlaneConnectStyle

```
TPlaneConnectStyle   = ( eReliefConnectToPlane,
                         eDirectConnectToPlane,
                         eNoConnect);
```

TPlaneConnectionStyle

```
TPlaneConnectionStyle = ( ePlaneNoConnect, ePlaneReliefConnect,
ePlaneDirectConnect);
```

TPlaneDrawMode

```
TPlaneDrawMode       = ( ePlaneDrawoOutlineLayerColoured // <- Protel 99 SE
style.
                         ePlaneDrawOutlineNetColoured,
                         ePlaneDrawInvertedNetColoured);
```

TPlotLayer

```
TPlotLayer           = ( eNullPlot                                  ,
                         eTopLayerPlot                              ,
                         eMidLayer1Plot                             ,
                         eMidLayer2Plot                             ,
                         eMidLayer3Plot                             ,
                         eMidLayer4Plot                             ,
                         eMidLayer5Plot                             ,
                         eMidLayer6Plot                             ,
                         eMidLayer7Plot                             ,
                         eMidLayer8Plot                             ,
                         eMidLayer9Plot                             ,
                         eMidLayer10Plot                            ,
                         eMidLayer11Plot                            ,
                         eMidLayer12Plot                            ,
                         eMidLayer13Plot                            ,
                         eMidLayer14Plot                            ,
                         eMidLayer15Plot                            ,
                         eMidLayer16Plot                            ,
                         eMidLayer17Plot                            ,
                         eMidLayer18Plot                            ,
```

```
eMidLayer19Plot                          ,
eMidLayer20Plot                          ,
eMidLayer21Plot                          ,
eMidLayer22Plot                          ,
eMidLayer23Plot                          ,
eMidLayer24Plot                          ,
eMidLayer25Plot                          ,
eMidLayer26Plot                          ,
eMidLayer27Plot                          ,
eMidLayer28Plot                          ,
eMidLayer29Plot                          ,
eMidLayer30Plot                          ,
eBottomLayerPlot                         ,
eTopOverlayPlot                          ,
eBottomOverlayPlot                       ,
eTopPastePlot                            ,
eBottomPastePlot                         ,
eTopSolderPlot                           ,
eBottomSolderPlot                        ,
eInternalPlane1Plot                      ,
eInternalPlane2Plot                      ,
eInternalPlane3Plot                      ,
eInternalPlane4Plot                      ,
eInternalPlane5Plot                      ,
eInternalPlane6Plot                      ,
eInternalPlane7Plot                      ,
eInternalPlane8Plot                      ,
eInternalPlane9Plot                      ,
eInternalPlane10Plot                     ,
eInternalPlane11Plot                     ,
eInternalPlane12Plot                     ,
eInternalPlane13Plot                     ,
eInternalPlane14Plot                     ,
eInternalPlane15Plot                     ,
eInternalPlane16Plot                     ,
eDrillGuide_Top_BottomPlot               ,
```

```
eDrillGuide_Top_Mid1Plot                ,
eDrillGuide_Mid2_Mid3Plot               ,
eDrillGuide_Mid4_Mid5Plot               ,
eDrillGuide_Mid6_Mid7Plot               ,
eDrillGuide_Mid8_Mid9Plot               ,
eDrillGuide_Mid10_Mid11Plot             ,
eDrillGuide_Mid12_Mid13Plot             ,
eDrillGuide_Mid14_Mid15Plot             ,
eDrillGuide_Mid16_Mid17Plot             ,
eDrillGuide_Mid18_Mid19Plot             ,
eDrillGuide_Mid20_Mid21Plot             ,
eDrillGuide_Mid22_Mid23Plot             ,
eDrillGuide_Mid24_Mid25Plot             ,
eDrillGuide_Mid26_Mid27Plot             ,
eDrillGuide_Mid28_Mid29Plot             ,
eDrillGuide_Mid30_BottomPlot            ,
eDrillGuide_SpecialPlot                 ,
eKeepOutLayerPlot                       ,
eMechanical1Plot                        ,
eMechanical2Plot                        ,
eMechanical3Plot                        ,
eMechanical4Plot                        ,
eMechanical5Plot                        ,
eMechanical6Plot                        ,
eMechanical7Plot                        ,
eMechanical8Plot                        ,
eMechanical9Plot                        ,
eMechanical10Plot                       ,
eMechanical11Plot                       ,
eMechanical12Plot                       ,
eMechanical13Plot                       ,
eMechanical14Plot                       ,
eMechanical15Plot                       ,
eMechanical16Plot                       ,
eDrillDrawing_Top_BottomPlot            ,
eDrillDrawing_Top_Mid1Plot              ,
```

```
                         eDrillDrawing_Mid2_Mid3Plot                ,
                         eDrillDrawing_Mid4_Mid5Plot                ,
                         eDrillDrawing_Mid6_Mid7Plot                ,
                         eDrillDrawing_Mid8_Mid9Plot                ,
                         eDrillDrawing_Mid10_Mid11Plot              ,
                         eDrillDrawing_Mid12_Mid13Plot              ,
                         eDrillDrawing_Mid14_Mid15Plot              ,
                         eDrillDrawing_Mid16_Mid17Plot              ,
                         eDrillDrawing_Mid18_Mid19Plot              ,
                         eDrillDrawing_Mid20_Mid21Plot              ,
                         eDrillDrawing_Mid22_Mid23Plot              ,
                         eDrillDrawing_Mid24_Mid25Plot              ,
                         eDrillDrawing_Mid26_Mid27Plot              ,
                         eDrillDrawing_Mid28_Mid29Plot              ,
                         eDrillDrawing_Mid30_BottomPlot             ,
                         eDrillDrawing_SpecialPlot                  ,
                         eTopPadMasterPlot                          ,
                         eBottomPadMasterPlot
                       );
```

TPlotterLanguage

```
TPlotterLanguage     = ( ePlotterLanguageHPGL,
                         ePlotterLanguageDMPL);
```

TPolyHatchStyle

```
TPolyHatchStyle      = ( ePolyHatch90,
                         ePolyHatch45,
                         ePolyVHatch,
                         ePolyHHatch,
                         ePolyNoHatch,
                         ePolySolid);
```

TPolygonReliefAngle

```
 TPolygonReliefAngle   = ( ePolygonReliefAngle_45,
                           ePolygonReliefAngle_90
                         );
```

TPolygonRepourMode

```
TPolygonRepourMode    = ( eNeverRepour
                          eThresholdRepour
                          eAlwaysRepour
                        );
```

TPolygonType

```
TPolygonType          = ( eSignalLayerPolygon,
                          eSplitPlanePolygon);
```

TPolySegmentType

```
TPolySegmentType      = ( ePolySegmentLine
                          ePolySegmentArc
                        );
```

TPrinterBatch

```
TPrinterBatch         = ( ePlotPerSheet,
                          ePanelize);
```

TPrinterComposite

```
TPrinterComposite     = ( eColorComposite,
                          eMonoComposite);
```

TRegionKind

```
TRegionKind = ( eRegionKind_Copper,
                eRegionKind_Cutout,
                eRegionKind_NamedRegion);
```

TRouteLayer

```
    TRouteLayer           = ( eRLLayerNotUsed                             ,
                              eRLRouteHorizontal                          ,
                              eRLRouteVertical                            ,
                              eRLRouteSingleLayer                         ,
                              eRLRoute_1_OClock                           ,
                              eRLRoute_2_OClock                           ,
                              eRLRoute_4_OClock                           ,
                              eRLRoute_5_OClock                           ,
                              eRLRoute_45_Up                              ,
```

```
        eRLRoute_45_Down                    ,
        eRLRouteFanout                      ,
        eRLRouteAuto
       );
```

TRouteVia

```
TRouteVia           = ( eViaThruHole                ,
                        eViaBlindBuriedPair         ,
                        eViaBlindBuriedAny          ,
                        eViaNone
                      );
```

TRuleKind

```
TRuleKind           = ( eRule_Clearance             ,
                        eRule_ParallelSegment       ,
                        eRule_MaxMinWidth           ,
                        eRule_MaxMinLength          ,
                        eRule_MatchedLengths        ,
                        eRule_DaisyChainStubLength  ,
                        eRule_PowerPlaneConnectStyle ,
                        eRule_RoutingTopology       ,
                        eRule_RoutingPriority       ,
                        eRule_RoutingLayers         ,
                        eRule_RoutingCornerStyle    ,
                        eRule_RoutingViaStyle       ,
                        eRule_PowerPlaneClearance   ,
                        eRule_SolderMaskExpansion   ,
                        eRule_PasteMaskExpansion    ,
                        eRule_ShortCircuit          ,
                        eRule_BrokenNets            ,
                        eRule_ViasUnderSMD          ,
                        eRule_MaximumViaCount       ,
                        eRule_MinimumAnnularRing    ,
                        eRule_PolygonConnectStyle   ,
                        eRule_AcuteAngle            ,
                        eRule_ConfinementConstraint ,
                        eRule_SMDToCorner           ,
```

```
                    eRule_ComponentClearance                   ,
                    eRule_ComponentRotations                   ,
                    eRule_PermittedLayers                      ,
                    eRule_NetsToIgnore                         ,
                    eRule_SignalStimulus                       ,
                    eRule_Overshoot_FallingEdge                ,
                    eRule_Overshoot_RisingEdge                 ,
                    eRule_Undershoot_FallingEdge               ,
                    eRule_Undershoot_RisingEdge                ,
                    eRule_MaxMinImpedance                      ,
                    eRule_SignalTopValue                       ,
                    eRule_SignalBaseValue                      ,
                    eRule_FlightTime_RisingEdge                ,
                    eRule_FlightTime_FallingEdge               ,
                    eRule_LayerStack                           ,
                    eRule_MaxSlope_RisingEdge                  ,
                    eRule_MaxSlope_FallingEdge                 ,
                    eRule_SupplyNets                           ,
                    eRule_MaxMinHoleSize                       ,
                    eRule_TestPointStyle                       ,
                    eRule_TestPointUsage                       ,
                    eRule_UnconnectedPin                       ,
                    eRule_SMDToPlane                           ,
                    eRule_SMDNeckDown                          ,
                    eRule_LayerPair                            ,
                    eRule_FanoutControl                        ,
                    eRule_MaxMinHeight
                  );
```

TRuleLayerKind

```
TRuleLayerKind    = ( eRuleLayerKind_SameLayer                 ,
                      eRuleLayerKind_AdjacentLayer
                    );
```

TSameNamePadstackReplacementMode

```
TSameNamePadstackReplacementMode
                  = ( eAskUser
```

```
                        eReplaceOne
                        eReplaceAll
                        eRenameOne
                        eRenameAll
                        eKeepOneExisting
                        eKeepAllExisting
                      );
```

TScopeId

```
ScopeId            = ( eScope1                                ,
                       eScope2
                     );
```

TScopeObjectId

```
TScopeObjectId      = ( eRuleObject_None                      ,
                        eRuleObject_Wire                      ,
                        eRuleObject_Pin                       ,
                        eRuleObject_Smd                       ,
                        eRuleObject_Via                       ,
                        eRuleObject_Fill                      ,
                        eRuleObject_Polygon                   ,
                        eRuleObject_KeepOut
                      );
```

TScopeKind

```
TScopeKind          = ( eScopeKindBoard                       ,
           {Lowest Precedence}
                        eScopeKindLayerClass                  ,
                        eScopeKindLayer                       ,
                        eScopeKindObjectKind                  ,
                        eScopeKindFootprint                   ,
                        eScopeKindComponentClass              ,
                        eScopeKindComponent                   ,
                        eScopeKindNetClass                    ,
                        eScopeKindNet                         ,
                        eScopeKindFromToClass                 ,
                        eScopeKindFromTo                      ,
```

```
                    eScopeKindPadClass                          ,
                    eScopeKindPadSpec                           ,
                    eScopeKindViaSpec                           ,
                    eScopeKindFootprintPad                      ,
                    eScopeKindPad                               ,
                    eScopeKindRegion
                                  {Highest Precedence}
                  );
```

TShape

```
TShape = (eNoShape,
          eRounded,
          eRectangular,
          eOctagonal,
          eCircleShape,
          eArcShape,
          eTerminator,
          eRoundRectShape,
          eRotatedRectShape);
```

TSignalLevel

```
TSignalLevel = ( eLowLevel,
                 eHighLevel);
```

TSortBy

```
TSortBy             = ( eSortByAXThenAY                        ,
                        eSortByAXThenDY                        ,
                        eSortByAYThenAX                        ,
                        eSortByDYThenAX                        ,
                        eSortByName
                      );
```

TStimulusType

```
TStimulusType       = ( eConstantLevel                         ,
                        eSinglePulse                           ,
                       ePeriodicPulse);
```

TStopBits

```
TStopBits            = ( eStopBits1                           ,
                         eStopBits1_5                         ,
                         eDataBits2
                       );
```

TString (PCB)

```
TString = ShortString;
```

TTestpointAllowedSide

```
TTestpointAllowedSide
                     = ( eAllowTopSide                        ,
                         eAllowBottomSide                     ,
                         eAllowThruHoleTop                    ,
                         eAllowThruHoleBottom
                       );
```

TTestPointStyle

```
TTestPointStyle      = ( eExistingSMDBottom                   ,
                         eExistingTHPadBottom                 ,
                         eExistingTHViaBottom                 ,
                         eNewSMDBottom                        ,
                         eNewTHBottom                         ,
                         eExistingSMDTop                      ,
                         eExistingTHPadTop                    ,
                         eExistingTHViaTop                    ,
                         eNewSMDTop                           ,
                         eNewTHTop
                       );
```

TTestpointValid

```
TTestpointValid      = ( eRequire                             ,
                         eInvalid                             ,
                         eIgnore
                       );
```

TTextAlignment

```
TTextAlignment        = ( eNoneAlign                                      ,
                          eCentreAlign                                    ,
                          eLeftAlign                                      ,
                          eRightAlign                                     ,
                          eTopAlign                                       ,
                          eBottomAlign
                        );
```

TTextAutoposition

```
TTextAutoposition     = ( eAutoPos_Manual                                 ,
                          eAutoPos_TopLeft                                ,
                          eAutoPos_CenterLeft                             ,
                          eAutoPos_BottomLeft                             ,
                          eAutoPos_TopCenter                              ,
                          eAutoPos_CenterCenter                           ,
                          eAutoPos_BottomCenter                           ,
                          eAutoPos_TopRight                               ,
                          eAutoPos_CenterRight                            ,
                          eAutoPos_BottomRight
                        );
```

TUnitStyle

```
TUnitStyle = ( eNoUnits,
               eYesUnits,
               eParenthUnits);
```

TUnit

```
TUnit = (eMetric, eImperial);
```

TViewableObjectID

```
TViewableObjectID     = ( eViewableObject_None                            ,
                          eViewableObject_Arc                             ,
                          eViewableObject_Pad                             ,
                          eViewableObject_Via                             ,
                          eViewableObject_Track                           ,
                          eViewableObject_Text                            ,
```

```
eViewableObject_Fill                        ,
eViewableObject_Connection                  ,
eViewableObject_Net                         ,
eViewableObject_Component                   ,
eViewableObject_Poly                        ,
eViewableObject_LinearDimension             ,
eViewableObject_AngularDimension            ,
eViewableObject_RadialDimension             ,
eViewableObject_LeaderDimension             ,
eViewableObject_DatumDimension              ,
eViewableObject_BaselineDimension           ,
eViewableObject_CenterDimension             ,
eViewableObject_OriginalDimension           ,
eViewableObject_LinearDiameterDimension     ,
eViewableObject_RadialDiameterDimension     ,
eViewableObject_Coordinate                  ,
eViewableObject_Class                       ,
eViewableObject_Rule_Clearance              ,
eViewableObject_Rule_ParallelSegment        ,
eViewableObject_Rule_MaxMinWidth            ,
eViewableObject_Rule_MaxMinLength           ,
eViewableObject_Rule_MatchedLengths         ,
eViewableObject_Rule_DaisyChainStubLength   ,
eViewableObject_Rule_PowerPlaneConnectStyle,
eViewableObject_Rule_RoutingTopology        ,
eViewableObject_Rule_RoutingPriority        ,
eViewableObject_Rule_RoutingLayers          ,
eViewableObject_Rule_RoutingCornerStyle     ,
eViewableObject_Rule_RoutingViaStyle        ,
eViewableObject_Rule_PowerPlaneClearance    ,
eViewableObject_Rule_SolderMaskExpansion    ,
eViewableObject_Rule_PasteMaskExpansion     ,
eViewableObject_Rule_ShortCircuit           ,
eViewableObject_Rule_BrokenNets             ,
eViewableObject_Rule_ViasUnderSMD           ,
eViewableObject_Rule_MaximumViaCount        ,
```

```
eViewableObject_Rule_MinimumAnnularRing    ,
eViewableObject_Rule_PolygonConnectStyle   ,
eViewableObject_Rule_AcuteAngle            ,
eViewableObject_Rule_ConfinementConstraint ,
eViewableObject_Rule_SMDToCorner           ,
eViewableObject_Rule_ComponentClearance    ,
eViewableObject_Rule_ComponentRotations    ,
eViewableObject_Rule_PermittedLayers       ,
eViewableObject_Rule_NetsToIgnore          ,
eViewableObject_Rule_SignalStimulus        ,
eViewableObject_Rule_Overshoot_FallingEdge ,
eViewableObject_Rule_Overshoot_RisingEdge  ,
eViewableObject_Rule_Undershoot_FallingEdge,
eViewableObject_Rule_Undershoot_RisingEdge ,
eViewableObject_Rule_MaxMinImpedance       ,
eViewableObject_Rule_SignalTopValue        ,
eViewableObject_Rule_SignalBaseValue       ,
eViewableObject_Rule_FlightTime_RisingEdge ,
eViewableObject_Rule_FlightTime_FallingEdge,
eViewableObject_Rule_LayerStack            ,
eViewableObject_Rule_MaxSlope_RisingEdge   ,
eViewableObject_Rule_MaxSlope_FallingEdge  ,
eViewableObject_Rule_SupplyNets            ,
eViewableObject_Rule_MaxMinHoleSize        ,
eViewableObject_Rule_TestPointStyle        ,
eViewableObject_Rule_TestPointUsage        ,
eViewableObject_Rule_UnconnectedPin        ,
eViewableObject_Rule_SMDToPlane            ,
eViewableObject_Rule_SMDNeckDown           ,
eViewableObject_Rule_LayerPair             ,
eViewableObject_Rule_FanoutControl         ,
eViewableObject_Rule_MaxMinHeight          ,
eViewableObject_FromTo                     ,
eViewableObject_Violation                  ,
eViewableObject_Board                      ,
eViewableObject_BoardOutline               ,
```

```
                        eViewableObject_Group                          ,
                        eViewableObject_Clipboard                      ,
                        eViewableObject_SplitPlane,
                        eViewableObject_EmbeddedBoard,
                        eViewableObject_Region);
```

PCB Constants

AllLayers

```
AllLayers = [MinLayer..eConnectLayer];
```

AllObjects

```
AllObjects = [FirstObjectId..LastObjectId];
```

AllPrimitives

```
AllPrimitives = [ eArcObject          ,
                  eViaObject          ,
                  eTrackObject        ,
                  eTextObject         ,
                  eFillObject         ,
                  ePadObject          ,
                  eComponentObject    ,
                  eNetObject          ,
                  ePolyObject         ,
                  eDimensionObject    ,
                  eCoordinateObject   ,
                  eEmbeddedObject     ,
                  eEmbeddedBoardObject,
                  eFromToObject       ,
                  eConnectionObject
                  eRegionObject];
```

cAdvPCB

```
cAdvPCB = 'AdvPCB';
```

cLayerStrings

```
cLayerStrings : Array[TLayer] Of Strin
              = ( 'NoLayer'        ,
```

```
'TopLayer'       ,
'MidLayer1'      ,
'MidLayer2'      ,
'MidLayer3'      ,
'MidLayer4'      ,
'MidLayer5'      ,
'MidLayer6'      ,
'MidLayer7'      ,
'MidLayer8'      ,
'MidLayer9'      ,
'MidLayer10'     ,
'MidLayer11'     ,
'MidLayer12'     ,
'MidLayer13'     ,
'MidLayer14'     ,
'MidLayer15'     ,
'MidLayer16'     ,
'MidLayer17'     ,
'MidLayer18'     ,
'MidLayer19'     ,
'MidLayer20'     ,
'MidLayer21'     ,
'MidLayer22'     ,
'MidLayer23'     ,
'MidLayer24'     ,
'MidLayer25'     ,
'MidLayer26'     ,
'MidLayer27'     ,
'MidLayer28'     ,
'MidLayer29'     ,
'MidLayer30'     ,
'BottomLayer'    ,
'TopOverlay'     ,
'BottomOverlay'  ,
'TopPaste'       ,
'BottomPaste'    ,
```

```
'TopSolder'       ,
'BottomSolder'    ,
'InternalPlane1'  ,
'InternalPlane2'  ,
'InternalPlane3'  ,
'InternalPlane4'  ,
'InternalPlane5'  ,
'InternalPlane6'  ,
'InternalPlane7'  ,
'InternalPlane8'  ,
'InternalPlane9'  ,
'InternalPlane10',
'InternalPlane11',
'InternalPlane12',
'InternalPlane13',
'InternalPlane14',
'InternalPlane15',
'InternalPlane16',
'DrillGuide'      ,
'KeepOutLayer'    ,
'Mechanical1'     ,
'Mechanical2'     ,
'Mechanical3'     ,
'Mechanical4'     ,
'Mechanical5'     ,
'Mechanical6'     ,
'Mechanical7'     ,
'Mechanical8'     ,
'Mechanical9'     ,
'Mechanical10'    ,
'Mechanical11'    ,
'Mechanical12'    ,
'Mechanical13'    ,
'Mechanical14'    ,
'Mechanical15'    ,
'Mechanical16'    ,
```

```
               'DrillDrawing'   ,
               'MultiLayer'     ,
               'ConnectLayer'   ,
               'BackGroundLayer',
               'DRCErrorLayer'  ,
               'HighlightLayer' ,
               'GridColor1'     ,
               'GridColor10'    ,
               'PadHoleLayer'   ,
               'ViaHoleLayer');
```

cMaxTestPointStyle

```
cMaxTestPointStyle             = eNewTHTop;
```

cMinTestPointStyle

```
cMinTestPointStyle = eExistingSMDBottom;
```

cMidLayers

```
cMidLayers        : Set Of TLayer = [eMidLayer1 .. eMidLayer30];
```

cRuleIdStrings

```
cRuleIdStrings : Array [TRuleKind] Of String[21]
               = ( 'Clearance'            ,
                   'ParallelSegment'      ,
                   'Width'                ,
                   'Length'               ,
                   'MatchedLengths'       ,
                   'StubLength'           ,
                   'PlaneConnect'         ,
                   'RoutingTopology'      ,
                   'RoutingPriority'      ,
                   'RoutingLayers'        ,
                   'RoutingCorners'       ,
                   'RoutingVias'          ,
                   'PlaneClearance'       ,
                   'SolderMaskExpansion'  ,
                   'PasteMaskExpansion'   ,
```

```
'ShortCircuit'          ,
'UnRoutedNet'           ,
'ViasUnderSMD'          ,
'MaximumViaCount'       ,
'MinimumAnnularRing'    ,
'PolygonConnect'        ,
'AcuteAngle'            ,
'RoomDefinition'        ,
'SMDToCorner'           ,
'ComponentClearance'    ,
'ComponentOrientations',
'PermittedLayers'       ,
'NetsToIgnore'          ,
'SignalStimulus'        ,
'OvershootFalling'      ,
'OvershootRising'       ,
'UndershootFalling'     ,
'UndershootRising'      ,
'MaxMinImpedance'       ,
'SignalTopValue'        ,
'SignalBaseValue'       ,
'FlightTimeRising'      ,
'FlightTimeFalling'     ,
'LayerStack'            ,
'SlopeRising'           ,
'SlopeFalling'          ,
'SupplyNets'            ,
'HoleSize'              ,
'Testpoint'             ,
'TestPointUsage'        ,
'UnConnectedPin'        ,
'SMDToPlane'            ,
'SMDNeckDown'           ,
'LayerPairs'            ,
'FanoutControl'         ,
'Height');
```

cTextAutopositionStrings

```
cTextAutopositionStrings : Array[TTextAutoPosition] Of String[20]
                         = ( 'Manual'      ,
                             'Left-Above'  ,
                             'Left-Center' ,
                             'Left-Below'  ,
                             'Center-Above',
                             'Center'      ,
                             'Center-Below',
                             'Right-Above' ,
                             'Right-Center',
                             'Right-Below');
```

cTestPointPriorityHigh

```
cTestPointPriorityHigh         = Ord(cMinTestPointStyle);
```

cTestPointPriorityLow

```
cTestPointPriorityLow          = Ord(cMaxTestPointStyle);
```

FirstObjectId

```
FirstObjectId = eArcObject;
```

InternalUnits

```
InternalUnits = 10000;
```

InternalPlanes

```
InternalPlanes   : Set Of TLayer = [eInternalPlane1..eInternalPlane16];
```

k1Mil

```
k1Mil = 1 * InternalUnits;
```

Notes

- 1 mil = 10000 internal units
- 1 inch = 1000 mils
- 1 inch = 2.54 cm
- 1 inch = 25.4 mm and 1 cm = 10 mm

PCB object's coordinates are usually in mils or mm depending on the board's current measurement units.

kMaxCoord

```
kMaxCoord = 99999  * InternalUnits;
```

kMinCoord

```
kMinCoord = 0 * InternalUnits;
```

kMaxInternalPlane

```
kMaxInternalPlane = eInternalPlane16;
```

kMinInternalPlane

```
kMinInternalPlane = eInternalPlane1;
```

kMaxPolySize

```
kMaxPolySize = 5000;
```

LastObjectId

```
LastObjectId = eEmbeddedBoardObject;
```

MaxLayer

```
MaxLayer = eViaHoleLayer;
```

Notes

Refer to Layer2String and String2Layer functions in the PCB Functions topic.

MaxBoardLayer

```
MaxBoardLayer = eMultiLayer;
```

MaxRouteLayer

```
MaxRouteLayer = eBottomLayer;
```

MechanicalLayers

```
MechanicalLayers : Set Of TLayer = [eMechanical1..eMechanical16];
```

MinLayer

```
MinLayer = eTopLayer;
```

Notes

Refer to Layer2String and String2Layer functions in the PCB Functions topic.

Numbers

```
Numbers             : Set Of Char    = ['0'..'9'];
```

PCB Messages

Overview

The PCB Messages are messages that are broadcasted by the PCB Editor server. There are different types of messages that describe a specific action within the PCB server.

Normally the PCB message constants are used for the **IPCB_ServerInterface.SendMessageToRobots** method.

Syntax

```
PCBM_NullMessage          = 0;
PCBM_BeginModify          = 1;
PCBM_BoardRegisteration   = 2;
PCBM_EndModify            = 3;
PCBM_CancelModify         = 4;
PCBM_Create               = 5;
PCBM_Destroy              = 6;
PCBM_ProcessStart         = 7;
PCBM_ProcessEnd           = 8;
PCBM_ProcessCancel        = 9;
PCBM_YieldToRobots        = 10;
PCBM_CycleEnd             = 11;
PCBM_CycleStart           = 12;
PCBM_SystemInvalid        = 13;
PCBM_SystemValid          = 14;
PCBM_ViewUpdate           = 15;
PCBM_UnDoRegister         = 16;

c_BroadCast    = Nil;
c_NoEventData = Nil;
c_FromSystem   = Nil;
```

See also

SendMessageToRobots method

SignalLayers

```
SignalLayers        : Set Of TLayer = [eTopLayer.. eBottomLayer];
```

PCB Functions

General functions

```
Function PCBServer : IPCB_ServerInterface;
Function GetStateString_PcbObjectId(N : TObjectId) : TString;
```

Unit conversion functions

```
Function  RealToMils     (C : TReal)  : TReal;
Function  RealToMMs      (C : TReal)  : TReal;
Function  CoordToMils    (C : TCoord) : TReal;
Function  CoordToMMs     (C : TCoord) : TReal;
Function  MilsToCoord    (M : TReal)  : TCoord;
Function  MMsToCoord     (M : TReal)  : TCoord;
Function  MilsToRealCoord(M : TReal)  : TReal;
Function  MMsToRealCoord (M : TReal)  : TReal;

Function  MetricString  (Var S        : TString;
                         DefaultUnits : TUnit)  : Boolean;
Function  ImperialString(Var S        : TString;
                         DefaultUnits : TUnit)  : Boolean;

Procedure StringToCoordUnit(S     : TString;
                            Var C : TCoord;
                            Var U : TUnit);

Procedure StringToRealUnit (S     : TString;
                            Var R : TReal;
                            Var U : TUnit);

Function  CoordUnitToString(C : TCoord;
                            U : TUnit) : TString;

Function  RealUnitToString (R : TReal;
                            U : TUnit) : TString;
```

Trigonometric functions

```
Function  Degrees2Radians             (Angle             : TAngle)        :
TReal;
```

```
Function  AngleToFormattedString       (TextValue          : TReal;
                                         TextFormat         : TString;
                                         TextDimensionUnit  : TDimensionUnit;
                                         TextPrecision      : Integer;
                                         TextPrefix         : TString;
                                         TextSuffix         : TString)       :
TString;

Function  DistanceToFormattedString    (TextValue          : TReal;
                                         TextFormat         : TString;
                                         TextDimensionUnit  : TDimensionUnit;
                                         TextPrecision      : Integer;
                                         TextPrefix         : TString;
                                         TextSuffix         : TString;
                                         DisplayUnit        : TUnit)         :
TString;
```

Layer conversion functions

```
Function  Layer2String (Layer : TLayer) : TString;
Function  String2Layer (Layer : TString): TLayer;
```

Schematic API Reference

The Schematic Application Programming Interface reference covers interfaces for schematic objects such as schematic documents and schematic design objects.

What are interfaces?

Each method in the interface is implemented in the corresponding class. Interfaces are declared like classes but cannot be directly instantiated and do not have their own method definitions. Each interface, a class supports is actually a list of pointers to methods. Therefore, each time a method call is made to an interface, the interface actually diverts that call to one of it's pointers to a method, thus giving the object that really implements it, the chance to act.

The Schematic interfaces exist as long there are associated existing objects in memory, thus when writing a script, you have the responsibility of checking whether the interface you wish to query exists or not before you proceed to invoke the interface's methods.

To obtain the Schematic interface that points to the Schematic editor object, invoke the **SchServer** function in your script which returns you the **ISch_ServerInterface** interface. This object interface obtains the Schematic editor server object and then you can extract data from Schematic objects and invoke Schematic object's methods.

For example

```
Var
    Sheet : ISch_Sheet;
Begin
    Sheet := SchServer.GetCurrentSchDocument
    If Sheet = Nil then Exit;

    // do something here
End;
```

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** interface 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 DXP.

Script Examples

There are Schematic script examples in the **\Examples\Scripts\SCH** folder which demonstrate the use of Schematic interfaces.

See also

Schematic Interfaces Overview

Schematic Interfaces

Schematic Design Objects

Schematic Enumerated Types

Schematic Functions

Client API Reference

Integrated Library API Reference

Nexar API Reference

PCB API Reference

Work Space Manager API Reference

Server Process Reference

Using Schematic Interfaces

An interface is simply a list of methods that a class declares that it implements. That is, each method in the interface is implemented in the corresponding class. Interfaces are declared like classes but cannot be directly instantiated and do not have their own method definitions.

Each interface, a class supports is actually a list of pointers to methods. Therefore, each time a method call is made to an interface, the interface actually diverts that call to one of it's pointers to a method, thus giving the object that really implements it, the chance to act.

An interface is reference counted and whenever it goes out of scope its reference count is decremented. In this case the reference count is set at 1 then is decremented to 0 when it goes out of scope. When the reference count is 0, it means nothing else will be referring to it so the object associated with the interface will be released. Sometimes the reference counting is disabled in certain DXP interfaces, for example, the **IProject** interface which encapsulates the Projects panel in DXP does not have a reference counting mechanism.

The Schematic interfaces exist as long there are associated existing objects in memory, thus when writing code, you have the responsibility of checking whether the interface you wish to query exists or not before you proceed to invoke the interface's methods by checking that this interface doesn't have a **Nil** value or using the **Assigned** function such as If Not Assigned(fschDoc) Then statement.

To obtain the Schematic interface that points to the Schematic editor object, invoke the **SchServer** function in your code which returns you the **ISch_ServerInterface** interface. This object interface obtains the Schematic editor server object and then you can extract data from Schematic objects and invoke Schematic object's methods.

Getting the schematic sheet interface.

```
Var
    Sheet : ISch_Sheet;
Begin
    Sheet := SchServer.GetCurrentSchDocument
```

```
    If Sheet = Nil then Exit;
    // do something here
End;
```

Placing a schematic port example

```
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;
    // the port is placed at 500,500 mils respectively.
    SchPort.Location  := Point(MilsToCoord(500),MilsToCoord(500));
    SchPort.Style     := ePortRight;
    SchPort.IOType    := ePortBidirectional;
    SchPort.Alignment := eHorizontalCentreAlign;
    SchPort.Width     := MilsToCoord(1000);
    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;
End;
```

See also

There are Schematic script examples in the **\Examples\Scripts\DelphiScript Scripts\Sch** folder which demonstrate the use of Schematic interfaces.

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 DXP.

- The **ISch_Lib** interface represents the schematic library document.
- The **ISch_Sheet** interface represents the schematic sheet document.

The Schematic Design Objects hierarchy is composed of

- ISch_BasicContainer and ISch_GraphicalObject ancestor interfaces
- ISch_Arc and its descendants
  - ISch_Pie, ISch_EllipticalArc
- ISch_Line and its descendants
  - ISch_BusEntry, ISch_ConnectionLine
- ISch_Label and descendants
  - ISch_PowerObject, ISch_ComplexText, ISch_Designator, ISch_NetLabel, ISch_Parameter, ISch_SheetFileName
- ISch_ParameterizedGroup and its descendants
  - ISch_Port, ISch_Pin, ISch_Component, ISch_RectangularGroup, ISch_SheetSymbol, ISch_ParameterSet, ISch_Probe
- ISch_Polygon and its descendants
  - ISch_Polyline, ISch_Bezier, ISch_Wire, ISch_Bus
- ISch_Rectangle and its descendants
  - ISch_Image, ISch_RoundRectangle, ISch_TextFrame, ISch_CompileMask

When you need to deal with Schematic design objects in DXP, 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. From the Server Interface, you can get the document and the design objects on it.

SchServer function example

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

```
If CurrentSheet = Nil Then Exit;

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

Notes

The **Client** function returns the **IClient** interface and from this interface you can obtain the **IServerModule** to retrieve information about its associated document views and panel views for a specified server.

```
Var
    ServerModule : IServerModule;
Begin
    If Client = Nil Then Exit;
    ServerModule := Client.ServerModuleByName('SCH');
    ShowMessage('Doc Count = ' + IntToStr(ServerModule.DocumentCount));
End;
```

See also

Client and Server interfaces

IClient interface

IServerModule interface

ISch_GraphicalObject interface

ISch_Document interface

ISch_ServerInterface interface

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
```

Base Interface and its methods/properties

The **ISch_GraphicalObject** is the base interface for all descendant Schematic design object interfaces such as **ISch_Arc** and I**Sch_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 visible in the **ISch_Arc** declaration but you will notice that the **ISch_Arc** interface is inherited from the **ISch_GraphicalObject** ancestor interface and this interface has a **Selection** property along with its associated methods (GetState function and SetState procedure for example). Therefore the **Selection** property is available in the **ISch_Arc** interface.

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.

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 DXP

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, see Client API online reference for details.

Opening a text document, you pass in the ‘Text’ string along with the full file name string in the **OpenDocument** method. 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.

Opening a schematic document using Client.OpenDocument method

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

Creating Schematic or Library documents in DXP

There are situations when you need to programmatically create a blank stand-alone document. This is facilitated by using the **CreateNewDocumentFromDocumentKind** function. For example, creating a schematic document, you pass in the 'SCH' string.

CreateNewDocumentFromDocumentKind example

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

Create a blank schematic and add to the current project

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 WorkSpace Manager in DXP and invoke DM_FocusedProject and DM_AddSourceDocument functions.

Adding a document to a project

```
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('SCH', Path);
         Client.ShowDocument(Doc);
    End;
```

```
    SchDoc := SchServer.GetCurrentSchDocument;
    If SchDoc = Nil Then Exit;
    // do what you want with the schematic document!
End;
```

Checking the type of schematic documents in DXP

This code snippet checks if a document is a Schematic library format using the **CurrentView.OwnerDocument.Kind** function.

```
    If UpperCase(Client.CurrentView.OwnerDocument.Kind) <> 'SCHLIB' Then
Exit;
```

Setting a document dirty

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

Setting a document dirty example

```
Var
    AView           : IServerDocumentView;
    AServerDocument : IServerDocument;
Begin
    // grab the current document view using the Client module'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;
```

Few methods to refresh a schematic document

When you place or modify objects on a schematic document, you often need to do a refresh of the document. An example below demonstrates one way to update the document. You can use the ICommandLauncher.LaunchCommand method.

Commands.LaunchCommand example

```
Procedure RefreshSchematicDocument;
Var
    Commands        : ICommandLauncher;
```

```
    Parameters      : TChar;
    SchematicServer : IServerModule;
Begin
    Parameters := 'Action = Document';

    SchematicServer := SchServer;
    If SchematicServer <> Nil Then
    Begin
        Commands := SchematicServer.CommandLauncher;
        If Commands <> Nil Then
            Commands.LaunchCommand('Zoom', Parameters,
255,Client.CurrentView);
    End;
End;
```

Client.SendMessage example

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

The SchDoc.Invalidate 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;
End;
```

Using Schematic Measurement Units

The Schematic editor supports two measurement units, imperial (mils) and metric (mm). By default the design database base unit deals with mils (thousandths of an inch).

Internal to the Schematic design database, the coordinates of all Schematic objects are in Internal Units. The internal unit is 1/ 10 000 of a mil or 1 /10,000,000 inch. Note that 1 mil = 10,000 internal units.

Therefore the Schematic design objects' dimensions or coordinates are measured in Internal Unit coordinate values.

Notes

The internal units in prior versions ie DXP, P99SE and so on use an internal unit of 1 / 1000 of a mil. Therefore the units used in the DXP 2004 are the 10 times the size of the units used in the previous versions, ie more resolution in DXP 2004.

There are functions that convert from mils or mm values to an internal coordinate value. See an example that converts from Mils unit values to Coord values.

Example

```
SchPort.Location  := Point(MilsToCoord(500),MilsToCoord(500));
SchPort.Width     := MilsToCoord(1000);
```

Notes

- 1 mil = 10000 internal units
- 1 inch = 1000 mils
- 1 inch = 2.54 cm
- 1 inch = 25.4 mm and 1 cm = 10 mm

Converting Measurement Units

To convert from one measurement unit to another unit, use the following Schematic functions:

```
//Imperial units to Internal Coordinate values
Function  CoordToMils        (    C : TCoord) : TReal;
Function  CoordToDxps        (    C : TCoord) : TReal;
Function  CoordToInches      (    C : TCoord) : TReal;
Function  MilsToCoord        (    M : TReal)  : TCoord;
Function  DxpsToCoord        (    M : TReal)  : TCoord;
Function  InchesToCoord      (    M : TReal)  : TCoord;

//Metric To Internal Coordinate values
Function  CoordToMMs         (    C : TCoord) : TReal;
Function  CoordToCMs         (    C : TCoord) : TReal;
Function  CoordToMs          (    C : TCoord) : TReal;
Function  MMsToCoord         (    M : TReal)  : TCoord;
Function  CMsToCoord         (    M : TReal)  : TCoord;
Function   MsToCoord         (    M : TReal)  : TCoord;
```

```
Function  MetricString       (Var S : TDynamicString; DefaultUnits : TUnit)
: Boolean;
Function  ImperialString     (Var S : TDynamicString; DefaultUnits : TUnit)
: Boolean;


Function  ExtractValueAndUnitFromString(    AInString    : TDynamicString;
                                            ADefaultUnit : TUnit;
                                        Var AValue       : TDynamicString;
                                        Var AUnit        : TUnit) : Boolean;

Function  StringToCoordUnit       (S : TDynamicString; Var C : TCoord;
ADefaultUnit : TUnit) : Boolean;
Function  CoordUnitToString       (C : TCoord; U : TUnit) : TDynamicString;
Function  CoordUnitToStringFixedDecimals (C : TCoord; U : TUnit;
AFixedDecimals : Integer) : TDynamicString;
Function  CoordUnitToStringNoUnit (C : TCoord; U : TUnit) : TDynamicString;


Function  GetDisplayStringFromLocation(ALocation : TLocation; AUnit : TUnit)
: TDynamicString;

Function GetCurrentDocumentUnit : TUnit;
Function GetCurrentDocumentUnitSystem : TUnitSystem;
Function GetSchObjectOwnerDocumentUnit(Const AObject : ISch_BasicContainer)
: TUnit;
```

See also

TCoord type

Schematic functions

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 unqiue 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.

Creating 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.

```
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.RegisterSchObjectInContainer(SchPort);
    FSchDoc.GraphicallyInvalidate;
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

See also

See the PlaceAPort script from the \Examples\Scripts\Delphiscript\Sch\ 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 example is shown below.

```
Procedure CreateALibComponent;
Var
    CurrentLib   : ISch_Lib;
    SchComponent : ISch_Component;
    R            : ISch_Rectangle;
    Location     : TLocation;
    Corner       : TLocation;
Begin
```

```
    If SchServer = Nil Then Exit;

    CurrentLib := SchServer.GetCurrentSchDocument;
    If CurrentLib = Nil Then Exit;

    SchComponent := SchServer.SchObjectFactory(eSchComponent,
eCreate_Default);
    If SchComponent = Nil Then Exit;

    R := SchServer.SchObjectFactory(eRectangle, eCreate_Default);
    If R = Nil 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;
    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';
```

```
    // use of Server processes to refresh the screen.
    ResetParameters;
    AddStringParameter('Action', 'Document');
    RunProcess('Sch:Zoom');
End;
```

Looking for 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
- Group iterators,

Object iterators are used to conduct global searches, while 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.

Normally you will need an iterator to search for schematic objects. You can customize different iterators and you can specify which objects to look in the specified region of a document. You can also set up 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 (cut down example)

```
Var
    Pin         : ISch_Pin;
    PinIterator : ISch_Iterator;
    PinFound    : Boolean;
Begin
    PinFound := False;
    PinIterator := AComponent.SchIterator_Create;
    PinIterator.SetState_IterationDepth(eIterateAllLevels);
    PinIterator.AddFilter_ObjectSet(MkSet(ePin));
    Try
        Pin := PinIterator.FirstSchObject;
        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;
        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

CheckPins example in \Examples\Scripts\Delphiscript Scripts\Sch folder.

Creating/Deleting Schematic Objects and updating the Undo system

The simple creation of objects in the examples above does not refresh the Undo system in the Schematic Editor. To have the ability to undo objects created on a PCB document, you will need to employ the SchServer.RobotManager.SendMessage methods in your script to make the Undo system work.

The sequence is as follows:

- Invoke the PreProcess method which initializes the robots in the Schematic server
- Add new objects and register them in the database
- Send a SCHM_PrimitiveRegistration message
- Invoke the PostProcess method 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.

```
Procedure CreateSchObjectsWithUndo;
```

```
Var
    Doc         : ISch_Document;
    AName       : TDynamicString;
    Orientation : TRotationBy90;
    AElectrical : TPinElectrical;
    SchPort     : ISch_Port;
    Loc         : TLocation;
Begin
    // Check if Schematic server loaded in DXP.
    If SchServer = Nil Then Exit;
    // Retrieve the current schematic document otherwise exit;
    Doc := SchServer.GetCurrentSchDocument;
    If Doc = Nil Then Exit;

    // Initialize the robots in Schematic editor.
    SchServer.ProcessControl.PreProcess(Doc, '');
    // New port created.
    SchPort := SchServer.SchObjectFactory(ePort,eCreate_GlobalCopy);
    If SchPort = Nil Then Exit;
    SchPort.Location  := Point(150,150);
    SchPort.Style     := ePortRight;
    SchPort.IOType    := ePortBidirectional;
    SchPort.Alignment := eHorizontalCentreAlign;
    SchPort.Width     := 100;
    SchPort.AreaColor := 0;
    SchPort.TextColor := $00FF00;
    SchPort.Name      := 'New Port 3';
    // Add a new port object in the existing Schematic document.
    Doc.RegisterSchObjectInContainer(SchPort);
    SchServer.RobotManager.SendMessage(
                           Doc.I_ObjectAddress,
                           c_BroadCast,
                           SCHM_PrimitiveRegistration,
                           SchPort.I_ObjectAddress);

    // Clean up the robots in Schematic editor
```

```
    SchServer.ProcessControl.PostProcess(Doc, '');
    Doc.GraphicallyInvalidate;

    // use of Server processes to refresh the screen.
    ResetParameters;
    AddStringParameter('Action', 'Document');
    RunProcess('Sch:Zoom');
End;
```

See also

CreateSchObjects script in \Examples\Scripts\Delphiscript Scripts\Sch\ folder

UndoRedo script in \Examples\Scripts\Delphiscript Scripts\Sch\ folder

Removing schematic objects example

```
Procedure DeleteSchObjects;
Var
    OldPort       : ISch_Port;
    Port          : ISch_Port;
    CurrentSheet  : ISch_Document;
    Iterator      : ISch_Iterator;
Begin
    If SchServer = Nil Then Exit;
    CurrentSheet := SchServer.GetCurrentSchDocument;
    If CurrentSheet = Nil Then Exit;

    // Initialize the robots in Schematic editor.
    SchServer.ProcessControl.PreProcess(Client.CurrentView, '');

    Iterator := CurrentSheet.SchIterator_Create;
    If Iterator = Nil Then Exit;
    Iterator.AddFilter_ObjectSet(MkSet(ePort));
    Try
        Port := Iterator.FirstSchObject;
        While Port <> Nil Do
        Begin
            OldPort := Port;
            Port := Iterator.NextSchObject;
```

```
            CurrentSheet.RemoveSchObject(OldPort);
            SchServer.RobotManager.SendMessage(
               CurrentSheet.I_ObjectAddress,
               c_BroadCast,
               SCHM_PrimitiveRegistration,
               OldPort.I_ObjectAddress);
         End;
    Finally
         CurrentSheet.SchIterator_Destroy(Iterator);
    End;
    // Clean up robots in Schematic editor.
    SchServer.ProcessControl.PostProcess(Client.CurrentView, '');
    CurrentSheet.GraphicallyInvalidate;
End;
```

See also

DeleteSchObjects script in \Examples\Scripts\Delphiscript Scripts\Sch\ folder

Modifying Schematic Objects and updating the Undo system

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. You will need to invoke the **PreProcess** and **PostProcess** methods and the **SendMessage** methods with appropriate parameters.

The sequence is as follows;

- Invoke the **PreProcess** method which Initializes the robots in the schematic server
- Send a **SCHM_BeginModify** message
- Modify the Schematic object
- Send a **SCHM_EndModify** message
- Invoke the **PostProcess** method which cleans up the robots in the Schematic server

Changing Schematic object's attributes example

```
Procedure FetchAndModifyObjects;
Var
    AnObject        : ISch_GraphicalObject;
    Iterator        : ISch_Iterator;
    Doc             : ISch_Document;
Begin
    If SchServer = Nil Then Exit;
```

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

    // Initialize the robots in Schematic editor.
    SchServer.ProcessControl.PreProcess(Doc, '');

    Iterator        := Doc.SchIterator_Create;
    Iterator.AddFilter_ObjectSet(MkSet(ePort, eWire));
    If Iterator = Nil Then Exit;
    Try
        AnObject := Iterator.FirstSchObject;
        While AnObject <> Nil Do
        Begin
            SchServer.RobotManager.SendMessage(
                AnObject.I_ObjectAddress,
                c_BroadCast,
                SCHM_BeginModify,
                c_NoEventData);

            Case AnObject.ObjectId Of
               eWire   : AnObject.Color     := $0000FF; //red color in bgr
format
               ePort   : AnObject.AreaColor := $00FF00; //green color in bgr
format
            End;

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

            AnObject := Iterator.NextSchObject;
        End;
    Finally
        Doc.SchIterator_Destroy(Iterator);
```

```
    End;
    // Clean up the robots in Schematic editor
    SchServer.ProcessControl.PostProcess(Doc, '');
    Doc.GraphicallyInvalidate;
End;
```

Notes

When you change the properties of a schematic object on a Schematic document, it is necessary to employ the ProcessControl interface's PrePost and PostProcess methods and 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** interface in this document.

See also

ISch_RobotManager interface

IProcessControl interface from the IClient interface.

ModifySchObjects script in the \Examples\Scripts\Delphiscript Scripts\Sch\ folder.

UndoRedo script in the \Examples\Scripts\Delphiscript Scripts\Sch\ 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 Spatial iterator where it needs the bounds of a rectangle on the schematic document to search within.

- **ChooseLocationInteractively**
- **ChooseRectangleInteractively**

Interactive Methods

```
Function  ChooseLocationInteractively(Var ALocation : TLocation;
                                        Prompt : TDynamicString) : Boolean;
Function  ChooseRectangleInteractively(Var ARect : TCoordRect;
                                         Prompt1    : TDynamicString;
                                         Prompt2    : TDynamicString) :
Boolean;
```

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);
    End;
End;
```

See also

ISch_Document interface

Using a Spatial iterator script in \Examples\Scripts\DephiScript Scripts\Sch\ folder.

Schematic Interfaces

Schematic Object Model

An interface is just a means of access to an object in memory. To have access to the schematic server and massage certain schematic design objects, you need to invoke the SchServer function which extracts the ISch_ServerInterface interface which represents the loaded schematic server in DXP. This is the main interface and contains many interfaces within.

With the ISch_ServerInterface interface, you can get the ISch_Document interface either by invoking the GetSchDocumentByPath or GetCurrentSchDocument interface method and then with the ISch_Document interface, you can proceed further by iterating for certain schematic objects.

A simplified Schematic Interfaces hierarchy

ISch_BasicContainer
- ISch_GraphicalObject
  - ISch_Arc
    - ISch_EllipticalArc

The ISch_ServerInterface and ISch_Document interfaces to name the few are the main interfaces that you will be dealing with, when you are extracting data from a schematic document.

See also

ISch_Arc

ISch_BusEntry

ISch_Bezier

ISch_Bus

ISch_ConnectionLine

ISch_Circle

ISch_ComplexText

ISch_Component

ISch_CrossSheetConnector

ISch_Designator

ISch_Directive

ISch_Document

ISch_EllipticalArc

ISch_Lib

ISch_Iterator

ISch_Pie

ISch_Line

ISch_Ellipse

ISch_ErrorMarker

ISch_Image

ISch_Junction

ISch_Label

ISch_NetLabel

ISch_NoERC

ISch_Parameter

ISch_ParametrizedGroup

ISch_ParameterSet

ISch_Pin

ISch_Port

ISch_Polygon

ISch_Polyline

ISch_PowerObject

ISch_Probe

ISch_Rectangle

ISch_RectanglarGroup

ISch_RoundRectangle

ISch_SheetEntry

ISch_SheetSymbol

ISch_ServerInterface

ISch_Sheet

ISch_SheetFileName

ISch_SheetName

ISch_Symbol

ISch_Template

ISch_TextFrame

ISch_Wire

ISCH_ServerInterface

Overview

This interface is an entry interface to the schematic server loaded in DXP. You can fetch the Preferences, Robot Manager (for sending messages into the schematic system), the font manager for managing fonts on a schematic document. You can also create or delete schematic design objects.

Note that these **IServerModule** interfaces represent loaded servers in DXP. The DXP application manages single instances of different server modules. Each server can have multiple server document kinds, for example the PCB server supports two server document kinds – SCH and SCHLIB design documents. A loaded server in DXP typically hosts documents and each document in turn hosts a document view and panel views. Thus a Schematic Editor server also has the **IServerModule** interface along with the **ISCH_ServerInterface** interface.

Notes

- Invoke the **SchServer** function to obtain the **ISch_ServerInterface** object interface.

ServerInterface Methods

```
//Methods documents
Function  GetSchDocumentByPath(APath : WideString) : ISch_Document;
Function  GetCurrentSchDocument  : ISch_Document;

//Methods Sch Objects Creation/destruction
Function  SchObjectFactory(AObjectId     : TObjectId;
                           ACreationMode : TObjectCreationMode) :
ISch_BasicContainer;
Procedure DestroySchObject(Var ASchObject : ISch_BasicContainer);


Function  LoadComponentFromLibrary(ALibReference : WideString;
                                   ALibraryName  : WideString) :
 ISch_Component;
Function ReportSchObjectsDifferences(Const AObject1, AObject2 :
ISch_BasicContainer;
                                     AIgnoreSpatialAttributes : Boolean;
                                     ADiffDescription         : PChar) :
Integer;

// Schematic Library Information Extractor
Function CreateLibCompInfoReader(ALibFileName : WideString) :
ILibCompInfoReader;
Procedure DestroyCompInfoReader(Var ALibCompReader : ILibCompInfoReader);
```

Properties

```
Property Preferences   : ISch_Preferences
Property RobotManager  : ISch_RobotManager
Property FontManager   : ISch_FontManager
Property ProbesTimerEnabled : Boolean
```

See also

ISch_Preferences interface

ISch_RobotManager interface

ISch_FontManager interface

ISch_Document interface

ISch_Document

Overview

This interface is the immediate ancestor interface for ISch_Sheet and ISch_Lib interfaces. You can iterate design objects in a Schematic or library document, see ISch_Iterator interface for details.

With scripts, you can invoke the ChooseLocationInteractively or ChooseRectangleInteractively methods to obtain coordinates from the Schematic sheet or library sheet.

Notes

- ISch_Document interface's ancestors
- ISch_BasicContainer
  - ISch_GraphicalObject
    - ISch_ParameterizedGroup
      - ISch_Document

Methods

```
Procedure RegisterSchObjectInContainer       (AObject   :
ISch_BasicContainer);
Procedure UnRegisterSchObjectFromContainer   (AObject   :
ISch_BasicContainer);
Function  ObjectReferenceZone(AObject : ISch_BasicContainer): WideString;

Procedure RedrawToDC(DC : HDC; PrintKind : Integer);

Procedure LockViewUpdate;

Procedure UnLockViewUpdate;

Function  CreateHitTest (ATestMode : THitTestMode;
                         ALocation : TLocation) : ISch_HitTest;

Procedure PlaceSchComponent (ALibraryPath  : WideString;
                             ALibRef       : WideString;
```

```
                             Var SchObject : TSchObjectHandle);

Procedure CreateLibraryFromProject (AddLibToProject : Boolean;
                                    FileName        : WideString;
                                    RunQuiet        : Boolean);

Procedure UpdateDocumentProperties;


Function CountContextMenuObjects(AObjectSet : TObjectSet) : Integer
// Interactive Methods
Function  ChooseLocationInteractively(Var ALocation : TLocation;
                                      Prompt : TDynamicString) : Boolean;
Function  ChooseRectangleInteractively(Var ARect : TCoordRect;
                                       Prompt1   : TDynamicString;
                                       Prompt2   : TDynamicString) :
Boolean;
Function CountContextMenuObjects (AObjectSet : TObjectSet) : Integer;
```

Properties

```
Property DocumentName          : WideString
Property DocumentBorderStyle   : TSheetDocumentBorderStyle
Property CustomSheetStyle      : WideString
Property SheetStyle            : TSheetStyle
Property WorkspaceOrientation  : TSheetOrientation
Property TitleBlockOn          : Boolean
Property BorderOn              : Boolean
Property ReferenceZonesOn      : Boolean
Property UseCustomSheet        : Boolean
Property CustomX               : TCoord
Property CustomY               : TCoord
Property CustomXZones          : TCoord
Property CustomYZones          : TCoord
Property CustomMarginWidth     : TCoord
Property SnapGridOn            : Boolean
Property SnapGridSize          : TCoord
Property ShowTemplateGraphics  : Boolean
Property TemplateFileName      : WideString
```

```
Property VisibleGridOn        : Boolean
Property VisibleGridSize      : TCoord
Property HotSpotGridOn        : Boolean
Property HotSpotGridSize      : TCoord
Property SheetSizeX           : TCoord
Property SheetSizeY           : TCoord
Property SheetZonesX          : Integer
Property SheetZonesY          : Integer
Property SheetMarginWidth     : TCoord
Property SystemFont           : TFontId
Property LoadFormat           : WideString
Property DisplayUnit          : TUnit
Property UnitSystem           : TUnitSystem
```

RegisterSchObjectInContainer example

```
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;

    // Create a new port object
    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 port object onto the existing schematic document
    FSchDoc.RegisterSchObjectInContainer(SchPort);
    FSchDoc.GraphicallyInvalidate;

    // Invoke a Schematic Zoom process to refresh the screen.
    ResetParameters;
    AddStringParameter('Action', 'Document');
    RunProcess('Sch:Zoom');
```

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
second 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);
    End;
End;
```

See also

ISch_Sheet interface

ISch_Lib interface

TSheetDocumentBorderStyle enumerated type

TSheetStyle enumerated type

TSheetOrientation enumerated type

TCoord enumerated type

TFontId enumerated type

UsingASpatialIterator example from the **\Examples\Delphi Scripts\Scripts\Sch folder**

ISch_Lib

Overview

This interface represents an existing library document open in DXP. You can iterate design objects in a library document, however you will need to create a library iterator with the SchLibIterator_Create function. You can invoke the **LibIsEmpty** method to check if the library is empty (ie no symbols in the library) or not.

Important Notes

- The ISch_Document interface's methods and properties are available to use as well
- Due to the nature of a library document, all symbols (library components) are displayed on separate library documents, so you iterate through library documents of a library to fetch symbols.
- ISch_Lib interface's ancestors
- ISch_BasicContainer
  - ISch_GraphicalObject
    - ISch_ParameterizedGroup
      - ISch_Document
        - ISch_Lib

ISch_Lib Methods

```
Procedure SetState_CurrentSchComponentAddPart;

Procedure SetState_CurrentSchComponentAddDisplayMode;


Procedure SetState_CurrentSchComponentRemovePart;

Procedure SetState_CurrentSchComponentRemoveDisplayMode;


Procedure SetState_CurrentSchComponentPartId(APartId : Integer);

Function  GetState_CurrentSchComponentPartId : Integer;


Procedure SetState_CurrentSchComponentDisplayMode(ADisplayMode :
TDisplayMode);
Function  GetState_CurrentSchComponentDisplayMode : TDisplayMode;


Procedure AddSchComponent   (Const AComponent : ISch_Component);
Procedure RemoveSchComponent(Const AComponent : ISch_Component);
Function  SchLibIterator_Create : ISch_Iterator;
Function  LibIsEmpty : Boolean;
Procedure TransferComponentsPrimitivesBackFromEditor;
Procedure TransferComponentsPrimitivesToEditor;

Function  Sch_LibraryRuleChecker_Create : ISch_LibraryRuleChecker;

Procedure Sch_LibraryRuleChecker_Destroy (Var ARuleChecker :
ISch_LibraryRuleChecker);
```

ISch_Lib Properties

```
Property ShowHiddenPins       : Boolean
Property Description          : WideString
Property CurrentSchComponent : ISch_Component
```

See also

ISch_Iterator interface

ISch_Component interface

ISch_Document interface

ISch_ParametrizedGroup interface

ISch_GraphicalObject interface

ISch_BasicContainer interface

CreateCompInLib script in **Examples\Delphi Scripts\Scripts\Sch** folder

LibIterator script in **\Examples\Delphi Scripts\Scripts\Sch folder**

ISch_Sheet

Overview

The ISch_Sheet interface represents an existing schematic document open in DXP. You can iterate for design objects on a currently focussed Schematic document in DXP.

Important Notes

- ISch_Sheet interface's ancestors
- ISch_BasicContainer
  - ISch_GraphicalObject
    - ISch_ParameterizedGroup
      - ISch_Document
        - ISch_Sheet

Properties

```
Property WireConnections : IConnectionsArray
Property BusConnections  : IConnectionsArray
```

See also

ISch_Iterator interface

ISch_Document interface

IConnectionArray interface

ISch_Iterator

Overview

An iterator object interface represents an existing iterator object which iterates through a design database to fetch specified objects within a specified region if necessary.

Important Notes

- Delphi Script does not support sets. Therefore, to specify the object set or the layer set, you need to use the **MkSet** function to create a set of objects, for example Iterator.AddFilter_ObjectSet(MkSet(ePort));
- The TIterationDepth type denotes how deep the iterator can look - look for first level objects (for example standalone system parameters of the document only, or all levels for example all parameters on the document including system parameters, objects' parameters such as component's parameters. By default, eIterateAllLevels value is used.
- SetState_FilterAll denotes that all objects and the whole schematic document is to be searched within. Otherwise, use the following AddFilter_ObjectSet, AddFilter_Area etc methods to set up a restricted search.

ISch_Iterator Methods

```
Function  I_ObjectAddress : TSCHObjectHandle;

Procedure SetState_FilterAll;



Procedure AddFilter_ObjectSet(AObjectSet : TObjectSet);

Procedure AddFilter_CurrentPartPrimitives;

Procedure AddFilter_CurrentDisplayModePrimitives;

Procedure AddFilter_PartPrimitives(APartId : Integer; ADisplayMode :
TDisplayMode);
Procedure AddFilter_Area(X1, Y1, X2, Y2 : TCoord);



Procedure SetState_IterationDepth(AIterationDepth : TIterationDepth);



Function FirstSchObject : ISch_BasicContainer;

Function NextSchObject  : ISch_BasicContainer;
```

Example

```
Procedure CountPortObjects;
Var
    Port          : ISch_Port;
```

```
    CurrentSheet : ISch_Sheet;
    Iterator     : ISch_Iterator;
    PortNumber   : Integer;
Begin
    If SchServer = Nil Then Exit;

    CurrentSheet := SchServer.GetCurrentSchDocument;
    If CurrentSheet = Nil Then Exit;
    PortNumber := 0;

    Iterator := CurrentSheet.SchIterator_Create;
    Iterator.AddFilter_ObjectSet(MkSet(ePort));
    Try
        Port := Iterator.FirstSchObject;
        While Port <> Nil Do
        Begin
            If Port.ObjectId = ePort Then
                PortNumber := PortNumber + 1;
            Port := Iterator.NextSchObject;
        End;
        ShowInfo ('The number of ports on the page is : ' +
IntToStr(PortNumber));
    Finally
        CurrentSheet.SchIterator_Destroy(Iterator);
    End;
End;
```

See also

MkSet keyword in DelphiScript Reference

TIterationDepth type

Script examples in the **\Altium2004\Examples\Scripts** folder

ISch_RobotManager

Overview

The ISch_RobotManager interface represents an object that can send Schematic messages into the Schematic Editor server from a script to update the sub-systems such as the Undo system.

Important Notes

- Part of ISch_ServerInterface object interface

MessageID table

```
SCHM_NullMessage              = 0;
SCHM_PrimitiveRegistration    = 1;
SCHM_BeginModify              = 2;
SCHM_EndModify                = 3;
SCHM_YieldToRobots            = 4;
SCHM_CancelModify             = 5;
SCHM_Create                   = 6;
SCHM_Destroy                  = 7;
SCHM_ProcessStart             = 8;
SCHM_ProcessEnd               = 9;
SCHM_ProcessCancel            = 10;
SCHM_CycleEnd                 = 11;
SCHM_CycleStart               = 12;
SCHM_SystemInvalid            = 13;
SCHM_SystemValid              = 14;
```

Message types table

```
c_BroadCast     = Nil;
c_NoEventData   = Nil;
c_FromSystem    = Nil;
```

ISch_RobotManager Methods

```
Procedure SendMessage(Source,Destination : Pointer; MessageID : Word;
MessageData : Pointer);
```

Example

```
    Client.ProcessControl.PreProcess(Client.CurrentView, '');
    Try
       // Add component to schematic with undo enabled
       Rect.OwnerPartId := Component.CurrentPartID;
       Rect.OwnerPartDisplayMode := Component.DisplayMode;

       Rect.Location := Point(0, 0);
       Rect.Corner := Point(20, 20);
```

```
        Pin.OwnerPartId := Component.CurrentPartID;
        Pin.OwnerPartDisplayMode := Component.DisplayMode;
        Pin.Location := Point(20, 10);

        Component.AddSchObject(Rect);
        Component.AddSchObject(Pin);
        SchDoc.AddSchObject(Component);
        Component.MoveByXY(100, 100);
        SchServer.RobotManager.SendMessage(SchDoc.I_ObjectAddress,
c_BroadCast, SCHM_PrimitiveRegistration, Component.I_ObjectAddress);
    Finally
        Client.ProcessControl.PostProcess(Client.CurrentView, '');
    End;
```

See also

ISch_ServerInterface interface.

DeleteSchObjects in **\Examples\Scripts\Sch** folder.

ModifySchObjects in **\Examples\Scripts\Sch** folder.

UndoRedo script in **\Examples\Scripts\Sch** folder.

ISch_Preferences

Overview

The ISch_Preferences interface represents the global schematic preference settings for schematic and library documents.

ISch_Preferences Properties

```
Property SelectionColor              : TColor
Property MultiSelectionColor         : TColor
Property ResizeColor                 : TColor
Property TranslateRotateColor        : TColor
Property VisibleGridColor            : TColor
Property VisibleGridStyle            : TVisibleGrid
Property GraphicsCursorStyle         : TCursorShape
Property OrcadFootPrint              : TOrcadFootPrint
Property SnapToCenter                : Boolean
Property UseOrcadPortWidth           : Boolean
Property AutoBackupTime              : Integer
```

```
Property AutoBackupFileCount          : Integer
Property SelectionReference           : Boolean
Property UndoRedoStackSize            : Integer
Property ConvertSpcialStrings        : Boolean
Property MaintainOrthogonal           : Boolean
Property DisplayPrinterFonts          : Boolean
Property AutoZoom                     : Boolean
Property HotSpotGridDistance          : Integer
Property SnapToHotSpot                : Boolean
Property AutoJunction                 : Boolean
Property OptimizePolylines            : Boolean
Property ComponentsCutWires           : Boolean
Property AddTemplateToClipBoard       : Boolean
Property AutoPanStyle                 : TAutoPanStyle
Property AutoPanJumpDistance          : TCoord
Property AutoPanShiftJumpDistance     : TCoord
Property PinNameMargin                : Integer
Property PinNumberMargin              : Integer
Property DefaultPrimsPermanent        : Boolean
Property IgnoreSelection              : Boolean
Property ClickClearsSelection         : Boolean
Property DoubleClickRunsInspector     : Boolean
Property MultiPartNamingMethod        : Integer
Property Sensitivity                  : Integer
Property SingleSlashNegation          : Boolean
Property RunInPlaceEditing            : Boolean
Property DefaultPowerGndName          : WideString
Property DefaultSignalGndName         : WideString
Property DefaultEarthName             : WideString
Property DefaultTemplateFileName      : WideString
Property BufferedPainting             : Boolean
Property Metafile_NoERCMarkers        : Boolean
Property Metafile_ParameterSets       : Boolean
Property DocumentScope                : TChosenDocumentScope
Property LibraryScope                 : TLibraryScope
Property ConfirmSelectionMemoryClear : Boolean
```

```
Property LastModelType                 : WideString
Property StringIncA                    : WideString
Property StringIncB                    : WideString
Property MarkManualParameters          : Boolean
Property CtrlDbleClickGoesDown         : Boolean

Property SheetStyle_XSize      [S : TSheetStyle]: TCoord   // Read only
Property SheetStyle_YSize      [S : TSheetStyle]: TCoord   // Read only
Property SheetStyle_XZones     [S : TSheetStyle]: TCoord   // Read only
Property SheetStyle_YZones     [S : TSheetStyle]: TCoord   // Read only
Property SheetStyle_MarginWidth[S : TSheetStyle]: TCoord   // Read only

Property PolylineCutterMode     : TPolylineCutterMode
Property CutterGridSizeMultiple : Integer
Property CutterFixedLength      : TCoord
Property ShowCutterBoxMode      : TShowCutterBoxMode
Property ShowCutterMarkersMode  : TShowCutterMarkersMode
Property PolylineCutterMode : TPolylineCutterMode
Property CutterFixedLength : Integer
Property ShowCutterBoxMode : TShowCutterBoxMode
Property ShowCutterMarkersMode : TShowCutterMarkersMode
Property ViolationDisplay [L : TErrorLevel] : Boolean;
Property ViolationColor [L : TErrorLevel] : Boolean;
Property AlwaysDrag             : Boolean
Property DocMenuID              : Widestring
Property LibMenuID              : Widestring
Property DefaultSheetStyle      : TSheetStyle
Property WireAutoJunctionsColor : TColor
Property ManualJunctionsColor   : TColor
Property BusAutoJunctionsColor  : TColor
Property DefaultDisplayUnit     : TUnit
Property DefaultUnitSystem      : TUnitSystem
```

See also

IServerInterface interface

ISch_FontManager

Overview

The ISch_FontManager interface represents the internal font manager in Schematic Editor that manages fonts for text based objects on schematic documents.

Important Notes

ISch_FontManager Methods

```
Function  GetFontID  (Size,Rotation : Integer;
Underline,Italic,Bold,StrikeOut : Boolean;
FontName : WideString) : TFontID;

Procedure GetFontSpec   (FontID : TFontID; Var Size,Rotation  : Integer; Var
Underline,Italic,Bold,StrikeOut : Boolean ;Var FontName : WideString);

Function  GetFontSize   (FontID : TFontID) : Integer;
```

ISch_FontManager Properties

```
Property DefaultHorizontalSysFontId : Integer          // Read only
Property DefaultVerticalSysFontId   : Integer          // Read only
Property FontCount : Integer                           // Read only
Property Rotation            [Id : Integer] : Integer   // Read only
Property Size                [Id : Integer] : Integer   // Read only
Property Italic              [Id : Integer] : Boolean   // Read only
Property Bold                [Id : Integer] : Boolean   // Read only
Property UnderLine           [Id : Integer] : Boolean   // Read only
Property StrikeOut           [Id : Integer] : Boolean   // Read only
Property SaveFlag            [Id : Integer] : Boolean   // Read only
Property FontName            [Id : Integer] : TFontName // Read only
```

See also

Font Manager script example in the **\Examples\Delphi Scripts\Scripts\Sch\Font Editor folder**

ISch_LibraryRuleChecker interface

Overview

The **ISch_LibraryRuleChecker** interface represents the internal library rule checker facility that checks the validity of symbols in schematic libraries.

Important Notes

Deals with schematic library documents only.

ISch_FontManager Methods

```
Function  SetState_FromParameters(Parameters : PChar) : Boolean;
Function  Import_FromUser : Boolean;
Function  Run : Boolean;
Function  I_ObjectAddress : TSCHObjectHandle;
```

ISch_FontManager Properties

```
Property Duplicate_Pins               : Boolean
Property Duplicate_Component          : Boolean
Property Missing_Pin_Number           : Boolean
Property Missing_Default_Designator : Boolean
Property Missing_Footprint            : Boolean
Property Missing_Description          : Boolean
Property Missing_Pin_Name             : Boolean
Property Missing_Pins_In_Sequence    : Boolean
Property ShowReport                   : Boolean
```

See also

IClient interface

IExternalForm interface

IConnectionsArray

Overview

The **IConnectionsArray** represents the bus and wire connections in a schematic document. Bus and wire connections could be connected by an automatic junction or a manual junction (placed by an user).

Important Notes

The **IConnectionsArray** interface is extracted from the **ILibCompInfoReader.ComponentInfos[Index]** method.

Methods

```
Procedure AddConnection  (ALocation : TLocation);
Procedure AddConnectionXY(X, Y : TCoord);
Procedure ResetAllConnections;
Procedure GraphicallyInvalidate;
Function  RemoveAllConnectionsAt(ALocation : TLocation) : Boolean;
```

```
Function  RemoveAllConnectionsForLine(L1, L2 : TLocation) : Boolean;
Function  GetConnectionAt(ALocation : TLocation) : IConnection;
```

Properties

```
Property ConnectionsCount        : Integer
Property Connection[i : Integer] : IConnection
```

See also

IConnection interface

IConnection

Overview

The **IConnection** interface represents whether the connection has a junction on it or not, with location and objects count. A manual junction (placed by an user) may signify a forced connection on a schematic document.

Important Notes

The **IConnection** interface is extracted from the **IConnectionArray.Connection** method.

Properties

```
Property Location : TLocation
Property ObjectsCount : Integer
Property IsManualJunction : Boolean
```

See also

IConnectionsArray interface

ILibCompInfoReader

Overview

The ILibCompInfoReader interface encapsulates the object that obtains component information of a specified schematic library with the filename of the schematic library parameter.

Important Notes

1/ Create and obtain the ILibCompInfoReader interface from the SchServer.CreateLibCompInfoReader method with the specified filename parameter.

2/ Invoke the ReadAllComponentInfo method.

3/ Invoke the NumComponentInfos method to obtain the count

4/ Invoke the indexed ComponentInfos property to obtain the IComponentInfo interface.

5/ Destroy the object by invoking the SchServer.DestroyCompInfoReader.

ILibCompInfoReader Methods

```
Procedure   ReadAllComponentInfo;
Function    NumComponentInfos            : Integer;
Function    I_ObjectAddress              : TSCHObjectHandle;
```

ILibCompInfoReader Properties

```
Property ComponentInfos[i : Integer] : IComponentInfo // Read only
Property FileName                    : WideString     // Read only
```

Example

```
Var
    ALibCompReader : ILibCompInfoReader;
    CompInfo       : IComponentInfo;
    ReportInfo     : TStringList;
    Filename         : String;
    CompNum        : Integer;
Begin
    If SchServer = Nil Then Exit;
    ReportInfo := TStringList.Create;

        Filename := '';
        ALibCompReader := SchServer.CreateLibCompInfoReader(FileName);
        ALibCompReader.ReadAllComponentInfo;
        CompNum := ALIbCompReader.NumComponentInfos;
        For J := 0 To CompNum - 1 Do
        Begin
            ReportInfo.Add(FileName);
            CompInfo := ALibCompReader.ComponentInfos[J];
            ReportInfo.Add(' Name : ' + CompInfo.CompName);
            ReportInfo.Add('  Alias Name : ' + CompInfo.AliasName);
            ReportInfo.Add('  Part Count : ' +
IntToStr(CompInfo.PartCount));
            ReportInfo.Add('  Description : ' + CompInfo.Description);
            ReportInfo.Add('  Offset : ' + IntToStr(CompInfo.Offset));
            ReportInfo.Add('');
        End;
      //  SchServer.DestroyCompInfoReader(ALibCompReader);
        ReportInfo.Add('');
```

```
End;
```

See also

IComponentInfo interface

See CompLibReader script in **Examples\Delphi Scripts\Scripts\Sch** folder

IComponentInfo

Overview

The **IComponentInfo** interface is an item within the **ILibCompInfoReader** interface which represents an existing schematic library file. This **IComponentInfo** interface represents a schematic symbol in a specified schematic library file with a SchLib extension.

Important Notes

The IComponentInfo interface is extracted from the **ILibCompInfoReader.ComponentInfos[Index]** method.

IComponentInfo Properties

```
Property Offset      : Integer    // Read only
Property AliasName   : WideString // Read only
Property CompName    : WideString // Read only
Property PartCount   : Integer    // Read only
Property Description : WideString // Read only
```

See also

ILibCompInfoReader interface

ISch_HitTest

Overview

The ISch_HitText interface returns the object that has been clicked on by the mouse.

ISch_HitTest Properties

```
Property HitTestCount          : Integer              // Read only
Property HitObject[i : Integer] : ISch_GraphicalObject // Read only
```

See also

ISch_GraphicalObject interface

ISch_Document interface

ISch_Implementation

Overview

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.

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 DXP.

- 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 DXP's central database, then each signal integrity model would then have multiple data files, each one for each type of pin.

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

ISch_Implementation Methods

```
Procedure AddDataFileLink(anEntityName, aLocation, aFileKind : WideString);
Procedure ClearAllDatafileLinks;
Function  Map_Import_FromUser  (AlowOneToMany    : Boolean): Boolean;
Procedure LockImplementation;
```

ISch_Implementation Properties

```
Property Description        : WideString      Read GetState_Description
Property ModelName          : WideString      Read GetState_ModelName
Property ModelType          : WideString      Read GetState_ModelType
Property IntegratedModel    : Boolean         Read GetState_IntegratedModel
Property DatalinksLocked    : Boolean         Read GetState_DatalinksLocked
Property IsCurrent          : Boolean         Read GetState_IsCurrent
Property MapAsString        : WideString      Read GetState_MapAsString
Property DatafileLinkCount : Integer
Property DefinerByInterfaceDesignator[S : WideString] : ISch_MapDefiner
Property DatafileLink                   [i : Integer]    :
ISch_ModelDatafileLink
```

ISch_MapDefiner

ISch_MapDefiner Methods

```
Procedure SetState_Designator_ImplementationClear;
Procedure SetState_Designator_ImplementationAdd(AValue : WideString);
Procedure SetState_AllFromString               (AValue : WideString);
```

ISch_MapDefiner Properties

```
Property Designator_Interface                   : WideString
Property Designator_ImplementationCount         : Integer
Property Designator_Implementation[i : Integer] : WideString
Property Designator_Implementations_AsString    : WideString
Property IsTrivial : Boolean
```

ISch_ModelDatafileLink

Overview

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. 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 DXP.

- 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 DXP's central database, then each signal integrity model would then have multiple data files, each one for each type of pin.

ISch_ModeldatafileLink Properties

```
Property EntityName : WideString
Property Location   : WideString
Property FileKind   : WideString
```

Schematic Design Objects

A schematic design object on a schematic document is represented by its interface. An interface represents an existing object in memory and its properties and methods can be invoked.

Since many design objects are descended from ancestor interfaces and thus the ancestor methods and properties are also available to use. For example the **ISch_Image** interface is inherited from an immediate **ISch_Rectangle** interface and in turn inherited from the **ISch_GraphicalObject** interface. If you check the **ISch_Image** entry in this online help you will see the following information;

The ISch_Image interface hierachy is as follows;

- ISch_GraphicalObject
  - ISch_Rectangle
    - ISch_Image

Immediate ancestor ISch_Rectangle properties

```
Corner    : TLocation
LineWidth : TSize
IsSolid   : Boolean
```

ISch_Image Properties

```
EmbedImage : Boolean
FileName   : WideString
KeepAspect : Boolean
```

Therefore you have the Image object properties, along with **ISch_Rectangle** methods and properties AND **ISch_GraphicalObject** methods and properties as well to use in your scripts.

See also

Schematic Documents

Schematic Objects

Creating/Deleting objects and updating the Undo system

Modifying objects and updating the Undo system

ISch_Arc

ISch_EllipticalArc

ISch_Pie

ISch_Line

ISch_BusEntry

ISch_ConnectionLine

ISch_Circle

ISch_Ellipse

ISch_Directive

ISch_ErrorMarker

ISch_Junction

ISch_NoERC

ISch_Label

ISch_NetLabel

ISch_PowerObject

ISch_CrossSheetConnector

ISch_ComplexText

ISch_Parameter

ISch_Designator

ISch_SheetFileName

ISch_SheetName

ISch_Rectangle

ISch_RoundRectangle

ISch_TextFrame

ISch_Image

ISch_SheetEntry

ISch_Symbol

ISch_Template

ISch_Polygon

ISch_Polyline

ISch_Bezier

ISch_Wire

ISch_Bus

ISch_ParameterSet

ISch_Port

ISch_Probe

ISch_Pin

ISch_Component

ISch_SheetSymbol

ISch_BasicContainer

Overview

The **ISch_BasicContainer** interface represents as a parent object or a child object for a schematic object in DXP. A sheet symbol object for example is a parent object, and its child objects are sheet entries, thus to fetch the sheet entries, you would create an iterator for the sheet symbol and iterate for sheet entry objects. A schematic document is a parent object as well thus you also create an iterator for this document and iterate for objects on this document.

Important Notes

- ISch_BasicContainer is the ancestor interface object for many schematic object interfaces.

ISch_BasicContainer Methods

```
Function  I_ObjectAddress : TSCHObjectHandle;
Procedure AddSchObject           (AObject : ISch_BasicContainer);
Procedure AddAndPositionSchObject(AObject : ISch_BasicContainer);
Procedure RemoveSchObject        (AObject : ISch_BasicContainer);

Function  SchIterator_Create : ISch_Iterator;
Procedure SchIterator_Destroy(Var AIterator : ISch_Iterator);

Procedure DeleteAll;
Procedure FreeAllContainedObjects;
Procedure Setstate_Default(AUnit : TUnitSystem);
Function  Import_FromUser             : Boolean;
Function  GetState_IdentifierString  : WideString;
Function  GetState_DescriptionString : WideString;
Function  Replicate                   : ISch_BasicContainer;
```

ISch_BasicContainer Properties

```
Property ObjectId      : TObjectId           // Read only
Property Container     : ISch_BasicContainer // Read only
Property OwnerDocument : ISch_Document;
```

See also

TObjectId enumerated values

Schematic Design Objects overview

ISch_GraphicalObject

Overview

The **ISch_GraphicalObject** interface represents an object that has graphical properties on a schematic document. All graphic objects such as arcs, ports, rectangles etc have bounding rectangles of TCoordRect type.

Important Notes

- Derived from ISch_BasicContainer interface

ISch_GraphicalObject Methods

```
Procedure RotateBy90(Center : TLocation; A : TRotationBy90);
Procedure MoveByXY  (x,y    : TCoord);
Procedure Mirror    (Axis   : TLocation);
Procedure SetState_xSizeySize;
Procedure GraphicallyInvalidate;
Function  BoundingRectangle : TCoordRect;
Function  BoundingRectangle_Full : TCoordRect;
Procedure AddErrorString(Const AErrorString : WideString; AtEnd : LongBool);
Procedure ResetErrorFields;
```

ISch_GraphicalObject Properties

```
Property Location               : TLocation
Property Color                  : TColor
Property AreaColor              : TColor
Property Selection              : Boolean
Property EnableDraw             : Boolean
Property Disabled               : Boolean
Property OwnerPartId            : Integer
Property OwnerPartDisplayMode   : TDisplayMode
Property LiveHighlightValue     : WideString
Property ErrorKind              : TErrorKind
Property ErrorColor             : TColor
Property DisplayError           : Boolean
Property ErrorString            : WideString
Property CompilationMasked      : Boolean
```

See also

TLocation enumerated values

TColor enumerated values

TDisplayMode enumerated values

TErrorKind enumerated values

TCoordRect enumerated values

Schematic Design Objects overview

ISch_Directive

Overview

An ISch_Directive interface represents an object that stores a text string. It is an ancestor interface for the ISch_ErrorMarker interface.

Notes

- The ISch_Directive interface is derived from ISch_GraphicalObject interface

ISch_Directive Properties

```
Property Text : WideString
```

See also

ISch_GraphicalObject interface

Schematic Design Objects overview

ISch_ErrorMarker

Overview

ErrorMarkers are placed on a sheet at the site of each ERC violation.

Notes

- The ISch_ErrorMarker interface is derived from ISch_Directive interface

Immediate ancestor ISch_Directive properties

```
Property Text : WideString
```

See also

ISch_Directive interface

Schematic Design Objects overview

ISch_NoERC

Overview

The NoERC directive is a special symbol that identifies a pin as one that you want the Electrical Rules Checker to ignore.

Notes

- Derived from ISch_GraphicalObject interface

Immediate ancestor ISch_GraphicalObject Methods

```
Procedure RotateBy90(Center : TLocation; A : TRotationBy90);
Procedure MoveByXY  (x,y    : TCoord);
Procedure Mirror    (Axis   : TLocation);
```

```
Procedure SetState_xSizeySize;
Procedure GraphicallyInvalidate;
Function  BoundingRectangle : TCoordRect;
Function  BoundingRectangle_Full : TCoordRect;
```

Immediate ancestor ISch_GraphicalObject Properties

```
Property Location              : TLocation
Property Color                 : TColor
Property AreaColor             : TColor
Property Selection             : Boolean
Property EnableDraw            : Boolean
Property Disabled              : Boolean
Property OwnerPartId           : Integer
Property OwnerPartDisplayMode : TDisplayMode
Property LiveHighlightValue    : WideString
Property ErrorKind             : TErrorKind
```

See also

ISch_GraphicalObject interface

Schematic Design Objects overview

ISch_Junction

Overview

Junctions are small circular objects used to logically join intersecting wires on the schematic sheet.

Notes

- The ISch_Junction interface is derived from ISch_GraphicalObject interface

Properties

```
Property Size   : TSize
Property Locked : Boolean
```

Example

```
Procedure PlaceASchJunction;
Var
    SchDoc    : ISch_Document;
    WorkSpace : IWorkSpace;
    SchJunction      : ISch_Junction;
Begin
```

```
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('SCH');
    If SchServer = Nil Then Exit;
    SchDoc := SchServer.GetCurrentSchDocument;
    If SchDoc = Nil Then Exit;

     SchJunction :=
SchServer.SchObjectFactory(eJunction,eCreate_GlobalCopy);
     If SchJunction = Nil Then Exit;
     SchJunction.Location        := Point(300, 200);
     SchJunction.SetState_Size   := eMedium;
     SchJunction.SetState_Locked := False;
     SchDoc.RegisterSchObjectInContainer(SchJunction);
End;
```

See also

TSize enumerated values

ISch_GraphicalObject interface

Schematic Design Objects overview

ISch_SheetEntry

Overview

A sheet entry within a Sheet Symbol object creates a connection between the net touching on the parent sheet, to a Port with the same name on the child sheet.

Notes

- Derived from the ISch_GraphicalObject interface

ISch_SheetEntry Methods

```
Function  IsVertical : Boolean;
```

ISch_SheetEntry Properties

```
Property Name                : WideString
Property Style               : TPortArrowStyle
Property Side                : TLeftRightSide
Property DistanceFromTop     : TCoord
Property IOType              : TPortIO
Property TextColor           : TColor
```

```
Property OwnerSheetSymbol  : ISch_SheetSymbol
```

See also

ISch_GraphicalObject interface

ISch_SheetSymbol interface

TPortArrowStyle enumerated types

TLeftRightSide enumerated types

Schematic Design Objects overview

ISch_Symbol

Overview

The symbol objects are special markers used for components in the Schematic Library.

Notes

- Descended from ISch_GraphicalObject

Properties

```
Property Orientation : TRotationBy90
Property Symbol      : TIeeeSymbol
Property IsMirrored  : Boolean
Property LineWidth   : TSize
Property ScaleFactor : TCoord
```

See also

ISchGraphicalObject interface

TIeeeSymbol enumerated values

TSize enumerated values

TCoord value

Schematic Design Objects overview

ISch_Template

Overview

The schematic templates represent the sheet border, title block and graphics for a schematic document.

Notes

- Descended from ISch_GraphicalObject

Properties

```
Property FileName : WideString
```

See also

ISch_GraphicalObject interface

Schematic Design Objects overview

ISch_Circle

Overview

A circle is a close arch object.

Notes

- The ISch_Circle interface is derived from ISch_GraphicalObject interface

Properties

```
Property LineWidth   : TSize
Property IsSolid     : Boolean
Property Radius      : TDistance
Property Transparent : Boolean
```

See also

ISch_GraphicalObject interface

TSize enumerated values

TDistance value

Schematic Design Objects overview

ISch_Ellipse

Overview

An ellipse is a drawing object which is filled or unfilled graphic elements.

Notes

- The ISch_Circle interface hierarchy is as follows;
- **ISch_GraphicalObject**
  - **ISch_Circle**
    - **ISch_Ellipse**

Immediate ancestor ISch_Circle properties

```
Property LineWidth : TSize
Property IsSolid   : Boolean
Property Radius    : TDistance
```

Properties

```
Property SecondaryRadius : TDistance
```

See also

TDistance enumerated values

ISch_Circle interface

Schematic Design Objects overview

ISch_Arc

Overview

An arc object is a circular curve used to place on the schematic sheet.

Notes

- The ISch_Arc interface is derived from ISch_GraphicalObject interface

Properties

```
Property Radius     : TDistance
Property StartAngle : TAngle
Property EndAngle   : TAngle
Property LineWidth  : TSize
```

See also

ISch_GraphicalObject interface

Schematic Design Objects overview

ISch_Pie

Overview

Pie objects are unfilled or filled graphic elements.

Notes

- The ISch_Pie interface hierarchy is as follows;
- **ISch_GraphicalObject**
  - **ISch_Arc**
    - **ISch_Pie**

Immediate ancestor ISch_Arc Properties

```
Property Radius     : TDistance
Property StartAngle : TAngle
Property EndAngle   : TAngle
Property LineWidth  : TSize
```

Properties

```
Property IsSolid : Boolean
```

See also

ISch_Arc interface

Schematic Design Objects overview

ISch_EllipticalArc

Overview

Elliptical arc objects are drawing objects which represent open circular or elliptical curves

Notes

- The ISch_EllipticalArc interface hierarchy is as follows;
- **ISch_GraphicalObject**
  - **ISch_Arc**
    - **ISch_EllipticalArc**

Immediate ancestor ISch_Arc Properties

```
Property Radius     : TDistance
Property StartAngle : TAngle
Property EndAngle   : TAngle
Property LineWidth  : TSize
```

ISch_EllipticalArc Properties

```
Property SecondaryRadius : TDistance
```

See also

ISch_Arc interface

Schematic Design Objects overview

ISch_Line

Overview

Lines are graphical drawing objects with any number of joined segments.

Notes

- The ISch_Line interface is derived from the ISch_GraphicalObject interface.

Properties

```
Property Corner      : TLocation
Property LineWidth   : TSize
Property LineStyle   : TLineStyle
```

Example

```
Procedure PlaceASchLine;
Var
    SchDoc    : ISch_Document;
    WorkSpace : IWorkSpace;
    SchLine : ISch_Line;
Begin
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('SCH');
    If SchServer = Nil Then Exit;
    SchDoc := SchServer.GetCurrentSchDocument;
    If SchDoc = Nil Then Exit;

     SchLine := SchServer.SchObjectFactory(eLine,eCreate_GlobalCopy);
     If SchLine = Nil Then Exit;
     SchLine.Location  := Point(180, 200);
     SchLine.Corner    := Point(180, 400);
     SchLine.LineWidth := eMedium;
     SchLine.LineStyle := eLineStyleSolid;
     SchLine.Color := $FF00FF;
     SchDoc.RegisterSchObjectInContainer(SchLine);
End;
```

See also

ISch_GraphicalObject interface

TLocation enumerated values

TSize enumerated values

TLineStyle enumerated values

Schematic Design Objects overview

ISch_BusEntry

Overview

A bus entry is a special wire at an angle of 45 degrees which is used to connect a wire to the bus line.

Notes

The ISch_BusEntry interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_Line**
    - **ISch_BusEntry**

Immediate ISch_Line properties

```
Property Corner      : TLocation
Property LineWidth   : TSize
Property LineStyle   : TLineStyle
```

See also

ISch_Line interface

Schematic Design Objects overview

ISch_ConnectionLine

Overview

A connection line represents a line that has corner properties as well as width and style properties between two nodes on a schematic document. An inferred property indicates that a connection between documents has been detected by the Schematic Navigation system after the project has been compiled.

An inferred property denotes whether the object is an inferred object with respect to connective objects. Bus and Sheet Symbols can be defined in ranges using the NetLabel [] and Repeat statements respectively and once the project has been compiled, inferred objects created in memory for navigation/connective purposes. For example, a Bus with a range of A[0..4] ends up with five wires with A0...A5 net labels (only in memory). This property is useful for multi – channel projects and for sheets that have Bus objects.

Notes

- The ISch_ConnectionLIne interface ancestors are;
- **ISch_GraphicalObject**
  - **ISch_Line**
    - **ISch_BusEntry**
    - **ISch_ConnectionLine**

Ancestor ISch_Line properties

```
Property Corner      : TLocation
Property LineWidth   : TSize
Property LineStyle   : TLineStyle
```

Properties

```
Property IsInferred : Boolean
```

See also

ISCh_BusEntry interface

ISch_Label

Overview

The ISch_Label interface is a wrapper for an existing label object on a schematic document. This interface is the ancestor interface for the ISch_NetLabel interface.

Notes

- The ISch_Label interface is derived from ISch_GraphicalObject interface
- The Text property can be used as a net string.

Properties

```
FontId                : TFontID
Orientation           : TRotationBy90
Justification         : TTextJustification
Text                  : WideString
OverrideDisplayString : WideString
DisplayString         : WideString
Formula               : WideString
CalculatedValueString : WideString
IsMirrored            : Boolean
```

See also

TFontID enumerated values

TRotationBy90 enumerated values

TTextJustification enumerated values

ISch_GraphicalObject interface

ISch_NetLabel interface

ISch_PowerObject interface

Schematic Design Objects overview

ISch_PowerObject

Overview

Power ports are special symbols that represent a power supply and are always identified by their net names.

Notes

The ISch_PowerObject interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_Label**
    - **ISch_PowerObject**
- Text property is the net name of the power object.

Immediate ancestor ISch_Label properties

```
Property FontId                : TFontID
Property Orientation           : TRotationBy90
Property Justification         : TTextJustification
Property Text                  : WideString
Property OverrideDisplayString : WideString
Property DisplayString         : WideString
Property Formula               : WideString
Property CalculatedValueString : WideString
Property IsMirrored            : Boolean
```

Properties

```
Property Style : TPowerObjectStyle
```

See also

ISch_Label interface

TPowerObjectStyle enumerated values

Schematic Design Objects overview

ISch_ComplexText

Overview

An immediate ancestor interface for ISch_SheetFilename and ISch_SheetName interfaces.

Notes

The ISch_ComplexText interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_Label**
    - **ISch_ComplexText**

Immediate ancestor ISch_Label properties

```
Property FontId                : TFontID
Property Orientation           : TRotationBy90
Property Justification         : TTextJustification
Property Text                  : WideString
```

```
Property OverrideDisplayString : WideString
Property DisplayString         : WideString
Property Formula               : WideString
Property CalculatedValueString : WideString
Property IsMirrored            : Boolean
```

Properties

```
Property Autoposition : Boolean
Property IsHidden : Boolean
Property TextHorzAnchor : TTextHorzAnchor
Property TextVertAnchor : TTextVertAnchor
```

See also

ISch_Label interface

Schematic Design Objects overview

ISch_Designator

Overview

The ISch_Designator interface represents a designator which is part of the component object.

Notes

The ISch_Designator interface hierarchy is as follows;

- **ISch_GraphicalObject**
  - **ISch_Label**
    - **ISch_ComplexText**
      - **ISch_Parameter**
        - **ISch_Designator**

Ancestor ISch_Label Properties

```
Property FontId                : TFontID
Property Orientation           : TRotationBy90
Property Justification         : TTextJustification
Property Text                  : WideString
Property OverrideDisplayString : WideString
Property DisplayString         : WideString
Property Formula               : WideString
Property CalculatedValueString : WideString
Property IsMirrored            : Boolean
```

Ancestor ISch_ComplexText Properties

```
Property IsHidden : Boolean
```

Immediate ancestor ISch_Parameter Properties

```
Property Name                         : WideString
Property ShowName                     : Boolean
Property ParamType                    : TParameterType
Property ReadOnlyState                : TParameter_ReadOnlyState
Property UniqueId                     : WideString
Property Description                  : WideString
Property AllowLibrarySynchronize      : Boolean
Property AllowDatabaseSynchronize     : Boolean
Property Autoposition                 : Boolean
Property NameIsReadOnly               : Boolean
Property ValueIsReadOnly              : Boolean
Property IsRule                       : Boolean
```

See also

ISch_Parameter interface

Schematic Design Objects overview

ISch_NetLabel

Overview

A net describes a connection from one component pin, to a second pin, and then to a third pin and so on. A net label is a text string with the text property that holds the net name that attachs to a connection such as wires.

Notes

The ISch_NetLabel interface hierachy is as follows:

- **ISch_GraphicalObject**
  - **ISch_Label**
    - **ISch_NetLabel**
- Text property is the net name of the net label.
- ISch_NetLabel itself has no properties or methods but has inherited properties and methods.

Immediate Ancestor ISch_Label Properties

```
Property FontId                   : TFontID
Property Orientation              : TRotationBy90
Property Justification            : TTextJustification
```

```
Property Text                   : WideString
Property OverrideDisplayString  : WideString
Property DisplayString          : WideString
Property Formula                : WideString
Property CalculatedValueString  : WideString
Property IsMirrored             : Boolean
```

Example

```
Procedure PlaceASchNetLabel;
Var
    SchDoc     : ISch_Document;
    WorkSpace  : IWorkSpace;
    SchNetlabel : ISch_Netlabel;
Begin
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('SCH');
    If SchServer = Nil Then Exit;
    SchDoc := SchServer.GetCurrentSchDocument;
    If SchDoc = Nil Then Exit;

    SchNetlabel := SchServer.SchObjectFactory(eNetlabel,eCreate_GlobalCopy);
    If SchNetlabel = Nil Then Exit;
    SchNetlabel.Location    := Point(250, 250);
    SchNetlabel.Orientation := eRotate90;
    SchNetlabel.Text        := 'Netname';
    SchDoc.RegisterSchObjectInContainer(SchNetlabel);
End;
```

See also

ISch_Label interface

Schematic Design Objects overview

PlaceSchObjects script in \Examples\Scripts\Sch folder.

ISch_Parameter

Overview

There are two types of parameters – system parameters which are owned by a schematic document and parameters owned by certain schematic design objects.

A parameter is a child object of a Parameter Set, Part, Pin, Port, or Sheet Symbol object. A Parameter object has a Name property and Value property which can be used to store information, thus the parameters are a way of defining and associating information and could include strings that identify component manufacturer, date added to the document and also a string for the component's value (e.g. 100K for a resistor or 10PF for a capacitor).

Each parameter has a Unique Id assigned to it. This is used for those parameters that have been added as design rule directives. When transferring the design to the PCB document, any defined rule parameters will be used to generate the relevant design rules in the PCB. These generated rules will be given the same Unique Ids, allowing you to change rule constraints in either schematic or PCB and push the change across when performing a synchronization.

Notes

- To look for system wide parameters (not associated with a schematic design object), you would set up an iterator to look for parameters, but you will have to define the iteration depth with the method **SetState_IterationDepth(eIterateFirstLevel).**

Notes

The interface hierarchy for the ISch_Parameter interface is as follows:

- **ISch_GraphicalObject**
  - **ISch_Label**
    - **ISch_ComplexText**
      - **ISch_Parameter**

Ancestor ISch_Label Properties

```
Property FontId                 : TFontID
Property Orientation            : TRotationBy90
Property Justification          : TTextJustification
Property Text                   : WideString
Property OverrideDisplayString : WideString
Property DisplayString          : WideString
Property Formula                : WideString
Property CalculatedValueString : WideString
Property IsMirrored             : Boolean
```

Immediate Ancestor ISch_ComplexText Properties

```
Property IsHidden : Boolean
```

Properties

```
Property Name                      : WideString
Property ShowName                  : Boolean
Property ParamType                 : TParameterType
Property ReadOnlyState             : TParameter_ReadOnlyState
Property UniqueId                  : WideString
Property Description               : WideString
Property AllowLibrarySynchronize   : Boolean
Property AllowDatabaseSynchronize  : Boolean
Property NameIsReadOnly            : Boolean
Property ValueIsReadOnly           : Boolean
Property IsRule                    : Boolean
```

Fetching system (standalone) parameters Example

```
Procedure FetchParameters;
Var
    CurrentSch : ISch_Sheet;
    Iterator   : ISch_Iterator;
    Parameter  : ISch_Parameter;
Begin
    // Check if schematic server exists or not.
    If SchServer = Nil Then Exit;
    // Obtain the current schematic document interface.
    CurrentSch := SchServer.GetCurrentSchDocument;
    If CurrentSch = Nil Then Exit;

    Iterator := CurrentSch.SchIterator_Create;
    // look for stand alone parameters
    Iterator.SetState_IterationDepth(eIterateFirstLevel);
    Iterator.AddFilter_ObjectSet(MkSet(eParameter));

    Try
       Parameter := Iterator.FirstSchObject;
       While Parameter <> Nil Do
       Begin
          // do what you want with the parameter
          Parameter := Iterator.NextSchObject;
```

```
        End;
    Finally
        CurrentSch.SchIterator_Destroy(Iterator);
    End;
End;
```

See also

ISch_ComplexText interface

ISch_Component interface

ISch_ParameterSet itnerface

ISch_Pin interface

ISch_Port interface

ISch_SheetSymbol interface

Schematic Design Objects overview

Examples in the \Scripts\Delphiscript Scripts\Sch folder

ISch_SheetFileName

Overview

A sheet filename is part of a complex text object such as the sheet symbol object.

Notes

The ISch_SheetFileName interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_Label**
    - **ISch_ComplexText**
      - **ISch_SheetFileName**

Ancestor ISch_Label properties

```
Property FontId                : TFontID
Property Orientation           : TRotationBy90
Property Justification         : TTextJustification
Property Text                  : WideString
Property OverrideDisplayString : WideString
Property DisplayString         : WideString
Property Formula               : WideString
Property CalculatedValueString : WideString
Property IsMirrored            : Boolean
```

Immediate ancestor ISch_ComplexText Properties

```
Property IsHidden : Boolean
```

See also

ISch_ComplexText interface

Schematic Design Objects overview

ISch_SheetName

Overview

A sheetname is part of a complex text object such as the sheet symbol object.

Notes

The ISch_SheetName interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_Label**
    - **ISch_ComplexText**
      - **ISch_SheetName**

Ancestor ISch_Label properties

```
Property FontId                : TFontID
Property Orientation           : TRotationBy90
Property Justification         : TTextJustification
Property Text                  : WideString
Property OverrideDisplayString : WideString
Property DisplayString         : WideString
Property Formula               : WideString
Property CalculatedValueString : WideString
Property IsMirrored            : Boolean
```

Immediate ancestor ISch_ComplexText properties

```
Property IsHidden : Boolean
```

See also

ISch_ComplexText interface

Schematic Design Objects overview

ISch_CrossSheetConnector

Overview

Cross sheet connector objects can be used to link a net from a sheet to other sheets within a project. This method defines global connections between sheets within a project.

Notes

The ISch_CrossSheetConnector interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_Label**
    - **ISch_PowerObject**
      - **ISch_CrossSheetConnector**

Ancestor ISch_Label properties

```
Property FontId                 : TFontID
Property Orientation            : TRotationBy90
Property Justification          : TTextJustification
Property Text                   : WideString
Property OverrideDisplayString : WideString
Property DisplayString          : WideString
Property Formula                : WideString
Property CalculatedValueString : WideString
Property IsMirrored             : Boolean
```

Immediate ancestor ISch_PowerObject Properties

```
Property Style : TPowerObjectStyle
```

Properties

```
Property CrossSheetStyle : TCrossSheetConnectorStyle
```

See also

TCrossSheetConnectorStyle enumerated values

ISch_PowerObject interface

Schematic Design Objects overview

ISch_ParametrizedGroup

Overview

The ISch_ParametrizedGroup is an immediate ancestor interface for ParameterSet, Port, Pin, Component and SheetSymbol interfaces. This interface deals with positions of parameters of such objects..

Notes

The ISch_ParameterizedGroup interface is derived from ISch_GraphicalObject interface.

Methods

```
Function     Import_FromUser_Parameters : Boolean;
Procedure    ResetAllSchParametersPosition;
```

See also

ISch_GraphicalObject ancestor interface

ISch_ParameterSet descendent interface

ISch_Port descendent interface

ISch_Pin descendent interface

ISch_Component descendent interface

ISch_RectangularGroup descendent interface

ISch_SheetSymbol descendent interface

Schematic Design Objects overview

ISch_Port

Overview

A port is used to connect a net on one sheet to Ports with the same name on other sheets. Ports can also connect from child sheets to Sheet entries, in the appropriate sheet symbol on the parent sheet. The port cross referencing information for ports on different schematics linked to sheet entries of a sheet symbol can be added to schematic sheets by executing the Reports » Port Cross Reference » Add To Sheet or Add to Project command.

Notes

- To obtain the cross reference field of a port, the design project needs to be compiled first and then port cross-referencing information added to the project or the sheet.
- Port cross references are a calculated attribute of ports, they can not be edited and are not stored with the design.
- The location of each port reference is determined by the location of the port on the sheet and the position of the connecting wire.
- The CrossReference property returns the name of the sheet the port is linked to and the grid where the port is located at. Example : 4 Port Serial Interface [3C].

The ISch_Port hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_ParametrizedGroup**
    - **ISch_Port**

Immediate ancestor ISch_ParametrizedGroup Methods

```
Function     Import_FromUser_Parameters : Boolean;
Procedure    ResetAllSchParametersPosition;
```

Methods

```
Function  IsVertical : Boolean;
```

Properties

```
Property Name             : WideString
Property Style            : TPortArrowStyle
Property IOType           : TPortIO
Property Alignment        : THorizontalAlign
Property TextColor        : TColor
Property Width            : TCoord
Property CrossReference : WideString
Property UniqueId         : WideString
Property ConnectedEnd     : TPortConnectedEnd
```

Example

```
Procedure PlaceASchPort;
Var
    SchDoc     : ISch_Document;
    WorkSpace : IWorkSpace;
    AName        : TDynamicString;
    Orientation : TRotationBy90;
    AElectrical : TPinElectrical;
    SchPort      : ISch_Port;
    Loc          : TLocation;

Begin
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('SCH');
    If SchServer = Nil Then Exit;
    SchDoc := SchServer.GetCurrentSchDocument;
    If SchDoc = 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';
    SchDoc.RegisterSchObjectInContainer(SchPort);
End;
```

See also

TPortArrowStyle enumerated values

TPortIO enumerated values

THorizontalAlign enumerated values

TColor values

TCoord values

TPortConnectedEnd enumerated values

Schematic Design Objects overview

ISch_Pin

Overview

Pins are special objects that have electrical characteristics and are used to direct signals in and out of components. Pins connect directly to other pins, wires, net labels, sheet entries or ports.

Notes

The ISch_Pin interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_ParameterizedGroup**
    - **ISch_Pin**

Immediate ancestor ISch_ParametrizedGroup Methods

```
Function     Import_FromUser_Parameters : Boolean;
Procedure    ResetAllSchParametersPosition;
```

Methods

```
Function   OwnerSchComponent          : ISch_Component;
Function   FullDesignator             : WideString;
```

Properties

```
Property Name              : WideString
Property Designator        : WideString
Property Orientation       : TRotationBy90
Property Width             : Integer
Property FormalType        : TStdLogicState
Property DefaultValue      : WideString
Property Description       : WideString
Property ShowName          : Boolean
Property ShowDesignator    : Boolean
Property Electrical        : TPinElectrical
Property PinLength         : TCoord
Property IsHidden          : Boolean
Property HiddenNetName     : WideString
Property Symbol_Inner      : TIeeeSymbol
Property Symbol_Outer      : TIeeeSymbol
Property Symbol_InnerEdge  : TIeeeSymbol
Property Symbol_OuterEdge  : TIeeeSymbol
Property SwapId_Part       : WideString
Property SwapId_Pin        : WideString
Property SwapId_PartPin    : WideString
Property UniqueId          : WideString
```

See also

TRotationBy90 enumerated values

TStdLogicState enumerated values

TPinElectrical enumerated values

TCoord enumerated values

TIeeeSymbol enumerated values

ISch_ParametrizedGroup interface

Schematic Design Objects overview

ISch_Component

Overview

The **ISch_Component** references a component that can contain links to different model implementations such as PCB, Signal Integrity and Simulation models. Only one model of a particular model type (PCB footprint, SIM, SI, EDIF Macro and VHDL) can be enabled as the currently linked model, at any one time.

Each schematic component has two system parameters – the **Designator** parameter and the **Comment** parameter. Custom parameters can be added anytime. The Comment parameter can be assigned an indirect name parameter. Once a name parameter (with a equal sign character as a prefix to the name parameter) is assigned to the Comment field of the Component properties dialog, the value for this parameter appears on the document, ensure that the **Convert Special Strings** option in the *Schematic Preferences* dialog is enabled

Notes

The ISch_Component interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_ParametrizedGroup**
    - **ISch_Component**
- The Unique ID (UID) is an system generated value that uniquely identifies this current component. It is used for linking to an associated PCB component on a PCB document. Enter a new UID value or click the Reset button to generate a new UID if you wish to force the Schematic component to be linked to a different PCB component. You will need to run the *Component Links...* dialog to update the linkage on the corresponding PCB document.
- This SourceLibraryName property denotes the source library where the symbol and its associated model links are from. The * character in this field denotes the current library of the current project. Note a schematic component is a symbol with a defined designator placed on a schematic document.
- The LibraryRef property is the name of the symbol. The symbol is from the library specified in the Library field below.
- The SheetPartyFilename property, enter a sub design project file name to be linked to the current schematic component. An example of a sub design project is a programmable logic device project or a schematic sub-sheet.

Immediate ancestor ISch_ParametrizedGroup methods

```
Function     Import_FromUser_Parameters : Boolean;
Procedure    ResetAllSchParametersPosition;
```

Methods

```
//Methods Alias
Procedure    Alias_Add    (S : WideString);
Procedure    Alias_Remove(S : WideString);
Procedure    Alias_Delete(i : Integer);
Procedure    Alias_Clear;

//Methods Part & DisplayMode
```

```
Procedure   AddPart;
Procedure   AddDisplayMode;
Procedure   DeletePart       (APartId : Integer);
Procedure   DeleteDisplayMode(AMode   : TDisplayMode);
Function    FullPartDesignator(APartId : Integer) : WideString;

//Methods Implementations
Function    AddSchImplementation : ISch_Implementation;
Procedure   RemoveSchImplementation(AnImplementation : ISch_Implementation)

//Methods Concerning Attributes
Function    IsIntegratedComponent : Boolean;
Function    IsMultiPartComponent  : Boolean;
Function    InSheet   : Boolean;
Function    InLibrary : Boolean;
Procedure UpdatePrimitivesAccessability;
```

Properties

```
Property DisplayMode            : TDisplayMode
Property DisplayModeCount       : Integer
Property CurrentPartID          : Integer
Property PartCount              : Integer
Property ShowHiddenPins         : Boolean
Property DisplayFieldNames      : Boolean
Property Orientation            : TRotationBy90
Property DesignatorLocked       : Boolean
Property PartIdLocked           : Boolean
Property PinsMoveable           : Boolean
Property UniqueId               : WideString
Property PinColor               : TColor
Property OverideColors          : Boolean
Property IsMirrored             : Boolean
Property ShowHiddenFields       : Boolean
Property ComponentKind          : TComponentKind
Property LibraryPath            : WideString
Property SourceLibraryName      : WideString
Property LibReference           : WideString
```

```
Property SheetPartFileName     : WideString
Property TargetFileName        : WideString
Property ComponentDescription : WideString
Property AliasAsText           : WideString
Property Alias[i : Integer]    : WideString
Property AliasCount            : Integer        // Read only
Property Designator : ISch_Designator           // Read only
Property Comment    : ISch_Parameter            // Read only
```

See also

TComponentKind enumerated values

TColor values

TDisplayMode enumerated values

TRotationBy90 enumerated values

ISch_ParametrizedGroup interface

Schematic Design Objects overview

ISch_RectangularGroup

Overview

An ancestor interface for the ISch_SheetSymbol interface.

Notes

The interface hierarchy for the ISch_RectangularGroup interface is as follows:

- **ISch_GraphicalObject**
  - **ISch_ParametrizedGroup**
    - **ISch_RectangularGroup**

Properties

```
Property XSize : TCoord
Property YSize : TCoord
```

See also

ISch_ParametrizedGroup interface

Schematic Design Objects overview

ISch_SheetSymbol

Overview

Sheet symbols represent other schematic sheets (often referred to as a child sheet). The link between a sheet symbol and other schematic sheets is the FileName attribute, which must be the same as the name of the child sheet.

Notes

The ISch_SheetSymbol interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_ParametrizedGroup**
    - **ISch_RectangularGroup**
      - **ISch_SheetSymbol**

Immediate ancestor ISch_RectangularGroup Properties

```
Property XSize : TCoord
Property YSize : TCoord
```

Properties

```
Property UniqueId         : WideString
Property LineWidth        : TSize
Property IsSolid          : Boolean
Property ShowHiddenFields : Boolean
Property SheetFileName    : ISch_SheetFileName
Property SheetName        : ISch_SheetName
```

See also

ISch_RectangularGroup interface

ISch_SheetFileName interface

ISch_SheetName interface

TSize enumerated values

Schematic Design Objects overview

ISch_ParameterSet

Overview

The ISch_ParameterSet interface is a group of parameters as a design parameter set directive for a wire or a net on the schematic document that can be transferred to its corresponding PCB document.

Notes

The ISch_ParameterSet interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_ParametrizedGroup**
    - **ISch_ParameterSet**

Immediate ancestor ISch_ParameterizedGroup interface

```
Function    Import_FromUser_Parameters : Boolean;
Procedure   ResetAllSchParametersPosition;
```

Properties

```
Property Orientation   : TRotationBy90
Property Name          : WideString
```

See also

ISch_ParametrizedGroup interface

Schematic Design Objects overview

ISch_Probe

Overview

A probe is a special marker which is placed on a schematic document to identify nodes for digital simulation.

Notes

The ISch_Probe interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_ParametrizedGroup**
    - **ISch_ParameterSet**
      - **ISch_Probe**

Ancestor ISch_ParameterizedGroup interface

```
Function    Import_FromUser_Parameters : Boolean;
Procedure   ResetAllSchParametersPosition;
```

Immediate ancestor ISch_ParameterSet Properties

```
Property Orientation   : TRotationBy90
Property Name          : WideString
```

See also

ISch_ParameterSet interface

Schematic Design Objects overview

ISch_Polygon

Overview

Polygons are multi-sided graphical elements. The vertices of a polygon object denote the link of lines to describe its outline.

Notes

The ISch_Polygon interface is descended from the ancestor ISch_GraphicalObject interface.

Methods

```
Function  InsertVertex (    Index : Integer) : Boolean;
Function  RemoveVertex (Var Index : Integer) : Boolean;
Procedure ClearAllVertices;
```

Properties

```
Property IsSolid   : Boolean
Property LineWidth : TSize
Property Vertex[i : Integer] : TLocation
Property VerticesCount : Integer
Property Transparent   : Boolean
```

See also

ISch_GraphicalObject interface

ISch_Polyline interface

ISch_Wire interface

ISch_Bus interface

TLocation values

TSize enumerated values

Schematic Design Objects overview

ISch_Polyline

Overview

Lines are graphical drawing objects with any number of joined segments.

Notes

The ISch_Polyline Interface is as follows:

- **ISch_GraphicalObject**
  - **ISch_Polygon**
    - **ISch_Polyline**

Immediate Ancestor ISch_Polygon Methods

```
Function  InsertVertex (    Index : Integer) : Boolean;
Function  RemoveVertex (Var Index : Integer) : Boolean;
Procedure ClearAllVertices;
```

Immediate Ancestor ISch_Polygon Properties

```
Property IsSolid   : Boolean
Property LineWidth : TSize
Property Vertex[i : Integer] : TLocation
Property VerticesCount : Integer
```

Properties

```
Property LineStyle : TLineStyle
```

See also

ISch_Polygon interface

TLineStyle enumerated values

Schematic Design Objects overview

ISch_Bezier

Overview

A bezier curve is used to create curved line shapes (For example a section of a sine wave or a pulse). At least four points are required to define a bezier curve. More than four points used will define another bezier curve and so on.

Notes

The interface ancestors for the ISch_Bezier

- **ISch_GraphicalObject**
  - **ISch_Polygon**
    - **ISch_Polyline**
      - **ISch_Bezier**

Ancestor ISch_Polygon Methods

```
Function  InsertVertex (    Index : Integer) : Boolean;
Function  RemoveVertex (Var Index : Integer) : Boolean;
Procedure ClearAllVertices;
```

Ancestor ISch_Polygon Properties

```
Property IsSolid   : Boolean
Property LineWidth : TSize
```

```
Property Vertex[i : Integer] : TLocation
Property VerticesCount : Integer
```

Immediate ancestor ISch_Polyline Properties

```
LineStyle : TLineStyle
```

See also

ISch_Polyline interface

Schematic Design Objects overview

ISch_Wire

Overview

Wires are straight line segments which are placed on a schematic document to create the electrical connections.

Notes

The ISchWire is descended fromthe immediate ancestor ISch_Polyline interface and the interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISchPolygon**
    - **ISch_Polyline**
      - **ISch_Wire**

Ancestor ISch_Polygon Methods

```
Function  InsertVertex (    Index : Integer) : Boolean;
Function  RemoveVertex (Var Index : Integer) : Boolean;
Procedure ClearAllVertices;
```

Ancestor ISch_Polygon Properties

```
Property IsSolid   : Boolean
Property LineWidth : TSize
Property Vertex[i : Integer] : TLocation
Property VerticesCount : Integer
```

Immediate ancestor ISch_Polyline Properties

```
Property LineStyle : TLineStyle
```

ISch_Wire Properties

```
Property CompilationMaskedSegment[AIndex : Integer] : Boolean
```

Example

```
Function SortVertices(WireVertices : String) : Integer;
Var
   NewValue : String;
Begin
     //X1=454|Y1=454|X2=454|Y2=345|X2=354|Y2=456|....
     If POS('|', WireVertices) > 0 Then
     Begin
          NewValue := copy(WireVertices, pos('=', WireVertices) + 1,
pos('|', WireVertices) - pos('=', WireVertices) - 1);
          result := NewValue;
     End;
End;
{.........................................................................
...}
{.........................................................................
...}
Function VerticesTrim(WireVertices : String) : String;
Var
   NewValue : String;
Begin
     If POS('|', WireVertices) > 0 Then
     Begin
          Delete(WireVertices, 1, pos('|', WireVertices));
          Result := WireVertices;
     End;
End;
{.........................................................................
...}
{.........................................................................
...}
Procedure PlaceASchWire(NumberOfVertices : Integer, Vertices : String,
LineWidth : TSize);
Var
   ScriptParametres : String;
   SchWire          : ISch_Wire;
   I                : Integer;
```

```
   X                 : Integer;
   Y                 : Integer;
   WireVertices      : String;
Begin
     SchWire := SchServer.SchObjectFactory(eWire,eCreate_GlobalCopy);
     If SchWire = Nil Then Exit;
     // Number of vertices. Always 2 for a single wire
     WireVertices := Vertices;
     X := SortVertices(WireVertices);
     WireVertices := VerticesTrim(WireVertices);
     Y := SortVertices(WireVertices);
     WireVertices := VerticesTrim(WireVertices);
     // Set the line width based on TSize type
     SchWire.SetState_LineWidth := LineWidth;
     // Starting point for the vertex
     Schwire.Location := Point(X, Y);
     Schwire.InsertVertex := 1;
     SchWire.SetState_Vertex(1, Point(X, Y));
     For I := 2 to NumberOfVertices Do
     Begin
          Schwire.InsertVertex := I;
          X                    := SortVertices(WireVertices);
          WireVertices         := VerticesTrim(WireVertices);
          Y                    := SortVertices(WireVertices);
          WireVertices         := VerticesTrim(WireVertices);
          SchWire.SetState_Vertex(I, Point(X, Y));
     End;
     SchDoc.RegisterSchObjectInContainer(SchWire);
End;
{............................................................................
...}
{............................................................................
...}
Procedure PlaceWires;
Var
    SchDoc    : ISch_Document;
    WorkSpace : IWorkSpace;
```

```
Begin
    WorkSpace := GetWorkSpace;
    If WorkSpace = Nil Then Exit;
    Workspace.DM_CreateNewDocument('SCH');
    If SchServer = Nil Then Exit;
    SchDoc := SchServer.GetCurrentSchDocument;
    If SchDoc = Nil Then Exit;

    PlaceASchWire(2, 'X1=200|Y1=200|X2=250|Y2=300|', eSmall);
    PlaceASchWire(2, 'X1=250|Y1=300|X2=300|Y2=200|', eMedium);
    PlaceASchWire(2, 'X1=300|Y1=200|X2=200|Y2=200|', eLarge);
End.
```

See also

ISch_Polyline interface

Schematic Design Objects overview

ISch_Bus

Overview

Buses are special graphical elements that represent a common pathway for multiple signals on a schematic document. Buses have no electrical properties, and they must be correctly identified by net labels and ports.

Notes

ISch_Bus Interface ancestors are:

- **ISch_GraphicalObject**
  - **ISch_Polygon**
    - **ISch_Polyline**
      - **ISch_Wire**
        - **ISch_Bus**
- Note that the ISch_Wire interface has no extra properties and methods but has inherited properties and methods only.

Ancestor ISch_Polygon Methods

```
Function  InsertVertex (    Index : Integer) : Boolean;
Function  RemoveVertex (Var Index : Integer) : Boolean;
Procedure ClearAllVertices;
```

Ancestor ISch_Polygon Properties

```
Property IsSolid   : Boolean
Property LineWidth : TSize
Property Vertex[i : Integer] : TLocation
Property VerticesCount : Integer
```

Immediate ancestor ISch_Polyline Properties

```
Property LineStyle : TLineStyle
```

See also

ISch_Wire interface

Schematic Design Objects overview

ISch_Rectangle

Overview

Rectangles are drawing objects which are unfilled or filled graphic elements.

Notes

The ISch_Rectangle interface is derived from ISch_GraphicalObject interface.

Rectangle Properties

```
Property Corner      : TLocation
Property LineWidth   : TSize
Property IsSolid     : Boolean
Property Transparent : Boolean
```

See also

Schematic Design Objects overview

ISch_GraphicalObject interface

ISch_Image interface

ISch_RoundRectangle interface

ISch_TextFrame interface

ISch_Image

Overview

Graphical Images are used to represent images.

Important Notes

The ISch_Image interface hierarchy is as follows:

- **ISch_GraphicalObject**

- **ISch_Rectangle**
  - **ISch_Image**

Immediate ancestor ISch_Rectangle properties

```
Property Corner    : TLocation
Property LineWidth : TSize
Property IsSolid   : Boolean
```

Properties

```
Property EmbedImage : Boolean
Property FileName   : WideString
Property KeepAspect : Boolean
```

See also

ISch_Rectangle interface

Schematic Design Objects overview

ISch_RoundRectangle

Overview

Rounded rectangles are drawing objects which are unfilled or filled graphic elements.

Notes

The ISch_RoundRectangle interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_Rectangle**
    - **ISch_RoundRectangle**

Immediate ancestor ISch_Rectangle properties

```
Property Corner    : TLocation
Property LineWidth : TSize
Property IsSolid   : Boolean
```

Properties

```
Property CornerXRadius  : TDistance
Property CornerYRadius  : TDistance
```

See also

ISch_Rectangle interface

Schematic Design Objects overview

ISch_TextFrame

Overview

Text frames hold multiple lines of free text.

Notes

ISch_TextFrame interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_Rectangle**
    - **ISch_TextFrame**
- The FontID denotes the font type of the TextFrame object. Windows True Type fonts are fully supported. The FontID value denotes which font has been used. The FontID is the index to an entry in the font table in the Schematic editor. Each font used in the Schematic editor has its own FontID. When a new font is used (through a Change Font dialog of a Change object dialog), a new FontID is added to the internal table in the Schematic editor. The FontID value can be extracted from the following Schematic objects (TextField, Sheet, Annotation, TextFrame and NetLabel objects)..

Immediate ancestor ISch_Rectangle properties

```
Property Corner    : TLocation
Property LineWidth : TSize
Property IsSolid   : Boolean
```

Properties

```
Property FontId       : Integer
Property TextColor    : TColor
Property Alignment    : THorizontalAlign
Property WordWrap     : Boolean
Property ShowBorder   : Boolean
Property ClipToRect   : Boolean
Property Text         : WideString
```

See also

ISch_Rectangle interface

THorizontalAlign enumerated values

Schematic Design Objects overview

ISch_CompileMask interface

Overview

CompileMask hold multiple lines of free text that can be collapsed or not.

Notes

ISch_TextFrame interface hierarchy is as follows:

- **ISch_GraphicalObject**
  - **ISch_Rectangle**
    - **ISch_CompileMask**

Immediate ancestor ISch_Rectangle properties

```
Property Corner    : TLocation
Property LineWidth : TSize
Property IsSolid   : Boolean
```

Properties

```
Property Collapsed      : Boolean
```

See also

ISch_Rectangle interface

THorizontalAlign enumerated values

Schematic Design Objects overview

Schematic Enumerated Types

The enumerated types are used for many of the schematic interfaces methods which are covered in this section. For example the ISch_Port interface has a ConnectedEnd : TPortConnectedEnd property. You can use this Enumerated Types section to check what the range is for the TPortConnectedEnd type.

See also

Schematic API Reference

TColor

TCoord

TCoordRect

TConnectivityScope

TCrossSheetConnectorStyle

TCursorShape

TDistance

TFontID

THorizontalAlign

TIeeeSymbol

TIterationDepth

TLeftRightSide

TLineStyle

TLocation

TObjectCreationMode

TObjectId

TParameterType

TPinElectrical

TPortArrowStyle

TPortConnectedEnd

TPortIO

TPowerObjectStyle

TRotationBy90

TSheetDocumentBorderStyle

TSheetOrientation

TSheetStyle

TStdLogicState

TSize

TTextJustification

TVisibleGrid

TAngle (Sch)

```
TAngle = TReal;
```

TAutoPanStyle

```
TAutoPanStyle = (
    eAutoPanOff,
    eAutoPanFixedJump,
    eAutoPanReCenter
     );
```

TColor

```
TColor  = TColorRef;
```

Notes

The TColor value specifies a 6 digit hexadecimal number of the $FFFFFF format. For example the color blue would be RGB:0,0,255 and Hex:FF0000 therefore the converted decimal value would be 16711680. The following formula may be used to calculate the required value, R+256*(G+(256*B)).

Examples: Color=0 is black, Color=255 is red, Color=65280 is green Color=16711680 is blue Color=16777215 is white. Decimal or hexadecimal values can be assigned.

TComponentDisplay

```
TComponentDisplay = (
    eCompBlock,
    eCompDevice,
    eCompPower,
    eCompSymbol
     );
```

TCoord

```
TCoord = Integer;
```

Note

You can use MilsToCoord, MMsToCoord and CoordToMMs, CoordToMils functions to convert a value unit to a different value unit.

See also

PCB Functions

TCoordRect (Sch)

```
TCoordRect                    = Record
        Case Integer of
           0 :(left, bottom, right, top : TCoord);
           1 :(x1,   y1,     x2,    y2  : TCoord);
           2 :(Location1,    Location2  : TLocation);
End;
```

TConnectionNodeType

```
TConnectionNodeType = (eConnectionNode_IntraSheetLink,
eConnectionNode_InterSheetLink, eConnectionNode_Hidden);
```

TConnectivityScope

```
TConnectivityScope = (eConnectivity_ConnectionOnly, eConnectivity_WholeNet);
```

TCrossSheetConnectorStyle

```
TCrossSheetConnectorStyle = (
        eCrossSheetLeft,
        eCrossSheetRight
    );
```

TCursorMove

```
TCursorMove = (
    eCursorLeft,
    eCursorRight,
    eCursorTop,
    eCursorBottom
     );
```

TCursorShape

```
TCursorShape = (
       eLargeCursor90,
       eSmallCursor90,
       eSmallCursor45,
       eTinyCursor45
        );
```

TDistance

```
TDistance = Integer;
```

TDrawMode

```
TDrawMode = (
    eDrawFull,
    eDrawDraft,
    eDrawHidden
     );
```

TDrawQuality

```
TDrawQuality = (eFullQuality,eDraftQuality);
```

TEditingAction (Sch)

```
TEditingAction = (eEditAction_DontCare, eEditAction_Move,
eEditAction_Change,eEditAction_Delete,eEditAction_Select);
```

TFileName

```
TFileName = TString;
```

TFontID

```
TFontID = Integer;
```

TFontName

```
TFontName = String[lf_FaceSize + 1];
```

TGridPreset

```
TGridPreset = (eDXPPreset, eCoarse2, eCoarse3, eFine2, eFine3, eElectrical);
```

THitTestMode

```
THitTestMode = (
       eHitTest_AllObjects,
       eHitTest_OnlyAccessible
       );
```

THitTestResult

```
THitTestResult   = (eHitTest_Fail,
                    eHitTest_NoAction,
                    eHitTest_Move,
```

```
                    eHitTest_InPlaceEdit,
                    eHitTest_CopyPaste,
                    eHitTest_Resize_Any,
                    eHitTest_Resize_EndAngle,
                    eHitTest_Resize_StartAngle,
                    eHitTest_Resize_SecondaryRadius,
                    eHitTest_Resize_Radius,
                    eHitTest_Resize_CornerTopLeft,
                    eHitTest_Resize_CornerTopRight,
                    eHitTest_Resize_CornerBottomRight,
                    eHitTest_Resize_CornerBottomLeft,
                    eHitTest_Resize_SideLeft,
                    eHitTest_Resize_SideRight,
                    eHitTest_Resize_SideTop  ,
                    eHitTest_Resize_SideBottom,
                    eHitTest_Resize_Vertical,
                    eHitTest_Resize_Horizontal,
                    eHitTest_Resize_SE_NW,
                    eHitTest_Resize_SW_NE);
```

THorizontalAlign

```
THorizontalAlign = (
    eHorizontalCentreAlign, // eVerticalCentreAlign
    eLeftAlign,             // eTopAlign
    eRightAlign             // eBottomAlign
     );
```

TIeeeSymbol

```
TIeeeSymbol = (
    eNoSymbol,
    eDot,
    eRightLeftSignalFlow,
    eClock,
    eActiveLowInput,
    eAnalogSignalIn,
    eNotLogicConnection,
    eShiftRight,
```

```
    ePostPonedOutput,
    eOpenCollector,
    eHiz,
    eHighCurrent,
    ePulse,
    eSchmitt,
    eDelay,
    eGroupLine,
    eGroupBin,
    eActiveLowOutput,
    ePiSymbol,
    eGreaterEqual,
    eLessEqual,
    eSigma,
    eOpenCollectorPullUp,
    eOpenEmitter,
    eOpenEmitterPullUp,
    eDigitalSignalIn,
    eAnd,
    eInvertor,
    eOr,
    eXor,
    eShiftLeft,
    eInputOutput,
    eOpenCircuitOutput,
    eLeftRightSignalFlow,
    eBidirectionalSignalFlow
     );
```

IeeeSymbolPosition

```
TIeeeSymbolPosition = (eInner, eInnerEdge, eOuterEdge, eOuter);
```

TIterationDepth

```
TIterationDepth     = (eIterateFirstLevel, eIterateFilteredLevels,
eIterateAllLevels);
```

TLeftRightSide

```
TLeftRightSide = (
    eLeftSide,
    eRightSide,
    eTopSide,
    eBottomSide
     );
```

TLinePlaceMode

```
TLinePlaceMode  = (eLineAnyAngle,
                   eLine90Start,
                   eLine90End,
                   eLine45Start,
                   eLine45End,
                   eLineArcStart,
                   eLineArcEnd,
                   eAutoWire );
```

TLineStyle

```
   TLineStyle = (
       eLineStyleSolid,
       eLineStyleDashed,
       eLineStyleDotted
        );
```

TLocation

```
TLocation = TPoint;
```

Where the TPoint = packed record X: Longint; Y: Longint;end;

TMyRect

```
TMyRect = Record
    Left,Right,Top, Bottom, Width, Height : Integer;
End;
```

TObjectCreationMode (Sch)

```
TObjectCreationMode = (eCreate_Default, eCreate_GlobalCopy);
```

TObjectId (Sch)

```
TObjectId     = (eFirstObjectID,
                  eClipBoardContainer,
                  eNote,
                  eProbe,
                  eRectangle,
                  eLine,
                  eConnectionLine,
                  eBusEntry,
                  eArc,
                  eEllipticalArc,
                  eRoundRectangle,
                  eImage,
                  ePie,
                  eTextFrame,
                  eEllipse,
                  eJunction,
                  ePolygon,
                  ePolyline,
                  eWire,
                  eBus,
                  eBezier,
                  eLabel,
                  eNetLabel,
                  eDesignator,
                  eSchComponent,
                  eParameter,
                  eParameterSet,
                  eParameterList,
                  eSheetName,
                  eSheetFileName,
                  eSheet,
                  eSchLib,
                  eSymbol,
                  eNoERC,
                  eErrorMarker,
```

```
    ePin,
    ePort,
    ePowerObject,
    eSheetEntry,
    eSheetSymbol,
    eTemplate,
    eTaskHolder,
    eMapDefiner,
    eImplementationMap,
    eImplementation,
    eImplementationsList,
    eCrossSheetConnector,
    eCompileMark,
    eLastObjectId
   );
```

TObjectAttribute

```
TObjectAttribute = (eObjectAttribute_ObjectId,
                    eObjectAttribute_DocumentName,
                    eObjectAttribute_Color,
                    eObjectAttribute_TextColor,
                    eObjectAttribute_AreaColor,
                    eObjectAttribute_LocationX,
                    eObjectAttribute_LocationY,
                    eObjectAttribute_CornerLocationX,
                    eObjectAttribute_CornerLocationY,
                    eObjectAttribute_OwnerPartId,
                    eObjectAttribute_OwnerPartDisplayMode,
                    eObjectAttribute_Width,
                    eObjectAttribute_Radius,
                    eObjectAttribute_Solid,
                    eObjectAttribute_Transparent,
                    eObjectAttribute_StartAngle,
                    eObjectAttribute_EndAngle,
                    eObjectAttribute_SecondaryRadius,
                    eObjectAttribute_StringText,
```

```
eObjectAttribute_LongStringText,
eObjectAttribute_LineStyle,
eObjectAttribute_IsHidden,
eObjectAttribute_FontId,
eObjectAttribute_Orientation,
eObjectAttribute_HorizontalJustification,
eObjectAttribute_VerticalJustification,
eObjectAttribute_TextHorizontalAnchor,
eObjectAttribute_TextVerticalAnchor,
eObjectAttribute_Alignment,
eObjectAttribute_BorderWidth,
eObjectAttribute_LineWidth,
eObjectAttribute_JunctionSize,
eObjectAttribute_Locked,
eObjectAttribute_Accessible,
eObjectAttribute_Name,
eObjectAttribute_OwnerName,
eObjectAttribute_Description,
eObjectAttribute_ShowName,
eObjectAttribute_IsMirrored,
eObjectAttribute_DesignatorLocked,
eObjectAttribute_PartIdLocked,
eObjectAttribute_PinsMoveable,
eObjectAttribute_FileName,
eObjectAttribute_TargetFileName,
eObjectAttribute_ImageKeepAspect,
eObjectAttribute_ImageEmbed,
eObjectAttribute_ParametersList,
eObjectAttribute_ParameterValue,
eObjectAttribute_ParameterName,
eObjectAttribute_ParameterType,
eObjectAttribute_ParameterReadOnlyState,
eObjectAttribute_ParameterAllowLibrarySynchronize,
eObjectAttribute_ParameterAllowDatabaseSynchronize,
eObjectAttribute_TextAutoposition,
eObjectAttribute_PinWidth,
```

```
eObjectAttribute_PinFormalType,
eObjectAttribute_PinDefaultValue,
eObjectAttribute_PinDesignator,
eObjectAttribute_PinHiddenNetName,
eObjectAttribute_PinShowDesignator,
eObjectAttribute_PinElectrical,
eObjectAttribute_PinLength,
eObjectAttribute_PinIeeeSymbolInner,
eObjectAttribute_PinIeeeSymbolOuter,
eObjectAttribute_PinIeeeSymbolInnerEdge,
eObjectAttribute_PinIeeeSymbolOuterEdge,
eObjectAttribute_PinSwapId_Pin,
eObjectAttribute_PinSwapId_Part,
eObjectAttribute_PinSwapId_PartPin,
eObjectAttribute_PortArrowStyle,
eObjectAttribute_PortIOType,
eObjectAttribute_PowerObjectStyle,
eObjectAttribute_CrossSheetConnectorStyle,
eObjectAttribute_RoundRectangleCornerRadiusX,
eObjectAttribute_RoundRectangleCornerRadiusY,
eObjectAttribute_SchComponentLibraryName,
eObjectAttribute_SchComponentLibReference,
eObjectAttribute_SchComponentDesignator,
eObjectAttribute_SchComponentDisplayMode,
eObjectAttribute_SchComponentPartId,
eObjectAttribute_SchComponentComment,
eObjectAttribute_SchComponentFootprint,
eObjectAttribute_SchComponentKind,
eObjectAttribute_ShowHiddenFields,
eObjectAttribute_ShowHiddenPins,
eObjectAttribute_ShowDesignator,
eObjectAttribute_SheetFileName,
eObjectAttribute_SheetName,
eObjectAttribute_SheetEntrySide,
eObjectAttribute_SheetEntryDistanceFromTop,
eObjectAttribute_IeeeSymbol,
```

```
                    eObjectAttribute_SymbolScaleFactor,
                    eObjectAttribute_TaskHolderProcess,
                    eObjectAttribute_TaskHolderInstanceName,
                    eObjectAttribute_TaskHolderConfiguration,
                    eObjectAttribute_TextFrameWordWrap,
                    eObjectAttribute_TextFrameShowBorder,
                    eObjectAttribute_TextFrameClipToRect,
                    eObjectAttribute_Author,
                    eObjectAttribute_Collapsed,
                    eObjectAttribute_ErrorKind);
```

TOrcadFootprint

```
TOrcadFootPrint = (
    ePartfield1,
    ePartfield2,
    ePartfield3,
    ePartfield4,
    ePartfield5,
    ePartfield6,
    ePartfield7,
    ePartfield8,
    eIgnore);
```

TParameter_ReadOnlyState

```
TParameter_ReadOnlyState = (
       eReadOnly_None,
       eReadOnly_Name,
       eReadOnly_Value,
       eReadOnly_NameAndValue
       );
```

TParameterType

```
TParameterType = (eParameterType_String,
                  eParameterType_Boolean,
                  eParameterType_Integer,
                  eParameterType_Float);
```

TPinElectrical

```
TPinElectrical = (
       eElectricInput,
       eElectricIO,
       eElectricOutput,
       eElectricOpenCollector,
       eElectricPassive,
       eElectricHiZ,
       eElectricOpenEmitter,
       eElectricPower);
```

TPlacementResult

```
TPlacementResult =
(eSingleObjectPlacementProcessAborted,eWholeObjectPlacementAborted,
eObjectPlacementSuccessfull);
```

TPlacementMode

```
TPlacementMode   = (ePlacementMode_Single, ePlacementMode_Multiple);
```

TPolylineCutterMode

```
TPolylineCutterMode    = (eCutterSnapToSegment, eCutterGridSize,
eCutterFixedLength);
```

TPortArrowStyle

```
TPortArrowStyle = (
       ePortNone,
       ePortLeft,
       ePortRight,
       ePortLeftRight,
       ePortNoneVertical,
       ePortTop,
       ePortBottom,
       ePortTopBottom
        );
```

TPortConnectedEnd

```
TPortConnectedEnd = (
    ePortConnectedEnd_None,
```

```
    ePortConnectedEnd_Origin,     //connected at port Location
    ePortConnectedEnd_Extremity,  //connected at the other end
    ePortConnectedEnd_Both        //connected at both ends
     );
```

TPortIO

```
TPortIO = (
    ePortUnspecified,
    ePortOutput,
    ePortInput,
    ePortBidirectional
     );
```

TPowerObjectStyle

```
TPowerObjectStyle = (
       ePowerCircle,
       ePowerArrow,
       ePowerBar,
       ePowerWave,
       ePowerGndPower,
       ePowerGndSignal,
       ePowerGndEarth
        );
```

TProbeMethod

```
TProbeMethod = (
    eProbeMethodAllNets,
    eProbeMethodProbedNetsOnly
     );
```

TPrintKind

```
TPrintKind =
(ePrintKind_FullColor,ePrintKind_GrayScale,ePrintKind_Monochrome);
```

TReal (Sch)

```
TReal = Double;
```

TRectangleStyle

```
TRectangleStyle = (
    eRectangleHollow,
    eRectangleSolid
     );
```

TRotationBy90

```
TRotationBy90 =
    eRotate0,
    eRotate90,
    eRotate180,
    eRotate270
     );
```

TSchDropAction

```
TSchDropAction = (eDropAction_None,
                  eDropAction_AskOpenOrInsertText,
                  eDropAction_WarnBinaryAsText,
                  eDropAction_OpenInEditor,
                  eDropAction_OpenAsText,
                  eDropAction_Insert);
```

TSelectionMatch

```
TypeTSelectionMatch = (
    eMatchSelected,
    eMatchedNotSelected,
    eMatchAnySelection
     );
```

TSelectionState

```
TSelectionState  = (eSelectionState_None,
                    eSelectionState_FirstSelected,
                    eSelectionState_MultiSelected);
```

TSheetDocumentBorderStyle

```
TSheetDocumentBorderStyle = (
    eSheetStandard,
```

```
    eSheetAnsi
     );
```

TSheetOrientation

```
TSheetOrientation = (
    eLandscape,
    ePortrait
     );
```

TSheetStyle

```
TSheetStyle = (
    eSheetA4,
    eSheetA3,
    eSheetA2,
    eSheetA1,
    eSheetA0,
    eSheetA,
    eSheetB,
    eSheetC,
    eSheetD,
    eSheetE,
    eSheetLetter,
    eSheetLegal,
    eSheetTabloid,
    eSheetOrcadA,
    eSheetOrcadB,
    eSheetOrcadC,
    eSheetOrcadD,
    eSheetOrcadE
     );
```

TShowCutterBoxMode

```
TShowCutterBoxMode     = (eBoxNever, eBoxAlways, eBoxOnPolyline);
```

TShowCutterMarkersMode

```
TShowCutterMarkersMode = (eMarkersNever, eMarkersAlways,
eMarkersOnPolyline);
```

TSide

```
TSide = (
    eLeft,
    eBottom,
    eRight,
    eTop
     );
```

TSignalLayer

```
TSignalLayer = (
    eNoSignalLayer,
    eTopSignalLayer,
    eMidSignalLayer1,
    eMidSignalLayer2,
    eMidSignalLayer3,
    eMidSignalLayer4,
    eMidSignalLayer5,
    eMidSignalLayer6,
    eMidSignalLayer7,
    eMidSignalLayer8,
    eMidSignalLayer9,
    eMidSignalLayer10,
    eMidSignalLayer11,
    eMidSignalLayer12,
    eMidSignalLayer13,
    eMidSignalLayer14,
    eBottomSignalLayer,
    eMultiSignalLayer,
    ePowerLayer1,
    ePowerLayer2,
    ePowerLayer3,
    ePowerLayer4
     );
```

TSize

```
TSize = (
    eZeroSize,
```

```
    eSmall,
    eMedium,
    eLarge
     );
```

TStdLogicState

```
TStdLogicState = (eStdLogic_Unitialized,
                  eStdLogic_ForcingUnknown,
                  eStdLogic_Forcing0,
                  eStdLogic_Forcing1,
                  eStdLogic_HiZ,
                  eStdLogic_WeakUnknown,
                  eStdLogic_Weak0,
                  eStdLogic_Weak1,
                  eStdLogic_DontCare);
```

TTextHorzAnchor

```
TTextHorzAnchor = (
       eTextHorzAnchor_None,
       eTextHorzAnchor_Both,
       eTextHorzAnchor_Left,
       eTextHorzAnchor_Right
       );
```

TTextVertAnchor

```
TTextVertAnchor = (
       eTextVertAnchor_None,
       eTextVertAnchor_Both,
       eTextVertAnchor_Top,
       eTextVertAnchor_Bottom
       );
```

TTextJustification

```
TTextJustification = (
    eJustify_BottomLeft,
    eJustify_BottomCenter,
    eJustify_BottomRight,
```

```
    eJustify_CenterLeft,
    eJustify_Center,
    eJustify_CenterRight,
    eJustify_TopLeft,
    eJustify_TopCenter,
    eJustify_TopRight
    );
```

TUpperLowerCase

```
TUpperLowerCase = (eUpperCase, eLowerCase, eAnyCase);
```

TUnit

```
TUnit = (eMil, eMM, eIN, eCM, eDXP, eM, eAutoImperial, eAutoMetric);
```

TUnitSet

```
TUnitSet = Set Of TUnit;
```

TUnitSystem

```
TUnitSystem = (eImperial, eMetric);
```

TVerticalAlign

```
TVerticalAlign = (
    eVerticalCentreAlign,
    eTopAlign,
    eBottomAlign
     );
```

TVHOrientation

```
THVOrientation = (
    eHorizontal,
    eVertical
     );
```

TVisibleGrid

```
TVisibleGrid = (
    eDotGrid,
    eLineGrid
     );
```

TWidthArray

```
TWidthArray = Array [TSize] of Integer;
```

Schematic Constants

Millimetre and Internal Unit constants

Notes

Each Millimetre constant value is expressed in internal units (rounded to nearest integer value).

```
c0_25MM = 98425;
c0_50MM = 196850;
c0_75MM = 295275;
c1_00MM = 393701;
c1_5MM  = 590551;
c2_0MM  = 787402;
c2_5MM  = 984252;
c3_0MM  = 1181102;
c3_5MM  = 1377953;
c4_0MM  = 1574803;
c4_5MM  = 1771654;
c5_0MM  = 1968504;
c5_5MM  = 2165354;
c6_0MM  = 2362205;
c6_5MM  = 2559055;
c7_0MM  = 2755906;
c7_5MM  = 2952756;
c8_0MM  = 3149606;
c8_5MM  = 3346457;
c9_0MM  = 3543307;
c9_5MM  = 3740157;
c10_0MM = 3937008;
c15_0MM = 5905512;
c20_0MM = 7874016;
c25_0MM = 9842520;
c30_0MM = 11811024;
c35_0MM = 13779528;
c40_0MM = 15748031;
```

```
c45_0MM = 17716535;
c50_0MM = 19685039;
c55_0MM = 21653543;
c60_0MM = 23622047;
c65_0MM = 25590551;
c70_0MM = 27559055;
c75_0MM = 29527559;
c80_0MM = 31496063;
c85_0MM = 33464567;
c90_0MM = 35433071;
c95_0MM = 37401575;
c100_0MM  = 39370078;
c1000_0MM = 393700787;
```

Power Object constants

```
cPowerObjectLineWidth   = 1 * cBaseUnit;
cPowerGndPowerXOffset1 = 0 * cBaseUnit;
cPowerGndPowerXOffset2 = 3 * cBaseUnit;
cPowerGndPowerXOffset3 = 6 * cBaseUnit;
cPowerGndPowerXOffset4 = 9 * cBaseUnit;
cPowerGndPowerYOffset1 = 10 * cBaseUnit;
cPowerGndPowerYOffset2 = 7  * cBaseUnit;
cPowerGndPowerYOffset3 = 4  * cBaseUnit;
cPowerGndPowerYOffset4 = 1  * cBaseUnit;
cPowerNameXOffset1      = 2  * cBaseUnit;
```

Parameter Set constants

```
cParameterSetLineWidth          = 1   *cBaseUnit;
cParameterSetLineLength         = 6   *cBaseUnit;
cParameterSetCircleRadius       = 6   *cBaseUnit;
cParameterSetCircleCenterOffset = 12  *cBaseUnit;
cParameterSetIOffsetX           = 12  *cBaseUnit;
cParameterSetIOffsetY           = 5   *cBaseUnit;
cParameterSetTextOffetX         = 20  *cBaseUnit;
cParameterSetParamDefaultLength = 5   *cBaseUnit;
cParameterSetParam000XOffset    = 32  *cBaseUnit;
cParameterSetParam090XOffset    = 4   *cBaseUnit;
```

```
cParameterSetParam090YOffset    = 24  *cBaseUnit;
cParameterSetParam180XOffset    = 12   *cBaseUnit
cParameterSetParam270XOffset    = 10   *cBaseUnit
cParameterSetParam270YOffset    = 22  *cBaseUnit;
cParameterSetParamYOffset       = 2   *cBaseUnit;
cParameterSetParamDeltaYOffset1 = 12  *cBaseUnit;
```

Title Block constants

```
cTitleBlockWidth               = 350 *cBaseUnit;
cTitleBlockWidth1              = 100 *cBaseUnit;
cTitleBlockWidth2              = 150 *cBaseUnit;
cTitleBlockWidth3              = 300 *cBaseUnit;
cTitleBlockHeight              = 80  *cBaseUnit;
cTitleBlockHeight1             = 50  *cBaseUnit;
cTitleBlockHeight2             = 20  *cBaseUnit;
cTitleBlockHeight3             = 10  *cBaseUnit;
cTitleBlockTextXPos_Title      = 345 *cBaseUnit;
cTitleBlockTextXPos_Number     = 295 *cBaseUnit;
cTitleBlockTextXPos_Revision   = 95  *cBaseUnit;
cTitleBlockTextXPos_Size       = 345 *cBaseUnit;
cTitleBlockTextXPos_SheetStyle = 340 *cBaseUnit;
cTitleBlockTextYPos_SheetStyle = 35  *cBaseUnit;
cTitleBlockTextXPos_Date1      = 345 *cBaseUnit;
cTitleBlockTextXPos_Date2      = 300 *cBaseUnit;
cTitleBlockTextXPos_SheetNbr   = 145 *cBaseUnit;
cTitleBlockTextXPos_File1      = 345 *cBaseUnit;
cTitleBlockTextXPos_File2      = 300 *cBaseUnit;
cTitleBlockTextXPos_DrawnBy    = 145 *cBaseUnit;
cTitleBlockTextYPos_TextLine1  = 20  *cBaseUnit;
cTitleBlockTextYPos_TextLine2  = 10  *cBaseUnit;
cAnsiTitleBlock1               = 175 *cBaseUnit;
cAnsiTitleBlock2               = 625 *cBaseUnit;
cAnsiTitleBlock3               = 425 *cBaseUnit;
cAnsiTitleBlock4               = 125 *cBaseUnit;
cAnsiTitleBlock5               = 63  *cBaseUnit;
```

```
cAnsiTitleBlock6              = 25  *cBaseUnit;
cAnsiTitleBlock7              = 387 *cBaseUnit;
cAnsiTitleBlock8              = 325 *cBaseUnit;
cAnsiTitleBlock9              = 276 *cBaseUnit;
cAnsiTitleBlock10             = 36  *cBaseUnit;
cAnsiTitleBlock11             = 420 *cBaseUnit;
cAnsiTitleBlock12             = 170 *cBaseUnit;
cAnsiTitleBlock13             = 420 *cBaseUnit;
cAnsiTitleBlock14             = 382 *cBaseUnit;
cAnsiTitleBlock15             = 271 *cBaseUnit;
cAnsiTitleBlock16             = 31  *cBaseUnit;
```

General Constants

```
cCoordFractionalPartSaveExtensionName      = '_Frac';
cCoordFractionalPartSaveExtensionNameExt   = '_Frac1';

cUnits : Array [TUnit] Of TDynamicString   = ('mil', 'mm', 'in', 'cm', '',
'm', 'AutoImperial', 'AutoMetric');

cUnitSystems : Array[TUnitSystem] Of TUnitSet = ([eMil, eIN, eDXP,
eAutoImperial], [eMM,  eCM, eM, eAutoMetric]);

cAutoUnits = [eAutoImperial, eAutoMetric];

cDefaultUnit             : Array[TUnitSystem] Of TUnit = (eDXP, eMM);
cDefaultGridSettingsUnit : Array[TUnitSystem] Of TUnit = (eMil, eMM);

//1 DXP 2004 SP1 Internal Unit = 100000 DXP 2004 SP2 Internal Unit (= 10
mils)
cBaseUnit           = 100000;

//1 mil = 10000 DXP 2004 SP2 internal units
cInternalPrecision  = 10000;

//Size of workspace in DXP 2004 SP1 base logical unit
cMaxWorkspace       = 6500;
```

```
//Size of workspace in DXP 2004 SP1 base logical unit
cMinWorkspace       = 10;

//Size of workspace in the new logical unit - max
cMaxWorkspaceSize   = cMaxWorkspace*cBaseUnit;

//Size of workspace in the new logical unit - min
cMinWorkspaceSize   = cMinWorkspace*cBaseUnit;
```

Schematic Functions

Schematic server interface

```
Function SchServer : ISch_ServerInterface;
```

General functions

```
Function GetState_AllImplementations (Const ASchComponent  : ISch_Component)
       : TList;
Function GetState_PinsForCurrentMode (Const ASchComponent  : ISch_Component)
       : TList;
Function GetState_AllPins            (Const ASchComponent  : ISch_Component)
       : TList;
Function GetState_AllParameters      (Const ASchObject     :
ISch_BasicContainer)  : TList;

Function  HitTestResultToCursor(T : THitTestResult): TCursor;

Function  GetDefaultSchSheetStyle : TSheetStyle;
Procedure GetWholeAndFractionalPart_DXP2004SP2_To_DXP2004SP1(ACoord :
TCoord; Var AWholePart, AFractionalPart : Integer);
Function  GetCoord_DXP2004SP1_To_DXP2004SP2(AWholePart, AFractionalPart :
Integer) : TCoord;
```

Measurement Conversion functions

```
//Imperial
Function  CoordToMils        (    C : TCoord) : TReal;
Function  CoordToDxps        (    C : TCoord) : TReal;
Function  CoordToInches      (    C : TCoord) : TReal;
Function  MilsToCoord        (    M : TReal)  : TCoord;
Function  DxpsToCoord        (    M : TReal)  : TCoord;
Function  InchesToCoord      (    M : TReal)  : TCoord;
//Metric
Function  CoordToMMs         (    C : TCoord) : TReal;
Function  CoordToCMs         (    C : TCoord) : TReal;
Function  CoordToMs          (    C : TCoord) : TReal;
Function  MMsToCoord         (    M : TReal)  : TCoord;
Function  CMsToCoord         (    M : TReal)  : TCoord;
```

```
Function   MsToCoord         (    M : TReal)   : TCoord;
{.......................................................................
...................................}
{.......................................................................
...................................}
Function  MetricString       (Var S : TDynamicString; DefaultUnits : TUnit)
: Boolean;
Function  ImperialString     (Var S : TDynamicString; DefaultUnits : TUnit)
: Boolean;
{.......................................................................
...................................}
{.......................................................................
...................................}
Function  ExtractValueAndUnitFromString(    AInString    : TDynamicString;
                                            ADefaultUnit : TUnit;
                                        Var AValue       : TDynamicString;
                                        Var AUnit        : TUnit) : Boolean;
Function  StringToCoordUnit       (S : TDynamicString; Var C : TCoord;
ADefaultUnit : TUnit) : Boolean;
Function  CoordUnitToString       (C : TCoord; U : TUnit) : TDynamicString;
Function  CoordUnitToStringFixedDecimals (C : TCoord; U : TUnit;
AFixedDecimals : Integer) : TDynamicString;
Function  CoordUnitToStringNoUnit (C : TCoord; U : TUnit) : TDynamicString;
{.......................................................................
...................................}
{.......................................................................
...................................}
Function  GetDisplayStringFromLocation(ALocation : TLocation; AUnit : TUnit)
: TDynamicString;
{.......................................................................
...................................}
{.......................................................................
...................................}
Function GetCurrentDocumentUnit : TUnit;
Function GetCurrentDocumentUnitSystem : TUnitSystem;
Function GetSchObjectOwnerDocumentUnit(Const AObject : ISch_BasicContainer)
: TUnit;
```

Conversion functions

```
Function  GetStateString_ObjectId                    (N : TObjectId
                ) : TString;
Function  GetStateString_HorizontalAlign             (N : THorizontalAlign
         ) : TString;
Function  GetStateString_IeeeSymbol                  (N : TIeeeSymbol
              ) : TString;
Function  GetStateString_LeftRightSide               (N : TLeftRightSide
           ) : TString;
Function  GetStateString_LineStyle                   (N : TLineStyle
               ) : TString;
Function  GetStateString_PinElectrical               (N : TPinElectrical
           ) : TString;
Function  GetStateString_PortArrowStyle              (N : TPortArrowStyle
          ) : TString;
Function  GetStateString_PortIO                      (N : TPortIO
                  ) : TString;
Function  GetStateString_PowerObjectStyle            (N : TPowerObjectStyle
        ) : TString;
Function  GetStateString_CrossSheetConnectorStyle (N :
TCrossSheetConnectorStyle ) : TString;
Function  GetStateString_RotationBy90                (N : TRotationBy90
            ) : TString;
Function  GetStateString_Justification               (N : TTextJustification
       ) : TString;
Function  GetStateString_HorizontalJustification  (N : TTextJustification
       ) : TString;
Function  GetStateString_VerticalJustification    (N : TTextJustification
       ) : TString;
Function  GetStateString_SheetStyle                  (N : TSheetStyle
              ) : TString;
Function  GetStateString_Size                        (N : TSize
                    ) : TString;
Function  GetStateString_Location                    (N : TLocation
                ) : TString;
Function  GetStateString_DisplayMode                 (N : TDisplayMode
             ) : TString;
```

Justification functions

```
Function  IsJustified_Left    (N : TTextJustification) : Boolean;
```

```
Function  IsJustified_HCenter (N : TTextJustification) : Boolean;
Function  IsJustified_Right   (N : TTextJustification) : Boolean;
Function  IsJustified_Bottom  (N : TTextJustification) : Boolean;
Function  IsJustified_VCenter (N : TTextJustification) : Boolean;
Function  IsJustified_Top     (N : TTextJustification) : Boolean;

Procedure GetOrdinalValueFromHorizontalJustification(J :
TTextJustification;Var I : Integer);
Procedure GetOrdinalValueFromVerticalJustification  (J :
TTextJustification;Var I : Integer);
Procedure GetHorizontalJustificationFromOrdinalValue(I : Integer; Var J :
TTextJustification);
Procedure GetVerticalJustificationFromOrdinalValue  (I : Integer; Var J :
TTextJustification);
```

See also

Schematic API Reference

WorkSpace Manager API Reference

The WorkSpace Manager Application Programming Interface reference covers interfaces for the Workspace manager objects in the Workspace Manager Object Model.

What are interfaces?

An interface is simply a list of methods that a class declares that it implements. That is, each method in the interface is implemented in the corresponding class. Interfaces are declared like classes but cannot be directly instantiated and do not have their own method definitions. Each interface, a class supports is actually a list of pointers to methods. Therefore, each time a method call is made to an interface, the interface actually diverts that call to one of it's pointers to a method, thus giving the object that really implements it, the chance to act.

The workspace manager interfaces exist as long there are associated existing objects in memory, thus when writing a script, you have the responsibility of checking whether the interface you wish to query exists or not before you proceed to invoke the interface's methods. Remember to ensure that the project is compiled first, otherwise the workspace manager interfaces are in an invalid state and will be returning nil values.

The workspace manager 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 **IWorkSpace** interface is the main interface representing the WorkSpace Manager object in DXP. The **IWorkSpace** interface deals with projects, documents and objects on the open documents in DXP. To use workspace interfaces, the project needs to be compiled first refreshing all the linkages and nets up to date.

Main Workspace Manager interfaces

- The **IDMObject** interface is a generic 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 Design Explorer.
- The **IDocument** interface represents a document in Design Explorer.

An important note, for the schematic documents, there are logical and physical documents; they are used to differentiate documents in DXP. For example, a multi channel design means that a single sheet is referenced repeatedly for each channel. This sheet is called a logical document. A physical document on the other hand is an existing real document that can be opened and edited in DXP.

Example

Obtaining the project path from the current **IProject** interface.

```
// Get WSM interface (the shell of the WorkSpace Manager interface).
WSM := GetWorkSpace;
```

```
If WSM = Nil Then Exit;
Document := WSM.DM_Document;
If Document = Nil The Exit;
Project := Document.DM_Project;
```

Script Examples

There are script examples in the **\Examples\Scripts\WSM** folder

See also

WorkSpace Manager Interface Overview

WorkSpace Manager Interfaces

WorkSpace Manager Enumerated Types

WorkSpace Manager Functions

Client API Reference

Integrated Library API Reference

Nexar API Reference

PCB API Reference

Schematic API Reference

Using WorkSpace Manager interface

The Work-Space Manager is a system extensions server which is always running when DXP is loaded in memory. This system extensions server provides project functionality of linking a group of files. The Work-Space Manager defines a high level description of a PCB project, documents such as net lists, schematics and PCB documents for synchronization, report and output generation. This server also provides a plug-in system for output generation.

The workspace manager 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.

When you need to deal with projects and documents and other WorkSpace Manager related objects, the starting point is to Invoke the **GetWorkspace** function which returns the **IWorkspace** interface.

With the **IWorkspace** interface, you can extract the all other derived work space manager interfaces that are exposed in the **IWorkSpace** interface. A project containing at least schematic documents need to be open in Design Explorer if you wish to extract documents data of a project using interfaces.

Workspace manager interface methods often have specified parameters types, which are covered in the Workspace Enumerated Types section. For example the **TComponentKind** type denotes the component type in Schematic or PCB documents.

You will need to compile a project in Design Explorer, before you can invoke the Work-Space Manager and its associated interfaces so you can have access to the most current data. Every compile of a project provides a snapshot of the latest status of a design project.

Compile example

```
Var
    Project  : IProject;
Begin
    Project := GetWorkspace.DM_FocusedProject;
    If Project = Nil Then Exit;

    // Do a compile so the logical documents get expanded into physical
documents.
    Project.DM_Compile;
```

Obtaining the Logical Document count example

```
Var
    i        : Integer;
    Document : IDocument;
    Project  : IProject;
Begin
    Project := GetWorkspace.DM_FocusedProject;
    If Project = Nil Then Exit;

    // Do a compile so the logical documents get expanded into physical
documents.
    Project.DM_Compile;

    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;
```

There are logical and physical documents; these terms are used to differentiate the documents in multi-channel projects. A multi channel design means that a single sheet is referenced repeatedly for a channel design. This sheet is called a logical document. A physical document (usually a PCB document) has components with unique names within a room which is mapped to a channel on a Schematic sheet. So a multi channel design translates to multiple rooms with components with unique physical designators on a PCB.

A physical designator of a PCB component is calculated to have the hierarchy path of a schematic project as well as the logical designator of the associated Schematic component to ensure that this designator for the PCB component is unique.

See also

Workspace Manager API Reference

Workspace Manager Interfaces

Workspace Enumerated Types

Workspace Functions

Workspace Manager Interfaces

WorkSpace Manager Object Model

An interface is just a means of access to an object in memory. To have access to the workspace interface object which represents the workspace in DXP, you need to invoke the GetWorkspace function first. This function returns you the pointer to the IWorkspace interface object.

The workspace manager 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 the ability to manipulate the contents of a design project in DXP.

The IDMObject interface is a generic 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 Design Explorer.

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 Design Explorer.

The IComponentMappings interface is used for the Signal Integrity models mapping to Schematic components.

The IECO interface is used for the Engineering Change Order system in PCB and Schematic servers.

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.

See also

Work Space Manager API Reference

IWorkSpace interface

IProject interface

IFPGAProject

ICoreProject

IBoardProject

IEmbeddedProject

IIntegratedLibraryProject

IDocument

IPart

IComponent

INet

INetItem

ICrossSheet

IPin

IPort

IPowerObject

INetLabel

ISheetEntry

IBus

ISheetSymbol

IParameter

IRule

IObjectClass

IChannelClass

INetClass

IRoom

ILine

ITextFrame

IViolation

ISearchPath

IVhdlEntity

IECO

IDMObject interface

Overview

The IDMObject interface is the base object interface for all object interfaces used in the Work Space Manager system extension server for Design Explorer.

IDMObject methods

DM_ObjectAdress

DM_Parameters

DM_ParameterCount

DM_IsInferredObject

DM_LocationX

DM_LocationY

DM_GeneralField

IDMObject properties

DM_LocationString

DM_LongDescriptorString

DM_ShortDescriptorString

DM_ObjectKindString

DM_ObjectKindStringForCrossProbe

DM_PrimaryCrossProbeString

DM_SecondaryCrossProbeString

DM_FullCrossProbeString

DM_ImageIndex

DM_OwnerDocument

DM_OwnerDocumentName

DM_OwnerDocumentFullPath

DM_CurrentSheetInstanceNumber

DM_ValidForNavigation

DM_NetIndex_Flat

DM_NetIndex_Sheet

DM_NetIndex_SubNet

DM_SheetIndex_Logical

DM_SheetIndex_Physical

DM_PCBObjectHandle

DM_SCHObjectHandle

DM_VHDLEntity

IDMObject interface methods

DM_CurrentSheetInstanceNumber method

(IDMObject interface)

Syntax

```
Function DM_CurrentSheetInstanceNumber : Integer;
```

Description

The function returns the current sheet instance number of the schematic document.

See also

IDMObject interface

DM_FullCrossProbeString method

(IDMObject interface)

Syntax

```
Function DM_FullCrossProbeString : WideString;
```

Description

The function returns the full cross probe string.

See also

IDMObject interface

DM_GeneralField method

(IDMObject interface)

Syntax

```
Function DM_GeneralField : Integer;
```

Description

The function can returns an integral value for this general field. This General Field can be used for any purpose - as a tag property, as an index property or as a flag to denote something.

See also

IDMObject interface

DM_ImageIndex method

(IDMObject interface)

Syntax

```
Function DM_ImageIndex : Integer;
```

Description

The function returns the image index depending on what type of object the image represents.

See also

IDMObject interface

DM_IsInferredObject method

(IDMObject interface)

Syntax

```
Function DM_IsInferredObject : Boolean;
```

Description

The function denotes whether the object is an inferred object with respect to connective objects. Bus and Sheet Symbols can be defined in ranges using the NetLabel [] and Repeat statements respectively and once the project has been compiled, inferred objects are created in memory for navigation/connective purposes. For example, a Bus with a range of A[0..4] ends up with five wires with A0...A5 net labels (only in memory). This property is useful for multi – channel projects and for sheets that have Bus objects.

See also

IDMObject interface

DM_LocationString method

(IDMObject interface)

Syntax

```
Function DM_LocationString : WideString;
```

Description

The function returns the Location string formatted as a X,Y format or if the object kind is a Text Documnt set, then the string returned is a formatted Line: LocationY Offset: XLocation string.

See also

IDMObject interface

DM_LocationX method

(IDMObject interface)

Syntax

```
Function DM_LocationX : Integer;
```

Description

The function returns the location of this interface object on the X axis.

See also

IDMObject interface

DM_LocationY method

(IDMObject interface)

Syntax

```
Function DM_LocationY : Integer;
```

Description

The function returns the location of this interface object on the Y axis.

See also

IDMObject interface

DM_LongDescriptorString method

(IDMObject interface)

Syntax

```
Function DM_LongDescriptorString : WideString;
```

Description

The function returns the long description version string.

See also

IDMObject interface

DM_NetIndex_Flat method

(IDMObject interface)

Syntax

```
Function DM_NetIndex_Flat : Integer;
```

Description

The function returns the net index for a flattened design.

See also

IDMObject interface

DM_NetIndex_Sheet method

(IDMObject interface)

Syntax

```
Function DM_NetIndex_Sheet : Integer;
```

Description

The function returns the netindex for a schematic sheet.

See also

IDMObject interface

DM_NetIndex_SubNet method

(IDMObject interface)

Syntax

```
Function DM_NetIndex_SubNet : Integer;
```

Description

The function returns the net index within a sub net.

See also

IDMObject interface

DM_ObjectAdress method

(IDMObject interface)

Syntax

```
Function DM_ObjectAdress : Pointer;
```

Description

The function returns the pointer of the interface object itself. Also called a handle.

See also

IDMObject interface

DM_ObjectKindString method

(IDMObject interface)

Syntax

```
Function DM_ObjectKindString : WideString;
```

Description

The function returns the object kind string which denotes the design document type.

See also

IDMObject interface

DM_ObjectKindStringForCrossProbe method

(IDMObject interface)

Syntax

```
Function DM_ObjectKindStringForCrossProbe : WideString;
```

Description

The function returns the specially formatted object kind string for the cross probing mechanism.

See also

IDMObject interface

DM_OwnerDocument method

(IDMObject interface)

Syntax

```
Function DM_OwnerDocument : IDocument;
```

Description

The function returns the document interface object. Refer to IDocument interface for details.

See also

IDMObject interface

DM_OwnerDocumentFullPath method

(IDMObject interface)

Syntax

```
Function DM_OwnerDocumentFullPath : WideString;
```

Description

The function returns the full path of the document.

See also

IDMObject interface

DM_OwnerDocumentName method

(IDMObject interface)

Syntax

```
Function DM_OwnerDocumentName : WideString;
```

Description

The function returns the name of the document that this object interface is part of.

See also

IDMObject interface

DM_ParameterCount method

(IDMObject interface)

Syntax

```
Function DM_ParameterCount : Integer;
```

Description

The function returns the number of parameters this object has.

See also

IDMObject interface

DM_Parameters method

(IDMObject interface)

Syntax

```
Function DM_Parameters (Index : Integer) : IParameter;
```

Description

The function returns the indexed parameter object with the index parameter. Use the IParameter interface to wrap the returned result.

See also

IDMObject interface

DM_PCBObjectHandle method

(IDMObject interface)

Syntax

```
Function DM_PCBObjectHandle : Integer;
```

Description

The function returns the object handle of a PCB object. If void, a Nil value is returned.

See also

IDMObject interface

DM_PrimaryCrossProbeString method

(IDMObject interface)

Syntax

```
Function DM_PrimaryCrossProbeString : WideString;
```

Description

The function returns the primary cross probe string.

See also

IDMObject interface

DM_SCHObjectHandle method

(IDMObject interface)

Syntax

```
Function DM_SCHObjectHandle : Pointer;
```

Description

The function returns the object handle of a Schematic object. If void, a zero value is returned.

See also

IDMObject interface

DM_SecondaryCrossProbeString method

(IDMObject interface)

Syntax

```
Function DM_SecondaryCrossProbeString : WideString;
```

Description

The function returns the secondary cross probe string.

See also

IDMObject interface

DM_SheetIndex_Logical method

(IDMObject interface)

Syntax

```
Function DM_SheetIndex_Logical : Integer;
```

Description

The function returns the sheet index for a logical design (multi – channel designs for example).

See also

IDMObject interface

DM_SheetIndex_Physical method

(IDMObject interface)

Syntax

```
Function DM_SheetIndex_Physical : Integer;
```

Description

The function returns the sheet index for a physical design. (that have unique designators)

See also

IDMObject interface

DM_ShortDescriptorString method

(IDMObject interface)

Syntax

```
Function DM_ShortDescriptorString : WideString;
```

Description

The function returns the short description version string.

See also

IDMObject interface

DM_ValidForNavigation method

(IDMObject interface)

Syntax

```
Function DM_ValidForNavigation : Boolean;
```

Description

The function toggles whether navigation is valid for this object. Navigation is performed on net aware objects such as components, nets and busses.

See also

IDMObject interface

DM_VHDLEntity method

(IDMObject interface)

Syntax

```
Function DM_VHDLEntity : IVHDLEntity;
```

Description

The function returns the VHDL entity interface object if it exists on a VHDL document. Basically every object interface has an access to this VHDL entity interface, so to check whether VHDL entity exists for this particular object, you can check out the Name field within the IVHDLEntity interface.

See also

IDMObject interface

DM_GetVCSProject

(IDMObject interface)

See also

IClient interface

IDMObject interface

Properties

VCSProject property

(IDMObject interface)

See also

IClient interface

IVCSProjectAccessor interface

IDocument interface

Overview

The IDocument interface represents the existing document in DXP. A document can be a Schematic, PCB, VHDL, PCB Library document etc. The DM_DocumentKind method of the IDocument interface when invoked returns you the document type. A document can be part of a project or free documents project.

An existing document can be queried to return the project interface this document is associated with.

Notes

The **IDocument** interface is a standalone interface.

IDocument methods

DM_FullPath

DM_FileName

DM_DocumentIsLoaded

DM_LoadDocument

DM_DocumentKind

DM_IsPrimaryImplementationDocument

DM_LogicalDocument

DM_Project

DM_IsPhysicalDocument

DM_PhysicalInstancePath

DM_PhysicalInstanceName

DM_PhysicalRoomName

DM_PhysicalDocumentParent

DM_PhysicalDocumentCount

DM_CurrentInstanceNumber

DM_ChannelIndex

DM_ChannelPrefix

DM_ChannelRoomNamingStyle

DM_ChildDocuments

DM_ParentDocuments

DM_Ports

DM_CrossSheetConnectors

DM_Components

DM_UniqueComponents

IDocument properties

DM_Buses

DM_UniqueParts

DM_Parts

DM_SheetSymbols

DM_Nets

DM_TextFrames

DM_Rules

DM_ChannelClasses

DM_ComponentClasses

DM_NetClasses

DM_Rooms

DM_VHDLEntities

DM_ConstraintGroups

DM_ChildDocumentCount

DM_ParentDocumentCount

DM_PortCount

DM_CrossSheetConnectorCount

DM_ComponentCount

DM_UniqueComponentCount

DM_BusCount

DM_UniquePartCount

DM_PartCount

DM_SheetSymbolCount

DM_NetCount

DM_TextFrameCount

DM_RuleCount

DM_ChannelClassCount

DM_ComponentClassCount

DM_NetClassCount

DM_RoomCount

DM_VHDLEntityCount

DM_ConstraintGroupCount

DM_ModelKind

DM_IndentLevel

DM_UpdateDateModified

DM_Compile

DM_ScrapCompile
DM_CreateViolation
DM_DocumentIsTextual
DM_SignalManager

IDocument interface methods

DM_BusCount method

(IDocument interface)

Syntax

```
Function DM_BusCount : Integer;
```

Description

The function returns the number of bus objects from this document. Use this in conjunction with the DM_Buses(Index) to go through each bus object.

See also

IDocument interface

DM_Buses method

(IDocument interface)

Syntax

```
Function DM_Buses (Index : Integer) : IBus;
```

Description

The function returns the indexed Bus instance from this document.

See also

IDocument interface

DM_ChannelClassCount method

(IDocument interface)

Syntax

```
Function DM_ChannelClassCount : Integer;
```

Description

The function denotes the number of Channel Classes from this document. Use this Channel Class count in conjunction with the DM_ChannelClasses(index) to go through each channel class.

See also

IDocument interface

DM_ChannelClasses method

(IDocument interface)

Syntax

```
Function DM_ChannelClasses (Index : Integer) : IChannelClass;
```

Description

The function returns the indexed ChannelClass instance from this document. Use this in conjunction with the DM_ChannelClassCount function

See also

IDocument interface

DM_ChannelIndex method

(IDocument interface)

Syntax

```
Function DM_ChannelIndex : Integer;
```

Description

The function returns the channel index of this document. This is especially for multi-channel designs where a single source document can be referenced multiple times.

See also

IDocument interface

DM_ChannelPrefix method

(IDocument interface)

Syntax

```
Function DM_ChannelPrefix : WideString;
```

Description

The function returns the channel prefix of this document. This is especially for multi-channel designs where a single source document can be referenced multiple times.

See also

IDocument interface

DM_ChannelRoomNamingStyle method

(IDocument interface)

Syntax

```
Function DM_ChannelRoomNamingStyle : TChannelRoomNamingStyle;
```

Description

The function returns the channel room naming style value.

See also

IDocument interface

DM_ChildDocumentCount method

(IDocument interface)

Syntax

```
Function DM_ChildDocumentCount : Integer;
```

Description

The function returns the number of child documents relative to this document.

See also

IDocument interface

DM_ChildDocuments method

(IDocument interface)

Syntax

```
Function DM_ChildDocuments (Index : Integer) : IDocument;
```

Description

The function returns the indexed child document. A hierarchical design consists of multi layered parent-child documents.

See also

IDocument interface

DM_Compile method

(IDocument interface)

Syntax

```
Function DM_Compile : LongBool;
```

Description

The function invokes the compiler to compile this document. If the compile was successful, a true value is returned.

See also

IDocument interface

DM_ComponentClassCount method

(IDocument interface)

Syntax

```
Function DM_ComponentClassCount : Integer;
```

Description

The function denotes the number of component classes from this document. Use this Component class count in conjunction with the DM_ComponentClasses(index) to go through each component class.

See also

IDocument interface

DM_ComponentClasses method

(IDocument interface)

Syntax

```
Function DM_ComponentClasses (Index : Integer) : IComponentClass;
```

Description

The function returns the indexed ComponentClass instance from this document. Use this in conjuniton with the DM_ComponentClassCount function.

See also

IDocument interface

DM_ComponentCount method

(IDocument interface)

Syntax

```
Function DM_ComponentCount : Integer;
```

Description

The function returns the number of component instances on this document. Use this in conjunction with the DM_Components(Index) method to go through each component object.

See also

IDocument interface

DM_Components method

(IDocument interface)

Syntax

```
Function DM_Components (Index : Integer) : IComponent;
```

Description

The function returns the indexed component instance from this document. This is to be used in conjunction with the DM_ComponentCount method.

See also

IDocument interface

DM_ConstraintGroupCount method

(IDocument interface)

Syntax

```
Function DM_ConstraintGroupCount : Integer;
```

Description

The function denotes the number of constraint groups.

See also

IDocument interface

DM_ConstraintGroups method

(IDocument interface)

Syntax

```
Function DM_ConstraintGroups (Index : Integer) : IConstraintGroup;
```

Description

The function returns the indexed constraint group. Use the DM_ConstraintGroupCount function to get the number of constraint groups.

See also

IDocument interface

DM_CreateViolation method

(IDocument interface)

Syntax

```
Function DM_CreateViolation (AErrorKind : TErrorKind;AErrorString :
WideString) : IViolation;
```

Description

The function creates a violation based on the error kind and error string upon an incorrect design.

See also

IDocument interface

DM_CrossSheetConnectorCount method

(IDocument interface)

Syntax

```
Function DM_CrossSheetConnectorCount : Integer;
```

Description

The function returns the number of cross sheet connectors on this document. Use this in conjunction with the DM_CrossConnectors(index) to go through each cross connector object.

See also

IDocument interface

DM_CrossSheetConnectors method

(IDocument interface)

Syntax

```
Function DM_CrossSheetConnectors (Index : Integer) : ICrossSheet;
```

Description

The function returns the indexed cross sheet connector instance from this document. This is to be used in conjunction with the DM_CrossSheetConnectorCount method.

See also

IDocument interface

DM_CurrentInstanceNumber method

(IDocument interface)

Syntax

```
Function DM_CurrentInstanceNumber : Integer;
```

Description

The function returns the current instance number for this document. (especially for multi – channel designs where a design document can be referenced multiple times)

See also

IDocument

DM_DocumentIsLoaded method

(IDocument interface)

Syntax

```
Function DM_DocumentIsLoaded : Boolean;
```

Description

This function returns a boolean value whether this document has been loaded in DXP or not.

See also

IDocument

DM_DocumentIsTextual method

(IDocument interface)

Syntax

```
Function DM_DocumentIsTextual : Boolean;
```

Description

The function denotes whether the document is a text document.

See also

IDocument interface

DM_DocumentKind method

(IDocument interface)

Syntax

```
Function DM_DocumentKind : WideString;
```

Description

This function returns the document kind for the current document. A document could be a Schematic document and thus the string returned is 'SCH'. Check the installation file of each server for the Server Name.

See also

IDocument

DM_FileName method

(IDocument interface)

Syntax

```
Function DM_FileName : WideString;
```

Description

This function returns the file name string of this document.

See also

IDocument

DM_FullPath method

(IDocument interface)

Syntax

```
Function DM_FullPath : WideString;
```

Description

This function returns the full path of where this document lives.

See also

IDocument

DM_IndentLevel method

(IDocument interface)

Syntax

```
Function DM_IndentLevel : Integer;
```

Description

The function returns the indent level for this current document with respect to the current project.

See also

IDocument interface

DM_IsPhysicalDocument method

(IDocument interface)

Syntax

```
Function DM_IsPhysicalDocument : Boolean;
```

Description

This function returns a Boolean value whether this document is a physical document or not. There are logical and physical documents; these terms are used to differentiate the documents in multi-channel projects. A multi channel design means that a single sheet is referenced repeatedly for a channel design. This sheet is called a logical document. A physical document (usually a PCB document) has components with unique names within a room which is mapped to a channel on a Schematic sheet. So a multi channel design translates to multiple rooms with components with unique physical designators on a PCB.

A physical designator of a PCB component is calculated to have the hierarchy path of a schematic project as well as the logical designator of the associated Schematic component to ensure that this designator for the PCB component is unique.

See also

IDocument

DM_IsPrimaryImplementationDocument method

(IDocument interface)

Syntax

```
Function DM_IsPrimaryImplementationDocument : Boolean;
```

Description

This function returns a Boolean value whether this document is a primary implementation document (namely a PCB document for instance). A schematic document is a source document and is centric to a design project.

See also

IDocument

DM_LoadDocument method

(IDocument interface)

Syntax

```
Function DM_LoadDocument : Boolean;
```

Description

This function returns a Boolean value whether this document has been loaded or not.

See also

IDocument

DM_LogicalDocument method

(IDocument interface)

Syntax

```
Function DM_LogicalDocument : IDocument;
```

Description

This function returns the logical document if valid. Otherwise a nil value is returned. There are logical and physical documents; these terms are used to differentiate the documents in multi-channel projects. A multi channel design means that a single sheet is referenced repeatedly for a channel design. This sheet is called a logical document. A physical document (usually a PCB document) has components with unique names within a room which is mapped to a channel on a Schematic sheet. So a multi channel design translates to multiple rooms with components with unique physical designators on a PCB.

A physical designator of a PCB component is calculated to have the hierarchy path of a schematic project as well as the logical designator of the associated Schematic component to ensure that this designator for the PCB component is unique.

See also

IDocument

DM_ModelKind method

(IDocument interface)

Syntax

```
Function DM_ModelKind : WideString;
```

Description

The function returns the model kind string related to this document.

See also

IDocument interface

DM_NetClassCount method

(IDocument interface)

Syntax

```
Function DM_NetClassCount : Integer;
```

Description

The function denotes the number of net classes on this document. Use this NetClass count in conjunction with the DM_NetClasses(Index) method to go through each net class.

See also

IDocument interface

DM_NetClasses method

(IDocument interface)

Syntax

```
Function DM_NetClasses (Index : Integer) : INetClass;
```

Description

The function returns the indexed NetClass instance from this document. Use this in conjunction with the DM_NetClassCount function.

See also

IDocument interface

DM_NetCount method

(IDocument interface)

Syntax

```
Function DM_NetCount : Integer;
```

Description

The function returns the number of nets from this document. Use this Net count in conjunction with the DM_Nets(Index) to go through each sheet symbol object

See also

IDocument interface

DM_Nets method

(IDocument interface)

Syntax

```
Function DM_Nets (Index : Integer) : INet;
```

Description

The function returns an indexed net associated with this document.

See also

IDocument interface

DM_ParentDocumentCount method

(IDocument interface)

Syntax

```
Function DM_ParentDocumentCount : Integer;
```

Description

The function returns the number of parent documents relative to this document.

See also

IDocument interface

DM_ParentDocuments method

(IDocument interface)

Syntax

```
Function DM_ParentDocuments (Index : Integer) : IDocument;
```

Description

The function returns the indexed parent document. A hierarchical design consists of multi layered parent-child documents.

See also

IDocument

DM_PartCount method

(IDocument interface)

Syntax

```
Function DM_PartCount : Integer;
```

Description

The function returns the number of part objects from this document. Use this PartCount in conjunction with the DM_Parts(Index) to go through each part object.

See also

IDocument interface

DM_Parts method

(IDocument interface)

Syntax

```
Function DM_Parts (Index : Integer) : IPart;
```

Description

The function returns an indexed part associated with this document.

See also

IDocument interface

DM_PhysicalDocumentCount method

(IDocument interface)

Syntax

```
Function DM_PhysicalDocumentCount : Integer;
```

Description

The function returns the number of physical documents associated with this document.

See also

IDocument interface

DM_PhysicalDocumentParent method

(IDocument interface)

Syntax

```
Function DM_PhysicalDocumentParent : IDocument;
```

Description

The function returns the IDocument interface for a parent physical document. Could be a VHDL or a PCB document for example.

See also

IDocument interface

DM_PhysicalInstanceName method

(IDocument interface)

Syntax

```
Function DM_PhysicalInstanceName : WideString;
```

Description

The function returns the name of this physical document if valid. Otherwise an empty string is returned.

See also

IDocument

DM_PhysicalInstancePath method

(IDocument interface)

Syntax

```
Function DM_PhysicalInstancePath : WideString;
```

Description

The function returns the path to the physical document instance if valid. Otherwise an empty string is returned.

See also

IDocument interface

DM_PhysicalRoomName method

(IDocument interface)

Syntax

```
Function DM_PhysicalRoomName : WideString;
```

Description

The function returns the name of the room on this physical document if valid. Otherwise a nil value is returned.

See also

IDocument interface

DM_PortCount method

(IDocument interface)

Syntax

```
Function DM_PortCount : Integer;
```

Description

The function returns the number of port objects on this document. Use this in conjunction with the DM_Ports(index) to go through each port object.

See also

IDocument interface

DM_Ports method

(IDocument interface)

Syntax

```
Function DM_Ports (Index : Integer) : INetItem;
```

Description

The function returns the indexed port instance from this document. This is to be used in conjunction with the DM_PortCount method

See also

IDocument

DM_Project method

(IDocument interface)

Syntax

```
Function DM_Project : IProject;
```

Description

This function returns the IProject object interface that this document is associated with.

See also

IDocument

DM_RoomCount method

(IDocument interface)

Syntax

```
Function DM_RoomCount : Integer;
```

Description

The function denotes the number of rooms on this document. Use this RoomCount in conjunction with the DM_Rooms(Index) to go through each room object.

See also

IDocument interface

DM_Rooms method

(IDocument interface)

Syntax

```
Function DM_Rooms (Index : Integer) : IRoom;
```

Description

The function returns the indexed room instance from this document. Use this in conjunction with the DM_RoomCount function.

See also

IDocument interface

DM_RuleCount method

(IDocument interface)

Syntax

```
Function DM_RuleCount : Integer;
```

Description

The function returns the number of rules from this document. Use this Rule count in conjunction with the DM_Rules(Index) to go through each sheet symbol object

See also

IDocument interface

DM_Rules method

(IDocument interface)

Syntax

```
Function DM_Rules (Index : Integer) : IRule;
```

Description

The function denotes the indexed rule from this document. Use this DM_RuleCount in conjunction with the DM_Rules to go through each rule found from this document..

See also

IDocument interface

DM_ScrapCompile method

(IDocument interface)

Syntax

```
Function DM_ScrapCompile(ForceCompile : Boolean) : LongBool;
```

Description

The function invokes a scrap compile (by force or not). A scrap compile is the background compile in DXP on a design document and does all the auto - junctions for bus and wire objects. Also the scrap compile does the online rule checks in schematics. It is totally separate from the main compile which compile projects.

See also

IDocument interface

DM_SheetSymbolCount method

(IDocument interface)

Syntax

```
Function DM_SheetSymbolCount : Integer;
```

Description

The function returns the number of sheet symbols from this document. Use this SheetSymbol count in conjunction with the DM_SheetSymbols(Index) to go through each sheet symbol object.

See also

IDocument interface

DM_SheetSymbols method

(IDocument interface)

Syntax

```
Function DM_SheetSymbols (Index : Integer) : ISheetSymbol;
```

Description

The function returns an indexed sheet symbol associated with this document.

See also

IDocument interface

DM_SignalManager method

(IDocument interface)

Syntax

```
Function DM_SignalManager : ISignalManager;
```

Description

The function returns the signal manager interface.

See also

IDocument interface

ISignalManager interface

DM_TextFrameCount method

(IDocument interface)

Syntax

```
Function DM_TextFrameCount : Integer;
```

Description

The function returns the number of text frame objects from this document. Use this TextFrame count in conjunction with the DM_TextFrames(Index) to go through each sheet symbol object

See also

IDocument interface

DM_TextFrames method

(IDocument interface)

Syntax

```
Function DM_TextFrames (Index : Integer) : ITextFrame;
```

Description

The function returns an indexed textframe object associated with this document.

See also

IDocument interface

DM_UniqueComponentCount method

(IDocument interface)

Syntax

```
Function DM_UniqueComponentCount : Integer;
```

Description

The function returns the number of unique components according to the library (ies) they are placed from. A duplicate of components of the same component kind is counted as one (1). Use this in conjunction with the DM_UniqueComponents(Index) method to go through each unique component object.

See also

IDocument

DM_UniqueComponents method

(IDocument interface)

Syntax

```
Function DM_UniqueComponents (Index : Integer) : IComponent;
```

Description

The function returns the indexed unique component instance from this document. This function is to be used in conjunction with the DM_UniqueComponentCount method.

See also

IDocument interface

DM_UniquePartCount method

(IDocument interface)

Syntax

```
Function DM_UniquePartCount : Integer;
```

Description

The function denotes the number of unique parts from this document. Duplicates of the same part kind are only returned as a count of one (1).

See also

IDocument interface

DM_UniqueParts method

(IDocument interface)

Syntax

```
Function DM_UniqueParts (Index : Integer) : IPart;
```

Description

The function returns an indexed unique part associated with this document. Note, if multiple instances of the same part exist, then only one of these parts will be recognized.

See also

IDocument

DM_UpdateDateModified method

(IDocument interface)

Syntax

```
Procedure DM_UpdateDateModified;
```

Description

The procedure sets the modified date for this document.

See also

IDocument interface

DM_VHDLEntities method

(IDocument interface)

Syntax

```
Function DM_VHDLEntities (Index : Integer) : IVHDLEntity;
```

Description

The function returns the indexed VHDL entity instance from this document. Use this in conjunction with the DM_VHDLEntityCount function.

See also

IDocument interface

DM_VHDLEntityCount function

DM_VHDLEntityCount method

(IDocument interface)

Syntax

```
Function DM_VHDLEntityCount : Integer;
```

Description

The function denotes the number of VHDL entities from this document. Use this VHDL Entity count in conjunction with the DM_VHDLEntities(Index) to go through each VHDL entity.

See also

IDocument interface

DM_VHDLEntities method

IVhdlEntity interface

Overview

The IVhdlEntity interface represents the existing VHDL entity object on a VHDL document. Basically a VHDL document can contain many VHDL entities and each entity corresponds to a schematic document.

Since every object interface (inherited from the IDMObject interface) has a DM_VHDLEntity method. This method can be useful in cases such as determining which ports correspond to VHDL entities.

Interface Methods

Method	Description
Function DM_Name : WideString;	Returns the name of the VHDL entity.

IWorkspace interface

Overview

The **IWorkspace** interface represents the Work-Space manager in the Design Explorer which deals with project and documents and their related attributes and options. This interface object is the starting point and upon querying this object, it can return currently open projects, number of projects, installed libraries, and create a new document for example.

Remember the projects need to be compiled first, before you can invoke the **GetWorkSpace** function to obtain the **IWorkSpace** interface and its descendant interfaces which represent actual objects in DXP.

It is highly recommended not to hold an interface of the Workspace manager, but re-query the work-space manager every-time the access to the information within is required.

IWorkspace methods

DM_WorkspaceFullPath

DM_WorkspaceFileName

DM_Projects

DM_InstalledLibraries

DM_ProjectCount

DM_InstalledLibraryCount

IWorkspace properties

DM_FocusedProject

DM_FocusedDocument

DM_GenerateUniqueID

DM_ShowMessageView

DM_ShowToDoList

DM_ImageIndexForDocumentKind

DM_GetDocumentFromPath

DM_GetProjectFromPath

DM_ViolationTypeDescription

DM_ViolationTypeGroup

DM_OpenProject

DM_FreeDocumentsProject

DM_CreateNewDocument

DM_AddDocumentToActiveProject

DM_AddOutputLine

DM_GetOutputLine

DM_GetOutputLineCount

DM_ClearOutputLines

DM_OptionsStorage

DM_SetRecoveryParameters

DM_GetRecoveryIsEnabled

DM_GetRecoveryInterval

DM_ChangeManager

DM_MessagesManager

DM_PromptForDefaultPcbType

DM_GetDefaultPcbType

IWorkspace interface methods

DM_AddDocumentToActiveProject method

(IWorkspace interface)

Syntax

```
Procedure DM_AddDocumentToActiveProject(DocumentPath : WideString);
```

Description

This method adds an existing document with its valid full path into an active project within DXP.

See also

IWorkspace

DM_AddOutputLine method

(IWorkspace interface)

Syntax

```
Procedure DM_AddOutputLine(MessageLine : PChar;ReplaceLastLine : Boolean =
False);
```

Description

Outputs the line to the output's dialog. An Internal operation.

See also

IWorkspace

DM_ChangeManager method

(IWorkspace interface)

Syntax

```
Function DM_ChangeManager : IChangeManager;
```

Description

Returns the Engineering Change Order Manager interface object which compares with two projects and creates an ECO to perform a pin swapping process to synchronize the specified two projects.

See also

IWorkspace

DM_ClearOutputLines method

(IWorkspace interface)

Syntax

```
Procedure DM_ClearOutputLines;
```

Description

Clears out the Output Memo. An Internal operation.

See also

IWorkspace

DM_ComponentConfigurator method

(IWorkspace interface)

Syntax

```
Function DM_ComponentConfigurator(Const ALibRef : WideString) :
IComponentConfigurator;
```

See also

IClient interface

IWorkSpace interface

DM_CreateNewDocument method

(IWorkspace interface)

Syntax

```
Function DM_CreateNewDocument (ADocKind : WideString) : WideString;
```

Description

This method creates a new document based on the Document Kind. The Kinds include – 'PCBLIB','PCB','SCH','SCHLIB' and so on depending on which document servers are installed in DXP.

Example

//Creating a new PCB document in DXP

```
Var
    WSM         : IWorkSpace;
Begin
    WSM := GetWorkSpace;
    If WSM = Nil Then Exit;
    WSM.DM_CreateNewDocument('PCB');
End;
```

See also

IWorkspace

DM_FocusedDocument method

(IWorkspace interface)

Syntax

```
Function DM_FocusedDocument : IDocument;
```

Description

Returns the focussed document interface object (if any) in Design Explorer. A focussed document is a document that is currently open and in focus (this document is active).

See also

IWorkspace

DM_FocusedProject method

(IWorkspace interface)

Syntax

```
Function DM_FocusedProject : IProject;
```

Description

Returns the focussed project (if any) in Design Explorer.

See also

IWorkspace

DM_FreeDocumentsProject method

(IWorkspace interface)

Syntax

```
Function DM_FreeDocumentsProject : IProject;
```

Description

Returns the **IProject** interface that contains free documents. A free document is a standalone document that lives in the Free Documents project.

See also

IWorkspace

DM_GenerateUniqueID method

(IWorkspace interface)

Syntax

```
Function DM_GenerateUniqueID : WideString;
```

Description

Invoke this method, and a generated Unique ID will be returned which can be used for any newly created or existing object in DXP. Objects in Schematic have their own Unique IDs which are tracked by the synchronizator so that the objects on the PCB document are sychronized to their equivalents in a corresponding schematic project.

Example - an incomplete example that assigns new UIDs to Schematic components

```
// get the workspace manager interface so you can
// generate unique ids...
```

```
WSM := GetWorkspace;
If WSM = Nil Then Exit;
// get the schematic server interface
If SchServer = Nil Then Exit;
// get the current sch sheet
CurrentSheet := SchServer.GetCurrentSchDocument;
If CurrentSheet = Nil Then Exit;

// Set up an iterator to look for components
// on a schematic document.
Iterator := CurrentSheet.SchIterator_Create;
Iterator.AddFilter_ObjectSet(MkSet(eSchComponent));

Try
    Comp := Iterator.FirstSchObject;
    While Compt <> Nil Do
    Begin
        Comp.UniqueID := WSM. DM_GenerateUniqueID;
        Comp := Iterator.NextSchObject;
    End;
Finally
    CurrentSheet.SchIterator_Destroy(Iterator);
End;
```

See also

IWorkspace

DM_GetDefaultPcbType method

(IWorkspace interface)

Syntax

```
Function DM_GetDefaultPcbType : WideString;
```

See also

IWorkspace

DM_GetDocumentFromPath method

(IWorkspace interface)

Syntax

```
Function DM_GetDocumentFromPath(DocumentPath : WideString) : IDocument;
```

Description

Retrieves the IDocument interface object by passing the full document path to this document. With this IDocument interface, you have access to its functionality, such as compiling the document itself.

See also

IWorkspace

DM_GetOutputLine method

(IWorkspace interface)

Syntax

```
Function DM_GetOutputLine(Index : Integer) : WideString;
```

See also

IWorkspace

DM_GetOutputLineCount method

(IWorkspace interface)

Syntax

```
Function DM_GetOutputLineCount : Integer;
```

Description

Returns the number of output lines in the Output dialog. An Internal operation.

See also

IWorkspace

DM_GetProjectFromPath method

(IWorkspace interface)

Syntax

```
Function DM_GetProjectFromPath (ProjectPath : WideString) : IProject;
```

Description

Retrieves the IProject interface object by passing the full project path to this project. With this IProject interface, you have access to its interface methods. A project is a container that has links to associated design documents in an organized manner.

See also

IWorkspace

DM_GetRecoveryInterval method

(IWorkspace interface)

Syntax

```
Function DM_GetRecoveryInterval : Integer;
```

Description

Returns the number of minutes as the interval when the recovery mechanism kicks in.

See also

IWorkspace

DM_GetRecoveryIsEnabled method

(IWorkspace interface)

Syntax

```
Function DM_GetRecoveryIsEnabled : Boolean;
```

Description

Returns a Boolean value whether the recovery mechanism is active or not.

See also

IWorkspace

DM_ImageIndexForDocumentKind method

(IWorkspace interface)

Syntax

```
Function DM_ImageIndexForDocumentKind(ADocumentKind : WideString) : Integer;
```

Description

Returns the image index depending on the document kind for example PCB, CAMtastic etc.

See also

IWorkspace

Image Index Table

DM_InstalledLibraries method

(IWorkspace interface)

Syntax

```
Function DM_InstalledLibraries (Index : Integer) : IDocument;
```

Description

Returns an indexed library (currently installed in Design Explorer only), to be used in conjunction with the DM_InstalledLibraryCount.

See also

IWorkspace interface

DM_InstalledLibraryCount method

DM_InstalledLibraryCount method

(IWorkspace interface)

Syntax

```
Function DM_InstalledLibraryCount : Integer;
```

Description

Returns the number of installed libraries in Design Explorer.

See also

IWorkspace

DM_MessagesManager method

(IWorkspace interface)

Syntax

```
Function DM_MessagesManager : IMessagesManager;
```

Description

This function returns you the interface to the Messages panel in DXP.

See also

IWorkspace interface

IMessagesManager interface

DM_NavigationZoomPrecision method

(IWorkspace interface)

Syntax

```
Function DM_NavigationZoomPrecision : Integer;
```

Description

Sets how precise the document zoom is when the interactive navigator is being used to trace the connection in a project.

See also

IWorkspace

DM_OpenProject method

(IWorkspace interface)

Syntax

```
Function DM_OpenProject ( ProjectPath : WideString;Const Show : Boolean) :
IProject;
```

Description

Opens a project with the full project path and set this project in focus depending on its Show parameter.

See also

IWorkspace

DM_OptionsStorage method

(IWorkspace interface)

Syntax

```
Function DM_OptionsStorage : IOptionsStorage;
```

Description

Represents a options storage container where DXP can use to retrieve and store options for storing parameters of EDE options such as Toolchain name, folder and default options and project options.

See also

IWorkspace

DM_ProjectCount method

(IWorkspace interface)

Syntax

```
Function DM_ProjectCount : Integer;
```

Description

Returns the number of projects open in Design Explorer.

See also

IWorkspace

DM_Projects method

(IWorkspace interface)

Syntax

```
Function DM_Projects (Index : Integer) : IProject;
```

Description

Returns the indexed project (currently loaded in Design Explorer only), to be used in conjunction with the DM_ProjectCount interface.

See also

IWorkspace

DM_PromptForDefaultPcbType method

(IWorkspace interface)

Syntax

```
Function DM_PromptForDefaultPcbType(Var PcbType : WideString) : Boolean;
```

See also

IWorkspace

DM_SetRecoveryParameters method

(IWorkspace interface)

Syntax

```
Procedure DM_SetRecoveryParameters(IsEnabled : Boolean; Interval : Integer);
```

Description

Set the interval when the autosave / recovery mechanism in DXP kicks in. The interval is in minutes, and whether to enable the recovery mechanism.

See also

IWorkspace

DM_ShowMessageView method

(IWorkspace interface)

Syntax

```
Procedure DM_ShowMessageView;
```

Description

Invoke this method to refresh the Message panel.

See also

IWorkspace

DM_ShowToDoList method

(IWorkspace interface)

Syntax

```
Procedure DM_ShowToDoList;
```

Description

This method displays the To Do List manager panel. This To Do List panel can be used to define your To Dos.

See also

IWorkspace

DM_ViolationTypeDescription method

(IWorkspace interface)

Syntax

```
Function DM_ViolationTypeDescription(ErrorKind : TErrorKind) : WideString;
```

Description

Returns the violation type description string with the error kind value passed in. Check the TErrorKind for its range of values.

See also

IWorkspace

DM_ViolationTypeGroup method

(IWorkspace interface)

Syntax

```
Function DM_ViolationTypeGroup (ErrorKind : TErrorKind) : TErrorGroup;
```

Description

Returns the error group for which this error kind parameter belongs to. Check the TErrorGroup type for its range of values.

See also

IWorkspace

DM_WorkspaceFileName method

(IWorkspace interface)

Syntax

```
Function DM_WorkspaceFileName : WideString;
```

Description

Returns the filename only of the workspace.

See also

IWorkspace

DM_WorkspaceFullPath method

(IWorkspace interface)

Syntax

```
Function DM_WorkspaceFullPath : WideString;
```

Description

Returns the full path of the workspace.

See also

IWorkspace

System Interfaces

IChangeManager interface

Overview

The IChangeManager interface represents the change manager where you can execute an ECO of pins to be swapped for the target component of the target document.

Interface Methods

```
Procedure DM_SetProject1(AProject : IProject);
Procedure DM_SetProject2(AProject : IProject);
Function  DM_ExecuteChanges(IsSilent : LongBool) : LongBool;
Procedure DM_CreateECO_SwapPin         (TargetDocument : IDocument;
                                         TargetComponent: IComponent;
                                         TargetPin       : IPin;
                                         NewPinNumber    : WideString;
                                         OldPinNet       : WideString;
                                         NewPinNet       : WideString);
```

See also

Workspace Manager Interfaces

IDocument interface

IComponent interface

IPin interface

IComponentMappings interface

Overview

The IComponentMappings interface represents the mapping of source components and target components in schematic and PCB documents.

Interface Methods

Method	Description
Function DM_UnmatchedSourceComponent(Index : Integer) : IComponent;	Returns the indexed unmatched source component, that is, a target component could not be found to map to this source component. Use the DM_UnmatchedSourceComponentCount function.

Method	Description
Function DM_UnmatchedTargetComponent(Index : Integer) : IComponent;	Returns the indexed unmatched target component, that is, a source component could not be found to map to the target component. Use the DM_UnmatchedTargetComponentCount function.
Function DM_MatchedSourceComponent (Index : Integer) : IComponent;	Returns the indexed matched source component (that has been matched with a target component). Use the DM_MatchedSourceComponentCount function.
Function DM_MatchedTargetComponent (Index : Integer) : IComponent;	Returns the indexed matched source component (that has been matched with a target component). Use the DM_MatchedTargetComponentCount function.
Function DM_UnmatchedSourceComponentCount : Integer;	Returns the number of unmatched source components.
Function DM_UnmatchedTargetComponentCount : Integer;	Returns the number of unmatched target components.
Function DM_MatchedComponentCount: Integer;	Returns the number of matched components.

ICustomClipboardFormat interface

Interface Methods

```
Function RegisterCustomClipboardFormat(Const AFormatName : WideString) :
Longword;
```

See also

Workspace Manager Interfaces

IDoToManager

Overview

The **IDoToManager** interface represents the To Do panel in DXP. This To Do list manager allows you to manage a list of what to do and assign a priority to each what to do item.

Interface Methods

Function AddItem (Const AnItem : WideString) : LongBool;

Function RemoveItem (Const AnItem : WideString) : LongBool;

Function GetItem (Index : Integer) : WideString;

Function GetCount : Integer;

Procedure Clear;

Interface Properties

Property Item[Index : Integer] : WideString Read GetItem;

Property Count : Integer Read GetCount;

See also

Workspace Manager Interfaces

IDocumentBackups interface

Interface Properties

```
Property Count : Integer
Property Backups[AIndex : Integer] : WideString
```

See also

IClient interface

IECO interface

Overview

The **IECO** interface represents an Engineering Change Order interface in the Work Space Manager. Basically an Engineering Change Order attempts to keep a project containing source documents and its corresponding primary implementation documents synchronized. For example a schematic project and its PCB document, every time something changes in a schematic project, it is necessary to bring the changes forward to the PCB document via the Engineering Change Order feature.

Interface Methods

Method	Description
Procedure DM_Begin;	Denotes that the ECO manager has started.
Procedure DM_End;	Denotes that the ECO manager has ended.
Function DM_AddObject (Mode : TECO_Mode; ReferenceObject : IDMObject)	Adds a reference object for the ECO to compare the target document against this reference document.
Function DM_RemoveObject (Mode : TECO_Mode; ObjectToRemove : IDMObject)	Removes a reference object depending on what ECO mode is.

Method	Description
Function DM_AddMemberToObject (Mode : TECO_Mode; ReferenceMember : IDMObject; ReferenceParent : IDMObject; TargetParent : IDMObject)	Adds a specific action in the ECO manager.
Function DM_RemoveMemberFromObject (Mode : TECO_Mode; MemberObject : IDMObject; ParentObject : IDMObject)	Removes a specific action in the ECO manager.
Function DM_ChangeObject (Mode : TECO_Mode; Kind : TModificationKind; ObjectToChange : IDMObject; ReferenceObject : IDMObject)	Changes a specific action in the ECO manager.

IIntegratedLibraryProject interface

Overview

The IIntegratedLibraryProject interface represents the project that deals with integrated libraries.

IImportant notes

- Inherited from IProject interface

Interface Methods

- IProject methods

Interface Properties

- IProject Properties

See also

Workspace Manager Interfaces

IProject interface

IMessagesManager Interface

Overview

The IMessagesManager interface represents the Messages panel in DXP.

IMessagesManager methods

AddMessage

AddMessageParametric

ClearMessages

ClearMessagesOfClass

ClearMessagesForDocument

ClearMessageByIndex

BeginUpdate

EndUpdate

MessagesCount

Messages

IMessagesManager properties

Example

//Populating the Message Panel using the Workspace manager's functionality

```
Procedure InsertMessagesIntoMessagePanel;
Var
    WSM          : IWorkSpace;
    MM           : IMessagesManager;
    ImageIndex   : Integer;
    F            : Boolean;
Begin
    WSM := GetWorkSpace;
    If WSM = Nil Then Exit;

    // Tick icon for the lines in the Message panel
    // Refer to the Image Index table in the
    // Workspace Manager API reference online help.
    ImageIndex := 3;

    MM := WSM.DM_MessagesManager;
    If MM = Nil Then Exit;
```

```
    // Clear out messages from the Message panel...
    MM.DM_ClearMessages;
    MM.DM_ShowMessageView;
    WSM.DM_MessageViewBeginUpdate;

    F := False;
    MM.DM_AddMessage({MessageClass             } 'MessageClass 1',
                     {MessageText              } 'MessageText 1',
                     {MessageSource            } 'DXP Message',
                     {MessageDocument          } 'Pseudo Doc 1',
                     {MessageCallBackProcess   } '',
                     {MessageCallBackParameters} '',
                     ImageIndex,
                     F);

    MM.DM_AddMessage({MessageClass             } 'MessageClass 2',
                     {MessageText              } 'MessageText 2',
                     {MessageSource            } 'DXP Message 2',
                     {MessageDocument          } 'Pseudo Doc 2',
                     {MessageCallBackProcess   } '',
                     {MessageCallBackParameters} '',
                     ImageIndex,
                     F);

    MM.DM_MessageEndUpdate;
End;
```

See also

Image Index Table

IMessagesManager Interface methods

AddMessage method

(IMessagesManager interface)

Syntax

```
Procedure AddMessage(Const
MessageClass,MessageText,MessageSource,MessageDocument,MessageCallBackProces
s,MessageCallBackParameters : WideString;ImageIndex :
Integer;ReplaceLastMessageIfSameClass : Boolean = False);
```

Description

This method gives you the ability to a DXP Message on the Message panel.

MessageClass :- which sort of message it belongs to. (User defined)

MessageText :- the message text to appears in the Message panel.

MessageSource :- could be one of the following pre-defined strings such as : Comparator, Back-Annotate, Output Generator, Compiler or you can define your own MessageSource string.

MessageDocument :- Owner Document name – normally a full path name of the document that the Message is associated with.

MessageCallBackProcess :- process name to call back.

MessageCallbackParameters :- parameters for the CallBackProcess.

ImageIndex :- the index to the image depending on which Message Class. Refer to the Image Index Table topic to check out the appropriate image for each message.

ReplaceLastMessageIfSameClass :- (defaults to false).

See also

IMessagesManager

AddMessageParametric method

(IMessagesManager interface)

Syntax

```
Procedure AddMessageParametric(MessageParams :
PChar;MessageCallBackParameters : PChar);
```

Description

Inserts a DXP message in the Message panel. Similar to the DM_AddMessage only that you define the Name / Value blocks in the MessageParams nullterminated string.

Class:– Back-Annotate class, Error level, Differences.

Text:-text displayed in the Message panel.

Source:- could be one of the following: Comparator, Back-Annotate, Output Generator, Compiler,.

Document:- Owner Document name

CallBackProcess:- process name to call back.

UserId:- Unique ID

HelpFileName:- Name of the Help file

HelpTopic:- specific help topic string

ImageIndex::- the index to the image depending on which Message Class.

'ReplaceLastMessageIfSameClass':- Boolean. If Same MessageClass, specify whether this class is to be overridden or not by the current message class information.

The **MessageCallBackParameters** parameter :– parameters for the CallBackProcess.

See also

IMessagesManager

BeginUpdate method

(IMessagesManager interface)

Syntax

```
Procedure BeginUpdate;
```

Description

Invoke this method before you wish to add Messages (DM_AddMessage or DM_AddMessageParameteric methods) to the Message panel.

See also

IMessagesManager

ClearMessageByIndex method

(IMessagesManager interface)

Syntax

```
Procedure ClearMessageByIndex ( AIndex : Integer );
```

See also

IMessagesManager

ClearMessages method

(IMessagesManager interface)

Syntax

```
Procedure ClearMessages;
```

Description

Clears out the Messages panel.

See also

IMessagesManager

ClearMessagesForDocument method

(IMessagesManager interface)

Syntax

```
Procedure ClearMessagesForDocument(Const DocumentPath : WideString);
```

See also

IMessagesManager

ClearMessagesOfClass method

(IMessagesManager interface)

Syntax

```
Procedure ClearMessagesOfClass (Const AMsgClass : WideString);
```

Description

This method gives you the ability to clear messages of the same class type. Various class types include Back-Annotate class, Error level, Differences

See also

IMessagesManager

EndUpdate method

(IMessagesManager interface)

Syntax

```
Procedure EndUpdate;
```

Description

Invoke this method after you have added Messages to the Message panel.

See also

IMessagesManager

Messages method

(IMessagesManager interface)

Syntax

```
Function Messages(Index : Integer) : IMessageItem;
```

See also

IMessagesManager

MessagesCount method

(IMessagesManager interface)

Syntax

```
Function MessagesCount : Integer;
```

See also

IMessagesManager

IMessageItem interface

Overview

The IMessageItem interface represents the message item sent in DXP.

IMessageItem Properties

```
Property MsgClass             : WideString
Property Text                 : WideString
Property Source               : WideString
Property Document             : WideString
Property MsgDateTime          : TDateTime
Property ImageIndex           : Integer
Property UserId               : WideString
Property CallBackProcess      : WideString
Property CallBackParameters : WideString
Property HelpFileName         : WideString
Property HelpFileID           : WideString
Property MsgIndex             : Integer
```

See also

IClient interface

IOptionsStorage interface

Overview

The IOptionsStorage interface can retrieve, store or delete external parameters within the storage container. It is part of the IWorkspace and IProject interface objects.

Interface Methods

```
Procedure AddExternalParameter      (ASection     : WideString;
                                     AName        : WideString;
                                     AValue       : WideString);

Procedure ClearExternalParameters   (ASection     : WideString);

Function  GetExternalParameterValue(ASection     : WideString;
                                     AName        : WideString;
                                     DefaultValue : WideString) : WideString;

Procedure DeleteExternalParameter   (ASection     : WideString;
                                     AName        : WideString);

Function  GetSectionText            (ASection     : WideString) : WideString;

Function  SectionExists             (ASection     : WideString) : LongBool;

Function  GetFileName                                           : WideString;

Procedure BeginUpdate;

Procedure EndUpdate;
```

ISearchPath interface

Overview

The ISearchPath interface represents the paths of a project. This ISearchPath interface has a link to the associated open project in DXP.

Interface Methods

Method	Description
Function DM_Path : WideString;	Returns the path of the focussed project in DXP.
Function DM_AbsolutePath : WideString;	Returns the absolute path of the focussed project in DXP.
Function DM_IncludeSubFolders : Boolean;	Returns whether sub folders are included in the focussed project in DXP.
Function DM_Project : IProject;	Returns the project in which this ISearchPath interface is associated with.

ISymbolGenerator interface

Overview

The ISymbolGenerator interface represents the symbol with parameters added if necessary generated by the ICoreProject interface.

Important Notes

- ICoreProject interface's DM_CreateSymbolGenerator method returns a ISymbolGenerator interface.

Interface Methods

```
Procedure DM_ClearParameters;
Procedure DM_AddParameter(Name, Value : WideString);
Procedure DM_GenerateComponent;
Procedure DM_GenerateComponentFromUser;
```

See also

Workspace Manager Interfaces

ICoreProject interface

IWrapper interface

(IWrapper interface)

See also

IClient interface

IVCSProjectAccessor interface

Interface Methods

```
Function ObjectAddress : IInterface;
```

See also

IClient interface

IExternalForm interface

IVersionControlServer interface

Interface Methods

Function GetStatusString(Const AObejct : IDMObejct) : WideString;

See also

IClient interface

IExternalForm interface

Project Interfaces

IProject Interface

Overview

The IProject interface deals with an open project in DXP. There are project and document variants, that is actually a project or document can be specified to have project or document variants (actual project / document variants do not exist) and on these document variants have component variants.

To have access to the data of a project, you need to do a compile first. Projects deal with logical and physical documents. Logical documents are the connected documents which are part of a design which include a PCB document associated with this design. Physical documents are source documents expanded by the DXP compiler as in a flattened design project.

Thus, a project contains source documents and implementation documents. To have access to the most current data of a project, you need to compile the project first. The compiler maps (or expands) all the logical source documents into physical documents.

Normally there is a one logical document to a one physical document for a simple flat design project, but for hierarchical design projects (for example multi channel projects), the documents that have sheet symbols with a Repeat statement, then logical documents are expanded into multiple physical documents.

There are Output jobs consisting of available output generators installed in DXP.

The **IProject** interface hierarchy is as follows:

IProject methods

DM_ProjectVariants

DM_GeneratedDocuments

DM_LogicalDocuments

DM_PhysicalDocuments

DM_SearchPaths

DM_Configurations

DM_Outputers

DM_ProjectVariantCount

DM_GeneratedDocumentCount

DM_LogicalDocumentCount

DM_PhysicalDocumentCount

DM_SearchPathCount

DM_ConfigurationCount

DM_IndexOfSourceDocument

IProject properties

DM_MoveSourceDocument

DM_AddConfigurationParameters

DM_AddConfigurationParameters_Physical

DM_AddGeneratedDocument

DM_AddSourceDocument

DM_AddControlPanel

DM_RemoveSourceDocument

DM_AddSearchPath

DM_ProjectFullPath

DM_ProjectFileName

DM_HierarchyMode

DM_TopLevelLogicalDocument

DM_TopLevelPhysicalDocument

DM_ComponentMappings

DM_DocumentFlattened

DM_PrimaryImplementationDocument

DM_CurrentProjectVariant

DM_ViolationCount

DM_Violations

DM_ClearViolations

DM_ErrorLevels

DM_SetErrorLevels

DM_GetDocumentFromPath

DM_ChannelDesignatorFormat

DM_ChannelRoomLevelSeperator

DM_ChannelRoomNamingStyle

DM_UserID

DM_StartNavigation

DM_StartCrossProbing

DM_DoCrossSelection SafeCall

DM_NavigationZoomPrecision

DM_InitializeOutputPath

DM_SetOutputPath

DM_GetOutputPath

DM_Compile

DM_CompileEx

DM_EditOptions

DM_UpdateConstraints

GetNavigationHistory

DM_OptionsStorage

DM_ToDoManager

DM_SetAsCurrentProject

DM_GetAllowPortNetNames

DM_GetAllowSheetEntryNetNames

DM_GetAppendSheetNumberToLocalNets

DM_SetAllowPortNetNames

DM_SetAllowSheetEntryNetNames

DM_SetAppendSheetNumberToLocalNets

DM_SetHierarchyMode

DM_GetScrapDocument

DM_GetConfigurationByName

DM_GetDefaultConfiguration

DM_GetDefaultConfigurationName

DM_SetDefaultConfigurationName

DM_GetDefaultPcbType

DM_SetDefaultPcbType

DM_HierarchyModeForCompile

IProject Interface methods

DM_AddConfigurationParameters method

(IProject interface)

Syntax

```
Procedure DM_AddConfigurationParameters(Configuration : WideString);
```

Description

A configuration is a list of constraints file which manages the mapping of pins to ports of a FPGA project. Invoke this method to add parameters of a specified configuration file for a FPGA project.

See also

IProject interface

DM_AddConfigurationParameters_Physical method

(IProject interface)

Syntax

```
Procedure DM_AddConfigurationParameters_Physical(Configuration :
WideString);
```

Description

A configuration is a list of constraints file which manages the mapping of pins to ports of a FPGA project. Invoke this method to add parameters of a specified configuration file for a FPGA project.

See also

IProject interface

DM_AddControlPanel method

(IProject interface)

Syntax

```
Procedure DM_AddControlPanel (Filename : WideString);
```

Description

The procedure adds a document to the main section of the the panel which could be part of a project or free documents.

See also

IProject interface

DM_AddGeneratedDocument method

(IProject interface)

Syntax

```
Procedure DM_AddGeneratedDocument (Filename : WideString);
```

Description

This procedure adds a new generated document referenced by its filename parameter in this current project, and this document appears in the **Generated** folder of this project on DXP Projects panel.

See also

IProject interface

DM_AddSearchPath method

(IProject interface)

Syntax

```
Procedure DM_AddSearchPath (SearchPath : WideString; IncludeSubFolders :
Boolean);
```

Description

This procedure adds a new serach path for the current project.

See also

IProject interface

DM_AddSourceDocument method

(IProject interface)

Syntax

```
Procedure DM_AddSourceDocument (Filename : WideString);
```

Description

The procedure adds a source document referenced by its filename parameter in the current project.

See also

IProject interface

DM_ChannelDesignatorFormat method

(IProject interface)

Syntax

```
Function DM_ChannelDesignatorFormat : WideString;
```

Description

This function returns the formatted channel designator string. This string is basedon the settings defined in the Multi-Channel page of the Options for Project dialog from the Project » Project Options menu item.

See also

IProject interface

DM_ChannelRoomLevelSeperator method

(IProject interface)

Syntax

```
Function DM_ChannelRoomLevelSeperator : WideString;
```

Description

The function returns the separator character for the Channel Room Level string. The default is an underline character used for room naming styles when there are paths (based on hierarchical designs).

See also

IProject interface

DM_ChannelRoomNamingStyle method

(IProject interface)

Syntax

```
Function DM_ChannelRoomNamingStyle : TChannelRoomNamingStyle;
```

Description

The function returns the TChannelRoomNamingStyle type. There are alternative styles for naming rooms on a PCB document.

See also

IProject interface

DM_ClearViolations method

(IProject interface)

Syntax

```
Procedure DM_ClearViolations;
```

Description

The procedure clears all existing violations within the project.

See also

IProject interface

DM_Compile method

(IProject interface)

Syntax

```
Function DM_Compile : LongBool;
```

Description

Invoke this function to compile the current project. Once the project is compiled, navigation of nets and comparing the differences of documents and other tasks can be performed.

See also

IProject interface

DM_CompileEx method

(IProject interface)

Syntax

```
Function DM_CompileEx(All : LongBool; Var Cancelled : LongBool) : LongBool;
```

Description

Invoke this function to compile all documents of all opened projects in DXP. Pass a Boolean parmeter in to cancel the compiling process.

See also

IProject interface

DM_ComponentMappings method

(IProject interface)

Syntax

```
Function DM_ComponentMappings (AnImplementationDocument : WideString) :
IComponentMappings;
```

Description

The function returns the IComponentMapping interface which details which PCB components are linked to Schematic components. Check the IComponentMappings interface.

See also

IProject interface

DM_ConfigurationCount method

(IProject interface)

Syntax

```
Function DM_ConfigurationCount : Integer;
```

Description

The function returns the number of configurations for the current project. To be used in conjunction with DM_Configurations function.

See also

IProject interface

DM_Configurations method

(IProject interface)

Syntax

```
Function DM_Configurations (Index : Integer ) : IConfiguration;
```

Description

The function returns the indexed configuration of a FPGA project. A configuration can have a list of different constraint files.

See also

IProject interface

DM_CurrentProjectVariant method

(IProject interface)

Syntax

```
Function DM_CurrentProjectVariant : IProjectVariant;
```

Description

The function returns the current project variant from this current project. Check out the IProjectVariant interface.

See also

IProject interface

DM_DoCrossSelection SafeCall method

(IProject interface)

Syntax

```
Procedure DM_DoCrossSelection
```

Description

Activates the cross probing function where you can jump from a Schematic object to its corresponding PCB object (both source and primary implementation documents need to be open in DXP).

See also

IProject interface

DM_DocumentFlattened method

(IProject interface)

Syntax

```
Function DM_DocumentFlattened : IDocument;
```

Description

The function returns the flattened document. A flattened document is part of a flattened hierarchy of a project and all objects of this project appear in the Instance list of the Navigator panel.

See also

IProject interface

DM_DocumentBackups

(IProject interface)

Syntax

```
Function DM_DocumentBackups : IDocumentBackups;
```

Description

This interface represents the DocumentBackups interface.

See also

IProject interface

IDocumentBackups interface

DM_EditOptions method

(IProject interface)

Syntax

```
Function DM_EditOptions(DefaultPage : WideString) : LongBool;
```

See also

IProject interface

DM_ErrorLevels method

(IProject interface)

Syntax

```
Function DM_ErrorLevels (AErrorKind : TErrorKind) : TErrorLevel;
```

Description

The function returns the error level for the specified error type. For each violation type, you can have up to four different error levels, No Report, Warning, Error and Fatal Error with four different colored folders.

See also

IProject interface

DM_GeneratedDocumentCount method

(IProject interface)

Syntax

```
Function DM_GeneratedDocumentCount : Integer;
```

Description

The function returns the number of generated documents such as those documents generated by the OutPut generator (from a OutJob document). Use this function in conjunction with the DM_GeneratedDocuments function.

See also

IProject interface

DM_GeneratedDocuments method

(IProject interface)

Syntax

```
Function DM_GeneratedDocuments (Index : Integer ) : IDocument;
```

Description

The function returns the indexed generated document which is generated by the Output Generator.

See also

IProject interface

DM_GetAllowPortNetNames method

(IProject interface)

Syntax

```
Function DM_GetAllowPortNetNames : Boolean;
```

Description

Invoke this function to check whether port net names are used for navigation in DXP or not.

See also

IProject interface

DM_GetAllowSheetEntryNetNames method

(IProject interface)

Syntax

```
Function DM_GetAllowSheetEntryNetNames : Boolean;
```

Description

Invoke this function to check whether sheet entry net anmes are used for navigation in DXP or not.

See also

IProject interface

DM_GetAppendSheetNumberToLocalNets method

(IProject interface)

Syntax

```
Function DM_GetAppendSheetNumberToLocalNets : Boolean;
```

Description

Invoke this function to check whether sheet numbers are appended to local nets or not.

See also

IProject interface

DM_GetConfigurationByName method

(IProject interface)

Syntax

```
Function DM_GetConfigurationByName(Configuration : WideString) :
IConfiguration;
```

Description

The function returns you the configuration object for the project (normally for FPGA projects) if configuration parameter is valid. A configuration file contains mapping information to link from a FPGA project to a linked PCB project.

See also

IProject interface

DM_GetDefaultConfiguration method

(IProject interface)

Syntax

```
Function DM_GetDefaultConfiguration : IConfiguration;
```

Description

The function returns the default configuration for a FPGA project.

See also

IProject interface

DM_GetDefaultConfigurationName method

(IProject interface)

Syntax

```
Function DM_GetDefaultConfigurationName : WideString;
```

Description

Returns the name of the default configuration for a FPGA project

See also

IProject interface

DM_GetDefaultPcbType method

(IProject interface)

Syntax

```
Function DM_GetDefaultPcbType : WideString;
```

See also

IProject interface

DM_GetDocumentFromPath method

(IProject interface)

Syntax

```
Function DM_GetDocumentFromPath(DocumentPath : WideString) : IDocument;
```

Description

This function returns the IDocument interface associated with the document path parameter. Otherwise a Nil value is returned.

See also

IProject interface

DM_GetOutputPath method

(IProject interface)

Syntax

```
Function DM_GetOutputPath : WideString;
```

Description

The function returns the output path for generated documents for the current project.

See also

IProject interface

DM_GetScrapDocument method

(IProject interface)

Syntax

```
Function DM_GetScrapDocument(DocumentPath : WideString) : IDocument;
```

Description

Returns the scrap document for the project. A scrap document is a temporary document used when creating a new document and once a document is saved, the contents of the scrap document is copied and freed.

See also

IProject interface

DM_HierarchyMode method

(IProject interface)

Syntax

```
Function DM_HierarchyMode : TFlattenMode;
```

Description

This function returns the hierarchy mode as a TFlattenMode parameter.

See also

IProject interface

DM_HierarchyModeForCompile method

(IProject interface)

Syntax

```
Function DM_HierarchyModeForCompile : TFlattenMode;
```

See also

IProject interface

DM_IndexOfSourceDocument method

(IProject interface)

Syntax

```
Function DM_IndexOfSourceDocument(Filename : WideString) : Integer;
```

Description

The function returns the index of the source document based on the filename of this document. This is for hierarchical or connected schematic documents.

See also

IProject interface

DM_InitializeOutputPath method

(IProject interface)

Syntax

```
Function DM_InitializeOutputPath(AnOutputType : WideString) : WideString;
```

Description

The function returns the output path for the Output Generator based on the AnOutputType parameter.

See also

IProject interface

DM_LogicalDocumentCount method

(IProject interface)

Syntax

```
Function DM_LogicalDocumentCount : Integer;
```

Description

The function returns the number of logical documents which represent the actual documents of a design project (documents that exist in the design project but are not part of the design are not logical documents). Use this function in conjunction with the DM_LogicalDocuments function.

See also

IProject interface

DM_LogicalDocuments method

(IProject interface)

Syntax

```
Function DM_LogicalDocuments (Index : Integer ) : IDocument;
```

Description

The function returns the indexed logical document of a project.

See also

IProject interface

DM_MoveSourceDocument method

(IProject interface)

Syntax

```
Procedure DM_MoveSourceDocument (Filename : WideString; NewIndex : Integer);
```

Description

The procedure re-assigns the source document referenced by the filename a new index number.

See also

IProject interface

DM_NavigationZoomPrecision method

(IProject interface)

Syntax

```
Function DM_NavigationZoomPrecision : Integer;
```

Description

Sets how precise the document zoom is when the interactive navigator is being used to trace the connection in a project.

See also

IProject interface

DM_OptionsStorage method

(IProject interface)

Syntax

```
Function DM_OptionsStorage : IOptionsStorage;
```

See also

IProject interface

DM_Outputers method

(IProject interface)

Syntax

```
Function DM_Outputers (Name : WideString) : IOutputer;
```

Description

The function returns the indexed Output Generator. An output generator could be a Simple BOM.

See also

IProject interface

DM_PhysicalDocumentCount method

(IProject interface)

Syntax

```
Function DM_PhysicalDocumentCount : Integer;
```

Description

The function returns the number of physical source documents (which are expanded logical documents of the design project). Source documents are usually schematic documents. Use this function in conjunction with the DM_PhysicalDocuments function.

See also

IProject interface

DM_PhysicalDocuments method

(IProject interface)

Syntax

```
Function DM_PhysicalDocuments (Index : Integer ) : IDocument;
```

Description

The function returns the indexed physical document of a project.

See also

IProject interface

DM_PrimaryImplementationDocument method

(IProject interface)

Syntax

```
Function DM_PrimaryImplementationDocument : IDocument;
```

Description

The function returns the primary implementation document for example PCB documents. Source documents are Schematic documents for example.

See also

IProject interface

DM_ProjectFileName method

(IProject interface)

Syntax

```
Function DM_ProjectFileName : WideString;
```

Description

This function returns the file name of this current project in DXP.

See also

IProject interface

DM_ProjectFullPath method

(IProject interface)

Syntax

```
Function DM_ProjectFullPath : WideString;
```

Description

This function returns the full path of this current project in DXP.

See also

IProject interface

DM_ProjectVariantCount method

(IProject interface)

Syntax

```
Function DM_ProjectVariantCount : Integer;
```

Description

The function returns the number of project variants for this current project.

See also

IProject interface

DM_ProjectVariants method

(IProject interface)

Syntax

```
Function DM_ProjectVariants (Index : Integer ) : IProjectVariant;
```

Description

The function returns the indexed IProjectVariant interface. A project variant interface is only a conceptual representation of a project that can have project variants. That is there is only one physical board but this same board can have certain components disabled or enabled leading to document variants. The variations of a PCB board are referred to as the IDocumentVariant and to check which components are enabled or not for this particular document variant, check out the IComponentVariant interface.

This is to be used in conjunction with the DM_ProjectVariantCount method.

See also

IProject interface

DM_RemoveSourceDocument method

(IProject interface)

Syntax

```
Procedure DM_RemoveSourceDocument (Filename : WideString);
```

Description

This procedure removes a source document referenced by its filename from this current project.

See also

IProject interface

DM_SearchPathCount method

(IProject interface)

Syntax

```
Function DM_SearchPathCount : Integer;
```

Description

The function returns the number of search paths for this current project. Use this function in conjunction with the DM_SearchPaths function.

See also

IProject interface

DM_SearchPaths method

(IProject interface)

Syntax

```
Function DM_SearchPaths (Index : Integer ) : ISearchPath;
```

Description

The function returns the indexed search path object defined for this project.

See also

IProject interface

DM_SetAllowPortNetNames method

(IProject interface)

Syntax

```
Procedure DM_SetAllowPortNetNames (AAllow : Boolean);
```

Description

Invoke this procedure to allow port net names be used for navigation.

See also

IProject interface

DM_SetAllowSheetEntryNetNames method

(IProject interface)

Syntax

```
Procedure DM_SetAllowSheetEntryNetNames (AAllow : Boolean);
```

Description

Invoke this procedure to allow sheet entry net names be used for navigation in DXP.

See also

IProject interface

DM_SetAppendSheetNumberToLocalNets method

(IProject interface)

Syntax

```
Procedure DM_SetAppendSheetNumberToLocalNets (AAppend : Boolean);
```

Description

Invoke this procedure to have the ability to append sheet numbers to local nets on a document / project.

See also

IProject interface

DM_SetAsCurrentProject method

(IProject interface)

Syntax

```
Procedure DM_SetAsCurrentProject;
```

Description

Invoke this function to set the project as the current project in DXP.

See also

IProject interface

DM_SetDefaultConfigurationName method

(IProject interface)

Syntax

```
Procedure DM_SetDefaultConfigurationName(Configuration : WideString);
```

Description

The procedure sets the name for the default configuration of a FPGA project.

See also

IProject interface

DM_SetDefaultPcbType method

(IProject interface)

Syntax

```
Procedure DM_SetDefaultPcbType(PcbType : WideString);
```

See also

IProject interface

DM_SetErrorLevels method

(IProject interface)

Syntax

```
Procedure DM_SetErrorLevels(AErrorKind : TErrorKind;AErrorLevel :
TErrorLevel);
```

See also

IProject interface

DM_SetHierarchyMode method

(IProject interface)

Syntax

```
Procedure DM_SetHierarchyMode (AFlatten : TFlattenMode);
```

Description

Invoke this function to set which hierarchy mode for this project. It can be one of the following modes:
eFlatten_Smart,eFlatten_Flat,eFlatten_Hierarchical,eFlatten_Global

See also

IProject interface

DM_SetOutputPath method

(IProject interface)

Syntax

```
Procedure DM_SetOutputPath (AnOutputPath : WideString);
```

Description
Sets the output path for generated documents to go in by the DXP output generator.

See also
IProject interface

DM_StartCrossProbing method
(IProject interface)

Syntax
```
Procedure DM_StartCrossProbing(CtrlDoesSwitch : Boolean);
```

Description
This procedure invokes the cross probing function. Both source and primary implementation documents need to be open in DXP in order for the cross probing to work.

See also
IProject interface

DM_StartNavigation method
(IProject interface)

Syntax
```
Procedure DM_StartNavigation;
```

Description
This procedure invokes the navigation panel for the current project. The project needs to be compiled first.

See also
IProject interface

DM_ToDoManager method
(IProject interface)

Syntax
```
Function DM_ToDoManager : IToDoManager;
```

Description
Invoke this function to have access to the IToDoManager object. This ToDo manager allows you to define to dos for your current project.

See also
IProject interface

DM_TopLevelLogicalDocument method

(IProject interface)

Syntax

```
Function DM_TopLevelLogicalDocument : IDocument;
```

Description

This function returns the top level logical document of this current project. A logical document is usually a Schematic document and can represent a document of a multi channel project for example.

See also

IProject interface

DM_TopLevelPhysicalDocument method

(IProject interface)

Syntax

```
Function DM_TopLevelPhysicalDocument : IDocument;
```

Description

This function returns the top level physical document of this current project. A physical document usually is a PCB document.

See also

IProject interface

DM_UpdateConstraints method

(IProject interface)

Syntax

```
Function DM_UpdateConstraints : LongBool;
```

Description

Invoke this function to update the constraint files used for a FPGA project and for corresponding PCB projects with FPGA components.

See also

IProject interface

DM_UserID method

(IProject interface)

Syntax

```
Function DM_UserID : WideString;
```

Description

The function returns a value that represents the UserID of the project.

See also

IProject interface

DM_ViolationCount method

(IProject interface)

Syntax

```
Function DM_ViolationCount : Integer;
```

Description

This function returns the number of violations reported by DXP for this current project.

See also

IProject interface

DM_Violations method

(IProject interface)

Syntax

```
Function DM_Violations(Index : Integer) : IViolation;
```

Description

Returns the indexed violation for a current project. This is to be used in conjunction with the DM_ViolationCount method.

See also

IProject interface

GetNavigationHistory method

(IProject interface)

Syntax

```
Function GetNavigationHistory : INavigationHistory;
```

Description

This function returns the status of the navigation buttons on the Navigator panel for the current project in DXP. Check out INavigationHistory interface for details.

See also

IProject interface

IAbstractVHDLProject

Overview

The IAbstractVHDLProject interface represents a project that hosts VHDL documents.

Important notes

- Inherited from IProject interface

Interface Methods

```
Function  DM_GetTargetDeviceName(ConfigurationName : WideString) :
WideString;
Function DM_GetVHDL87
Function DM_GetVerilog95
```

See also

```
Workspace Manager Interfaces
IProject interface
```

IBoardProject

Overview

The IBoardProject interface represents a project compromising of Schematic and corresponding PCB documents along with other document kinds.

Important notes

- Inherited from IProject interface

Interface Methods

IProject methods

Interface Properties

IProject Properties

See also

Workspace Manager Interfaces

IProject interface

ICoreProject

Overview

The ICoreProject interface represents the project that hosts core designs. A core project is typically created to develop pre-synthesized user models whose EDIF output becomes the model for these user defined components.

Important notes

- Inherited from IAbstractVHDLProject interface

Interface Methods

```
Function DM_CreateSymbolGenerator     : ISymbolGenerator;
Function DM_GetIncludeModelsInArchive : LongBool;
```

See also

Workspace Manager Interfaces

IAbstractVHDLProject interface

ISymbolGenerator interface

IEmbeddedProject

Overview

The IEmbeddedProject interface represents the project that hosts embedded designs that can be targetted to the hard device on the Nanoboard.

Important notes

- Inherited from IProject interface

Interface Methods

- IProject methods

Interface Properties

- IProject Properties

See also

Workspace Manager Interfaces

IProject interface

IFPGAProject

Overview

The IFPGAProject interface represents the project that hosts FPGA designs.

Important notes

- Inherited from IAbstractVHDLProject interface

Interface Methods

```
Function  DM_GetTargetBoardName (ConfigurationName : WideString) :
WideString;
```

See also

Workspace Manager Interfaces

IAbstractVHDLProject Interface

IProjectVariant interface

Overview

The IProjectVariation interface represents the project that contains component variations. Physically, there is only one PCB document with components that are specified. So for each output requirement, each document variant is generated, although there is only one PCB design document.

Interface Methods

Method	Description
Function DM_Project : IProject;	Returns the IProject interface this variant is associated with.
Function DM_Name : WideString;	Returns the name of this variant.
Function DM_Description : WideString;	Returns the description of this variant.
Function DM_VariationCount : Integer;	Returns the count of variants. To be used in conjunction with the DM_Variations(index) method.
Function DM_Variations (Index : Integer) : IComponentVariation;	Returns the indexed component variation for this project. To be used in conjunction with the DM_VariationCount method.

Configuration Constraints Interfaces

IConfiguration interface

Overview

The IConfiguration interface represents the configuration container that contains a group of constraints that can be targetted to a specific device.

Interface Methods

```
Function    DM_Name                                 : WideString;
Function    DM_ConstraintGroupCount                 : Integer;
Function    DM_ConstraintGroups(Index : Integer)    : IConstraintGroup;
Function    DM_ConstraintsFileCount                   : Integer;
Function    DM_ConstraintsFilePath(Index : Integer)   : WideString;
Function    DM_GetTargetDeviceName                    : WideString;
```

See also

Workspace Manager Interfaces

IConstraintGroup interface

Overview

The IConstraintGroup interface represents a constraint file made up of constraints (as IConstraint interface).

Important notes

- Inherited from IDMObject interface

Interface Methods

```
Function   DM_TargetKindString                  : WideString;
Function   DM_TargetId                          : WideString;
Function   DM_ConstraintCount                   : Integer;
Function   DM_Constraints(Index : Integer)      : IConstraint;
```

See also

Workspace Manager Interfaces

IConstraint interface

IConstraint interface

Overview

The IConstraint interface represents the data entry in a constraint file represented by the IConstraintGroup interface.

Important notes

- Inherited from IDMObject interface

Interface Methods

```
Function   DM_Kind : WideString;
Function   DM_Data : WideString;
```

See also

Workspace Manager Interfaces

IConstraintGroup interface

IInstalledConstraintFiles interface

Overview

The IInstalledConstraintFiles interface represents the constraint files that are installed in DXP, ie available to a FPGA project.

Interface Methods

```
Function  InstalledConstraintFileCount                             : Integer;
Function  InstalledConstraintFile    (aIndex    : Integer)     : WideString;
Function  ConstraintFileIsInstalled (aPath     : WideString) : LongBool;
Function  DefaultConstraintFile                                    : WideString;
Function  EditInstalledConstraintFiles                             : LongBool;
```

See also

Workspace Manager Interfaces

Design Objects

IBus interface

Overview

The IBus interface represents a bus object on the schematic sheet. Buses are special graphical elements that represent a common pathway for multiple signals on a schematic document. Buses have no electrical properties, and they must be correctly identified by net labels and ports.

When a schematic document is compiled, bus objects have inferred objects (wires with netlabels on them) in memory that aids the connectivity and navigation features within DXP.

Interface Methods

Method	Description
Function DM_Wires(Index : Integer) : INet;	Returns the indexed wire. Used in conjunction with the DM_WireCount function.
Function DM_Sections(Index : Integer) : INet;	Returns the indexed section. Used in conjunction with the DM_SectionCount function. Each section denotes the outline.
Function DM_WireCount : Integer;	Returns the number of wires for this IBus interface. This is used for the DM_Wires function.
Function DM_SectionCount : Integer;	Returns the number of sections for this IBus interface. This is used for the DM_Sections function.
Function DM_Scope : TNetScope;	Denotes the net scope of this IBus interface.
Function DM_Electric : TPinElectrical;	Returns the electrical property for this bus. Various values include :eElectricInput, eElectricIO, eElectricOutput, eElectricOpenCollector, eElectricPassive, eElectricHiZ, eElectricOpenEmitter, eElectricPower
Function DM_SignalType : WideString;	Returns the signal type string for this bus.
Function DM_FullBusName : WideString;	Returns the full bus name of this bus interface.
Function DM_BusName : WideString;	Returns the name of this bus interface.
Function DM_BusRange1 : WideString;	Returns the Bus range 1 value.
Function DM_BusRange2	Returns the Bus range 2 value.

Method	Description
: WideString;	
Function DM_BusRangeValue1 : Integer;	Returns the first value of the Bus range. Eg A[0..3], the first value is 0.
Function DM_BusRangeValue2 : Integer;	Returns the second value of the Bus range. Eg A[0..3], the second value is 3.
Function DM_BusKind : TBusKind;	Returns the bus kind.
Function DM_BusWidth : Integer;	Returns the bus width.
Function DM_PrefixList : TSortedUniqueStringList;	Not implemented.
Function DM_RangeDefinedByValue : Boolean;	Returns a Boolean value whether this range is defined by a two specific range values or not.
Function DM_IsLocal : Boolean;	Returns a Boolean value whether this bus is a local object or not.

IChannelClass interface

Overview

The IChannelClass interface is a PCB Channel class object interface for an existing Channel Class on a PCB document. An existing Channel (room) class contains members of specific components. Each component within a Channel Class object can either be a member or not. The ‘All Components’ Channel Class exists in every PCB document by default, it includes all Components in the document. It is not possible to change which components are members of that Channel class, but the user has full control over which components are members of any other Channel classes (which are created and named by the User)

Notes

- Inherited from IObjectClass interface.

See also

IObjectClass interface

IComponent interface

Overview

The IComponent interface is the interface or the front end of an existing schematic component on a Schematic sheet. Note that a part object is “part” of a component, that is, a multi-part component consists of part objects. For example a multiple gate integrated circuit has duplicate gates, and that a component represents the multi-part gate and a part represents the gate itself. The IComponent interface is inherited from the IPart interface.

The ISch_Component interface from Schematic API represents an existing component that can contain links to different model implementations such as PCB, Signal Integrity and Simulation models. Only one model of a particular model type (PCB footprint, SIM, SI, EDIF Macro and VHDL) can be enabled as the currently linked model, at any one time.

Interface Methods

Method	Description
Function DM_SubParts (Index : Integer) : IPart;	Returns the indexed sub-part of a multi-part component. Use the DM_SubPartCount function.
Function DM_PhysicalComponents (Index : Integer) : IComponent;	Returns the indexed physical component. Use this in conjunction with the DM_PhysicalComponentCount function.
Function DM_SubPartCount : Integer;	Returns the number of parts for this multi-part component. A standalone component returns 1 (only one part for a standalone component).
Function DM_PhysicalComponentCount : Integer;	Returns the number of physical components.
Function DM_PhysicalPath : WideString;	Returns the full physical path for this component. For example the string can consist of the schematic filename \ channel name and instance.
Function DM_UniqueId : WideString;	Returns the Unique ID string for this component so this component can be synchronized on the source document and the primary implementation document (PCB)
Function DM_UniqueIdName : WideString;	Returns the unique name portion of the Unique ID for this component.
Function DM_UniqueIdPath : WideString;	Returns the unique path portion of the Unique ID for this component. Includes the back slash.

IComponentClass interface

Overview

The IComponentClass interface is a PCB Component class object interface for an existing Component Class on a PCB document. An existing Component class contains members of specific Components. Each Component within a ComponentClass object can either be a member or not. The ‘All Components’ Component Class exists in every PCB document by default, it includes all Components in the document. It is not possible to change which components are members of that Component class, but the user has full control over which components are members of any other Component classes (which are created and named by the User).

Notes

- Inherited from IObjectClass interface.

See also

IObjectClass interface

IComponentImplementation interface

Overview

The IComponentImplementation interface is associated with an IPart/IComponent interface in terms of model linking. Note that the IComponent interface is inherited from the IPart interface.

A model represents all the information needed for a component in a given domain (a model can be a PCB footprint, Simulation file or a Signal Integrity model). A model is also called an implementation.

Each schematic component can contain links to different model implementations such as PCB, Signal Integrity and Simulation models. Only one model of a particular model type (PCB footprint, SIM, SI, EDIF Macro and VHDL) can be enabled as the currently linked model, at any one time.

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 PCB footprints, the data file link and the model is the same since the external file is the PCB footprint library.

For simulation models, there can be no data file links because these models are defined using the Spice format.

However for signal integrity models, each pin can have different pieces of information represented by ibis data files. These signal integrity models can have multiple data files, that is, each pin of a component can have a separate IBIS file. A signal integrity model can however use the DXP's central Signal Integrity database.

Thus depending on which model type, you can have a number of data file links. Each data file link describes the model name, the path to where the library is stored in and what sort of model it is.

Interface Methods

Method	Description
Function DM_Description : WideString;	Denotes the description string of the implementation model.
Function DM_ModelName : WideString;	Denotes the model name of the implementation model.
Function DM_ModelType : WideString;	Denotes the model type string.

Method	Description
Function DM_DatafileCount : Integer;	Denotes the number of data files for the model.A data file is an internal aggregrate and each data file describes the model name, the path to where the library is stored in and what implementation model type.
Function DM_DatafileLocation (Index : Integer) : WideString;	Returns the indexed data file location. Used in conjunction with the DM_DataFileCount function.
Function DM_DatafileEntity (Index : Integer) : WideString;	Returns the indexed data file entity (the name of the implementation model). Used in conjunction with the DM_DataFileCount function.
Function DM_DatafileKind (Index : Integer) : WideString;	Returns the indexed data file kind (the model kind eg PCB etc)Used in conjunction with the DM_DataFileCount function.
Procedure DM_SetDatafileLocation (Index : Integer; ALocation : WideString);	Sets the data file location which denotes the full path of the implementation model associated with the IPart/IComponent interface.
Procedure DM_SetDatafileEntity (Index : Integer; AEntity : WideString);	Sets the data file entity which denotes the name of the implementation model linked to a schematic component/part.
Procedure DM_SetDatafileKind (Index : Integer; AKind : WideString);	Sets the data file kind which denotes the type of implementation model. Example, a PCB Footprint is a PCBLIB data file kind.
Procedure DM_SetDatafileCount (ACount : Integer);	Sets the number of data files associated with the IPart/IComponent interface.
Function DM_DatafileFullPath (Index : Integer; EntityName, FileKind : WideString Var FoundIn : WideString) : WideString;	This function returns you the full path to the data file via the FoundIn parameter, if the Entity name, fhte file Kind are valid and Found In strings Used in conjunction with the DM_DataFileCount function.
Function DM_IntegratedModel : Boolean;	Denotes a boolean value whether this is a model from an integrated library or not.
Function DM_DatalinksLocked : Boolean;	Denotes a boolean value whether datalinks are locked or not.
Function DM_IsCurrent : Boolean;	Denotes a boolean value whether this model implementation is current or not.

Method	Description
Function DM_Part : IPart;	Denotes the IPart interface this IComponentImplementation interface is associated with.
Function DM_PortMap : WideString;	Denotes the mapping of pins of a component and its corresponding model.
Function DM_PortMapList : WideString;	Same as DM_PortMap function.

IComponentVariation interface

Overview

The IComponentVariation interface represents the component variant on a PCB document. There is only one physical document, but each component on this document can be specified to be a variant and when the output is generated, a specific variant document is generated. This variant output is controlled by the Output Job files.

Interface Methods

Method	Description
Function DM_ProjectVariant : IDocumentVariant;	This function returns the IProjectVariant interface which represents a container that stores the component variants for the project.
Function DM_VariationKind : TVariationKind;	This function returns the variation kind for this component.
Function DM_PhysicalDesignator : WideString;	Returns the full physical designator string for this component variant.
Function DM_UniqueId : WideString;	Returns the unique ID for this component variant.
Function DM_AlternatePart : WideString;	Returns the alternate part string for this component variant.

ICrossSheet interface

Overview

The ICrossSheet interface is a cross sheet connector object interface. Cross sheet connector objects can be used to link a net from a sheet to other sheets within a project. This method defines global connections between sheets within a project. An active cross sheet object is associated with a net.

An equivalent Cross Sheet Connector object representation is the ISch_CrossSheetConnector interface in Schematic API Reference.

Important notes

- ICrossSheet interface is inherited from INetItem interface.

See also

INetItem interface.

ILine interface

Overview

The ILine interface is a line object interface for an existing line object on a Schematic document. A line is a graphical drawing object with any number of joined segments.

An equivalent Line object representation is the ISch_Line interface in the Schematic API reference.

See also

IDMObject interface

INet interface

Overview

The INet interface is associated with an existing net object of a design document. A net is a series of connections of net identifiers (electrically aware objects such as sheet entries, pins, wires and ports) with the same net name.

That is, all connections sharing the same net name is a net and can be connected on a sheet or between sheets in a project.

Interface Methods

Method	Description
Function DM_AllNetItems (Index : Integer) : INetItem;	Returns an indexed net aware object. Use the DM_AllNetItemCount function.
Function DM_RemovedNetItems (Index : Integer) : INetItem;	Returns an indexed net item that has been removed from the schematic document.
Function DM_Directives (Index : Integer) : INetItem;	Returns an indexed directive (which could be a PCB layout directive that contains PCB fules). Use the DM_DirectiveCount function.
Function DM_Pins(Index : Integer) : INetItem;	Returns an indexed pin that is part of the current net. Use the DM_PinCount function.
Function DM_PowerObjects (Index :	Returns an indexed power object that is part of the

Method	Description
Integer) : INetItem;	current net. Use the DM_PowerObjectCount function.
Function DM_Ports (Index : Integer) : INetItem;	Returns an indexed port that is part of the current net. Use the DM_PortCount function.
Function DM_CrossSheetConnectors (Index : Integer) : ICrossSheet;	Returns an indexed cross sheet connector that is part of the current net. Use the DM_CrossSheetConnectorCount function.
Function DM_NetLabels (Index : Integer) : INetItem;	Returns an indexed net label that is part of the current net. Use DM_NetLabelCount function.
Function DM_SheetEntrys (Index : Integer) : INetItem;	Returns an indexed sheet entry that is part of the current net. Use DM_SheetEntryCount function.
Function DM_Lines (Index : Integer) : ILine;	Returns an indexed line that is part of the current net. use the DM_LineCount function.
Function DM_SubWires (Index : Integer) : INet;	Returns an indexed sub wire (part of a bus object). Use the DM_SubWireCount. A bus object conceptually carries multiple wires.
Function DM_AllNetItemCount : Integer;	Returns the number of net aware objects (that is inherited from the INetItem interface).
Function DM_RemovedNetItemCount : Integer;	Returns the number of net items that have been removed from the nets.
Function DM_DirectiveCount : Integer;	Returns the number of directives associated with this net.
Function DM_PinCount : Integer;	Returns the number of pins associated with this net.
Function DM_PowerObjectCount : Integer;	Returns the number of power objects associated with this net.
Function DM_PortCount : Integer;	Returns the number of ports associated with this net.
Function DM_CrossSheetConnectorCount : Integer;	Returns the number of cross sheet connectors associated with this net.
Function DM_NetLabelCount : Integer;	Returns the number of net labels associated with this net.
Function DM_SheetEntryCount : Integer;	Returns the number of sheet entries associated with this net.

Method	Description
Function DM_LineCount : Integer;	Returns the number of lines associated with this net.
Function DM_SubWireCount : Integer;	Returns the number of sub wires associated with this net.
Function DM_Electric : TPinElectrical;	Returns the type of electrical property the pin is associated with. Various values include :eElectricInput, eElectricIO, eElectricOutput, eElectricOpenCollector, eElectricPassive, eElectricHiZ, eElectricOpenEmitter, eElectricPower
Function DM_ElectricalString : WideString;	Returns the electrical property associated with this net.
Function DM_SignalType : WideString;	Returns the signal type property associated with this net.
Function DM_AutoNumber : Integer;	Returns the auto number value used for auto-numbering nets.
Function DM_Scope : TNetScope;	Denotes the scope of this net.
Function DM_CalculatedNetName : WideString;	Denotes the system generated name for this net.
Function DM_HiddenNetName : WideString;	Denotes the hidden net name (like power nets).
Function DM_IsAutoGenerated : Boolean;	Denotes a boolean value whether this net has been system generated or not.
Function DM_IsLocal : Boolean;	Denotes whether this net is a local net restricted to the document or not.
Function DM_NetNumber : WideString;	Denotes the net number of this net.
Function DM_NetName : WideString;	Denotes the net name of this net.
Function DM_FullNetName : WideString;	Denotes the full net name (includes the bus index and so on).
Function DM_BusRange1 : WideString;	Returns the bus range 1 string.

Method	Description
Function DM_BusRange2 : WideString;	Returns the bus range 2 string.
Function DM_BusRangeValue1 : Integer;	Returns the first index of the Bus range. Eg. A[1..6], the bus range1 is 1.
Function DM_BusRangeValue2 : Integer;	Returns the last index of the Bus Range. Eg A[0..4], the bus range 2 is 4.
Function DM_BusIndex : Integer;	Returns the bus index. An IBus interface is inherited from a INetItem interface.
Function DM_BusWidth : Integer;	Returns the bus width. An IBus interface is inherited from a INetItem interface.
Function DM_BusKind : TBusKind;	Returns the bus kind. Refer to the TBusKind for different types.
Function DM_IsBusElement : Boolean;	Returns a Boolean value whether this bus element exists or not for this INetItem interface. An IBus interface is inherited from a INetItem interface.
Function DM_IsBusSection : Boolean;	Returns a Boolean value whether the bus section exists or not for this INetItem interface. An IBus interface is inherited from a INetItem interface.
Function DM_IsBusMember : Boolean;	Returns a Boolean value whether this bus member exists or not for this INetItem interface. An IBus interface is inherited from a INetItem interface.
Function DM_RangeDefinedByValue : Boolean;	Returns a boolean value whether the range has been defined by a two specific range values or not.
Function DM_BusPrefix : WideString;	Returns the bus prefix as used in this net.
Function DM_CountOfNonPinItems : Integer;	Returns the number of non-pin objects used on the current sheet or the project.
Function DM_CountOfElectricalType (AElectric : TPinElectrical) : Integer;	Returns the number of electrical types used by the current sheet or the project.
Function DM_SuppressERC : Boolean;	Returns a boolean value whether the ERC has been suppressed for this net or not.
Function DM_BusSectionParent : INet;	Returns an INet interface for the bus section.

INetClass interface

Overview

The INetClass interface is a PCB Net Class object interface for an existing NetClass on a PCB document. An existing Net class contains members of specific Net objects. Each Net within a NetClass object can either be a member, or not. The 'All Nets' Net Class exists in every PCB file by default; it includes all Nets in the document. It is not possible to change which Nets are members of that Net Class, but the user has full control over which Nets are members of any other Net Classes (which are created and named by the user).

Notes

- An INetClass interface is inherited from the IObjectClass interface.

See also

IObjectClass

INetItem interface

Overview

The INetItem interface represents the ancestor or parent interface for the following interfaces – IBus, ICrossSheetConnector, IPin, IPort, INetlabel, ISheetEntry and IPowerObject interfaces. These interface objects have a net property and thus these objects can be part of a net.

Interface Methods

Method	**Description**
Function DM_OwnerNetLogical : INet;	Denotes whether this net aware object is associated with the net of a logical document.
Function DM_OwnerNetPhysical : INet;	Denotes whether this net aware object is associated with the netof a physical document.
Function DM_ParentID : WideString;	Denotes the parent ID or the Sheet document name / Net Name property where this interface is associated with. For example a sheet entry on a sheet symbol object's parent ID is the name of the schematic sheet where the port is.
Function DM_Electric : TPinElectrical;	Denotes the electrical pin property for a net aware object.
Function DM_Id : WideString;	Denotes the Id for this net aware object.
Function DM_NetName : WideString;	Returns the net name of the net where the net aware object is associated with.
Function DM_FlattenedNetName :	Returns the net name of the flattened net where the

Method	Description
WideString;	net aware object is associated with.
Function DM_Electrical : TPinElectrical;	Returns the electrical pin property.
Function DM_ElectricalString : WideString;	Returns the electrical property string.
Function DM_SignalType : WideString;	Returns the signal type string.
Function DM_BusRange1 : WideString;	Returns the bus range 1 string.
Function DM_BusRange2 : WideString;	Returns the bus range 2 string.
Function DM_BusRangeValue1 : WideString;	Returns the first index of the Bus range. Eg. A[1..6], the bus range1 is 1.
Function DM_BusRangeValue2:: Integer;	Returns the last index of the Bus Range. Eg A[0..4], the bus range 2 is 4.
Function DM_BusKind : TBusKind;	Returns the type of bus. An IBus interface is inherited from a INetItem interface.
Function DM_BusIndex : Integer;	Returns the bus index. An IBus interface is inherited from a INetItem interface.
Function DM_BusWidth : Integer;	Returns the bus width. An IBus interface is inherited from a INetItem interface.
Function DM_BusPrefix : WideString;	Returns the bus prefix. An example, a bus object could have this A[0..7] net label, and the prefix is A. An IBus interface is inherited from a INetItem interface.
Function DM_IsAutoGenerated : Boolean;	Returns a Boolean value whether this INetItem has been automatically generated by DXP or not.
Function DM_IsBusMember : Boolean;	Returns a Boolean value whether this bus member exists or not for this INetItem interface. An IBus interface is inherited from a INetItem interface.
Function DM_IsBusElement : Boolean;	Returns a Boolean value whether this bus element exists or not for this INetItem interface. An IBus interface is inherited from a INetItem interface.
Function DM_IsBusSection : Boolean;	Returns a Boolean value whether the bus section exists or not for this INetItem interface. An IBus interface is inherited from a INetItem interface.
Function DM_RangeDefinedByValue : Boolean;	Returns a Boolean value whether the range is defined by a two specific range values or not.

Method	Description
Function DM_Part : IPart;	Returns the IPart interface.
Function DM_PartId : Integer;	Returns the Part ID value. A part object is a composite of a multi-part component, and thus each part object is referenced by its Part Id.
Function DM_DisplayMode : TDisplayMode;	Returns the display mode for this part object. A part object can have up to 254 alternative graphical displays along with the normal graphical display.
Function DM_PinName : WideString;	Returns the Pin name that this INetItem interface is associated with. Since an IPin interface is inherited from an INetItem interface.
Function DM_PinNumber : WideString;	Returns the Pin Number.that this INetItem interface is associated with. An IPin interface is inherited from an INetItem interface.
Function DM_FullPinName : WideString;	Returns the full Pin name and number that this INetItem interface is associated with. An IPin interface is inherited from an INetItem interface.
Function DM_IsHidden : Boolean;	Returns whether this pin object is hidden or not. An IPin interface is inherited from an INetItem interface.
Function DM_LogicalPartDesignator : WideString;	Returns the logical part designator for this INetItem interface.
Function DM_FullLogicalPartDesignator : WideString;	Returns the full logical part designator for this INetItem interface.
Function DM_PhysicalPartDesignator : WideString;	Returns the logical part designator and the channel instance for this INetItem Interface.
Function DM_FullPhysicalPartDesignator : WideString;	Returns the full logical part designator and the channel instance for this INetItem Interface.
Function DM_PartUniqueId : WideString;	Returns the Unique ID for this part the NetItem is associated with.
Function DM_PartType : WideString;	Returns the part type for this INetItem associated with an IPart object.
Function DM_FootPrint : WideString;	Returns the Footprint string for this INetItem associated with an IPart object.
Function DM_PinNameNoPartId : WideString;	Returns the Pin Name Number and Part ID string for this INetItem associated with an Part object. A pin is

Method	Description
	part of a part / component.
Function DM_FullUniqueId : WideString;	Returns the full Unique ID string for this INetItem interface.
Function DM_PartSwapId : WideString;	Returns the wide string for the part swap Id.
Function DM_PinSwapId : WideString;	Returns the wide string for the pin swap Id.
Function DM_SheetSymbol : ISheetSymbol;	Returns the ISheetSymbol interface where this INetItem (representing a ISheetEntry interface if it exists) is associated with. If not, a nil value is returned.
Function DM_ParentSheetSymbolSheetName : WideString;	Returns the parent sheet symbol sheet name string associated with this INetItem interface (which is a sheet entry object).
Function DM_ParentSheetSymbolName : WideString;	Returns the parent sheet symbol name associated with this INetItem interface (which is a SheetEntry object).
Function DM_LinkObject : INetItem;	Denotes the linked object to a sheet entry or port from a port or a sheet entry respectively. This method is for port objects that are connected from child schematic sheets to sheet entries of sheet symbols on a parent sheet.

INetLabel interface

Overview

The INetLabel interface is a net label interface to an existing net label object on the schematic sheet document. A net describes a connection from one component pin, to a second pin, and then to a third pin and so on.

Notes

- The INetLabel interface is inherited from the INetItem interface.
- An equivalent NetLabel object representation is the ISch_NetLabel class in Schematic API Reference.

See also

INetItem interface.

IObjectClass interface

Overview

The IObjectClass interface is the ancestor object class interface for Channel Class, Component Class and Net Class interfaces.

Interface Methods

Method	Description
Function DM_Name : WideString;	Returns the name of the Object class (one of its descendants ie Channel Class, Component class or Net class)
Function DM_MemberCount : Integer;	Returns the number of members associated with the object class (one of its descendants ie Channel Class, Component class or Net class). This method is to be used in conjunction with the DM_Members(index) method.
Function DM_Members (Index : Integer) : WideString;	Returns the indexed member of the object class (one of its descendants that is, a channel class, component class or a net class).

IParameter interface

Overview

The IParameter interface is a parameter object interface to an existing parameter object on a schematic sheet. There are two types of parameters – system parameters which are owned by a schematic document and parameters owned by certain schematic design objects.

A parameter is a child object of a Parameter Set, Part, Pin, Port, or Sheet Symbol object. A Parameter object has a Name property and Value property which can be used to store information, thus the parameters are a way of defining and associating information and could include strings that identify component manufacturer, date added to the document and also a string for the component's value (e.g. 100K for a resistor or 10PF for a capacitor).

Each parameter has a Unique Id assigned to it. This is used for those parameters that have been added as design rule directives. When transferring the design to the PCB document, any defined rule parameters will be used to generate the relevant design rules in the PCB. These generated rules will be given the same Unique Ids, allowing you to change rule constraints in either schematic or PCB and push the change across when performing a synchronization.

An equivalent object representation is the ISch_Parameter class in the Sch API reference.

Interface Methods

Method	Description
Function DM_Name : WideString;	Denotes the name of the parameter object.

Method	Description
Function DM_ConfigurationName : WideString;	Returns the configuration name, that the parameter object is associated with.
Function DM_Kind : TParameterKind;	Denotes the specific kind that can be assigned to this parameter object. String, Boolean, Integer or float..
Function DM_Value : WideString;	Denotes the value placeholder for this parameter object.
Function DM_RawText : WideString	Returns the raw text for this parameter object.
Function DM_UniqueId : WideString;	Any parameter that is configured as a container for design rule directives need to have a unique ID that will be ported onto the corresponding PCB implementation document.
Function DM_Description : WideString;	Denotes the description of this parameter object.
Function DM_NewName : WideString;	Denotes the New Name for the parameter object, especially when there is an ECO change. You can then compare the original and new names.
Function DM_NewValue : WideString;	Denoes the New Value for the parameter object, especially when there is an ECO change. You can then compare the original and new values.
Function DM_OriginalOwner : IDMObject;	This function returns the interface of the owner object this parameter object is associated with.
Function DM_Visible : Boolean;	Denotes whether this parameter object is visible or not.

IPart interface

Overview

The IPart interface is the interface or the front end of an existing schematic part on a Schematic sheet. A part object is “part” of a component, that is, a multi-part component consists of part objects. For example a multiple gate integrated circuit has duplicate gates, and that a component represents the multi-part gate and a part represents the gate itself.

An equivalent component object representation is the ISch_Component class in Schematic API Reference. The ISch_Component interface represents a component that can contain links to different model implementations such as PCB, Signal Integrity and Simulation models. Only one model of a particular model type (PCB footprint, SIM, SI, EDIF Macro and VHDL) can be enabled as the currently linked model, at any one time.

Interface Methods

Method	Description
Function DM_Pins(Index : Integer) : INetItem;	Returns the INetItem interface for the specified indexed Pin of a Schematic Component.
Function DM_Implementations(Index : Integer) : IComponentImplementation;	Returns the particular IComponentImplementation for the specified indexed implementations of a Schematic component.
Function DM_CurrentImplementation (AType : WideSTring) : IComponentImplementation;	Returns the current implementation.
Function DM_PinCount : Integer;	Returns the number of pins for this schematic component.
Function DM_ImplementationCount : Integer;	Returns the number of implementations of this schematic component.
Function DM_DesignatorLocationX : Integer;	Returns the location X of the designator associated with this component.
Function DM_DesignatorLocationY : Integer;	Returns the location Y of the designator associated with this component.
Function DM_ReferenceLocationX : Integer;	Returns the reference location X of the designator associated with this component.
Function DM_ReferenceLocationY : Integer;	Returns the reference location Y of the designator associated with this component.
Function DM_CenterLocationX : Integer;	Returns the central location X of the designator associated with this component.
Function DM_CenterLocationY : Integer;	Returns the central location Y of the designator associated with this component.
Function DM_FirstPinLocationX : Integer;	Denotes the reference X location of the first pin of a part
Function DM_FirstPinLocationY : Integer;	Denotes the reference Y location of the first pin of a part
Function DM_Layer : WideString;	Denotes which layer this part is on.
Function DM_Rotation : Double;	Denotes the rotation property of a part.

Method	Description
Function DM_Footprint : WideString;	Denotes the footprint string that this part is associated with.
Function DM_Comment : WideString;	Denotes the comment string for this part.
Function DM_SubProject : WideString;	Returns the sub project string of this part. A part can represent a schematic sheet, like a sheet symbol.
Function DM_ChildVHDLEntity : WideString;	Returns the Child VHDL entity string
Function DM_PhysicalDesignator : WideString;	Denotes the physical designator of a part.
Function DM_FullPhysicalDesignator : WideString;	Denotes the full physical designator of a part (which includes the logical designator and the channel instance string).
Function DM_FullLogicalDesignator : WideString;	Denotes the full logical designator of a part.
Function DM_ChildProjectSheet : IDocument;	Denotes the IDocument interface representing the child project sheet associated with this part.
Function DM_InstanceCount : Integer;	Returns the number of instances of this part.
Function DM_LogicalDesignator : WideString;	Denotes the logical designator of this part.
Function DM_AssignedDesignator : WideString;	Denotes the assigned designator for this part.
Function DM_CalculatedDesignator : WideString;	Denotes the system generated designator for this part.
Function DM_UniqueId : WideString;	Denotes the Unique ID for this part. Unique IDs are used in Schematic – PCB documents synchronization so that Sch components and its corresponding PCB components are in sync.
Function DM_UniqueIdName : WideString;	Denotes the Unique ID name of this part.
Function DM_UniqueIdPath : WideString;	Denotes the Unique ID path of this part (includes the back slash).
Function DM_PartType: WideString;	Denotes the part type for this part. (Footprint type).

Method	Description
Function DM_LibraryReference : WideString;	Denotes the name of the component from the library
Function DM_SourceLibraryName : WideString;	Denotes the name of the source library where the schematic component and its associated part come from.
Function DM_SourceUniqueId : WideString;	Unique IDs (UIDs) are used to match each schematic component to the corresponding PCB component. When a schematic is transferred to a blank PCB using the Update command, the source reference links for each PCB footprint is populated with source library pathnames. The UID is a system generated value that uniquely identifies the source component.
Function DM_SourceHierarchicalPath : WideString;	Denotes the source reference path to the PCB component. The path can be multi level depending on whether it is a multi channel or a bormal design. When a schematic is transferred to a blank PCB using the Update command, the source reference links for each PCB footprint is populated with source library path names.
Function DM_SourceDesignator : WideString;	Denotes the current designator of the source component from the corresponding schematic.
Function DM_Description : WideString;	Denotes the description of the reference link to a source component or as a device name.
Function DM_PartID : Integer;	Denotes the PartID for this part. A multi-part component references each part by its PartID, for example a four part component has four unique PartIDs.
Function DM_DisplayMode : TDisplayMode;	Denotes one of the 255 display modes. The mode 0 is the normal graphical display for this part object. The other 254 modes are alternative graphical displays of this same part object.
Function DM_MaxPartCount : Integer;	Returns the maximum part count for this part object.
Function DM_LogicalOwnerDocument : IDocument;	Denotes the IDocument representing the logical owner document that this part is associated to a schematic component.
Function DM_ChannelOffset :	The offset represents which part is offset in relation to the reference channel and the associated channels are

Method	Description
Integer;	also affected.
Function DM_DesignatorLocked : Boolean;	Denotes whether or not the designator string is locked (unmoveable).
Function DM_PartIdLocked : Boolean;	Denotes whether or not the part id string is locked (unmoveable).
Function DM_ComponentKind : TComponentKind;	Denotes the component kind that this part is represented as. in the BOM and are maintained during synchronization. A component kind can be one of the following: eComponentKind_Standard : These components possess standard electrical properties, are always synchronized and are the type most commonly used on a board. eComponentKind_Mechanical: These components do not have electrical properties and will appear in the BOM. They are synchronized if the same components exist on both the Schematic and PCB documents. An example is a heatsink. eComponentKind_Graphical: These components are not used during synchronization or checked for electrical errors. These components are used, for example, when adding company logos to documents. eComponentKind_NetTie_BOM: These components short two or more different nets for routing and these components will appear. eComponentKind_NetTie_NoBOM: These components short two or more different nets for routing and these components will NOT appear in the BOM and are maintained during synchronization. Note the TComponentKind type is defined from RT_Workspace unit.
Function DM_NewDesignator : WideString;	Denotes the new designator for this part.
Function DM_NewPartId : Integer;	Denotes the new part id for this part.

Method	Description
Function DM_Height : Integer;	Denotes the height property of the part object. A part object is “part” of a multi-part component.
Procedure DM_AddConfigurationParameters;	Add configuration parameters to this part.

IPin interface

Overview

The IPin interface is a pin object interface to an existing pin object on the schematic. Pins are special objects that have electrical characteristics and are used to direct signals in and out of components. Pins connect directly to other pins, wires, net labels, sheet entries or ports.

Notes

- The IPin interface is inherited from the INetItem interface.
- The pins are part of a schematic component, thus if you wish to have access to the pins, invoke the DM_Pins and DM_PinCount method call from the part object interface.
- An equivalent Pin object representation is the ISch_Pin interface in Schematic API Reference

Example

```
For J := 0 to Doc.DM_ComponentCount - 1 Do
Begin
    Comp := Doc.DM_Components(J);
    //Comp.DM_Footprint;
    //Comp.DM_Comment;
    For K := 0 to Comp.DM_PinCount - 1 Do
    Begin
        Pin := Comp.DM_Pins(K);
        PinName := Pin.DM_PinNumber;
        // Check for parts of a multi-part component that are not used in
the project
        // then add 'No Net' for unused pins...
        If Pin.DM_FlattenedNetName = '?' Then
           // these pins of the part is not used on the schematic.
    End;
End;
```

See also

INetItem interface

IPort interface

Overview

The IPort interface is a port object interface to an existing port object on the schematic. A port is used to connect a net on one sheet to Ports with the same name on other sheets. Ports can also connect from child sheets to Sheet entries, in the appropriate sheet symbol on the parent sheet.

Notes

- The IPort interface is inherited from the INetItem interface.
- An equivalent Port object representation is the ISch_Port class in Schematic API Reference.

Example

```
Var
    DM_Port        : IPort;
    I              : Integer;
    S              : TDynamicString;
    ServerDocument : IServerDocument;
Begin
    If ADM_Document = Nil Then Exit;
    If Not ADM_Document.DM_ValidForNavigation Then Exit;

    S := ADM_Document.DM_FullPath;
    ServerDocument := Client.GetDocumentByPath(PChar(S));
    If ServerDocument = Nil Then Exit;

    If Not StringsEqual(TDynamicString(ServerDocument.Kind), 'Sch') Then
Exit;

    For i := 0 To ADM_Document.DM_PortCount - 1 Do
    Begin
        DM_Port := ADM_Document.DM_Ports(i);
        If DM_Port <> Nil Then
          If DM_Port.DM_ValidForNavigation Then
        Begin
           // port is available for manipulation here.
        End;
    End;
End;
```

See also

INetItem interface

IPowerObject interface

Overview

The IPowerObject interface is a power object interface to an existing power object on the schematic. Power ports are special symbols that represent a power supply and are always identified by their net names.

Notes

- The IPowerObject interface is inherited from the INetItem interface.
- An equivalent PowerObject object representation is the ISch_PowerObject class in Sch API Reference.

See also

INetItem interface.

IRoom interface

Overview

The IRoom interface is a PCB room object. A room is controlled by the room design rule. This room serves as a boundary constraint for a group of specified components as a component or channel class.

Interface Methods

Method	Description
Function DM_LX : Integer;	Returns the lower X coordinate of the room object.
Function DM_LY : Integer;	Returns the lower Y coordinate of the room object.
Function DM_HX : Integer;	Returns the higher X coordinate of the room object.
Function DM_HY : Integer;	Returns the higher Y coordinate of the room object.
Function DM_RoomName : WideString;	Returns the name of this room object.
Function DM_Scope1Expression : WideString;	Returns the scope 1 expression which describes the scope of this room object.
Function DM_Layer : Integer;	Returns the PCB layer where the room resides on.

IRule interface

Overview

The IRule interface represents the one of the rules attached to a parameter within the PCB Layout directive (as a Parameter Set object with a small flag symbol) on a net aware object on a schematic object. A parameter set object can be placed on the schematic sheet by the Place » Directives » PCB Layout menu item.

This PCB Layout directive allows you to assign PCB layout information to a net in the schematic. When a PCB is created from the schematic, the information in the PCB layout directive is used to create relevant PCB design rules.

Interface Methods

Method	Description
Function DM_RuleKind : Integer;	Denotes the type of PCB Rule
Function DM_Scope1Expression : WideString;	Denotes the first scope expression string. The scope of Design rules are determined by the defined boundary or objects.
Function DM_Scope2Expression : WideString;	Denotes the second scope expression string. The scope of Design rules are determined by the defined boundary or objects.
Function DM_MaxWidth : Integer;	Denotes the Maximum Width rule property of a PCB rule.
Function DM_MinWidth : Integer;	Denotes the Minimum Width rule property of a PCB Rule.
Function DM_PreferedWidth : Integer;	Denotes the preferred Width rule property of a PCB Rule.
Function DM_ViaHole : Integer;	Denotes the Via Hole rule property of a Routing Via style PCB Rule.
Function DM_ViaWidth : Integer;	Denotes the Via width rule property of a Routing Via style PCB Rule.
Function DM_MinViaHole : Integer;	Denotes the min Via Hole rule property of a Routing Via style PCB Rule.
Function DM_MaxViaHole : Integer;	Denotes the max Via Hole rule property of a Routing Via style PCB Rule.
Function DM_MinViaWidth :	Denotes the min Via width rule property of a Routing Via style PCB

Method	Description
Integer;	Rule.
Function DM_MaxViaWidth : Integer;	Denotes the max Via width rule property of a Routing Via style PCB Rule.
Function DM_ViaStyle : Integer;	Denotes the topology (Shortest, Horizontal, Vertical, Daisy-Simple, Daisy-MidDriven, Daisy-Balanced and Daisy-StarBurst) rule property of a Routing Topology PCB Rule.
Function DM_Topology : Integer;	Denotes the topology (Shortest, Horizontal, Vertical, Daisy-Simple, Daisy-MidDriven, Daisy-Balanced and Daisy-StarBurst) rule property of a Routing Topology PCB Rule.
Function DM_Priority : Integer;	Denotes the priority of the PCB Design Rule. The priority value of 1 denotes the highest priority.
Function DM_RoutingLayers (IndexLayer : Integer) : Integer;	Denotes the indexed routing layer rule property (Top layer, Mid1-Mid30, Bottom Layer) of a Routing Layers PCB rule.
Function DM_Attributes : WideString;	Denotes the attributes of the IRule interface.
Function DM_Description : WideString;	Denotes the description of this IRule interface.
Function DM_RuleName : WideString;	Denotes the name of this IRule interface representing a PCB rule.
Function DM_UniqueId : WideString;	Each rule has a Unique ID assigned so that when Schematic and PCB documents are synchronized, the ECO knows which rules to update or apply to/from.

ISheetSymbol interface

Overview

The ISheetSymbol interface is a sheet symbol interface to an existing sheet symbol object on the schematic. Sheet symbols represent other schematic sheets (often referred to as a child sheet). The link between a sheet symbol and other schematic sheets is the FileName attribute, which must be the same as the name of the child sheet.

An equivalent Sheet Symbol object representation is the ISch_SheetSymbol class in Sch API Reference.

Interface Methods

Method	Description
Function DM_SheetEntries (Index : Integer) : INetItem;	Returns the number of sheet entries that are associated with this sheet symbol. Since a sheet entry is of a INetItem type, thus a INetItem interface is returned.
Function DM_SheetEntryCount : Integer;	Returns the number of sheet entries associated with this sheet symbol object.
Function DM_ChildSheet (Index : Integer) : IDocument;	Returns the indexed child sheet associated with this sheet symbol object. Use in conjunction with the DM_ChildSheetCount method.
Function DM_ChildSheetCount Integer;	Returns the number of child sheets associated with this sheet symbol object.
Function DM_SheetSymbolFileName : WideString;	Returns the filename which is a link between this sheet symbol object and the other schematic sheet.
Function DM_LogicalDesignator : WideString;	Returns the logical designator of this sheet symbol. A logical designator is not unique, since logical designators are used in multi channel designs.
Function DM_CalculatedDesignator : WideString;	Returns the calculated designator string which contains the hierarchical path and the logical designator strings. Only when a project is compiled and up to date, designators of sheet symbols are calculated based on the physical documents they are on.
Function DM_PhysicalDesignator : WideString;	Returns the designator of this sheet symbol. Every physical designator is unique.
Function DM_UniqueId : WideString;	Returns the unique ID of this sheet symbol object.

ISheetEntry interface

Overview

The **ISheetEntry** interface is a sheet entry object interface to an existing sheet entry object on the schematic. A sheet entry creates a connection between the net touching on the parent sheet, to a Port with the same name on the child sheet.

Notes

- The **ISheetEntry** interface is inherited from the **INetItem** interface.

- An equivalent SheetEntry object representation is the **ISch_SheetEntry** class in Sch API Reference.

See also

INetItem interface.

ITextFrame interface

Overview

The ITextFrame interface is a text frame object for an existing text frame on a schematic document. It is a container holding lines of text like a memo.

An equivalent TextFrame object representation is the ISch_TextFrame interface in the Schematic API reference.

Interface Methods

Method	Description
Function DM_Text : WideString;	This function returns the text string from this current TextFrame object.

See also

IDMObject interface

IViolation interface

Overview

The IViolation interface represents a violation object on a design document in the Workspace Manager of DXP.

Interface Methods

Method	Description
Function DM_ErrorKind : TErrorKind;	Returns the kind of error this violation has been assigned to.
Function DM_ErrorLevel : TErrorLevel;	Returns the level of error this violation has been assigned to. Various error levels include : eErrorLevelNoReport,eErrorLevelWarning,eErrorLevelError,eErrorLevelFatal
Function DM_CompilationStage : TCompilationStage;	This function returns the status of the complation stage: during compilation or during flattening process.

Method	Description
Procedure DM_AddRelatedObject (AnObject : IDMObject);	This procedure adds the object that is part of the violation.
Function DM_RelatedObjectCount : Integer;	This function returns the number of related objects of the violation.
Function DM_RelatedObjects (Index : Integer) : IDMObject;	This function returns the indexed related object of the violation.
Function DM_DescriptorString : WideString;	This function returns the description string for this violation interface.
Function DM_DetailString : WideString;	This function returns the detailed description stirng of this violation interface.

Signals Manager interfaces

IEntityPort interface

Important notes

- Inherited from ISignalNode interface

Interface Methods

- All methods from ISignalNode interface.

See also

Workspace Manager Interfaces

ISignalManager interface

ISignalNode interface

IExternalParameter interface

Overview

The IExternalParameter interface defines the external parameter object

Interface Methods

Method	Description
Function DM_GetSection : WideString;	Returns the Section string of the external parameter interface.
Function DM_GetName : WideString;	Returns the Name string of the external parameter interface.
Function DM_GetValue : WideString;	Returns the Value string of the external parameter interface.
Procedure DM_SetValue(AValue : WideString);	Sets the new value string for this external parameter.

IInstance interface

Interface Methods

```
Function     DM_Part           : IPart;
Function     DM_SheetSymbol    : ISheetSymbol;
Function     DM_Ports (Index : Integer)  :  IInstancePort;
Function     DM_PortCount      : Integer;
```

```
Function     DM_Designator    : WideString;
Function     DM_InstanceType : WideString;
```

See also

Workspace Manager Interfaces

ISignalManager interface

IPart interface

ISheetSymbol interface

IInstancePort interface

IInstancePort interface

Important notes

- Inherited from ISignalNode interface

Interface Methods

- All methods from ISignalNode interface.

See also

Workspace Manager Interfaces

ISignalManager interface

ISignalNode interface

ISignal interface

Interface Methods

```
Function     DM_Namers      (Index : Integer) : ISignalNode;
Function     DM_SubNets     (Index : Integer) : ISubNet;
Function     DM_DriverLinks(Index : Integer) : ISignalLink;
Function     DM_TargetLinks(Index : Integer) : ISignalLink;
Function     DM_NamerCount       : Integer;
Function     DM_SubNetCount      : Integer;
Function     DM_DriverLinkCount : Integer;
Function     DM_TargetLinkCount : Integer;
Function     DM_DriverBits  (BitNo,Index : Integer) : ISignalNode;
Function     DM_TargetBits  (BitNo,Index : Integer) : ISignalNode;
Function     DM_DriverBitCount(BitNo : Integer) : Integer;
Function     DM_TargetBitCount(BitNo : Integer) : Integer;
Function     DM_Prefix           : WideString;
Function     DM_Range1           : WideString;
```

```
Function     DM_Range2           : WideString;
Function     DM_RangeValue1      : Integer;
Function     DM_RangeValue2      : Integer;
Function     DM_BusKind          : TBusKind;
Function     DM_Width            : Integer;
Function     DM_RangeMax         : Integer;
Function     DM_RangeMin         : Integer;
Function     DM_PrimaryNode      : ISignalNode;
Function     DM_PowerNode        : ISignalNode;
Function     DM_PowerName        : WideString;
```

See also

Workspace Manager Interfaces

ISignalManager interface

ISignalNode interface

ISubNet interface

ISignalLink interface

TBusKind interface

ISignalLink

Interface Methods

```
Function     DM_DriverNode                : ISignalNode;
Function     DM_TargetNode                : ISignalNode;
Function     DM_DriverSignal              : ISignal;
Function     DM_DriverNodeRange1          : WideString;
Function     DM_DriverNodeRange2          : WideString;
Function     DM_DriverNodeRangeValue1     : Integer;
Function     DM_DriverNodeRangeValue2     : Integer;
Function     DM_TargetSignal              : ISignal;
Function     DM_TargetNodeRange1          : WideString;
Function     DM_TargetNodeRange2          : WideString;
Function     DM_TargetNodeRangeValue1     : Integer;
Function     DM_TargetNodeRangeValue2     : Integer;
Function     DM_DriverRangeMax            : Integer;
Function     DM_DriverRangeMin            : Integer;
Function     DM_TargetRangeMax            : Integer;
Function     DM_TargetRangeMin            : Integer;
```

See also

Workspace Manager Interfaces

ISignalManager interface

ISignal interface

ISignalNode interface

ISignalManager interface

Interface Methods

```
Function     DM_SubNets        (Index : Integer) : ISubNet;
Function     DM_Instances      (Index : Integer) : IInstance;
Function     DM_InstanceKinds  (Index : Integer) : IInstance;
Function     DM_Signals        (Index : Integer) : ISignal;
Function     DM_EntityPorts    (Index : Integer) : IEntityPort;

Function     DM_SubNetCount        : Integer;
Function     DM_InstanceCount      : Integer;
Function     DM_InstanceKindCount  : Integer;
Function     DM_SignalCount        : Integer;
Function     DM_EntityPortCount    : Integer;
```

See also

Workspace Manager Interfaces

ISubNet interface

IInstance interface

ISignal interface

IEntityPort interface

ISignalNode

Interface Methods

```
Function     DM_NetItem              : INetItem;
Function     DM_SubNet               : ISubNet;
Function     DM_GetDescription       : WideString;
Function     DM_GetName              : WideString;
Function     DM_Direction            : TSignalDirection;
Function     DM_IsDriver             : LongBool;
Function     DM_Range1               : WideString;
Function     DM_Range2               : WideString;
```

```
Function    DM_RangeValue1           : Integer;
Function    DM_RangeValue2           : Integer;
Function    DM_RangeMax              : Integer;
Function    DM_RangeMin              : Integer;
Function    DM_BusIndex              : Integer;
Function    DM_Width                 : Integer;

Function    DM_TargetLinks    (Index : Integer) : ISignalLink;
Function    DM_DriverLinks    (Index : Integer) : ISignalLink;
Function    DM_TargetLinkCount       : Integer;
Function    DM_DriverLinkCount       : Integer;
Function    DM_Signal                : ISignal;
Function    DM_EntityPort            : IEntityPort;
Function    DM_ConstantExpression    : WideString;
```

See also

Workspace Manager Interfaces

ISignalManager interface

ISignal interface

ISignalLink interface

IEntityPort interface

TSignalDirection interface

ISubNet interface

Interface Methods

```
Function    DM_Lines             (Index : Integer) : ILine;
Function    DM_SignalLinks       (Index : Integer) : ISignalLink;
Function    DM_Signals           (Index : Integer) : ISignal;
Function    DM_Nodes             (Index : Integer) : ISignalNode;
Function    DM_PinNodes          (Index : Integer) : ISignalNode;
Function    DM_PowerObjectNodes  (Index : Integer) : ISignalNode;
Function    DM_PortNodes         (Index : Integer) : ISignalNode;
Function    DM_NetLabelNodes     (Index : Integer) : ISignalNode;
Function    DM_SheetEntryNodes   (Index : Integer) : ISignalNode;
Function    DM_CrossSheetNodes   (Index : Integer) : ISignalNode;
Function    DM_LineCount             : Integer;
Function    DM_SignalLinkCount       : Integer;
```

```
Function     DM_SignalCount          : Integer;
Function     DM_NodeCount            : Integer;
Function     DM_PinNodeCount         : Integer;
Function     DM_PowerObjectNodeCount : Integer;
Function     DM_PortNodeCount        : Integer;
Function     DM_NetLabelNodeCount    : Integer;
Function     DM_SheetEntryNodeCount  : Integer;
Function     DM_CrossSheetNodeCount  : Integer;
Function     DM_Net                  : INet;
```

See also

Workspace Manager Interfaces

ISignalManager interface

ISignal interface

ISignalNode interface

ISignalLink interface

ILine interface

INet interface

WorkSpace Enumerated Types

The enumerated types are used for many of the WorkSpace Manager interfaces methods which are covered in this section. For example the IPart interface has a Function DM_ComponentKind : TComponentKind; method. You can use this Enumerated Types section to check what the range is for the TComponentKind type.

See also

Work Space Manager API Reference

TCompilationStage type

TCompileMode type

TECO_Mode type

TErrorGroup type

TErrorKind type

TErrorLevel type

TFlattenMode type

TFlowState type

TModificationKind type

TChannelRoomNamingStyle type

TNetScope type

TParameterKind type

TPinElectrical type

TSystemParmeterKind type

TVariationKind type

TSignalDirection type

TChannelRoomNamingStyle

```
TChannelRoomNamingStyle = (eChannelRoomNamingStyle_FlatNumericWithNames,
                           eChannelRoomNamingStyle_FlatAlphaWithNames,
                           eChannelRoomNamingStyle_NumericNamePath,
                           eChannelRoomNamingStyle_AlphaNamePath,
                           eChannelRoomNamingStyle_MixedNamePath);
```

TCompilationStage

```
TCompilationStage =
(eCompilationStage_Compiling,eCompilationStage_Flattening);
```

TCompilationStageSet

```
TCompilationStageSet = Set of TCompilationStage;
```

TCompileMode

```
TCompileMode =
(eCompile_None,eCompile_Document,eCompile_All,eCompile_Smart);
```

TComponentKind

```
TComponentKind = (eComponentKind_Standard,
                  eComponentKind_Mechanical,
                  eComponentKind_Graphical,
                  eComponentKind_NetTie_BOM,
                  eComponentKind_NetTie_NoBOM,
                  eComponentKind_Standard_NoBOM);
```

TDisplayMode

```
TDisplayMode = Byte; // one of 255 display modes
```

TECO_Mode

```
TECO_Mode = (eECO_PerformAction,
             eECO_ValidateAction,
             eECO_CheckSupportForAction);
```

TErrorGroup

```
TErrorGroup = (eErrorGroupDocument,
               eErrorGroupComponent,
               eErrorGroupParameters,
               eErrorGroupBus,
               eErrorGroupNet,
               eErrorGroupMisc);
```

TErrorKind

```
TErrorKind = (eError_OffGridObject,
              eError_OffDocumentObject,
              eError_MissingChildDocument,
              eError_MissingChildProject,
              eError_PortNotLinkedToSheetSymbol,
              eError_SheetEntryNotLinkedToPort,
```

```
eError_DuplicateDocumentNumbers,
eError_UnconnectedWire,
eError_UnconnectedNetItem,
eError_NetWithNoDrivingSource,
eError_FloatingInputPinsOnNet,
eError_DifferentConnectionCodesOnNet,
eError_MultipleSameConnectionCodeOnNet,
eError_MultipleNamesForNet,
eError_AddingItemsFromHiddenNetToNet,
eError_AddingHiddenNet,
eError_PowerObjectScopeChange,
eError_NetParameterInvalidName,
eError_NetParameterInvalidValue,
eError_MismatchedBusSectionOrdering,
eError_MismatchedFirstGenericIndex,
eError_MismatchedSecondGenericIndex,
eError_MismatchedIOTypeOnBus,
eError_BusIndexOutOfRange,
eError_RangeSyntaxError,
eError_IllegalBusDefinition,
eError_IllegalBusRangeValue,
eError_MismatchedBusWidths,
eError_MismatchedBusLabelOrdering,
eError_MixedGenericAndNumericBusLabels,
eError_UnDesignatedPart,
eError_DuplicateComponentDesignator,
eError_DuplicateSheetSymbolDesignator,
eError_DuplicateNets,
eError_DuplicatePinsInComponent,
eError_DuplicateSheetEntrysInSheetSymbol,
eError_DuplicatePortsInDocument,
eError_DuplicateSubParts,
eError_MismatchedHiddenPinConnections,
eError_MismatchedPinVisibility,
eError_SameParameterWithDifferentValues,
eError_SameParameterWithDifferentTypes,
```

```
eError_MissingModel,
eError_ModelInDifferentLocation,
eError_MissingModelInFile,
eError_DuplicateModelsFound,
eError_MissingModelParameter,
eError_ErrorInModelParameter,
eError_DuplicatePinsInPortMap,
eError_MissingPinInPortMap,
eError_MissingPinsPortMapSequence,
eError_DuplicateImplementation,
eError_UnusedPartInComponent,
eError_ExtraPinInComponentDisplayMode,
eError_MissingPinInComponentDisplayMode,
eError_MismatchedBusAndWire,
eError_FloatingNetLabel,
eError_FloatingPowerObject,
eError_SinglePinNet,
eError_SignalWithNoLoad,
eError_SignalWithNoDriver,
eError_SignalWithMultipleDrivers,
eError_AutoAssignedPin,
eError_NoError,
eError_MultipleTopLevelDocuments,
eError_MultipleConfigurationTargets,
eError_ConflictingConstraints,
eError_MissingConfigurationTarget,
eError_DuplicateUniqueIds);
```

TErrorKindSet

```
TErrorKindSet = Set of TErrorKind;
```

TErrorLevel

```
TErrorLevel =
(eErrorLevelNoReport,eErrorLevelWarning,eErrorLevelError,eErrorLevelFatal);
```

TErrorLevelSet

```
TErrorLevelSet = set of TErrorLevel;
```

TFlattenMode

```
TFlattenMode =
(eFlatten_Smart,eFlatten_Flat,eFlatten_Hierarchical,eFlatten_Global);
```

TFlowState

```
TFlowState =
(eState_UpToDate,eState_OutOfDate,eState_Failed,eState_Missing,eState_Runnin
g,eState_None);
```

TModificationKind

```
TModificationKind =
   ( eModification_Unknown,
     eModification_RemoveNode,
     eModification_RemoveComponentClassMember,
     eModification_RemoveNetClassMember,
     eModification_RemoveChannelClassMember,
     eModification_RemoveRule,
     eModification_RemoveNet,
     eModification_RemoveComponent,
     eModification_ChangeComponentFootPrint,
     eModification_ChangeComponentComment,
     eModification_ChangeComponentDesignator,
     eModification_ChangeComponentKind,
     eModification_AnnotateComponent,
     eModification_AddComponent,
     eModification_ChangeNetName,
     eModification_AddNet,
     eModification_AddNode,
     eModification_RemoveComponentClass,
     eModification_RemoveNetClass,
     eModification_RemoveChannelClass,
     eModification_ChangeComponentClassName,
     eModification_ChangeNetClassName,
     eModification_ChangeChannelClassName,
     eModification_AddComponentClass,
     eModification_AddNetClass,
     eModification_AddChannelClass,
```

```
    eModification_AddComponentClassMember,
    eModification_AddNetClassMember,
    eModification_AddChannelClassMember,
    eModification_RemoveRoom,
    eModification_ChangeRoom,
    eModification_AddRoom,
    eModification_AddParameter,
    eModification_RemoveParameter,
    eModification_ChangeParameterName,
    eModification_ChangeParameterValue,
    eModification_ChangeParameterType,
    eModification_AddRule,
    eModification_ChangeRule,
    eModification_FullPartUpdate,
    eModification_UpdatePartSymbol,
    eModification_UpdateImplementationValues,
    eModification_AddImplementation,
    eModification_RemoveImplementation,
    eModification_UpdateCurrentImplementation,
    eModification_ChangePinName,
    eModification_ChangePinElectrical,
    eModification_ChangePortElectrical,
    eModification_SwapPin,
    eModification_ChangePinSwapId_Pin,
    eModification_AddConstraintGroup,
    eModification_RemoveConstraintGroup,
    eModification_AddPort,
    eModification_RemovePort,
    eModification_ChangePortName,
    eModification_ChangeComponentLibRef);
```

TNetScope (WSM)

```
TNetScope = (eScopeLocal,eScopeInterface,eScopeGlobal);
```

TParameterKind

```
TParameterKind = (eParameterKind_String,
                  eParameterKind_Boolean,
```

```
                  eParameterKind_Integer,
                  eParameterKind_Float);
```

TPathMode

```
TPathMode = (ePathAbsolute,ePathRelative);
```

TPinElectrical (WSM)

```
TPinElectrical    = (eElectricInput,
                     eElectricIO,
                     eElectricOutput,
                     eElectricOpenCollector,
                     eElectricPassive,
                     eElectricHiZ,
                     eElectricOpenEmitter,
                     eElectricPower);
```

TSearchMode

```
TSearchMode = (eSearchModeCurrentDatabase,
               eSearchModeSpecifiedDatabase,
               eSearchModeMultipleDatabases,
               eSearchmodeWitnodwsFileSystem);
```

TSignalDirection

```
TSignalDirection =
(eSignalUndefined,eSignalInput,eSignalOutput,eSignalInOut);
```

TSystemParameterKind

```
TSystemParameterKind = (eSystemParameter_UserDefined,
                        eSystemParameter_CurrentTime,
                        eSystemParameter_CurrentDate,
                        eSystemParameter_Time       ,
                        eSystemParameter_Date       ,
                        eSystemParameter_DocFullPath,
                        eSystemParameter_DocName    ,
                        eSystemParameter_ModifiedDate,
                        eSystemParameter_ApprovedBy  ,
                        eSystemParameter_CheckedBy   ,
                        eSystemParameter_Author      ,
```

```
                        eSystemParameter_CompanyName ,
                        eSystemParameter_DrawnBy     ,
                        eSystemParameter_Engineer    ,
                        eSystemParameter_Organization,
                        eSystemParameter_Address1    ,
                        eSystemParameter_Address2    ,
                        eSystemParameter_Address3    ,
                        eSystemParameter_Address4    ,
                        eSystemParameter_Title       ,
                        eSystemParameter_DocNum      ,
                        eSystemParameter_Revision    ,
                        eSystemParameter_SheetNum    ,
                        eSystemParameter_SheetCount  ,
                        eSystemParameter_Rule        ,
                        eSystemParameter_ImagePath   ,
                        eSystemParameter_ConfigurableComponent);
```

TSystemParameterKindSet

```
TSystemParameterKindSet = Set of TSystemParameterKind;
```

TVariationKind

```
TVariationKind =
(eVariation_None,eVariation_NotFitted,eVariation_Alternate);
```

TViolationTypeDescription

```
TViolationTypeDescription = Record
    DefaultLevel : TErrorevel;
    Group        : TErrorGroup;
    Description  : TDynamicString;
End;
```

WorkSpace Manager Constants

```
cDocKind_Asm                      =   'ASM';
cDocKind_C                        =   'C';
cDocKind_Camtastic                =   'CAMTASTIC';
cDocKind_Ckt                      =   'CKT';
cDocKind_Constraint               =   'CONSTRAINT';
cDocKind_CoreProject              =   'COREPROJECT';
cDocKind_Cupl                     =   'CUPL';
cDocKind_DatabaseLink             =   'DATABASELINK';
cDocKind_Disassembly              =   'DISASSEMBLY';
cDocKind_Edif                     =   'EDIF';
cDocKind_EditScript               =   'EDITSCRIPT';
cDocKind_EditScriptDSUnit         =   'EDITSCRIPTDSUNIT';
cDocKind_EditScriptDSForm         =   'EDITSCRIPTDSFORM';
cDocKind_EditScriptBasUnit        =   'EDITSCRIPTBAS';
cDocKind_EditScriptTclUnit        =   'EDITSCRIPTTCL';
cDocKind_EditScriptVBSUnit        =   'EDITSCRIPTVBSUnit';
cDocKind_EditScriptVBSForm        =   'EDITSCRIPTVBSForm';
cDocKind_EditScriptJSUnit         =   'EDITSCRIPTJSUNIT';
cDocKind_EditScriptJSForm         =   'EDITSCRIPTJSForm';
cDocKind_EmbeddedProject          =   'EMBEDDEDPROJECT';
cDocKind_FavLink                  =   'FAVLINK';
cDocKind_Fpgaflow                 =   'FPGAFLOW';
cDocKind_FpgaProject              =   'FPGAPROJECT';
cDocKind_FpgaWorkspace            =   'FPGAWORKSPACE';
cDocKind_FreeDocsProject          =   'FREEDOCSPROJECT';
cDocKind_Html                     =   'HTML';
cDocKind_HtmlHelp                 =   'HTMLHELP';
cDocKind_IntegratedLibrary        =   'INTEGRATEDLIBRARY';
cDocKind_IntLibrary               =   'INTLIBRARY';
cDocKind_LogicAnalyser            =   'LogicAnalyser';
cDocKind_LogicAnalyserAnalog      =   'LogicAnalyserAnalog';
cDocKind_Mdl                      =   'MDL';
cDocKind_Nsx                      =   'NSX';
cDocKind_OutputJob                =   'OUTPUTJOB';
```

```
cDocKind_PCADPCB                   =  'PCADPCB';
cDocKind_Pcb                       =  'PCB';
cDocKind_Situs                     =  'SITUS';
cDocKind_Pcb3DLib                  =  'PCB3DLIB';
cDocKind_PcbLib                    =  'PCBLIB';
cDocKind_PCADLIB                   =  'PCADLIB';
cDocKind_PcbProject                =  'PCBPROJECT';
cDocKind_PDF                       =  'PDF';
cDocKind_PickATask                 =  'PICKATASK';
cDocKind_Profiler                  =  'PROFILER';
cDocKind_ProjectGroup              =  'PROJECTGROUP';
cDocKind_ProtelNetlist             =  'PROTELNETLIST';
cDocKind_Sch                       =  'SCH';
cDocKind_Schlib                    =  'SCHLIB';
cDocKind_ScriptProject             =  'SCRIPTPROJECT';
cDocKind_Simdata                   =  'SIMDATA';
cDocKind_SIPinModelLibrary         =  'SIPINMODELLIBRARY';
cDocKind_Targets                   =  'TARGETS';
cDocKind_Text                      =  'TEXT';
cDocKind_Vhdl                      =  'VHDL';
cDocKind_Verilog                   =  'VERILOG';
cDocKind_VhdLib                    =  'VHDLIB';
cDocKind_VhdlSim                   =  'VHDLSIM';
cDocKind_VhdTst                    =  'VHDTST';
cDocKind_VQM                       =  'VQM';
cDocKind_Wave                      =  'WAVE';
cDocKind_WaveSim                   =  'WAVESIM';
cDocKind_DefaultPcb                =  'DefaultPcb';
cDocKind_DefaultPcbLib             =  'DefaultPcbLib';
cDocKind_SchTemplate               =  'SCHDOT';
cDocKind_DDB                       = 'DDB';
cDocKind_ORCAD7_DSN                = 'ORCAD7_DSN';
cDocKind_ORCAD7_OLB                = 'ORCAD7_OLB';
cDocKind_PCAD16_SCH                = 'PCAD16_SCH';
cDocKind_PCAD16_BIN_SCH            = 'PCAD16_BIN_SCH';
cDocKind_PCAD16_LIA                = 'PCAD16_LIA';
```

```
cDocKind_OLD_PCAD_LIB          = 'OLD_PCAD_LIB';
cDocKind_ORCAD7_LLB            = 'ORCAD7_LLB';
cDocKind_CIRCUITMAKER2000_CKT  = 'CM2000_CKT';
cDocKind_CIRCUITMAKER2000_LIB  = 'CM2000_LIB';
cDocKind_NGC                   = 'NGC';
```

WorkSpace Manager Functions

```
Function GetWorkspace : IWorkspace;
Function GetProjectOfDocument(Const ADocPath : WideString) : IProject;
Function IsFreeDocument(Const FileName : WideString) : LongBool;

Function IsBusConnector(ALibReference : TDynamicString) : Boolean;

Function GetViolationTypeInformation(ErrorKind : TErrorKind) :
TViolationTypeDescription;
Function GetViolationTypeDescription(ErrorKind : TErrorKind) :
TDynamicString;
Function GetViolationTypeDefaultLevel(ErrorKind : TErrorKind) : TErrorLevel;
Function GetViolationTypeGroup(ErrorKind : TErrorKind) : TErrorGroup;

Function GetErrorLevelColor(ErrorLevel : TErrorLevel) : TColor;

Function IsFreeDocument (Const Filename : WideString) : LongBool;
```

See also

Work Space Manager API Reference

IProject interface

TColor type

TDynamicString type

TErrorLevel type

TErrorGroup type

TViolationTypeDescription type

Image Index Table

The Message panel has icons which specify messages. The DM_AddMessage and DM_AddMessageParametric methods of the IWorkSpace interface require an icon.

Image Index Table

Index = -1;	IndexTick = 3;	IndexNoERC = 3;
IndexCross = 4;	IndexConnective = 4;	IndexConnectiveList = 6
Folder = 6;	IndexFreeDocumentsProject = 6	IndexSheetFileName = 15;
OpenDocument = 68;	CloseDocument = 69;	NewFromExistingDocument = 70;
IndexProjectGroup = 54;	IndexProjectGroup2 = 55;	IndexPcbLayer = 51;
IndexEmptySection = 9;	IndexCamJob = 67;	IndexBoardProject = 56;
IndexFpgaProject = 57;	IndexEmbeddedProject = 58;	IndexIntegratedLibrary = 59;
Search = 38;	SearchSelected = 39;	IndexPCB = 52;
IndexPCBVariant = 53;	IndexParameter = 24;	IndexDocumentList = 26;
IndexEdifDocument = 43;	IndexEdifDocumentSelected = 43;	IndexGenericDocument = 62;
IndexTextFile = 62;	IndexCUPLFile = 63;	IndexAdvSimModel = 64;
IndexAdvSimNSX = 48;	IndexAdvSimSubCircuit = 47;	IndexBasicScript = 65;
IndexDelphiScript = 66;	IndexCFile = 45;	IndexVHDLDocument = 44;
IndexVHDLDocumentSelected = 44;	IndexVHDLLibrary = 44;	IndexSheetSymbolList = 30;
HierarchyNets = 30;	IndexPartList = 32;	IndexPinList = 5;
IndexTextFrameList = 28;	IndexProtelNetlistFile = 46;	IndexSchematicSheetSelected = 10
IndexSchematicSheet = 15;	IndexSchematicLibrary = 32;	IndexFlattenedHierarchy = 15;
IndexPCBLibrary = 40;	IndexNet = 1;	IndexBus = 21;

IndexBusEntry = 74;	IndexPart = 2;	IndexComponent = 20;
IndexFootprint = 36;	IndexSubPart = 2;	IndexImplementation = 8;
IndexSheetSymbol = 13;	IndexTextFrame = 18;	IndexPin = 19;
IndexPad = 41;	IndexHiddenName = 19;	IndexNetLabel = 22;
IndexPowerObject = 16;	IndexPort = 17;	IndexSheetEntry = 14;
IndexViolation = 4;	IndexDesignatorMapping = 2	IndexDesignatorManager = 8;
IndexModification = 4;	IndexModificationList = 9;	IndexDifference = 4;
IndexDifferenceList = 8;	IndexNetParameter = 24;	IndexSchematicSheetProcessor = 15
IndexSchematicLibraryProcessor = 15;	IndexEdifDocumentProcessor = 15;	IndexVHDLDocumentProcessor = 15;
IndexVHDLLibraryProcessor = 15;	IndexNetlistFileProcessor = 15;	IndexBoardProcessor = 15;
IndexSpatialAnalyser = 15;	IndexBusSection = 21;	IndexBusElement = 34;
IndexErrorList = 6;	IndexSpatialLine = 1;	IndexComponentClass = 7;
IndexNetClass = 7;	IndexRule = 2;	IndexRoom = 3;
IndexGraphic = 75;	IndexJunction = 76;	IndexAnnotation = 77;
IndexBrowserNetIdentifiers = 78;	IndexLibRef = 79;	IndexComponentParameters = 80;
IndexSheetSymbolParameters = 81;	IndexPortParameters = 82;	IndexPinParameters = 83;
IndexErrorMarker = 84;	IndexParameterSet = 85;	IndexPinsAndParts = 86;
IndexRectangle = 87;	IndexArc = 88;	IndexEllipticalArc = 89;

IndexRoundRectangle = 90;	IndexDesignator = 91;	Indexellipse = 92;
IndexPie = 93;	IndexPolygon = 94;	IndexPolyline = 95;
IndexBezier = 96;	IndexSheetName = 97;	IndexSymbol = 98;
IndexTaskHolder = 99	IndexFolder_NoError = 6;	IndexFolder_Warning = 7;
IndexFolder_Error = 8;	IndexFolder_Fatal = 9;	IndexGeneratedPage = 33;
IndexPrintView = 61;	IndexPrinterJob = 67;	IndexPrinter = 49;
IndexOutput = 61;	IndexAlias = 71;	IndexAliases = 72;
IndexOffsheetPin = 73;	IndexOffSheetPart = 100;	IndexOffSheetNet = 101;
IndexOffSheetBus = 102;	IndexOffSheetPort = 103;	IndexOffSheetSheetEntry = 104;
IndexOffSheetNetLabel = 105;	IndexOffSheetPowerObject = 106;	IndexMarker_NoError = 107;
IndexMarker_Warning = 108;	IndexMarker_Error = 109;	IndexMarker_Fatal = 110;
Index_MainHotSpot1 = 0;	Index_MainHotSpot2 = 1;	Index_MainHotSpot3 = 2;
Index_MainHotSpot4 = 3;	Index_MainHotSpot5 = 4;	Index_MainHotSpot6 = 5;
Index_MainHotSpot7 = 6;	Index_MainHotSpot8 = 7;	Index_MainHotSpot9 = 8;
Index_MainHotSpot10 = 9;		

See also

Work Space Manager API Reference

General DXP RTL Reference

In this section, general routines from the DXP Run Time Library such as File IO routines and enumerated types in which you can use in your scripts using a supported language set.

The scripting system also supports the DXP Object Models which are outlined in Client Server API Reference, Integrated Library API Reference, Nexar API Reference, PCB API Reference, Schematic API Reference and Work Space Manager API Reference.

The Scripting system also supports a subset of Borland Delphi Run Time Library (RTL) which is covered in Supported Borland Delphi Units section.

Script Examples

There are script examples in the **\Examples\Scripts** folders.

In this section of General DXP RTL Reference:

- Enumerated Types
- Dialogs
- File IO
- Folder Routines
- Number Manipulation Routines
- Other Routines
- String Manipulation Routines
- Time and Date Routines

Enumerated Types

In this section, enumerated types are used by the variables of functions or procedures or objects in the DXP Run Time Library.

See also

General DXP RTL Reference

TAltShiftCtrlCombination

```
TAltShiftCtrlCombination = TShiftState;
```

TChar

```
TChar          = Array[0..256] of Char;
```

TBoolean

```
TBoolean       = Boolean;
```

TBusKind

```
TBusKind          =
(eBusKindUndefined,eBusKindLowValueFirst,eBusKindHighValueFirst,eBusKindGene
ric);
```

TByte

```
TByte          = Byte;
```

TDouble

```
TDouble = Double;
```

TExtended

```
TExtended      = Extended;
```

THugeInt

```
THugeInt       = Comp;
```

TMatchFileNameKind

```
TMatchFileNameKind = (eMatchByPath,eMatchByFileName);
```

TReal

```
TReal          = Single;
```

TString

```
TString = ShortString;
```

Dialogs

In this section, custom DXP dialogs are available to use in your scripts from the DXP Run Time Library.

See also

General DXP RTL Reference

ConfirmNoYesWithCaption

Declaration

```
Function  ConfirmNoYesWithCaption      (Caption   : TDynamicString; S :
TDynamicString) : TBoolean;
```

Description

The ConfirmNoYesWithCaption function displays a dialog with a Caption parameter for the title bar of the dialog, and the S parameter for the message body of the dialog and has ‘Yes’ and ‘No’ buttons.

This function returns a modal value, ie when the user user chose the No button an IDNo (7) is returned, or when the user chose the Yes button, an IDYES (6) value is returned

See also

ConfirmNoYes, ShowError, ShowInfo, ShowWarning procedures.

ConfirmNoYesCancelWithCaption

Declaration

```
Function  ConfirmNoYesCancelWithCaption(Const Caption, S         :
TDynamicString) : Integer;
```

Description

The ConfirmNoYesCancelWithCaption function displays a dialog with a Caption parameter for the title bar of the dialog, and the S parameter for the message body of the dialog and has ‘Yes’, ‘No’ and ‘Cancel’ buttons. This function returns a modal value, ie when the user chose the Cancel button, an IDCancel (2) is returned or when the user chose the No button an IDNo (7) is returned, or when the user chose the Yes button, an IDYES (6) value is returned.

See also

ConfirmNoYes, ShowError, ShowInfo, ShowWarning procedures.

ConfirmNoYesCancel

Declaration

```
Function ConfirmNoYesCancel(Const S: String) : Integer
```

Description

The procedure displays a message dialog with a YES button, NO button and Cancel buttons. The title of the message box is "Confirm". The Value parameter returns one of the following values as a TModalResult type (as defined in Borland Delphi) representing which button has been pressed.

See also

ConfirmNoYes, ShowError, ShowInfo, ShowWarning procedures.

ConfirmNoYes

Declaration

```
Function ConfirmNoYes(Const S: String) : Boolean
```

Description

The procedure displays a message dialog with a YES button and NO button buttons. The title of the message box is "Confirm". The Value parameter returns True for the button Yes and False for no.

See also

Dialogs

ShowWarning

Declaration

```
Procedure ShowWarning(Const S: String);
```

Description

This procedure displays a warning dialog containing an OK button and the warning icon.

See also

ShowError and ShowInfo procedures.

ShowInfoWithCaption

Declaration

```
Procedure ShowInfoWithCaption          (Caption,S : TDynamicString);
```

Description

Displays a dialog with the Information icon and with a Caption parameter for the title bar of the dialog, and the S parameter for the message body of the dialog.

See also

ShowError and ShowWarning procedures.

ShowInfo

Declaration

```
Procedure ShowInfo(Const S: String);
```

Description

The procedure displays an information dialog containing an OK button and the information icon.

See also

ShowError and ShowWarning procedures.

ShowError

Declaration

```
Procedure ShowError(Const S: String);
```

Description

This procedure displays an Error dialog containing an OK button and the warning icon.

See also

ShowInfo and ShowWarning procedures.

File IO

In this section, custom file IO routines are available to use in your scripts from the DXP Run Time Library.

See also

General DXP RTL Reference

AddBackSlashToFrontAndBack

Declaration

```
Function  RemoveBackSlashFromFrontAndBack(S: TDynamicString) :
TDynamicString;
```

Description

The RemoveBackSlashFromFrontAndBack function checks for the presence of a backslash character from the front of the string, S, and at the back of this string, S.

LowLevelRunTextEditorWithFile

Declaration

```
Procedure LowLevelRunTextEditorWithFile   (S : TDynamicString);
```

Description

This function invokes the Microsott Windows NotePad application and attempts to open the file denoted by the S parameter.

IsFullPathToExistingFile

Declaration

```
Function IsFullPathToExistingFile(FullPath : TDynamicString) : Boolean;
```

Description

This function returns True if the path including the filename to an existing file exists. Use this function to distinguish a path that contains the filename only.

HasExtension

Declaration

```
Function HasExtension(Const Name : TDynamicString; Var DotPos : Integer) :
TBoolean;
```

Description

This function checks if the Name parameter has an extension by scanning for the dot character. If the dot character is found, the index of the DotPos variable parameter is returned. Note that the invalid characters are '\' and ':' and if they exist in the Name parameter, then the function returns a false value.

GetFreeDiskSpaceString

Declaration

```
Function  GetFreeDiskSpaceString(DiskNumber : Integer) : TDynamicString;
```

Description

The GetFreeDiskSpaceString function returns a TDynamicString value which represents the number of free bytes on the specified drive number.

GetDiskSizeString

Declaration

```
Function  GetDiskSizeString     (DiskNumber : Integer) : TDynamicString;
```

Description

The GetDiskSizeString function returns a TDynamicString value which represents the size, in bytes, of the specified drive.

GetDiskFree

Declaration

```
Function GetDiskFree(Drive: Byte): Double;
```

Description

The GetDiskFree function returns a double value which reports the amount of free space on the disk. The Drive value (Byte value) represents the drive letter. A drive = 0, B Drive = 1 etc.

FileExists

Declaration

```
Function FileExists(const FileName: string): Boolean;
```

Description

The FileExists function returns True if the file specified by FileName exists. If the file does not exist, FileExists returns False.

Example

```
Function OpenProject(ProjectName : String) : Boolean;
Begin
```

```
    Result := True;
    If Not FileExists(ProjectName) Then Result := False;

    ResetParameters;
    AddStringParameter('ObjectKind','Project');
    AddStringParameter('FileName', ProjectName);
    RunProcess('WorkspaceManager:OpenObject');
End;
```

ExpandFile

Declaration

```
Function  ExpandFile   (S     : TDynamicString) : TDynamicString;
```

Description

The ExpandFile function converts the relative file name into a fully qualified path name by merging in the current drive and directory. A fully qualified path name includes the drive letter and any directory and subdirectories in addition to the file name and extension. ExpandFileName does not verify that the resulting fully qualified path name refers to an existing file, or even that the resulting path exists.

DocumentIsReadOnly

Declaration

```
Function DocumentIsReadOnly      (FullPath : TDynamicString) : Boolean;
```

Description

The DocumentIsReadOnly function returns True if a design document file has a read only property set true.

ConvertDiskSizeToString

Declaration

```
Function  ConvertDiskSizeToString    (Size : Integer) : TDynamicString;
```

Description

The ConvertDiskSizeToString function converts a number into a string representing the size of a storage space. For example, when Size = 345, then the function returns a ‘345 Bytes’ string.

ComputerName

Declaration

```
Function ComputerName : ShortString
```

Description

The ComputerName function retrieves the computer name of the current system. This name is established at system startup, when it is initialized from the registry.

CheckAgainstWildCard_CaseSensitive

Declaration

```
Function  CheckAgainstWildCard_CaseSensitive(WildCard,Name : TDynamicString)
```

Description

The CheckAgainstWildCard_CaseSensitive function allows the comparison of the Wildcard string containing wildcards to the Name string. Use the Wildcard string which can consist of upper case and lower case characters to determine if the Name string matches the format described by the Wildcard string. The wildcard string can contain wildcards that can match any character, and sets that match a single character that is included in the Name string.

CheckAgainstWildCard

Declaration

```
Function  CheckAgainstWildCard (WildCard,Name : TDynamicString)
```

Description

The CheckAgainstWildCard function allows the comparison of the Wildcard string containing wildcards to the Name string. Use the Wildcard string to determine if the Name string matches the format described by the Wildcard string. The wildcard string can contain wildcards that can match any character, and sets that match a single character that is included in the Name string. This function is not case sensitive.

Folders Routines

In this section, custom folder routines are available to use in your scripts from the DXP Run Time Library.

See also

General DXP RTL Reference

SpecialFolder_MyDesigns

Declaration

Function SpecialFolder_MyDesigns : TDynamicString;

Description

This function returns the path to the MyDesigns folder. Example C:\Documents and Settings\UserName\My Documents\My Designs

See also

File IO Routines

SpecialFolder_DesignExamples

Declaration

Function SpecialFolder_DesignExamples : TDynamicString;

Description

This function returns the path to the Design Examples folder. Example C:\Program Files\Altium\Examples\

See also

File IO Routines

SpecialFolder_DesignTemplates

Declaration

Function SpecialFolder_DesignTemplates : TDynamicString;

Description

This function returns the path to the DesignTemplates folder. Example C:\Program Files\Altium\Templates\

See also

File IO Routines

SpecialFolder_AltiumLibraryIntegrated

Declaration

Function SpecialFolder_AltiumLibraryIntegrated : TDynamicString;

Description

This function returns the path to the Altium Integrated Library folder. Example C:\Program Files\Altium\Library\

See also

File IO Routines

SpecialFolder_AltiumLibraryPld

Declaration

Function SpecialFolder_AltiumLibraryPld : TDynamicString;

Description

This function returns the path to the Altium PLD Library folder. Example C:\Program Files\Altium\Library\Pld\

See also

File IO Routines

SpecialFolder_AltiumLibrary

Declaration

Function SpecialFolder_AltiumLibrary : TDynamicString;

Description

This function returns the path to the Altium Library folder. Example C:\Program Files\Altium\Library\

See also

File IO Routines

SpecialFolder_AltiumSystemTemplates

Declaration

Function SpecialFolder_AltiumSystemTemplates : TDynamicString;

Description

This function returns the path to the Altium's System Templates folder. Example C:\Program Files\Altium\System\Templates\

See also

File IO Routines

SpecialFolder_AltiumSystem

Declaration

Function SpecialFolder_AltiumSystem : TDynamicString;

Description

This function returns the path to the Altium's system folder. Example C:\Program Files\Altium\System\

See also

File IO Routines

SpecialFolder_AltiumDesignExplorer

Declaration

Function SpecialFolder_AltiumDesignExplorer : TDynamicString;

Description

This function returns the path to the Altium folder. Example C:\Program Files\Altium\

See also

File IO Routines

SpecialFolder_AltiumApplicationData

Declaration

Function SpecialFolder_AltiumApplicationData : TDynamicString;

Description

This function returns the path to the Altium User Application Data folder. Example C:\Documents and Settings\UserName\Application Data\Altium

See also

File IO Routines

SpecialFolder_AltiumAllUserApplicationData

Declaration

Function SpecialFolder_AltiumAllUserApplicationData : TDynamicString;

Description

This function returns the path to the Altium All User Application Data folder. Example C:\Documents and Settings\All Users\Application Data\Altium

See also

File IO Routines

SpecialFolder_AltiumLocalApplicationData

Declaration

Function SpecialFolder_AltiumLocalApplicationData : TDynamicString;

Description

This function returns the path to the Altium Local Application Data folder. Example C:\Documents and Settings\UserName\My Documents\My Designs

See also

File IO Routines

SpecialFolder_Recovery

Declaration

Function SpecialFolder_Recovery : TDynamicString;

Description

This function returns the path to the Altium Recover folder. Example C:\Documents and Settings\UserName\Application Data\Recovery\

See also

File IO Routines

SpecialFolder_AdminTools

Declaration

Function SpecialFolder_AdminTools : TDynamicString;

Description

This function returns the path to the All User Application Data folder.

See also

File IO Routines

SpecialFolder_AllApplicationData

Declaration

Function SpecialFolder_AllUserApplicationData : TDynamicString;

Description

This function returns the path to the C:\Documents and settings\All Users\Application Data folder.

See also

File IO Routines

SpecialFolder_ApplicationData

Declaration

Function SpecialFolder_ApplicationData : TDynamicString;

Description

This function returns the path to the C:\Documents and settings\UserName\Application Data folder.

See also

File IO Routines

SpecialFolder_LocalApplicationdata

Declaration

Function SpecialFolder_LocalApplicationData : TDynamicString;

Description

This function returns the path to the C:\Documents and settings\UserName\Local Settings\Application Data folder

See also

File IO Routines

SpecialFolder_TemporarySlash

Declaration

Function SpecialFolder_TemporarySlash : TDynamicString;

Description

This function returns the path to the C:\Documents and settings\UserName\Local Settings\Temp\ folder.

See also

File IO Routines

SpecialFolder_Temporary

Declaration

Function SpecialFolder_Temporary : TDynamicString;

Description

This function returns the path to the C:\DOCUME~1\UserName\LOCALS~1\Temp\ folder.

See also

File IO Routines

SpecialFolder_MyComputer

Declaration

Function SpecialFolder_MyComputer : TDynamicString;

Description

This function returns the path to the MyComputer folder.

See also

File IO Routines

SpecialFolder_Fonts

Declaration

Function SpecialFolder_Fonts : TDynamicString;

Description

This function returns the path to the folder where fonts are stored. For example, C:\WinNT\Fonts

See also

File IO Routines

SpecialFolder_DesktopLocation

Declaration

Function SpecialFolder_DesktopLocation : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Desktop folder.

See also

File IO Routines

SpecialFolder_Favorites

Declaration

Function SpecialFolder_Favorites : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Cookies folder.

See also

File IO Routines

SpecialFolder_AllUserAdminTools

Declaration

Function SpecialFolder_AllUserAdminTools : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\All Users\Start Menu\Programs\Administrative Tools folder.

See also

File IO Routines

SpecialFolder_Desktop

Declaration

Function SpecialFolder_Desktop : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Desktop folder.

See also

File IO Routines

SpecialFolder_InternetCookies

Declaration

```
Function SpecialFolder_InternetCookies : TDynamicString;
```

Description

This function returns the path to the C:\Documents and Settings\UserName\Cookies folder.

See also

SpecialFolder_ControlPanel

Declaration

Function SpecialFolder_ControlPanel : TDynamicString;

Description

This function returns the path to the Control Panel folder.

See also

File IO Routines

SpecialFolder_TemplatesForAllUsers

Declaration

Function SpecialFolder_TemplatesForAllUsers : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\All Users\Templates folder.

See also

File IO Routines

SpecialFolder_CommonStartup

Declaration

Function SpecialFolder_CommonStartup : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\All Users\Start Menu folder.

See also

File IO Routines

SpecialFolder_CommonStartupPrograms

Declaration

Function SpecialFolder_CommonStartupPrograms : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\All Users\Start Menu\Programs folder.

See also

File IO Routines

SpecialFolder_CommonFavorites

Declaration

Function SpecialFolder_CommonFavorites : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\All Users\Favorites folder.

See also

File IO Routines

SpecialFolder_AllUserDesktop

Declaration

Function SpecialFolder_AllUserDesktop : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\All Users\Desktop folder.

See also

File IO Routines

SpecialFolder_RecycleBin

Declaration

Function SpecialFolder_RecycleBin : TDynamicString;

Description

This function returns the path to the Recycle Bin.

See also

File IO Routines

SpecialFolder_NonlocalizedStartupPrograms

Declaration

Function SpecialFolder_NonLocalizedStartupPrograms : TDynamicString;

Description

This function returns the path to the Non Localized Startup Programs folder.

See also

File IO Routines

SpecialFolder_AllUserDocuments

Declaration

Function SpecialFolder_AllUserDocuments : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\All Users\Desktop folder.

See also

File IO Routines

SpecialFolder_InstalledPrinters

Declaration

Function SpecialFolder_InstalledPrinters : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\PrintHood folder.

See also

File IO Routines

SpecialFolder_MyDocuments

Declaration

Function SpecialFolder_MyDocuments : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Local Settings\My Documents folder.

See also

File IO Routines

SpecialFolder_NetWorkRoot

Declaration

Function SpecialFolder_NetworkRoot : TDynamicString;

Description

This function returns the path to the Network Root directory.

See also

File IO Routines

SpecialFolder_MyNetworkPlaces

Declaration

Function SpecialFolder_MyNetworkPlaces : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\NetHood folder.

See also

File IO Routines

SpecialFolder_MyPictures

Declaration

Function SpecialFolder_MyPictures : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Local Settings\My Pictures folder.

See also

File IO Routines

SpecialFolder_MyMusic

Declaration

Function SpecialFolder_MyMusic : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Local Settings\My Music folder.

See also

File IO Routines

SpecialFolder_InternetTemporaryFiles

Declaration

Function SpecialFolder_InternetTemporaryFiles : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Local Settings\Temporary Internet Files folder.

See also

File IO Routines

SpecialFolder_InternetHistory

Declaration

Function SpecialFolder_InternetHistory : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Local Settings\History folder.

See also

File IO Routines

SpecialFolder_ProgramFiles

Declaration

Function SpecialFolder_ProgramFiles : TDynamicString;

Description

This function returns the path to the C:\Program Files folder

See also

File IO Routines

SpecialFolder_Internet

Declaration

Function SpecialFolder_Internet : TDynamicString;

Description

This function returns the path to the folder where the internet browser software is located in.

See also

File IO Routines

SpecialFolder_Printers

Declaration

Function SpecialFolder_Printers : TDynamicString;

Description

This function returns the path to the Printers folder.

SpecialFolder_Profile

Declaration

Function SpecialFolder_Profile : TDynamicString;

Description

This function returns the path to the C:\Program Files\UserName.

See also

File IO Routines

SpecialFolder_SendTo

Declaration

Function SpecialFolder_SendTo : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\SendTo folder.

See also

File IO Routines

SpecialFolder_Recent

Declaration

Function SpecialFolder_Recent : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Recent folder.

See also

File IO Routines

SpecialFolder_Programs

Declaration

Function SpecialFolder_Programs : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Start Menu\Programs folder.

See also

File IO Routines

SpecialFolder_CommonProgramFiles

Declaration

Function SpecialFolder_CommonProgramFiles : TDynamicString;

Description

This function returns the path to the C:\Program Files\Common Files folder.

See also

File IO Routines

SpecialFolder_WindowsFolder

Declaration

Function SpecialFolder_WindowsFolder : TDynamicString;

Description

This function returns the path to the C:\WINNT folder.

See also

File IO Routines

SpecialFolder_CommonDocumentTemplates

Declaration

Function SpecialFolder_CommonDocumentTemplates : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Templates folder.

See also

File IO Routines

SpecialFolder_SystemFolder

Declaration

Function SpecialFolder_SystemFolder : TDynamicString;

Description

This function returns the path to the C:\WINNT\System32 folder.

See also

File IO Routines

SpecialFolder_UserStartMenuItems

Declaration

Function SpecialFolder_UserStartMenuItems : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Recent folder.

See also

File IO Routines

SpecialFolder_StartMenuItems

Declaration

Function SpecialFolder_StartMenuItems : TDynamicString;

Description

This function returns the path to the C:\Documents and Settings\UserName\Recent folder.

See also

File IO Routines

Number Manipulation Routines

In this section, custom number manipulation routines are available to use in your scripts from the DXP Run Time Library.

See also

General DXP RTL Reference

GetBinaryStringFromInteger

Declaration

Function GetBinaryStringFromInteger(L : Integer) : TDynamicString;

Description

The GetBinaryStringFromInteger function converts an integer to a binary string (up to thirty two characters long). An integer contains 4 bytes = 32 bits.

ExtendedToEng

Declaration

Function ExtendedToEng (Const ExtVal : Extended) : String;

Description

The ExtendedToEng function converts the floating-point value given by Value to its string representation. Example: ShowInfo(ExtendedToEng(4.32e18)); //4.320e18

See also

Number Manipulation routines

EngToExtended

Declaration

Function EngToExtended (Const EngString : String) : Extended;

Description

The EngToExtended function converts the string value given by EngString to its extended representation. This function looks at the last character of the string and converts it accordingly - see scale factor table below. For example ‘3Meg’ will come out as 3M.

See also

Number Manipulation routines

DoubleToComp

Declaration

function DoubleToComp(Value: Double; var Result: Comp);

Description

The DoubleToComp function converts a Double value into a Comp value. The Comp (computational) type is native to the Intel CPU and represents a 64-bit integer. It is classified as a real, however, because it does not behave like an ordinal type. (For example, you cannot increment or decrement a Comp value.).

See also

Number Manipulation routines

IntToStr

Declaration

function IntToStr(Value: Integer): string;

Description

IntToStr converts a Value integer into a string containing the decimal representation of that number.

See also

Number Manipulation routines

IntToHex

Declaration

function IntToHex(Value: Integer; Digits: Integer): string;

Description

The IntToHex function converts a Value number into a string containing the number's hexadecimal (base 16) representation. The Digits parameter indicates the minimum number of hexadecimal digits to return.

See also

Number Manipulation routines

IntSwap

Declaration

Procedure IntSwap(Var a,b : Integer);

Description

The IntSwap procedure swaps the values for A and B. For example A = 2 and B = 5. After passing these values into IntSwap procedure, the new values are a = 5 and b = 2.

See also

Number Manipulation routines

IntMin

Declaration

Function IntMin(x,y : Integer) : Integer;

Description

The IntMin function returns the minimum value of X and Y integer types.

See also

Number Manipulation routines

IntegerToHex

Declaration

Function IntegerToHex(L : Integer) : TDynamicString;

Description

Convert an integer value to an hexadecimal value.

See also

Number Manipulation routines

HexToInteger

Declaration

Function HexToInteger(Const S : TDynamicString) : Integer;

Description

Convert a hexadecimal value (as a string value) to an Integer value.

See also

Number Manipulation routines

GetHexStringFromInteger

Declaration

Function GetHexStringFromInteger   (L : Integer) : TDynamicString;

Description

The GetHexStringFromInteger converts a word to a hexadecimal string (up to eight characters long). The hexadecimal number system is a base 16 system with 16 digits. A byte equals 2 hexademical digits because each hexadecimal digit corresponds to four binary digits thus 4 bytes equals 8 hexadecimal digits.

See also

Number Manipulation routines

Other Routines

In this section, other routines that do not fit in other categories in the General DXP RTL Reference are available to use in your scripts from the DXP Run Time Library.

AltKeyDown

Declaration

Function AltKeyDown: Integer;

Description

This function returns a value that indicates the state of the ALT key, that is, the function returns 1 if the ALT key is pressed down, otherwise it returns 0.

See also

Other Routines

CheckActiveServer

Declaration

Function CheckActiveServer(Const AServerName, AServerCaption: String; AWithDialog: Boolean): Boolean;

Description

The function checks whether the server for the nominated document is active or not.

See also

Other Routines

GetCurrentWindowHandle

Declaration

Procedure GetCurrentWindowHandle(Var Value: HWND);

Description

The procedure returns an HWND value which represent the window handle of the currently active window in DXP.

See also

Other Routines

GetCurrentDocumentFileName

Declaration

Function GetCurrentDocumentFileName : String;

Description

The GetCurrentDocumentFileName obtains the filename of the currently focussed document in DXP.

See also

SaveCurrentDocument function.

Other Routines

GetErrorMessage

Declaration

Function GetErrorMessage(Const ErrorNumber : Integer) : String;

Description

The GetErrorMessage function returns an error message string that corresponds to the specified Operating System error code.

See also

Other Routines

RunApplication

Declaration

Function RunApplication(Const CommandLine : String) : Integer;

Description

The RunApplication function executes an application program outside the DXP environment. You need to supply the full path including the filename to the application you wish to execute.

Example

```
CommandLine := 'notepad.exe' + NameOfTextFile;
ErrorCode   := RunApplication(CommandLine);
If ErrorCode <> 0 Then
```

```
    ShowError('System cannot start : ' + CommandLine + #13#10 +
GetErrorMessage(ErrorCode));
```

See also

Other Routines

ResetCursor

Declaration

Procedure ResetCursor;

Description

The ResetCursor resets the cursor to the default arrow cursor.

See also

SetCursorBusy

Other Routines

SetCursorBusy

Declaration

Procedure SetCursorBusy;

Description

The SetCursorBusy updates the cursor to the default busy cursor, to indicate that the system is busy. This procedure could be set before a time consuming loop within a script.

See also

ResetCursor

Other Routines

ShiftKeyDown

Declaration

Function ShiftKeyDown: Integer;

Description

The ShiftKeyDown function returns a value that indicates the state of the SHIFT key, that is, the function returns 1 if the SHIFT key is down, otherwise it returns 0.

See also

AltKeyDown and ControlKeyDown functions.

Other Routines

String Manipulation Routines

In this section, string manipulation routines are available to use in your scripts from the DXP Run Time Library.

See also

General DXP RTL Reference

Center

Declaration

Function Center (Const S : TDynamicString; Width : Integer) : TDynamicString;

Description

Return a string centered in a blank string of specified width.

See also

String Manipulation Routines

CenterCH

Declaration

Function CenterCh (Const S : TDynamicString; Ch : Char; Width : Integer) : TDynamicString;

Description

Returns a string centered in a string of character Ch, with specified width.

See also

String Manipulation Routines

CharStr

Declaration

Function CharStr (Ch : Char; Len : Integer) : TDynamicString;

Description

Returns a string of length len filled with Ch

See also

String Manipulation Routines

CropStringToLength

Declaration

Function CropStringToLength (Const StringToCrop : TDynamicString; Const MaximumLength : Integer) : TDynamicString;

Description

The CropStringToLength function removes leading and trailing spaces and control characters from the given string StringToCrop. The MaximumLength parameter specifies the string from index 0 to MaximumLength that will be returned by the function. The remaining portion of the string is chopped.

See also

String Manipulation Routines

GeneralStringInc

Declaration

Procedure GeneralStringInc (Var S : TString; Const IncValue : TDynamicString);

Description

The GeneralStringInc procedure analyses the S parameter to determine if it has a number value embedded. If there is a number in the string then it increments the existing number value by one..

Example

```
S := 'Part1';
GeneralStringInc(S,'4');
//Part5
```

See also

String Manipulation Routines

GetStringFromBoolean

Declaration

Function GetStringFromBoolean (B : Boolean) : TDynamicString;

Description

The GetStringFromBoolean function returns a ‘True’ if the B parameter is true otherwise a ‘False’ is returned.

See also

String Manipulation Routines

GetStringFromInteger

Declaration

Function GetStringFromInteger   (N : Integer) : TDynamicString;

Description

The GetStringFromInteger function converts any integer type to a string.

See also

String Manipulation Routines

IndentString

Declaration

Function IndentString(Indent : Integer) : TDynamicString;

Description

The function returns you a string which specifies the amount of indentation as white spaces (#32) in this string. So an indent of 4 produces a string of four white spaces for example.

See also

String Manipulation Routines

LeftJust

Declaration

Function LeftJust   (Const S : TDynamicString; Width : Integer) : TDynamicString;

Description

The LeftJust function left justifies a string by padding the string with (Width - Length of String) white spaces to the right of this string.

Example

```
S := LeftJust('smith',9) + '.';
//s := 'smith    .' (four empty spaces between the word 'smith' and the
fullstop '.')
```

See also

String Manipulation Routines

PadLeft

Declaration

Function PadLeft(S : TDynamicString; Len : Integer) : TDynamicString;

Description

Returns a string left-padded to length len with blanks.

See also

String Manipulation Routines

PadLeftCh

Declaration

Function PadLeftCh (S : TDynamicString; Ch : Char; Len : Integer) : TDynamicString;

Description

Returns a string left-padded to length len with the specified character, Ch.

See also

String Manipulation Routines

PadRight

Declaration

Function PadRight (S : TDynamicString;   Len : Integer) : TDynamicString;

Description

Returns a string right-padded to length len with blanks.

See also

String Manipulation Routines

PadRightCh

Declaration

Function PadRightCh(S : TDynamicString; Ch : Char; Len : Integer) : TDynamicString;

Description

Returns a string right-padded to length specified by the len parameter and with Ch characters.

See also

String Manipulation Routines

SameString

Declaration

Function SameString   (Const S1,S2 : TDynamicString; CaseSensitive : Boolean) : Boolean;

Description

This SameString function compares two strings and depending on the CaseSensitive parameter returns a boolean result. If CaseSensitive is set to false, then the two strings, ‘aaa’ and ‘AaA’ are considered the same.

See also

String Manipulation Routines

StringsEqual

Declaration

Function StringsEqual(S1,S2 : TDynamicString) :Boolean;

Description

This SameString function compares two strings and checks whether Strings S1 and S2 have equal lengths and have the same contents.

See also

String Manipulation Routines

StrToInt

Declaration

function StrToInt(const S: string): Integer;

Description

The StrToInt function converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number.

See also

String Manipulation Routines

TrimLead

Declaration

Function TrimLead (Const S : TDynamicString) : TDynamicString;

Description

Returns a string with leading white space removed.

See also

String Manipulation Routines

TrimTrail

Declaration

Function TrimTrail (Const S : TDynamicString) : TDynamicString;

Description

Returns a string with trailing white space removed.

See also

String Manipulation Routines.

Time and Date Routines

In this section, time and date routines are available to use in your scripts from the DXP Run Time Library.

See also

General DXP RTL Reference

DateString

Declaration

Function DateString (Const DateRecord : TDate) : TDynamicString;

Description

The DateString function returns a TString representing a date in '12-Jan-1985' format.

See also

Time and Date Routines

GetCurrentDate

Declaration

Procedure GetCurrentDate (Var DateRecord : TDate);

Description

The GetCurrentDate procedure is based on the WinAPI's DecodeDate procedure which breaks the value specified as the Date parameter into Year, Month, and Day values. If the given TDateTime value is less than or equal to zero, the year, month, and day return parameters are all set to zero.

See also

Time and Date Routines

GetCurrentDateString

Declaration

Function GetCurrentDateString : TDynamicString;

Description

The GetCurrentDateString function returns a TString representing date in ‘12-Jan-1985’ format

See also

Time and Date Routines

GetCurrentTimeString

Declaration

Function GetCurrentTimeString : TDynamicString;

Description

The GetCurrentTimeString function returns a TString representing a time of day in HH:MM:SS format.

See also

Time and Date Routines

GetCurrentTimeRec

Declaration

Procedure GetCurrentTimeRec (Var   TimeRecord   : TTime);

Description

The GetCurrentTimeRec procedure is based on WinAPI’s DecodeTime function which breaks the TDateTime record into hours, minutes, seconds, and milliseconds.

See also

Time and Date Routines

GetDateAndTimeStamp

Declaration

Function GetDateAndTimeStamp : TDynamicString;

Description

This function returns the string containing the current date and the time.

See also

Time and Date Routines

GetElapsedTime

Declaration

Procedure GetElapsedTime (Const Start : TTime; Const Stop : TTime;Var Elapsed : TTime);

Description

The GetElapsedTime procedure returns the Elapsed value in seconds between the Start and Stop timing intervals.

See also

Time and Date Routines

GetElapsedTimeDate

Declaration

```
Procedure GetElapsedTimeDate (Const Start     : TTime;          Const Stop     : TTime;
                              Var   Elapsed   : TTime;
                              Const StartDate : TDate;
                              Const StopDate  : TDate);
```

Description

The GetElapsedTimeDate procedure returns the Elapsed value derived from the StartDate, StopDate dates and Start, Stop times. The results can be retrieved as a string by the TimString_Elapsed function.

See also

Time and Date Routines

GetFileDateString

Declaration

Function GetFileDateString(Const AFileName : TDynamicString) : TDynamicString;

Description

The GetCurrentDateString function returns a TString representing date in '12-Jan-1985' format

See also

Time and Date Routines

GetMilliSecondTime

Declaration

Function GetMilliSecondTime : Integer;

Description

The GetMilliSecondTime function retrieves the number of milliseconds that have elapsed since Windows was started.

See also

Time and Date Routines

MakeDateAndTimeStampedFileName

Declaration

Function MakeDateAndTimeStampedFileName(BaseName : TDynamicString) : TDynamicString;

Description

This function returns the date and time inserted in the base file name string.

See also

Time and Date Routines

SecondsToTimeRecord

Declaration

Procedure SecondsToTimeRecord(Var   TimeRecord   : TTime;   Const Seconds   : Integer);

Description

This procedure does the reverse of the TimeRecordToSeconds procedure. It converts the seconds information into the TTime structure type.

See also

Time and Date Routines

TimeString_elapsed

Declaration

Function TimeString_Elapsed (Const TimeRecord   : TTime) : TDynamicString;

Description

This function returns the string containing the Time information that has elapsed. To find the timing information, invoke the GetElspasedTimeDate or GetElapsedTime function.

Example

```
Var
   ElapsedTime : TTime;
Begin
```

```
   GetCurrentTimeRec (EndTime);
   GetCurrentDate (EndDate);
   GetElapsedTimeDate (StartTime, EndTime, ElapsedTime, StartDate, EndDate);
   ShowInfo('Time Elapsed : ' + TimeString_Elapsed(ElapsedTime));
End;
```

See also

Time and Date Routines

TimeString

Declaration

Function TimeString (Const TimeRecord : TTime) : TDynamicString;

Description

The TimeString function returns a TString representing a time of day in HH:MM:SS format.

See also

Time and Date Routines

TimeRecordToSeconds

Declaration

Procedure TimeRecordToSeconds(Const TimeRecord : TTime; Var Seconds : Integer);

Description

This procedure converts a TTime type structure into number of seconds. This procedure is used for GetElapsedTime and GetElapsedTimeDate procedures.

See also

Time and Date Routines

WaitMilliSecondDelay

Declaration

```
Function  ExtendedToEng(Const ExtVal   : Extended) : String;
```

Description

The ExtendedToEng function converts the floating-point value given by Value to its string representation. Example: ShowInfo(ExtendedToEng(4.32e18)); //4.320e18

See also

Time and Date Routines

Supported Borland Delphi Units

In this section:

- Introduction
- CopyFile function
- TIniFile object
- TList object
- TStringList object

Borland Delphi Units used in the Scripting System

The Scripting System has provided a few Helper objects which are to help simplify your scripting tasks especially with creating and managing lists of strings or objects. Basically there are two different groups of Run Time Libraries used in the scripting system: the Borland Delphi Units and the DXP Units.

The objects and functions that are supported in the Scripting system are based on Borland Delphi version 6 and the units that are exposed in the scripting system are:

Supported Borland Component Units

actnlist	buttons	checklst	controls	comctrls
dialogs	extctrls	extdlgs	filectrl	forms
graphics	grids	imglist	mask	menus
mplayer	olectnrs olectrls	outline	stdactns	stdctrls
tabnotbk	tabs	toolwin		

Supported Borland Delphi System Units

classes	clipbrd	Contnrs	inifiles	math
printers	registry	sysutils	system	types
variants				

Few useful functions:

- CopyFile (windows unit)

Few useful classes:

- TStringLIst
- TList
- TIniFIle

Many routines and objects from the Borland Delphi Run Time Library cannot be used in the scripting system because the scripting system cannot support Int64 type parameters, for example the **TStream** and its descendant classes cannot be used in the scripting system because many of the methods use the Int64 parameter type. The other limitations are that you cannot define classes or records because the scripting system is type-less.

See also

Borland Delphi 6 toolkit or perform a search on the internet with one of the Net search engines for more information on Borland Run Time Library and its units.

CopyFile function

Declaration

The **CopyFile** function (exposed from the Borland Delphi's Windows unit) copies a file specified by the original filename to a new file with the new filename.

Syntax

```
CopyFile(SourceFileName, TargetFilename : PChar; FailIfExists : Boolean);
```

See also

Helper Classes and Functions

TIniFile object

The **TIniFile** object (derived from Borland Delphi's TIniFile class) stores and retrieves application-specific information and settings from a text file with an INI extension. When you instantiate the **TIniFile** object, you pass as a parameter to the **TIniFile**'s constructor, the filename of the INI file. If the file does not exist, the ini file is created automatically.

You then can read values using ReadString, ReadInteger, or ReadBool methods. Alternatively, if you want to read an entire section of the INI file, you can use the ReadSection method. As well, you can write values using WriteBool, WriteInteger, or WriteString methods.

Each of the Read routines takes three parameters. The first parameter identifies the section of the INI file. The second parameter identifies the value you want to read, and the third is a default value in case the section or value doesn't exist in the INI file. Similarly, the Write routines will create the section and/or value if they do not exist.

Script example

See at the end of this page the example code which creates an INI file.

TIniFile Methods

```
DeleteKey(const Section, Ident: String);
EraseSection(const Section: String);

ReadSection (const Section: String; Strings: TStrings);
ReadSections(Strings: TStrings);
ReadSectionValues(const Section: String; Strings: TStrings);

ReadString(const Section, Ident, Default: String): String;
WriteString(const Section, Ident, Value: String);

UpdateFile;
```

Derived from TCustomIniFile

```
Create(const FileName: String);
ReadBinaryStream(const Section, Name: string; Value: TStream): Integer;
ReadBool (const Section, Ident: String; Default: Boolean): Boolean ;
ReadDate (const Section, Ident: String; Default: TDateTime): TDateTime;
ReadDateTime (const Section, Ident: String; Default: TDateTime): TDateTime;
ReadFloat (const Section, Ident: String; Default: Double): Double;
ReadInteger(const Section, Ident: String; Default: Longint): Longint;
ReadTime (const Section, Ident: String; Default: TDateTime): TDateTime;
SectionExists (const Section: String): Boolean;

WriteBinaryStream(const Section, Name: string; Value: TStream);
WriteBool(const Section, Ident: String; Value: Boolean);
WriteDate(const Section, Ident: String; Value: TDateTime);
WriteDateTime(const Section, Ident: String; Value: TDateTime);
procedure WriteFloat(const Section, Ident: String; Value: Double);
WriteInteger(const Section, Ident: String; Value: Longint);
WriteTime(const Section, Ident: String; Value: TDateTime);
ValueExists (const Section, Ident: String): Boolean;
```

Derived from TObject

AfterConstruction

BeforeDestruction

ClassInfo

ClassName

ClassNameIs

ClassParent

ClassType

CleanupInstance

DefaultHandler

Destroy

Dispatch

FieldAddress

Free

FreeInstance

GetInterface

GetInterfaceEntry

GetInterfaceTable

InheritsFrom

InitInstance

InstanceSize

MethodAddress

MethodName

NewInstance

SafeCallException

Example of an Ini file creation

```
Procedure WriteToIniFile(AFileName : String);
Var
    IniFile : TIniFile;
    I,J     : Integer;
Begin
    IniFile := TIniFile.Create(AFileName);
    For I := 1 to 2 Do
       For J := 1 to 2 Do
          IniFile.WriteString('Section'+IntToStr(I),
          'Key' + IntToStr(I) + '_' + IntToStr(J),
          'Value' + IntToStr(I));
    IniFile.Free;

    (* The INIFILE object generates a text file of the
```

```
      following format;
    [Section1]
    Key1_1=Value1
    Key1_2=Value1
    [Section2]
    Key2_1=Value2
    Key2_2=Value2
    *)
End;
```

See also

Helper Classes and Functions

Refer to the **IniFileEg** script example in the **\Examples\Scripts\General** folder.

TList object

The TList class stores an array of pointers to objects. You can create an instance of a TList object and you can add, sort or delete individual objects from this TList object in your script.

TList Properties

- Capacity
- Count
- Items
- List

TList methods

- Add(Item: Pointer): Integer;
- Assign(ListA: TList; AOperator: TListAssignOp = laCopy; ListB: TList = nil);
- Clear
- Delete(Index: Integer);
- Destroy
- Exchange(Index1, Index2: Integer);
- Expand: TList;
- Extract(Item: Pointer): Pointer;
- First: Pointer;
- IndexOf
- IndexOf(Item: Pointer): Integer;
- function Last: Pointer;
- Move(CurIndex, NewIndex: Integer);

- Pack
- Remove(Item: Pointer): Integer;
- Sort

Methods derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Dispatch
- FieldAddress
- Free
- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

Example

```
//The following code adds an object to TheList container if the object is
not in the list.
Begin
    If TheList.IndexOf(AnObject)=-1 Then
        TheList.Add(AnObject);
    // do something
    TheList.Remove(AnObject);
```

```
End;
```

See also

Helper Classes and Functions

TStringList object

The TStringList object maintains a list of strings. You can create an instance of a TStringList object and you can add, sort or delete individual strings from this object in your script.

If you need to do a customized sorting of the TStringList container, you need to write your own sorting routine. See examples below.

TStringList Properties

- Capacity: Integer;
- CaseSensitive: Boolean;
- Count: Integer;
- Duplicates: TDuplicates;
- Objects[Index: Integer]: TObject;
- Sorted: Boolean;
- Strings[Index: Integer]: string;

Derived from TStrings

- CommaText: string;
- DelimitedText: string;
- Delimiter: Char;
- Names[Index: Integer]: string;
- QuoteChar: Char;
- StringsAdapter: IStringsAdapter;
- Text: string;
- Values[const Name: string]: string;

TStringList Methods

- Add(const S: string): Integer;
- AddObject(const S: string; AObject: TObject: Integer);
- Clear
- Delete(Index: Integer);
- Destroy
- Exchange(Index1, Index2: Integer);
- Find(const S: string; var Index: Integer): Boolean;
- IndexOf(const S: string): Integer;

- Insert(Index: Integer; const S: string);
- InsertObject(Index: Integer; const S: string; AObject: TObject);
- Sort

Methods derived from TStrings

- AddStrings(Strings: TStrings);
- Append(const S: string);
- Assign(Source: TPersistent);
- BeginUpdate
- EndUpdate
- Equals(Strings: TStrings): Boolean;
- GetText: PChar;
- IndexOfName(const Name: string): Integer;
- IndexOfObject(AObject: TObject): Integer;
- LoadFromFile(const FileName: string);
- LoadFromStream(Stream: TStream);
- Move(CurIndex, NewIndex: Integer);
- SaveToFile(const FileName: string);
- SaveToStream(Stream: TStream);
- SetText(Text: PChar);

Methods derived from TPersistent

- GetNamePath

Methods derived from TObject

- AfterConstruction
- BeforeDestruction
- ClassInfo
- ClassName
- ClassNameIs
- ClassParent
- ClassType
- CleanupInstance
- Create
- DefaultHandler
- Dispatch
- FieldAddress
- Free

- FreeInstance
- GetInterface
- GetInterfaceEntry
- GetInterfaceTable
- InheritsFrom
- InitInstance
- InstanceSize
- MethodAddress
- MethodName
- NewInstance
- SafeCallException

Example

```
Procedure TDialogForm.FormCreate(Sender: TObject);
Var
    StringsList : TStringList;
    Index       : Integer;
Begin
    StringsList := TStringList.Create;
    Try
        StringsList.Add('Capacitors');
        StringsList.Add('Resistors');
        StringsList.Add('Antennas');
        StringsList.Sort;

        // The Find method will only work on sorted lists.
        If StringsList.Find('Resistor', Index) then
        Begin
            ListBox.Items.AddStrings(StringsList);
            Label.Caption := 'Antennas has an index value of ' +
IntToStr(Index);
        End;
        Finally
            StringsList.Free;
        End;
    End;
End;
```

Example of a customized sorting routine

Refer to the Netlister script example in the **\Examples\Scripts\WSM** folder.

See also

Helper Classes and Functions

Index

E

F

Revision History

Date	Version No.	Revision
01-Dec-2004	1.0	New product release