Altium Designer System Reference

Summary

Technical Reference
TR0135 (v1.9) August 4, 2008

This reference provides a concise reference of the Altium Designer low level system API as part of the Altium Designer Run Time Library.

The System Reference contains low level Application Programming Interface information that can be used for scripting and server development in Altium Designer.

The Altium Designer Run time Library is composed of Units and some of them are automatically exposed for the scripting system. For the server projects, you need to add the `Units` in the `Uses` clause in the server project where appropriate.

Altium Designer Run Time Library

Scripting System

The scripting system implements a subset of the Altium Designer Run Time Library. Normally the units that are available from the Altium Designer RTL in the Scripting system are also available to use in server projects.

Server Development system

The Server Development system uses the full set of the Altium Designer RTL for development of servers and add-ons. Where the documentation is not covered in this online help it will be covered in the **Altium Designer RTL Reference for Servers** document as part of the Server Development Kit.

System Reference for Scripting and Server Development

Object Interfaces and Routines common to Scripting System and Server Development

- Client Server Interfaces (`RT_ClientServerInterface` unit)
- Routines that deal with server processes (`ClientAPIReg` and `RT_Param` units)
- Routines that deal with low level implementation (`RT_Util` unit and `RT_FileUnit`)
- Routines and objects exposed from Borland Delphi units (in Helper Functions and Objects section) for the Scripting System only. In server projects, you have access to any Borland Delphi units.

Separate API References for other APIs

- Schematic Object Model (`RT_Schematic`) refer to *Schematic API Reference*
- PCB Object Model (`RT_PCB` and `RT_PCBProcs`) refer to *PCB API Reference*
- FPGA Object Model (`RT_NexusWorkspace`, `RT_NexusDevices`, `RT_FPGA`) refer to *FPGA API Reference*
- Integrated Library Object Model (`RT_IntegratedLibrary` unit) refer to *Integrated Library API Reference*
- Workspace Manager Object Model (`RT_Workspace` unit) refer to *Workspace Manager API Reference*

Client Server API Reference

The Client/Server Application Programming Interface reference covers interfaces for Client/Server objects in the Client/Server Object Model as part of the `RT_ClientServerInterface` unit from the Altium Designer RTL and exposed for use in scripts from the Scripting System.

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 Altium Designer 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 Altium Designer.

Example

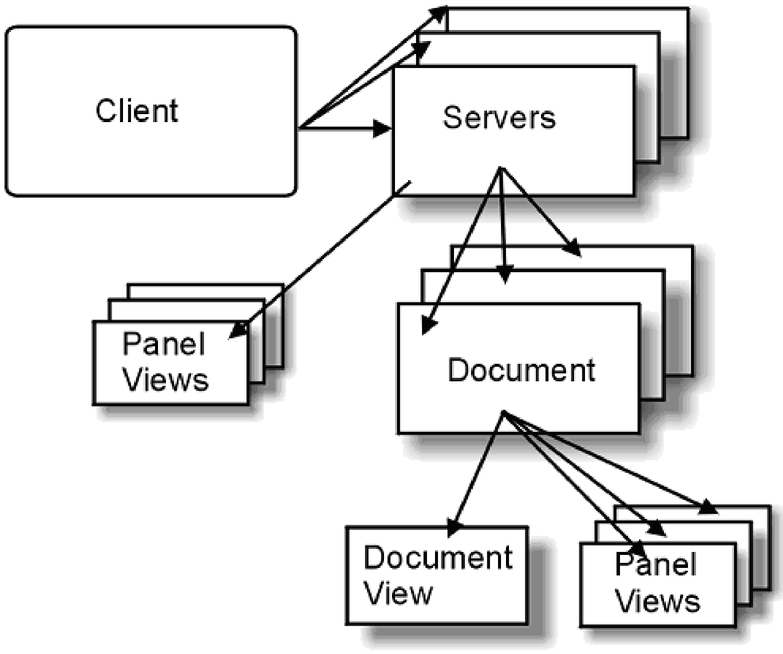
```
Var
    ReportDocument : IServerDocument;
Begin
    If Client = Nil Then Exit;
    // Opens and shows a text file in Altium Designer
    ReportDocument  := Client.OpenDocument('Text',FileName);
    If ReportDocument <> Nil Then
          Client.ShowDocument(ReportDocument);
End;
```

Script Examples

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

Using Client / Server Interfaces

Central to the Altium Designer 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 Altium Designer– there is one Client executable and several servers as loaded dynamic library linked modules as shown in the diagram below.



Object Interfaces

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

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, the locations and state of panels)
- `IServerModule` (deals with loaded servers)
- `INotification` (broadcast or dispatch notification messages to servers or to a specified server)

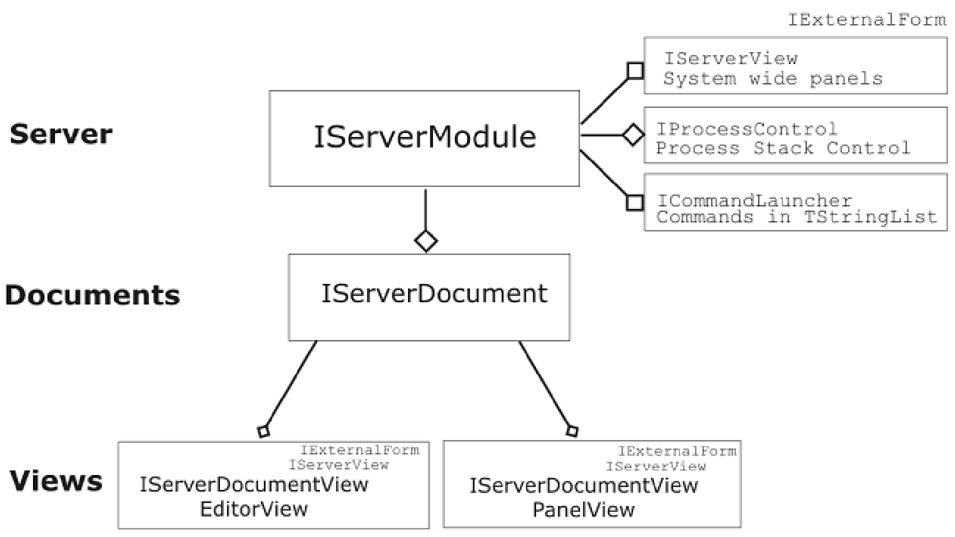
Server Interfaces

The `IServerModule` interfaces represent loaded servers in Altium Designer. 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:MoveCursorToTopOfDocument',
                                                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 Altium Designer)

• `IServerView` (represents a panel that can have a view of the system)

• `IServerDocumentView` (deals with a document view (either the document window or panel window))

• `IExternalForm` (represents Altium Designer 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)

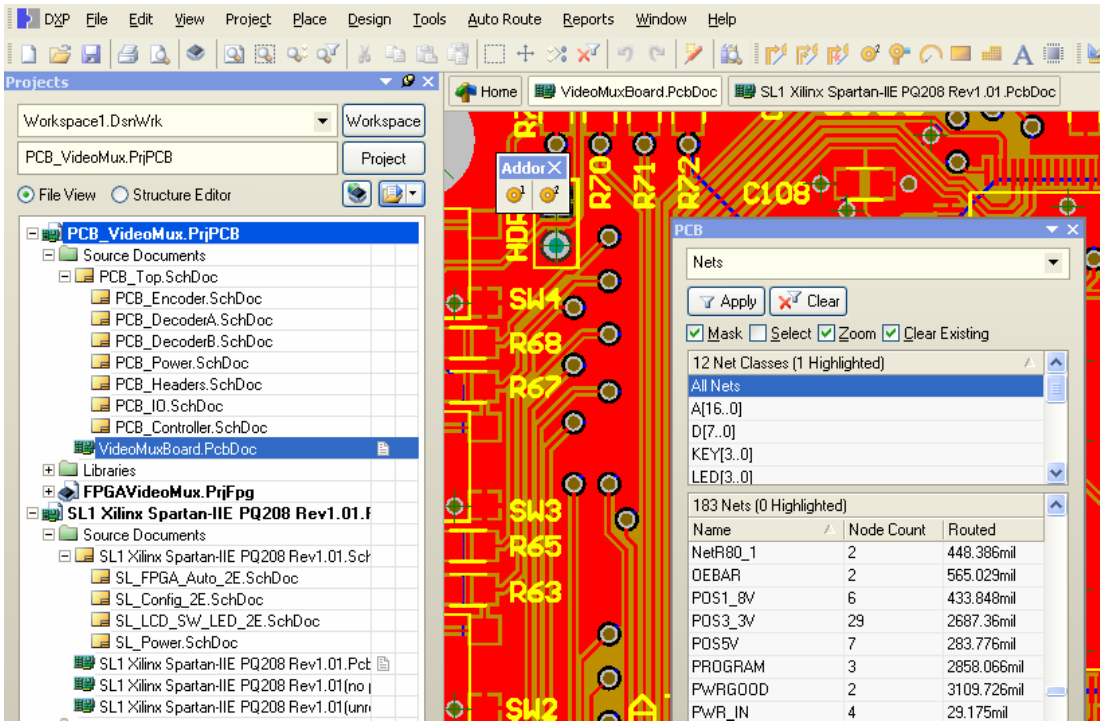
• `INotification` represents the system notifications from the Client system and all server modules receive these notifications. There is an ability to handle a notification and take it from there. Documents and associated panels can be synchronized through the use of notifications as well).

Servers Documents and Panels Interfaces in Altium Designer

The concept of documents and panels are central to understanding how servers work in Altium Designer. The servers manage their own panels and documents. Altium Designer has access to the currently active panels and documents and manages 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 Altium Designer 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 Altium Designer 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.



- PCB_VideoMux.PrjPCB
  - Source Documents
    - PCB_Top.SchDoc
      - PCB_Encoder.SchDoc
      - PCB_DecoderA.SchDoc
      - PCB_DecoderB.SchDoc
      - PCB_Power.SchDoc
      - PCB_Headers.SchDoc
      - PCB_IO.SchDoc
      - PCB_Controller.SchDoc
    - VideoMuxBoard.PcbDoc
  - Libraries
- FPGAVideoMux.PrjFpg
- SL1 Xilinx Spartan-IIE PQ208 Rev1.01.
  - Source Documents
    - SL1 Xilinx Spartan-IIE PQ208 Rev1.01.Sch
      - SL_FPGA_Auto_2E.SchDoc
      - SL_Config_2E.SchDoc
      - SL_LCD_SW_LED_2E.SchDoc
      - SL_Power.SchDoc
    - SL1 Xilinx Spartan-IIE PQ208 Rev1.01.Pcb
    - SL1 Xilinx Spartan-IIE PQ208 Rev1.01(no
    - SL1 Xilinx Spartan-IIE PQ208 Rev1.01(unr

12 Net Classes (1 Highlighted)

- All Nets
- A[16..0]
- D[7..0]
- KEY[3..0]
- LED[3..0]

183 Nets (0 Highlighted)

Name	Node Count	Routed
NetR80_1	2	448.386mil
OEBAR	2	565.029mil
POS1_8V	6	433.848mil
POS3_3V	29	2687.36mil
POS5V	7	283.776mil
PROGRAM	3	2858.066mil
PWRGOOD	2	3109.726mil
PWR_IN	4	29.175mil

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

The Client system within the Altium Designer 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 have 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 Altium Designer 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 Altium Designer is encapsulated by the `IServerModule` interface, so from figure above, there is an `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 above, 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

The major interfaces that are used in the client – server architecture within Altium Designer 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 Altium Designer, the locations and state of panels)

• `IServerModule` (deals with a loaded server in Altium Designer)

• `INotification` (Client can broadcast or dispatch notification messages to servers or to a specified server)

Altium Designer's Configuration Interfaces:

• IServerRecord (collect servers information at Altium Designer's start up – not loaded servers)

• IServerWindowKind (determines which document kinds open in Altium Designer)

• IServerProcess (contains the information of a current server process)

IServerModule Interfaces represent loaded servers in Altium Designer

An IServerModule interface has the following interfaces:

• `ICommandLauncher` interface (deals with a server's processes table)

• `IServerDocument` interface (represents a loaded design document in Altium Designer)

• `IServerView` interface (represents a panel that can have a view of the Altium Designer system)

• `IServerDocumentView` interface (deals with a document view (either the document window or panel window))

• `IExternalForm` interface (represents the Altium Designer aware Delphi form either as a document form or a panel form. These forms are wrapped by the `IServerDocumentView` or `IServerView` interface objects. 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` interface receives 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 (from `RT_ClientServerInterface` unit) represents the Client subsystem of the Altium Designer application and the Client manages the commands (pre packaged process launchers), process depths and documents. The every server module loaded in Altium Designer has hooks to the single client executable subsystem, so you have access to the specific documents of any loaded servers 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 ALtium Designer, the locations and state of panels)

• `IServerModule` (deals with loaded servers in ALtium Deisgner)

• `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 and Properties Table

IClient methods	**IClient Properties**
`AddServerView`	`ApplicationHandle`
`AddViewToFavorites`	`CommandLauncher`
`ApplicationIdle`	`Count`
`BeginDisableInterface`	`CurrentView`
`BeginDocumentLoad`	`GUIManager`
`BeginRecoverySave`	`MainWindowHandle`
`BroadcastNotification`	`NavigationSystem`
`CanServerStarted`	`ProcessControl`
`CloseDocument`	`ServerModule`
`DispatchNotification`	`ServerModuleByName`
`EndDisableInterface`	`TimerManager`
`EndDocumentLoad`	
`EndRecoverySave`	
`GetApplicationHandle`	
`GetCommandLauncher`	
`GetCount`	
`GetCurrentView`	
`GetDefaultExtensionForDocumentKind`	
`GetDocumentByPath`	
`GetDocumentKindFromDocumentPath`	
`GetDynamicHelpManager`	
`GetEncryptedTechnologySets`	
`GetGUIManager`	
`GetMainWindowHandle`	
`GetNavigationSystem`	
`GetOptionsSet`	
`GetOptionsSetByName`	

```
GetOptionsSetCount
GetPanelInfoByName
GetProcessControl
GetRealMainWindowHandle
GetServerModule
GetServerModuleByName
GetServerNameByPLID
GetServerRecord
GetServerRecordByName
GetServerRecordCount
GetServerViewFromName
GetTimerManager
GetWindowKindByName
HideDocument
InRecoverySave
IsDocumentOpen
IsQuitting
LastActiveDocumentOfType
LicenseInfoStillValid
OpenDocument
OpenDocumentShowOrHide
QuerySystemFont
RegisterNotificationHandler
RemoveServerView
SetCurrentView
ShowDocument
ShowDocumentDontFocus
StartServer
StopServer
UnregisterNotificationHandler
```

IClient Methods

AddServerView method

(IClient interface)

Syntax

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

Description

This procedure adds a document view such as a custom panel in the Client object within Altium Designer. In the `TServerModule` constructor, where the server commands are registered, this is the place to create global panel views. The `TServerModule.CreateServerViews` method will have the global panel form and the view created from this panel form. Then the view is added to the server module (`TServerModule.AddView()`) as well as in the client object (`Client.AddServerView`).

See also

IServerView interface

IClient interface

RT_ServerImplementation for the TServerModule class.

ApplicationIdle method

(IClient interface)

Syntax

```
Procedure ApplicationIdle;
```

Description

When the `ApplicationIdle` method is invoked, the procedure puts the Altium Designer in a mode where it has a chance to process Window and Altium Designer specific messages.

See also

IClient interface

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.

See also

EndDisableInterface method

IClient interface

BeginDocumentLoad method

(IClient interface)

Syntax

```
Procedure BeginDocumentLoad;
```

Description

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

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

IClient interface

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 Altium Designer. To check if the recovery save process is in progress, invoke the `InRecoverySave` method.

See also

EndRecoverySave method

InRecoverySave method

IClient interface

BroadcastNotification method

(IClient interface)

Syntax

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

Description

This procedure broadcasts a notification message in Altium Designer 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

IClient interface

Client_CanServerStarted method

(IClient interface)

Syntax

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

Description

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

See also

IClient interface

StartServer method

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 Altium Designer already). Note the document is not removed from Altium Designer, that is, the document still exists on the **Projects** panel for example.

See also

OpenDocument method

IClient interface

Count property

(IClient interface)

Syntax

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

Description

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

See also

GetCount method

IServerModule interface

IClient interface

DispatchNotification method

(IClient interface)

Syntax

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

Description

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

See also

INotification interface

IClient interface

EndDisableInterface method

(IClient interface)

Syntax

```
Procedure EndDisableInterface;
```

Description

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

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

See also

BeginDisableInterface method

IClient interface

EndDocumentLoad method

(IClient interface)

Syntax

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

Description

The `BeginDocumentLoad` and `EndDocumentLoad` procedures are used to load a group of documents in Altium Designer.

Example

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

See also

IClient interface

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 Altium Designer.

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

See also

BeginRecoverySave method

InRecoverySave method

IClient interface

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 Altium Designer will inherit Altium Designer’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

IClient interface

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

IClient interface

GetCount method

(IClient interface)

Syntax

```
Function  GetCount : Integer;
```

Description

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

See also

Count property

IClient interface

GetCurrentView method

(IClient interface)

Syntax

```
Function GetCurrentView : IServerDocumentView;
```

Description

This function fetches the current view (ie the open document in focus in Altium Designer). 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

IClient interface

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.

IClient interface

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.

See also

IClient interface

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.

See also

IClient interface

GetEncryptedTechnologySets method

(IClient interface)

Syntax

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

Description

Example

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 Altium Designer such as controlling the status bars of Altium Designer, the locations and the state of panels in Altium Designer.

See also

IGUIManager interface

IClient interface

GetLicenseManager function

(IClient interface)

Syntax

```
Function  GetLicenseManager : ILicenseManager;
```

Description

Example

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 Altium Designer will inherit Altium Designer’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

IClient interface

GetNavigationSystem method

(IClient interface)

Syntax

```
Function GetNavigationSystem : INavigationSystem;
```

Description

The function returns the Navigation system interface.

See also

INavigationSystem interface

IClient interface

GetOptionsManager function

(IClient interface)

Syntax

```
Function  GetOptionsManager : IOptionsManager;
```

Description

This method retrieves the `IOptionsManager` interface. With this interface, you can invoke the GetOptionsReader or GetOptionsWriter to retrieve or write options (settings) for the target server. Each editor server has options that manage its server documents.

Example

```
Var
    Reader : IOptionsReader;
Begin
    Reader := Client.OptionsManager.GetOptionsReader(NameOfServer,'');
    If Reader = Nil Then Exit;

    AValue := Reader.ReadBoolean(NameOfServerPreferences,SettingName,DefaultValue);
End;
```

See also

IClient interface

IOptionsManager

GetOptionsSetByName method

(IClient interface)

Syntax

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

Description

This function retrieves the IDocumentOptionsSet interface based on the valid Name string.

See also

GetOptionsSetCount method

GetOptionsSet method

IDocumentOptionsSet interface

IClient interface

GetOptionsSetCount method

(IClient interface)

Syntax

```
Function  GetOptionsSetCount : Integer;
```

Description

This function returns you the number of Options Set.

See also

GetOptionsSet method

GetOptionsSetByName method

IClient interface

GetOptionsSet method

(IClient interface)

Syntax

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

Description

This function returns you the indexed Options set (IDocumentOptionsSet type).

See also

GetOptionsSetCount method

GetOptionsSetByName method

IClient interface

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

IClient 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 Altium Designer.

See also

IProcessControl interface

IClient interface

GetRealMainWindowHandle method

(IClient interface)

Syntax

```
Function  GetRealMainWindowHandle : THandle;
```

Description

The function returns the window handle of the main window in Altium Designer.

See also

IClient interface

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).

See also

IClient interface

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 Altium Designer.

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 Altium Designer

```
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

IClient interface

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

IClient interface

GetServerRecord method

(IClient interface)

Syntax

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

Description

The `GetServerRecord` function reports the number of installed servers based on the installation `*.INS` files in the System folder of Altium Designer installation). 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 `\System\` folder of Altium Designer installation).

The `GetServerModule` merely returns the active (loaded) server in Altium Designer 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

IClient interface

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 `\System\` folder of the Altium Designer software installation. This is to be used in conjunction with the `GetServerRecord` function.

See also

IServerRecord interface

IClient interface

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).

Example

```
Var
    ClientModule : IClient;
    ServerRecord : IServerRecord;
    Version      : WideString;
Begin
    ClientModule := Client;
    If ClientModule = Nil Then Exit;

    //The IServerRecord interface encapsulates the details
    // of a server's installation file

    //We are interested in the Altium Designer's Client Module
    // and fetch the product version.
    ServerRecord := ClientModule.GetServerRecordByName('CLIENT');
    Version := ServerRecord.GetVersion;

    ShowMessage(Version);
End;
```

See also

IServerRecord interface

IClient interface

GetServerViewFromName method

(IClient interface)

Syntax

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

Description

This function returns the server view object interface depending on the name of the server view. A IServerView interface represents a panel view as well as an ancestor for a document view.

See also

IExternalForm interface

IServerView interface

IClient interface

GetTimerManager Interface

(IClient interface)

Syntax

```
Function  GetTimerManager : ITimerManager;
```

Description

This function returns the timer manager interface associated with the client sub system.

See also

ITimerManager interface

IClient 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

IClient interface

HideDocument method

(IClient interface)

Syntax

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

Description

This procedure hides the document, ie puts it out of focus but not closed or destroyed.

See also

CloseDocument method

OpenDocument method

ShowDocument method

IServerDocument interface

IClient interface

OpenDocumentShowOrHide method

(IClient interface)

Syntax

```
Function  OpenDocumentShowOrHide              (Const AKind, AFileName : WideString;
AShowInTree : Boolean) : IServerDocument;
```

Description

This function opens a specific document but you can control how it is displayed in the Altium Designer workspace.

See also

IClient interface

HandleException method

(IClient interface)

Syntax

```
Procedure HandleException (Const AMessage   : WideString);
```

Description

Example

See also

IClient interface

InRecoverySave method

(IClient interface)

Syntax

```
Function  InRecoverySave : LongBool
```

Description

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

See also

BeginRecoverySave method

EndRecoverySave method

IClient interface

IsDocumentOpen method

(IClient interface)

Syntax

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

Description

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

See also

IClient interface

IsQuitting method

(IClient interface)

Syntax

```
Function  IsQuitting : Boolean;
```

Description

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

See also

IClient interface

LastActiveDocumentOfType method

(IClient interface)

Syntax

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

Description

This function returns the last active loaded document in Altium Designer by the document type. Types include PCB, SCH, TEXT, WAVE, PCBLIB, SCHLIB.

See also

IClient interface

IsInitialized function

(IClient interface)

Syntax

```
Function  IsInitialized : LongBool;
```

Description

Example

See also

Client interface

LicenseInfoStillValid method

(IClient interface)

Syntax

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

Description

See also

IClient interface

MainWindowHandle property

(IClient interface)

Syntax

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

Description

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

See also

GetMainWindowHandle method

ApplicationHandle property

IClient interface

OpenDocument method

(IClient interface)

Syntax

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

Description

The OpenDocument method returns the `IServerDocument` interface depending on the DocumentKind and FileName values of this document are valid.

Example

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

See also

ShowDocument method

IClient interface

OpenNewDocument method

(IClient interface)

Syntax

```
Function OpenNewDocument (Const AKind, AFileName, ANewName : Widestring; ReuseExisting : 
Boolean) : IServerDocument;
```

Description

Example

See also

IClient interface

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.

See also

IClient interface

RegisterNotificationHandler method

(IClient interface)

Syntax

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

Description

The `RegisterNotificationHandler` method registers the notification handler in the Client module part of Altium Designer once the server object is created and loaded in computer memory. The Handler parameter contains the server module object.

Notes

The `INotificationHandler` object interface is responsible for handling notifications raised in Altium Designer.

Each server object has a `HandleNotification` procedure to handle notifications when the options values have been adjusted from the system wide Preferences dialog.

The `HandleNotification` procedure would involve calls to update the server preferences values on the server panel for example every-time a specific server notification code is intercepted.

This method is normally used for in developing servers and not for scripts.

See also

BroadcastNotification method

DispatchNotification method

UnRegisterNotificationHandler method

INotificationHandler interface

IClient interface

RemoveServerView method

(IClient interface)

Syntax

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

Description

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

See also

GetCurrentView method

IClient interface

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

IClient interface

ShowDocument method

(IClient interface)

Syntax

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

Description

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

IServerDocument example

This example gets the client interface and then opens and shows a document.

```
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

IClient 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 Altium Designer.

See also

GetCurrentView method

CurrentView property

IClient interface

StopServer method

(IClient interface)

Syntax

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

Description

The `StartServer` and `StopServer` properties can be used to load a server in Altium Designer if it has not loaded 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 Altium Designer 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

IClient interface

StartServer method

(IClient interface)

Syntax

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

Description

The `StartServer` and `StopServer` properties can be used to load a server in Altium Designer 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 Altium Designer 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

IClient interface

UnregisterNotificationHandler method

(IClient interface)

Syntax

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

Description

The `UnregisterNotificationHandler` method un registers the notification handler from Client once the server object goes out of scope (destroyed). The Handler parameter contains the server module object.

Notes

The `INotificationHandler` object interface is responsible for handling notifications raised in Altium Designer.

Each server object has a `HandleNotification` procedure to handle notifications when the options values have been adjusted from the system wide Preferences dialog.

The `HandleNotification` procedure would involve calls to update the server preferences values on the server panel for example every-time a specific server notification code is intercepted.

This method is normally used for in developing servers and not for scripts.

See also

BroadcastNotification

DispatchNotification

RegisterNotificationHandler method

INotificationHandler interface

IClient interface

AddViewToFavorites method

(IClient interface)

Syntax

```
Function  AddViewToFavorites(Const AView : IServerDocumentView; AIsSnippet : Boolean) :
Boolean;
```

Description

Example

See also

IClient interface

GetDynamicHelpManager method

(IClient interface)

Syntax

```
Function GetDynamicHelpManager : IDynamicHelpManager;
```

Description

The method returns the Dynamic Help manager which represents the Knowledge Center panel in Altium Designer.

See also

IClient interface

IDynamicHelpManager interface.

IClient 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 Altium Designer will inherit Altium Designer'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.

Note

Normally script writers will not need to worry about this applicationhandle property. This property is used by the server writers as part of the Altium Designer SDK.

Server Example

```
In the server project's main unit
Function ServerFactory (AClient : IClient) : IServerModule; Safecall;
Begin
    Result := TAddOn.Create(AClient, 'AddOn');
    Application.Handle := Client.ApplicationHandle;
End;

In the server project's commands unit
Procedure DisplayResultsOnDialog(PadCount : TDynamicString);
Var
    DisplayForm : TDialog;
Begin
    DisplayForm := TDialog.Create(Application);
    DisplayForm.Label1.Caption := PadCount;
    DisplayForm.ShowModal;
    DisplayForm.Free;
End;
```

See also

IClient interface

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 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

IClient interface

CurrentView property

(IClient interface)

Syntax

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

Description

This property returns the current document view interface which represents the current design document view in Altium Designer.

SendMessage Example

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

CurrentView example

```
Procedure GrabACurrentDocumentView;
Var
    ServerDocumentView : IServerDocumentView;
    FileName   : WideString;
Begin
    ServerDocumentView := Client.CurrentView;
    FileName := ServerDocumentView.GetOwnerDocument.FileName;
End;
```

ViewName example

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

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

See also

GetCurrentView method

SetCurrentView method

IServerDocumentView interface

IClient 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 Altium Designer's Graphical User Interface such as controlling the status bars, the locations and the state of panels.

See also

IGUIManager interface

IClient interface

NavigationSystem property

(IClient interface)

Syntax

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

Description

The NavigationSystem property represents the Navigation system in Altium Designer. The navigation system is the workhouse for the Navigation panel which is the center-piece for net connectivity for the design project. There are three ways a design can be arranged - as a list of compiled sheets, flattened hierarchy and as a structural tree.

Example

See also

IClient interface

INavigationSystem interface

ProcessControl property

(IClient interface)

Syntax

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

Description

This property returns the **IProcessControl** 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 Altium Designer. Refer to the **IProcessControl** interface for details.

ProcessDepth Example

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

See also

IClient interface

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 Altium Designer.

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 Altium Designer

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

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

See also

IClient interface

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.

Server Names

Example

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

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

See also

IClient interface

IServerModule interface

TimerManager property

(IClient interface)

Syntax

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

Description

This property returns the timer manager object interface.

See also

IClient interface

ITimerManager interface

OptionsManager property

(IClient interface)

Syntax

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

Description

This is a read only property that returns the `IOptionsManager` interface. This interface is responsible for managing (reading and writing) values to/from the system wide Preferences dialog in Altium Designer for the specified server.

This interface is useful for server writers who wish to add their options pages in the system wide preferences dialog and manage the controls on these options pages.

Example

```
Var
    Reader : IOptionsReader;
Begin
    Reader := Client.OptionsManager.GetOptionsReader(NameOfServer,'');
    If Reader = Nil Then Exit;

    AValue := Reader.ReadBoolean(NameOfServerPreferences,SettingName,DefaultValue);
End;
```

See also

IClient interface

IOptionsManager interface

IServerModule Interface

Overview

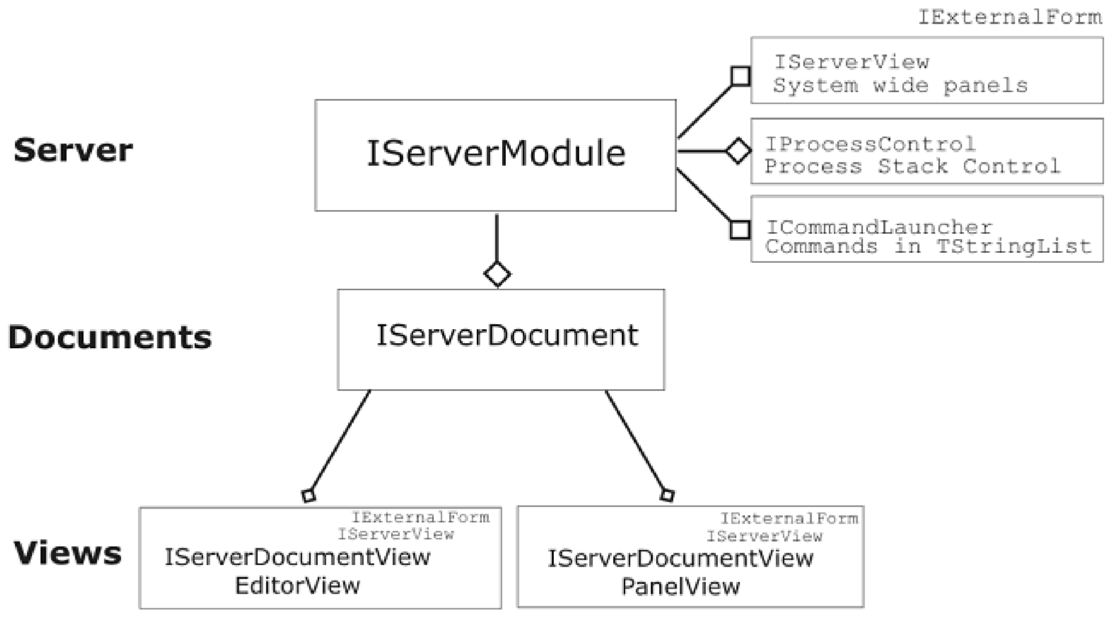
A server deals with its own server documents. There can be different design document types, for example the Schematic Editor has two Schematic and Schematic Library document types.

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 below.

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 Altium Designer client / server architecture.

These `IServerModule` interfaces (from the `RT_ClientServerInterface` unit) represent loaded servers in Altium Designer. This 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 Altium Designer 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 Altium Designer

• `IServerView` represents a panel that can have a view of the Altium Designer system

• `IServerDocumentView` (deals with a document view (either the document window or panel window)

• `IExternalForm` represents a Altium Designer 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` receives 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 and Properties Table

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

See also

IPCB_ServerInterface interface

ISCH_ServerInterface interface

IServerModule GetState and SetState Methods

GetClient method

(IServerModule interface)

Syntax

```
Function GetClient : IClient;
```

Description

The `GetClient` method returns the `IClient` interface of the client subsystem of Altium Designer. This `IClient` interface can be used to invoke its methods.

The `GetClient` method is used for the Client property.

Example

See also

IServerModule interface

GetCommandLauncher method

(IServerModule interface)

Syntax

```
Function GetCommandLauncher : ICommandLauncher;
```

Description

The `CommandLauncher` function returns 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 then nothing will happen.

This `CommandLauncher` object is initialized in the main.pas unit of a server project. See the `ICommandLauncher` interface for more details.

This method is used for the `CommandLauncher` property.

Example

See also

IServerModule interface

GetDocumentCount method

(IServerModule interface)

Syntax

```
Function GetDocumentCount : Integer;
```

Description

The `DocumentCount` method returns you the number of Document Kinds. An important note is that a View is the actual design document. A Document type is a container that stores specific Views.

This method is used for the `DocumentCount` property.

Example

See also

IServerModule interface

GetDocuments method

(IServerModule interface)

Syntax

```
Function GetDocuments (Index : Integer) : IServerDocument;
```

Description

An editor type of server can have different document types, such as Schematic Editor and PCB Editor - these editor 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 acting like a piggy back and utilising the editor server's API services.

This method returns you the indexed document container which is represented by the `IServerDocument` interface.

This method is used for the `Documents` property.

Example

See also

IServerModule interface

IServerDocument interface

GetHandle method

(IServerModule interface)

Syntax

```
Function GetHandle : THandle;
```

Description

The method returns the handle of the server.

This method is used for the Handle property.

Example

See also

IServerModule interface

GetModuleName method

(IServerModule interface)

Syntax

```
Function GetModuleName : Widestring;
```

Description

The method returns the module name of this server.

For example the texteditor server's module name is TextEdit. This server name property is defined in the associated server installation file (with an INS file extension).

This method is used for the `ModuleName` property.

Example

See also

IServerModule interface

GetProcessControl method

(IServerModule interface)

Syntax

```
Function GetProcessControl : IProcessControl;
```

Description

The method returns the `IProcessControl` interface. This interface controls the process depth for each design document in Altium Designer.

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 read only property is supported by the `GetProcessControl` method.

Example

See also

IServerModule interface

GetViewCount method

(IServerModule interface)

Syntax

```
Function GetViewCount : Integer;
```

Description

The `ViewCount` method returns you the number of views for the specified server.

A View object encapsulates a form/window object in Altium Designer normally as a global panel supported by its associated server.

This method is used for the `ViewCount` property.

Example

See also

IServerModule interface

GetViews method

(IServerModule interface)

Syntax

```
Function GetViews (Index : Integer) : IServerView;
```

Description

The `GetViews` method in conjunction with the `GetViewCount` method returns you the indexed View object. A view is a form supported by its associated server.

This method is used for the `Views` property.

Example

See also

IServerModule interface

IServerModule Methods

AddServerView method

(IServerModule interface)

Syntax

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

Description

This procedure adds a panel in the Server Module where this new panel can be used by the module.

Invoke this function after you have created a `IServerView` object with the `CreateServerView` function or pass in the `IServerView` interface parameter.

Example

See also

IServerModule interface

IServerView interface

ApplicationIdle method

(IServerModule interface)

Syntax

```
Procedure ApplicationIdle;
```

Description

The `ApplicationIdle` procedure is an internal procedure that gets invoked when Altium Designer is idling. The `ApplicationIdle` procedure in all active running servers gets invoked. The messages sent by Altium Designer get the chance to be followed up.

Example

See also

IServerModule interface

CreateDocument method

(IServerModule interface)

Syntax

```
Function CreateDocument (Const AKind, AFileName : Widestring) : IServerDocument;
```

Description

The `CreateDocument` function creates a document supported by the server based on the `AKind` and `AFilename` parameters.

The `AKind` parameter represents the document kind that the server supports and the AFileName parameter is assigned to the new document.

Example

See also

IServerModule interface

CreateServerDocView method

(IServerModule interface)

Syntax

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

Description

The `CreateServerDocView` function creates an `IServerDocumentView` (which could be the document or its associated panel view) object based on the Name of the document view and the `IServerDocument` container.

Example

See also

IServerModule interface

CreateServerView method

(IServerModule interface)

Syntax

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

Description

The `CreateServerView` function creates a `IServerView` object representing a system panel. You need to invoke the `AddServerView` procedure to add the object within Altium Designer.

Example

See also

IServerModule interface

CreateOptionsView method

(IServerModule interface)

Syntax

```
Function CreateOptionsView (Const AName : Widestring) : IDocumentOptionsView;
```

Description

The `CreateOptionsView` creates a `IDocumentOptions` view to be used in the system wide Preferences dialog in Altium Designer.

Example

See also

IServerModule interface

DestroyDocument method

(IServerModule interface)

Syntax

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

Description

The `DestroyDocument` procedure closes and removes the design document as specified by the `ADocument` parameter.

Example

See also

IServerModule interface

ReceiveNotification method

(IServerModule interface)

Syntax

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

Description

The `ReceiveNotification` procedure of the server module intercepts notifications broadcasted by Altium Designer.

The system has a `BroadCastNotification` or a `DispatchNotification` function which all running servers in Altium Designer can receive and process accordingly.

This procedure needs to be overridden and implemented.

Example

See also

IServerModule interface

RemoveServerView method

(IServerModule interface)

Syntax

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

Description

The `RemoveServerView` procedure removes a `IServerView` object in Altium Designer which represents a system panel.

Example

See also

IServerModule interface

CreateDocumentShowOrHide method

(IServerModule interface)

Syntax

```
Function  CreateDocumentShowOrHide(Const AKind, AFileName : Widestring;
            AShowInTree : Boolean) : IServerDocument;
```

Description

The `CreateDocumentShowOrHide` function controls how a document when created is displayed in Altium Designer.

Example

See also

IServerModule interface

Properties

Client property

(IServerModule interface)

Syntax

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

Description

The Client property returns the `IClient` interface of the client subsystem of Altium Designer. This `IClient` interface can be used to invoke its methods.

This readonly property is supported by the `GetClient` method.

Example

See also

IServerModule 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. See the `ICommandLauncher` interface for more details.

This read-only property is supported by the GetCommandLauncher method.

Example

See also

IServerModule interface

DocumentCount property

(IServerModule interface)

Syntax

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

Description

The `DocumentCount` property returns you the number of Document Kinds. An important note is that a View is the actual design document. A Document type is a container that stores specific Views.

This property is supported by the `GetDocumentCount` method.

Example

See also

IServerModule interface

Documents property

(IDocuments interface)

Syntax

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

Description

An editor type of server can have different document types, such as Schematic Editor and PCB Editor - these editor 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 acting like a piggy back and utilising the editor server's API services.

This property returns you the indexed document container which is represented by the `IServerDocument` interface.

This read only property is supported by the GetDocuments method.

Example

See also

IClient interface

IServerModule interface

DocumentCount property

Handle property

(IServerModule interface)

Syntax

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

Description

The Handle property returns the handle of the server. This read only property is supported by the `GetHandle` method.

Example

See also

IServerModule interface

ModuleName property

(IServerModule interface)

Syntax

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

Description

The `ModuleName` property returns the module name of this server.

For example the Texteditor server's module name is TextEdit. This server name property is defined in the associated server installation file (with an INS file extension).

This read only property is supported by the `GetModuleName` method.

Example

```
If StringsEqual(ServerModule.ModuleName,'TextEdit') Then
Begin
...
End;
```

See also

IServerModule interface

ProcessControl property

(IServerModule interface)

Syntax

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

Description

The `ProcessControl` property returns the pointer to the IProcessControl interface. This interface controls the process depth for each design document in Altium Designer.

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 read only property is supported by the `GetProcessControl` method.

Example

See also

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 Altium Designer normally as a global panel supported by its associated server.

This read only property is supported by the `GetViewCount` method.

Example

See also

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.

This read only property is supported by the `GetViews` method.

Example

See also

IClient interface

IServerModule interface

Document and Panel View Interfaces

IExternalForm

Overview

The `IExternalForm` interface represents a 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.

Notes

The Altium Designer platform is based on the object interfaces technology by Borland(TM), thererfore TForm, TFrame, and other VCL controls to object interfaces are not passed into object interfaces that can be exposed to third party development in different programming systems. For example VCL technology is not compatible with MS C++ toolkit.

Therefore to work with windows in the Altium Designer platform, you use the `IExternalForm` interface to have access to windows and manipulate them. The `IExternalFormHolder` interface and the `TExternalFormComponent` class are used to work with Delphi windows in a server plugged into the Altium Designer platform and accessible to other servers plugged in.

IExternalForm Methods and Properties Table

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

See also

IServerView interface

IServerDocumentView interface

IExternalFormHolder interface

TExternalFormComponent class from ExternalForm unit

TServerExternalFormComponent class from ExternalForm unit.

IExternalForm Methods

FocusFirstTabStop method

(IExternalForm interface)

Syntax

```
Procedure FocusFirstTabStop;
```

Description

Example

See also

IClient interface

IExternalForm interface

GetBounds method

(IExternalForm interface)

Syntax

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

Description

This procedure retrieves the four bounds (left, top, width and height) of the form.

Example

See also

IClient interface

IExternalForm interface

Hide method

(IExternalForm interface)

Syntax

```
Procedure Hide;
```

Description

This `Hide` method hides the form from view in Altium Designer.

Example

See also

IClient interface

IExternalForm interface

ParentWindowCreated method

(IExternalForm interface)

Syntax

```
Procedure ParentWindowCreated;
```

Description

Example

See also

IClient interface

IExternalForm interface

ParentWindowDestroyed method

(IExternalForm interface)

Syntax

```
Procedure ParentWindowDestroyed;
```

Description

Example

See also

IClient interface

IExternalForm interface

SetBounds method

(IExternalForm interface)

Syntax

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

Description

This procedure sets the bounds of the external form.

Example

See also

IClient interface

IExternalForm interface

SetFocus method

(IExternalForm interface)

Syntax

```
Procedure SetFocus;
```

Description

This procedure sets the Delphi based form in focus in Altium Designer.

Example

See also

IClient interface

IExternalForm interface

SetParentWindow method

(IExternalForm interface)

Syntax

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

Description

Example

See also

IClient interface

IExternalForm interface

Show method

(IExternalForm interface)

Syntax

```
Procedure Show;
```

Description

This procedure displays the hidden form.

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.

Example

See also

IClient interface

IExternalForm interface

Handle property

(IExternalForm interface)

Syntax

`Property Handle : HWND`

Description

A read only property that returns the handle of the Delphi based form.

Example

See also

IClient interface

IExternalForm interface

IExternalFormHolder interface

Overview

The `IExternalFormHolder` interface represents the `TExternalFormComponent` object and holds the `IExternalForm` interface.

Notes

The DXP platform is based on the object interfaces technology by Borland(TM), therefore TForm, TFrame, and other VCL controls to object interfaces are not passed into object interfaces that can be exposed to third party development in different programming systems. For example VCL technology is not compatible with MS C++ toolkit.

Therefore to work with windows in the Altium Designer platform, you use the `IExternalForm` interface to have access to windows and manipulate them. The `IExternalFormHolder` interface and the `TExternalFormComponent` class are used to work with Delphi windows in a server plugged into the Altium Designer platform.

IExternalFormHolder Methods and Properties Table

IExternalFormHolder methods	**IExternalFormHolder properties**
GetParentWindow	
SetDialogHandle	

See also

IExternalForm interface

TExternalFormComponent class in ExternalForm unit.

IExternalFormHolder Methods

GetParentWindow method

(IExternalFormHolder interface)

Syntax

`Function GetParentWindow : THandle;`

Description

This function retrieves the `THandle` of the parent window that can be used in the IExternalForm interface.

Example

See also

IExternalFormHolder interface

SetDialogHandle method

(IExternalFormHolder interface)

Syntax

```
Procedure SetDialogHandle (AHandle : THandle);
```

Description

This procedure sets the dialog handle for this external form.

Example

See also

IExternalFormHolder interface

IHTMLViewExternalForm interface

Overview

The **IHTMLViewExternalForm** interface represents a HTML document.

IHTMLViewExternalForm methods	IHTMLViewExternalForm properties
GetCtrlClickInNewWindow	CtrlClickInNewWindow
SetCtrlClickInNewWindow	
NavigateTo	
GetHTMLDocument	

ISceneViewinterface

Overview

The `ISceneView` interface represents a specific view.

ISceneView methods	ISceneView properties
CanClose	

INavigationDocument

Overview

The `INavigationDocument` interface represents a specific navigation view.

INavigationDocument methods	INavigationDocument properties
GetDocumentScene	

See also

IExternalForm interface

IServerView interface

Overview

The `IServerView` interface is the ancestor interface for a document or panel view object interface.

This `IServerView` interface also represents a global panel in Altium Designer, for example the Messages or ToDo panels.

The IServerView interface hierarchy is as follows;

IExternalForm

IServerView interface

IServerView Methods and Properties Table

IServerView Methods	**IServerView Properties**
`GetViewState`	`IsPanel`
`SetViewState`	`ViewName`
`ReceiveNotification`	

See also

IExternalForm interface

IServerDocumentView interface

IServerDocument interface

IServerView GetState and SetState methods

GetIsPanel method

(IServerView interface)

Syntax

```
Function  GetIsPanel : LongBool;
```

Description

The IsPanel property determines whether the IServerDocumentView object is a panel or not. A IServerDocument container stores IServerDocumentView objects and they are can be a panel view or a document view.

This property is supported by the GetIsPanel method.

Example

```
Var
ServerDocumentView : IServerDocumentView;
Begin
ServerDocumentView := ServerDocument.View[j];
If Not(ServerDocumentView.IsPanel) Then
      ShowMessage('Document Name ' + ServerDocument.FileName);
End;
```

See also

IClient interface

IExternalForm interface

GetViewName method

(IServerView interface)

Syntax

```
Function GetViewName : Widestring;
```

Description

The ViewName property represents the view name and is not the same as the document filename. A view can be a global panel that can be seen globally within Altium Designer, as a document view or as a panel view.

This read only property is supported by the GetViewName method.

For example a library document open in Altium Designer yields the following information:

View Name: `PCBEditor`

Document Name: `C:\Program Files\Altium Designer\Examples\Reference Designs\4 Port Serial Interface\Libraries\4 Port Serial Interface.PcbLib`

Caption: PCBView_GraphicalForm

ViewName example

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

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

See also

IClient interface

IServerView interface

IExternalForm interface

IServerView Methods

GetViewState method

(IServerView interface)

Syntax

```
Function  GetViewState : Widestring;
```

Description

Example

See also

IClient interface

IServerView interface

SetViewState method

ReceiveNotification method

(IServerView interface)

Syntax

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

Description

The `ReceiveNotification` procedure captures the notification generated by Altium Designer. A global panel, a document view or a panel view has the ability to intercept a notification and take action accordingly.

Example

See also

IClient interface

IServerView interface

INotification interface

SetViewState method

(IServerView interface)

Syntax

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

Description

Example

See also

IClient interface

IExternalForm interface

GetViewState method

IServerView 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.

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.

This read only property is supported by the GetIsPanel method.

Example

```
Var
ServerDocumentView : IServerDocumentView;
Begin
ServerDocumentView := ServerDocument.View[j];
If Not(ServerDocumentView.IsPanel) Then
      ShowMessage('Document Name ' + ServerDocument.FileName);
End;
```

See also

IServerView interface

ViewName property

(IServerView interface)

Syntax

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

Description

The `ViewName` property represents the view name and is not the same as the document filename. A view can be a global panel that can be seen globally within Altium Designer, as a document view or as a panel view.

This read only property is supported by the GetViewName method.

For example a library document open in Altium Designer yields the following information:

View Name: `PCBEditor`

Document Name: `C:\Program Files\Altium Designer\Examples\Reference Designs\4 Port Serial Interface\Libraries\4 Port Serial Interface.PcbLib`

Caption: PCBView_GraphicalForm

ViewName example

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

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

See also

IClient interface

IServerView interface

IServerDocumentView Interface

Overview

The `IServerDocumentView` represents either the document view or one of the associated panel views in Altium Designer. This interface is inherited from the `IServerView` interface.

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

The hierarchy is as follows;

IExternalForm

IServerView interface

IServerDocumentView interface

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

IServerView Methods	**IServerView Properties**
`GetViewState`	`IsPanel`
`SetViewState`	`ViewName`
`ReceiveNotification`	

IServerDocumentView Methods and Properties Table

IServerDocumentView Methods	**IServerDocumentView Properties**
`GetOwnerDocument`	`OwnerDocument`
`PerformAutoZoom`	
`UpdateStatusBar`	

See also

IClient interface

IServerModule interface

IServerDocument interface

IServerView interface

IExternalForm interface

IServerDocumentView GetState and SetState Methods

GetOwnerDocument method

(IServerDocumentView interface)

Syntax

```
Function GetOwnerDocument : IServerDocument;
```

Description

The `OwnerDocument` property returns the IServerDocument interface that the `IServerDocumentView` itnerface is associated with. An `IServerDocument` container stores `IServerDocumentView` interfaces which represent a document or panel view.

This read only property is supported by the `GetOwnerDocument` method.

Example

See also

IClient interface

IServerDocumentView interface

IServerDocumentView Methods

PerformAutoZoom method

(IServerDocumentView interface)

Syntax

```
Procedure PerformAutoZoom;
```

Description

This procedure forces a refresh or repaint of the document / panel view.

Example

See also

IClient interface

IServerDocumentView interface

UpdateStatusBar method

(IServerDocumentView interface)

Syntax

```
Procedure UpdateStatusBar;
```

Description

This procedure forces an update of the status bar when a string is submitted to the status bar.

Example

See also

IClient interface

IServerDocumentView interface

IServerDocumentView Properties

OwnerDocument property

(IServerDocumentView interface)

Syntax

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

Description

This property returns the `IServerDocument` interface that the `IServerDocumentView` interface is associated with. An IServerDocument container stores `IServerDocumentView` interfaces which represent a document or panel view.

This read only property is supported by the `GetOwnerDocument` method.

Example

See also

IClient interface

IExternalForm interface

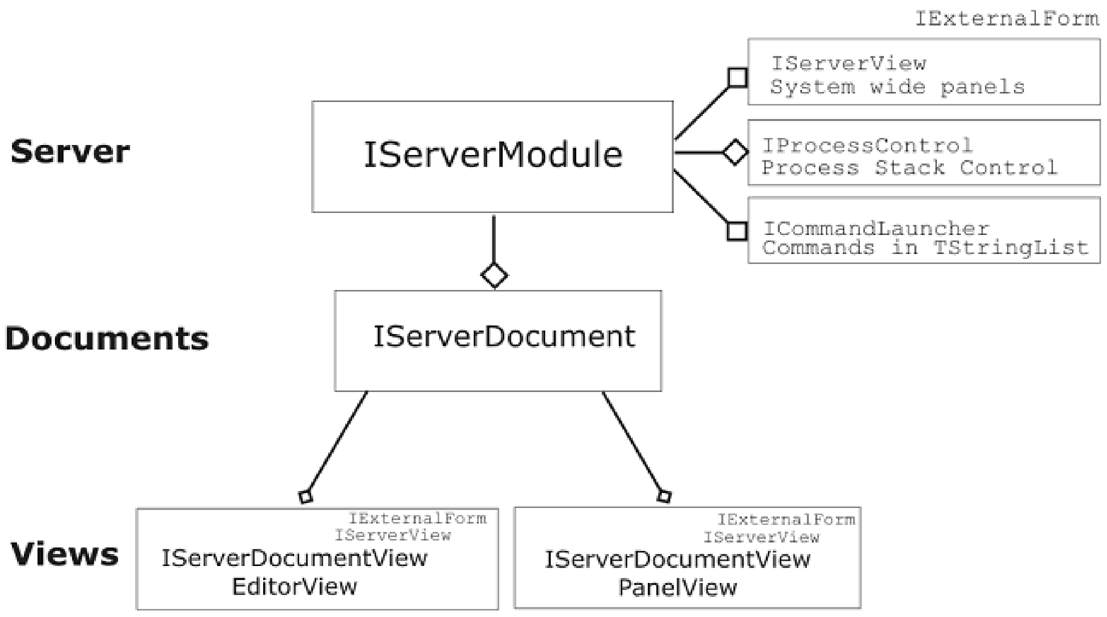
IServerDocument Interface

Overview

The `IServerDocument` interface represents the document container. Each `IServerDocument` interface is a document containter made up of views of the same kind.

A view can be a design document form or a panel form.

Every document editor server (encapsulated by the `IServerModule` interface) that supports creation of documents will have a `IServerDocument` interface.



The **IServerDocument** interface hierarchy is as follows;

IServerDocument Methods and Properties Table

IServerDocument methods	**IServerDocument properties**
AddView	CanClose
SetModified	Count
SetIsShown	FileName
SetBeingClosed	Kind
Focus	Modified
DoFileLoad	IsShown
DoFileSave	BeingClosed
SupportsReload	ServerModule
GetCanClose	View
GetCount	SupportsOwnSave
GetFileName	
SetFileName	
GetKind	
GetModified	
GetIsShown	
GetBeingClosed	
GetFileModifiedDate	

UpdateModifiedDate

GetServerModule

GetView

GetViewByName

NotifyViews

GetSupportsOwnSave

GetContextHelpTopicName

SetFileModifiedDate

WarnIfOwnedByOther

AcquireFileOwnership

ReleaseFileOwnership

ReleaseDataFileHandle

AcquireDataFileHandle

OwnsFile

DoSafeFileSave

DoSafeChangeFileNameAndSave

CreateSnippetFile

ZoomSnippetContents

GetSnippetView

PlaceSnippet

CanPlaceSnippet

CanCreateSnippet

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

IServerDocumentView interface

IServerView interface

CS server example in the \Developer Kit\Examples\DXP\ClientServer Interfaces\ folder.

IServerDocument Methods

AddView method

(IServerDocument interface)

Syntax

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

Description

This procedure adds a IServerDocumentView object in the server document. A IServerDocument object is a container containing views of document views and panel views.

Example

See also

IServerDocument interface

IServerDocumentView interface

DoFileLoad method

(IServerDocument interface)

Syntax

```
Function DoFileLoad : LongBool;
```

Description

This function allows the re-loading of the document. This is useful if the document has been modified and saved and it needs to be re-loaded to ensure that the document is in the latest state.

Example

See also

IServerDocument interface

DoFileSave method

(IServerDocument interface)

Syntax

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

Description

This function provides you an option to save the document in a different format if the document supported by the specific document editor provides the option of saving in a different format other than the default format. Normally these file formats are stored in the SaveFilters block within the EditorWindowKind section within a server installation file (with an INS extension).

File Formats

For example with PCB documents in Altium Designer, you can save them as a PCB ASCII format, PCB Binary 3 format etc - PCB Binary, PCB 3.0 Binary, PCB 4.0 Binary, PCB ASCII. By default its PCB Binary 5.0.

With Schematic documents, you can save them as a Advanced Schematic binary, Advanced Schematic ascii, Schematic binary 4.0, Orcad SDT Schematic, Advanced Schematic template.

Server Installation files

The file formats supported by editors can be found in the server installation files within the **SaveFilters** - **End** blocks.

DelphiScript Example

```
Var
    Board           : IPCB_Document;
    AView           : IServerDocumentView;
    AServerDocument : IServerDocument;
Begin
    // save the file in a different PCB format
    //check if current document is a PCB document otherwise exit!
    Board := PCBServer.GetCurrentPCBBoard;
    If Board = Nil Then Exit;
    If Client = Nil Then Exit;

    // Grab the current document view using the Client's Interface.
    AView := Client.GetCurrentView;
    AServerDocument := AView.OwnerDocument;
    AServerDocument.DoFileSave('PCB ASCII');
    Close;
End;
```

See also

IServerDocument interface

IServerDocument interface

GetCanClose method

GetModified method

GetFileName method

Focus method

(IServerDocument interface)

Syntax

```
Procedure Focus;
```

Description

The procedure forces the document to be the focussed document in Altium Designer. A focussed document is the top level document and in view in Altium Designer workspace that responds to commands etc.

Example

See also

IServerDocument interface

GetBeingClosed method

(IServerDocument interface)

Syntax

```
Function GetBeingClosed : LongBool;
```

Description

The function determines whether the server document is being closed or not. Use the GetCanClose function to check if the document can be closed or not.

Example

See also

IServerDocument interface

GetCanClose method

GetModified method

GetFileName method

DoFileSave method

GetCanClose method

(IServerDocument interface)

Syntax

```
Function GetCanClose : LongBool;
```

Description

This function checks whether the document can be closed or not. This method is used for the CanClose property.

Example

See also

IServerDocument interface

GetModified method

GetFileName method

DoFileSave method

GetContextHelpTopicName method

(IServerDocument interface)

Syntax

```
Function GetContextHelpTopicName : Widestring;
```

Description

The GetContextHelpTopicName function retrieves the help topic name for the document. Normally the returned string would be the ServerModuleName.DocumentKind format for example 'SCH.SCH' Some servers provide more detailed information, for example Schematic Editor server returns Sch.Sheet.Port when the mouse is over the Port object on a schematic sheet.

Notes

Third party developers can use this function to provide context sensitive help.

To implement the help for your server, you should have a .HELPID file in the Help folder where the link between the string returned by the GetContextHelpTopicName and the actual help document is established.

For example the CXTSystemDesignCapture.HelpID contains a Sch.Sheet.Port = CXTSystemDesignCapture.chm,Document_Objects\Port.htm. This means when the F1 key is pressed and the Sch.Sheet.Port string is returned, it will use the CXTSystemDesignCapture.chm filename and display the Document_Objects\Port.htm topic.

Example

See also

IServerDocument interface

GetCount method

(IServerDocument interface)

Syntax

```
Function GetCount : Integer;
```

Description

The `Count` property returns the number of views (of the same type) in the `IServerDocument` container. Use in conjunction with the `View` property.

This method is used for the `Count` property.

Example

```
Var
    ServerModule       : IServerModule;
    ServerDocument     : IServerDocument;
    ServerDocumentView : IServerDocumentView;
Begin
ServerModule := Client.ServerModuleByName['PCB'];
If ServerModule = Nil Then Exit;

For I := 0 to ServerModule.DocumentCount - 1 Do
Begin
    ServerDocument := ServerModule.Documents[I];
    ShowMessage('Document View Count ' +
    IntToStr(ServerDocument.Count) + #13 +
                'Kind ' + ServerDocument.Kind));
End;
End;
```

See also

IServerDocument interface

GetFileModifiedDate method

(IServerDocument interface)

Syntax

```
Function GetFileModifiedDate: TDateTime;
```

Description

This function returns the date and time of the modified file.

Example

See also

IServerDocument interface

GetFileModifiedDate method

SetFileModifiedDate method

TDateTime type from Borland Delphi Run Time Library.

GetFileName method

(IServerDocument interface)

Syntax

```
Function GetFileName : Widestring;
```

Description

This function retrieves the file name as a string for the server document. Note a server document can be a document view or a panel view, and thus if it is a panel view, the `GetFileName` method is invalid.

Example

```
ServerDocumentView := ServerDocument.View[j];
If Not(ServerDocumentView.IsPanel) Then
    ShowMessage('  Document Name ' +
                ServerDocument.FileName);
```

See also

IServerDocument interface

GetIsShown method

(IServerDocument interface)

Syntax

```
Function GetIsShown : LongBool;
```

Description

The IsShown property denotes whether or not this document is displayed in Altium Designer. This property is supported by the `GetIsShown` and `SetIsShown` methods.

Example

See also

IServerDocument interface

GetKind method

(IServerDocument interface)

Syntax

```
Function GetKind : Widestring;
```

Description

This function returns the Kind string for this document and this function is used for the Kind property. Examples include ‘PCB’, ‘PCBLIB’,’SCH’,’SCHLIB’ etc.

Example

```
ServerModule := Client.ServerModuleByName['PCB'];
If ServerModule = Nil Then Exit;

For I := 0 to ServerModule.DocumentCount - 1 Do
Begin
    ServerDocument := ServerModule.Documents[I];
    ShowMessage('Document View Count ' +
         IntToStr(ServerDocument.Count) + #13 +
                'Kind ' + ServerDocument.GetKind));
End;
```

See also

IServerDocument interface

GetModified method

(IServerDocument interface)

Syntax

```
Function GetModified : LongBool;
```

Description

The `Modified` 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.

This property is supported by the `GetModified` and `SetModified` methods.

Example

```
Var
    AView           : IServerDocumentView;
    AServerDocument : IServerDocument;
Begin
    If Client = Nil Then Exit;
    // Grab the current document view using the Client's Interface.
    AView := Client.GetCurrentView;

    // Grab the server document which stores views by extracting the ownerdocument field.
    AServerDocument := AView.OwnerDocument;

    // Set the document dirty.
    AServerDocument.Modified := True;
End;
```

See also

IServerDocument interface

GetServerModule method

(IServerDocument interface)

Syntax

```
Function GetServerModule : IServerModule;
```

Description

The `ServerModule` is a read-only property which returns the `IServerModule` interface that the document is associated with. The server module represents the server object installed and running in Altium Designer.

A server module manages its own documents and panels. This property is supported by the `GetServerModule` method.

Example

```
//IServerModule interface
ServerModule := Client.ServerModuleByName['PCB'];
If ServerModule = Nil Then Exit;

ShowMessage(IntToStr(ServerModule.DocumentCount));
For I := 0 to ServerModule.DocumentCount - 1 Do
Begin
    //IServerDocument interface
    ServerDocument := ServerModule.Documents[I];
    // do what you want with server documents
End;
```

See also

IServerDocument interface

IServerModule interface

GetSupportsOwnSave method

(IServerDocument interface)

Syntax

```
Function GetSupportsOwnSave : LongBool;
```

Description

The `SupportsOwnSave` property returns a boolean value whether a save routine has been provided to save these documents associated with the server. This is a read only property and is supported by the GetSupportsOwnSave method.

Example

See also

IServerDocument interface

GetView method

(IServerDocument interface)

Syntax

```
Function GetView (Index : Integer) : IServerDocumentView;
```

Description

The `View` property is an indexed property and represents a document or panel view. The `IServerDocument.Count` method returns the list of views (which could be document or panel windows) as part of the `IServerDocument` container.

This property is supported by the `GetView` method.

Example

```
For J := 0 to ServerDocument.Count - 1 Do
Begin
    ServerDocumentView := ServerDocument.View[j];
    ShowMessage('View Name ' + ServerDocumentView.ViewName);

    If Not(ServerDocumentView.IsPanel) Then
        ShowMessage('  Document Name ' +
                    ServerDocument.FileName);
```

```
End;
```

See also

IServerDocument interface

GetViewByName method

(IServerDocument interface)

Syntax

```
Function GetViewByName (Const ViewName : Widestring) : IServerDocumentView;
```

Description

The `GetViewByName` function returns the View object which represents a document or panel view.

Example

```
ServerDocumentView := ServerDocument.GetViewByName(PCBExpressionFilter);
If ServerDocumentView.IsPanel Then
    ShowMessage('This Server Document View is a Panel');
```

See also

IServerDocument interface

IServerDocumentView interface

SetBeingClosed method

(IServerDocument interface)

Syntax

```
Procedure SetBeingClosed (Const Value : LongBool);
```

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.

This property is supported by the `GetBeingClosed` and `SetBeingClosed` methods.

Example

See also

IServerDocument interface

SetFileModifiedDate method

(IServerDocument interface)

Syntax

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

Description

The procedure sets the modified date for the document if the document has been modified by an outside agent.

Example

See also

IServerDocument interface

GetModified method

SetModified method

SetFileName method

(IServerDocument interface)

Syntax

```
Function SetFileName (Const AFileName : Widestring): Widestring;
```

Description

The SetFileName function sets the filename for the document.

Example

See also

IServerDocument interface

SetIsShown method

(IServerDocument interface)

Syntax

```
Procedure SetIsShown (Const Value : LongBool);
```

Description

The `IsShown` property denotes whether or not this document is displayed in Altium Designer. This property is supported by the GetIsShown and SetIsShown methods.

Example

See also

IServerDocument interface

SetModified method

(IServerDocument interface)

Syntax

```
Procedure SetModified (Const Value : LongBool);
```

Description

The `Modified` 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.

This property is supported by the GetModified and SetModified methods.

Example

```
Var
    AView           : IServerDocumentView;
    AServerDocument : IServerDocument;
Begin
    If Client = Nil Then Exit;
    // Grab the current document view using the Client's Interface.
    AView := Client.GetCurrentView;

    // Grab the server document which stores views by extracting the ownerdocument field.
    AServerDocument := AView.OwnerDocument;

    // Set the document dirty.
    AServerDocument.Modified := True;
End;
```

See also

IServerDocument interface

NotifyViews method

(IServerDocument interface)

Syntax

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

Description

This procedure sends a notification to all the views associated with the **IServerDocument** container.

Example

See also

IServerDocument interface

INotification interface

SupportsReload method

(IServerDocument interface)

Syntax

```
Function SupportsReload : LongBool;
```

Description

This method determines whether the document in Altium Designer can be re loaded or not (to refresh and to make sure that the document state is the latest).

Example

See also

IServerDocument interface

DoFileLoad method

UpdateModifiedDate method

(IServerDocument interface)

Syntax

```
Procedure UpdateModifiedDate;
```

Description

The procedure updates the modified document's date after this document has been modified.

Example

See also

IServerDocument interface

GetModified method

SetModified method

ReleaseFileOwnership method

(IServerDocument interface)

Syntax

```
Procedure ReleaseFileOwnership;
```

Description

For internal use only.

Example

See also

IServerDocument interface

ReleaseDataFileHandle method

(IServerDocument interface)

Syntax

```
Procedure ReleaseDataFileHandle;
```

Description

For internal use only.

Example

See also

IServerDocument interface

OwnsFile method

(IServerDocument interface)

Syntax

```
Function OwnsFile : Boolean;
```

Description

The `OwnsFile` function determines whether the document is owned by the Altium Designer product and thus this document can be saved or not.

Example

See also

IServerDocument interface

DoSafeFileSave method

(IServerDocument interface)

Syntax

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

Description

The function determines whether the document can be saved of specified document type safely.

Example

See also

IServerDocument interface

DoSafeChangeFileNameAndSave method

(IServerDocument interface)

Syntax

```
Function DoSafeChangeFileNameAndSave(Const ANewFileName, AKind : Widestring) : LongBool;
```

Description

The function determines whether the current document can be saved with the new file name and new document type or not.

Example

See also

IServerDocument interface

AcquireFileOwnership method

(IServerDocument interface)

Syntax

```
Procedure AcquireFileOwnership;
```

Description

For internal use only.

Example

See also

IServerDocument interface

AcquireDataFileHandle method

(IServerDocument interface)

Syntax

```
Procedure AcquireDataFileHandle;
```

Description

For internal use only.

Example

See also

IServerDocument interface

WarnIfOwnedByOther method

(IServerDocument interface)

Syntax

```
Function WarnIfOwnedByOther(AWarningLevel : TFileOwnershipWarningLevel) : LongBool;
```

Description

This function determines whether the document is owned by another user. A document can be shared amongst other users but the other users cannot save this document when this document is owned solely by one user.

Example

See also

IServerDocument interface

IServerDocument 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.

This property is supported by the `GetBeingClosed` and `SetBeingClosed` methods.

Example

See also

IClient interface

IServerDocument interface

CanClose property

(IServerDocument interface)

Syntax

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

Description

This CanClose property determines whether the document can be closed or not.

Example

See also

IClient interface

IServerDocument interface

Count property

(IServerDocument interface)

Syntax

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

Description

The `Count` property returns the number of views (of the same type) in the `IServerDocument` container. Use in conjunction with the `View` property.

This property is supported by the `GetCount` method.

Example

```
Var
    ServerModule       : IServerModule;
    ServerDocument     : IServerDocument;
    ServerDocumentView : IServerDocumentView;
Begin
ServerModule := Client.ServerModuleByName['PCB'];
If ServerModule = Nil Then Exit;

For I := 0 to ServerModule.DocumentCount - 1 Do
Begin
    ServerDocument := ServerModule.Documents[I];
    ShowMessage('Document View Count ' +
    IntToStr(ServerDocument.Count) + #13 +
                'Kind ' + ServerDocument.Kind));
End;
End;
```

See also

IClient interface

IServerDocument interface

Filename property

(IServerDocument interface)

Syntax

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

Description

The `FileName` property returns the filename for the server document (not the corresponding server panel). This property is a read-only property and is supported by the `GetFileName` method.

Note a server document can be a document view or a panel view, and thus if it is a panel view, the FileName property is invalid.

Example

```
ServerDocumentView := ServerDocument.View[j];
If Not(ServerDocumentView.IsPanel) Then
    ShowMessage('  Document Name ' +
                ServerDocument.FileName);
```

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 Altium Designer. This property is supported by the `GetIsShown` and `SetIsShown` methods.

Example

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 Altium Designer.

Examples include ‘PCB’, ‘PCBLIB’,’SCH’,’SCHLIB’ etc. This property is a read-only property. This property is supported by the `GetKind` method.

Example

```
ServerModule := Client.ServerModuleByName['PCB'];
If ServerModule = Nil Then Exit;

For I := 0 to ServerModule.DocumentCount - 1 Do
Begin
    ServerDocument := ServerModule.Documents[I];
    ShowMessage('Document View Count ' +
         IntToStr(ServerDocument.Count) + #13 +
                'Kind ' + ServerDocument.Kind));
End;
```

See also

IClient interface

IServerDocument interface

Modified property

(IServerDocument interface)

Syntax

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

Description

The `Modified` 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.

This property is supported by the GetModified and SetModified methods.

Example

```
Var
```

```
    AView           : IServerDocumentView;
    AServerDocument : IServerDocument;
Begin
    If Client = Nil Then Exit;
    // Grab the current document view using the Client's Interface.
    AView := Client.GetCurrentView;

    // Grab the server document which stores views by extracting the ownerdocument field.
    AServerDocument := AView.OwnerDocument;

    // Set the document dirty.
    AServerDocument.Modified := True;
End;
```

See also

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 `IServerModule` interface that the document is associated with. The server module represents the server object installed and running in Altium Designer.

A server module manages its own documents and panels. This property is supported by the `GetServerModule` method.

Example

```
//IServerModule interface
ServerModule := Client.ServerModuleByName['PCB'];
If ServerModule = Nil Then Exit;

ShowMessage(IntToStr(ServerModule.DocumentCount));
For I := 0 to ServerModule.DocumentCount - 1 Do
Begin
    //IServerDocument interface
    ServerDocument := ServerModule.Documents[I];
    // do what you want with server documents
End;
```

See also

IClient interface

IServerDocument interface

IServerModule 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.

Example

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 and represents a document or panel view part of the `IDocument` container associated with a specific IServerModule interface. The `IServerDocument.Count` method returns the list of views (which could be document or panel windows) as part of the `IServerDocument` container.

This property is supported by the `GetView` method.

Example

```
For J := 0 to ServerDocument.Count - 1 Do
Begin
    ServerDocumentView := ServerDocument.View[j];
    ShowMessage('View Name ' + ServerDocumentView.ViewName);

    If Not(ServerDocumentView.IsPanel) Then
        ShowMessage('  Document Name ' +
                        ServerDocument.FileName);
End;
```

See also

IClient interface

IServerDocument interface

IHighlightedDocument Interface

Overview

This `IHighlightedDocument` interface represents a mechanism that deals with highlighting of objects on a design document (especially Schematic and PCB documents) in Altium Designer when objects are being selected or deselected and when being masked or not.

This interface and its methods are for internal use.

Notes

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

IHighlightedDocument Methods and Properties Table

IHighlightedDocument methods	IHighlightedDocument properties
`HL_Begin`	`Property  HL_HighlightedNet : INet`
`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
```

See also

IServerDocument interface

IServerPanelInfo Interface

Overview

The `IServerPanelInfo` interface encapsulates the details of a panel in Altium Designer and the details can be Name, Bitmap, whether the panel can be docked horizontally or vertically and so on.

This interface is used by the `IServerRecord` interface and the `IClient` interface.

IServerPanelInfo Methods and Properties Table

IServerPanelInfo methods	**IServerPanelInfo properties**
GetName	DocumentKindCount
GetCategory	DocumentKinds[Index
GetBitmap	ProjectKindCount
GetHotkey	ProjectKinds
GetButtonVisible	
GetMultipleCreation	
GetCreationClassName	
GetCanDockVertical	
GetCanDockHorizontal	
SupportsDocumentKind	
SupportsProjectKind	
GetDocumentKindCount	
GetDocumentKinds	
GetProjectKindCount	
GetProjectKinds	

See also

IServerRecord interface

IClient Interface

IServerPanelInfo Methods

GetBitmap method

(IServerPanelInfo interface)

Syntax

```
Function GetBitmap : Widestring;
```

Description

The function returns the name of the bitmap.

Example

See also

IServerPanelInfo interface

GetButtonVisible method

(IServerPanelInfo interface)

Syntax

```
Function GetButtonVisible : Boolean;
```

Description

The function returns whether the button on the panel is visible or not.

Example

See also

IServerPanelInfo interface

GetCanDockHorizontal method

(IServerPanelInfo interface)

Syntax

```
Function GetCanDockHorizontal: Boolean;
```

Description

This function determines whether the panel can be docked horizontally to the Altium Designer User Interface.

Example

See also

IServerPanelInfo interface

GetCanDockVertical method

(IServerPanelInfo interface)

Syntax

```
Function GetCanDockVertical : Boolean;
```

Description

This function determines whether the panel can be docked vertically to the Altium Designer User Interface.

Example

See also

IServerPanelInfo interface

GetCategory method

(IServerPanelInfo interface)

Syntax

```
Function GetCategory : Widestring;
```

Description

This function returns the Category string, ie which module it is part of within Altium Designer. For example the Favorites panel is part of the System.

Example

See also

IServerPanelInfo interface

GetCreationClassName method

(IServerPanelInfo interface)

Syntax

```
Function GetCreationClassName: Widestring;
```

Description

Internal use.

Example

See also

IServerPanelInfo interface

GetDocumentKindCount method

(IServerPanelInfo interface)

Syntax

```
Function GetDocumentKindCount : Integer;
```

Description

This function reports how many document kinds this panel can be associated with. For example with Simulation Breakpoints panel, it can be associated with VHDL and VHDTST documents.

Use this function with the GetDocumentKinds function.

Example

See also

IServerPanelInfo interface

GetDocumentKinds method

(IServerPanelInfo interface)

Syntax

```
Function GetDocumentKinds(Index : Integer) : WideString;
```

Description

This function returns the indexed Document Kind string that this panel is associated with. For example with Simulation Breakpoints panel, it can be associated with VHDL and VHDTST documents. This function is to be used in conjunction with the GetDocumentKindCount function.

Example

See also

IServerPanelInfo interface

GetHotkey method

(IServerPanelInfo interface)

Syntax

```
Function GetHotkey : Widestring;
```

Description

The function returns the HotKey string that is used to render the panel visible or not.

Example

See also

IServerPanelInfo interface

GetMultipleCreation method

(IServerPanelInfo interface)

Syntax

```
Function GetMultipleCreation : Boolean;
```

Description

Internal use.

Example

See also

IServerPanelInfo interface

GetName method

(IServerPanelInfo interface)

Syntax

```
Function GetName : Widestring;
```

Description

This function returns the name of the panel. For example the PCB Library panel has a PCBLibPanel name.

Example

See also

IServerPanelInfo interface

GetProjectKindCount method

(IServerPanelInfo interface)

Syntax

```
Function GetProjectKindCount : Integer;
```

Description

Internal use.

Example

See also

IServerPanelInfo interface

GetProjectKinds method

(IServerPanelInfo interface)

Syntax

```
Function GetProjectKinds(Index : Integer) : WideString;
```

Description

Internal use.

Example

See also

IServerPanelInfo interface

SupportsDocumentKind method

(IServerPanelInfo interface)

Syntax

```
Function SupportsDocumentKind(Const AKind : Widestring) : Boolean;
```

Description

This function determines whether the document kind is supported by the panel.

Example

See also

IServerPanelInfo interface

SupportsProjectKind method

(IServerPanelInfo interface)

Syntax

```
Function SupportsProjectKind (Const AKind : Widestring) : Boolean;
```

Description

Internal use.

Example

See also

IServerPanelInfo interface

IServerPanelInfo Properties

DocumentKindCount property

(IServerPanelInfo interface)

Syntax

```
Property DocumentKindCount : Integer read GetDocumentKindCount;
```

Description

This property reports how many document kinds this panel can be associated with. For example with Simulation Breakpoints panel, it can be associated with VHDL and VHDTST documents.

Use this property with the DocumentKinds property.

Example

See also

IServerPanelInfo interface

DocumentKinds property

(IServerPanelInfo interface)

Syntax

```
Property DocumentKinds[Index : Integer] : WideString read GetDocumentKinds;
```

Description

This property returns the indexed Document Kind string that this panel is associated with. For example with Simulation Breakpoints panel, it can be associated with VHDL and VHDTST documents. This property is to be used in conjunction with the GetDocumentKindCount function.

Example

See also

IServerPanelInfo interface

ProjectKindCount property

(IServerPanelInfo interface)

Syntax

```
Property ProjectKindCount : Integer read GetProjectKindCount;
```

Description

Internal use

Example

See also

IServerPanelInfo interface

ProjectKinds property

(IServerPanelInfo interface)

Syntax

```
Property ProjectKinds[Index : Integer] : WideString read GetProjectKinds;
```

Description

Internal use

Example

See also

IServerPanelInfo interface

System Interfaces

ICommandLauncher Interface

Overview

The `ICommandLauncher` interface encapsulates the functionality of launching a command (which is a pre packaged process) in Altium Designer. 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 Altium Designer 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 and Properties Table

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 Altium Designer), 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 Altium Designer, process procedures or commands registered in the CommandLauncher object within the server module are loaded in Altium Designer.

See also

IClient interface

IServerModule interface

ICommandLauncher Methods

GetCommandState

(ICommandLauncher interface)

Syntax

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

Description

The GetCommandState procedure fetches the current snapshot of the server command (internal server process) and the parameters are returned for the specified server command name.

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:MoveCursorToTopOfDocument',
                                               Nil,0,ServerDocument.View[0]);
End;
```

See also

IServerDocumentView interface

IGUIManager Interface

Overview

The IGUIManager interface represents the Graphical User interface portions of the Altium Designer application such as resizing panels, checking for certain hot key maps and status bars.

IGUIManager methods	**IGUIManager properties**
AddKeyStrokeAndLaunch	
AddKeyToBuffer	
BeginDragDrop	
CanResizePanel	
CurrentProcessLauncherAvailable	
DoneTransparentToolbars	
DXPShortcutToDelphiShortcut	
GetActivePLByCommand	
GetAllAvailableHotkeys	
GetFocusedPanelName	
GetPanelIsOpen	
GetPanelIsOpenInAnyForm	
GetPanelIsVisibleInAnyForm	
GetProcessLauncherInfoByID	
GetShortcutTextForPLID	
InitTransparentToolbars	
IsPanelValidInCurrentForm	
IsPanelVisibleInCurrentForm	
IsSysLevelHotKey	
LaunchCurrentHotkey	
ProcessMessage	
RegisterFloatingWindow	
ResizePanel	
SetFocusLock	
SetPanelActiveInCurrentForm	
SetPanelVisibleInCurrentForm	
ShowCurrentProcessLauncherHelp	
ShowTreeAsPopup	
StatusBar_GetState	
StatusBar_SetState	
UnregisterFloatingWindow	
UpdateInterfaceState	
UpdateTransparentToolbars	

See also

IGUIManager Methods

AddKeyStrokeAndLaunch method

(IGUIManager interface)

Syntax

```
Function AddKeyStrokeAndLaunch (AKey : Word) : LongBool;
```

Description

Example

See also

IGUIManager interface

AddKeyToBuffer method

(IGUIManager interface)

Syntax

```
Function AddKeyToBuffer (KeyId : Integer;Alt, Shift, Ctrl : LongBool) : LongBool;
```

Description

Example

See also

IGUIManager interface

BeginDragDrop method

(IGUIManager interface)

Syntax

```
Procedure BeginDragDrop (ADragDropInfo : IDragDropObject);
```

Description

Example

See also

IGUIManager interface

CanResizePanel method

(IGUIManager interface)

Syntax

```
Function CanResizePanel (Const AViewName : Widestring) : LongBool;
```

Description

This function determines whether the panel can be resized or not. The name of the panel need to be supplied.

Example

See also

IGUIManager interface

CurrentProcessLauncherAvailable method

(IGUIManager interface)

Syntax

```
Function CurrentProcessLauncherAvailable : LongBool;
```

Description

This function determines whether the current process launcher is available or not to use.

Example

See also

IGUIManager interface

DoneTransparentToolbars method

(IGUIManager interface)

Syntax

```
Procedure DoneTransparentToolbars;
```

Description

Example

See also

IGUIManager interface

GetActivePLByCommand method

(IGUIManager interface)

Syntax

```
Function GetActivePLByCommand (Const DocumentKind, ACommand, AParams : Widestring) :
IProcessLauncherInfo;
```

Description

Example

See also

IGUIManager interface

GetFocusedPanelName method

(IGUIManager interface)

Syntax

```
Function GetFocusedPanelName : Widestring;
```

Description

Example

See also

IGUIManager interface

GetPanelIsOpen method

(IGUIManager interface)

Syntax

```
Function GetPanelIsOpen (Const AViewName : Widestring) : LongBool;
```

Description

Example

See also

IGUIManager interface

GetProcessLauncherInfoByID method

(IGUIManager interface)

Syntax

```
Function GetProcessLauncherInfoByID (Const PLID : Widestring) : IProcessLauncherInfo;
```

Description

Example

See also

IGUIManager interface

InitTransparentToolbars method

(IGUIManager interface)

Syntax

```
Procedure InitTransparentToolbars (Const ViewRect : TRect);
```

Description

Example

See also

IGUIManager interface

IsPanelValidInCurrentForm method

(IGUIManager interface)

Syntax

```
Function IsPanelValidInCurrentForm (Const AViewName : Widestring) : LongBool;
```

Description

Example

See also

IGUIManager interface

IsPanelVisibleInCurrentForm method

(IGUIManager interface)

Syntax

```
Function IsPanelVisibleInCurrentForm (Const AViewName : Widestring) : LongBool;
```

Description

Example

See also

IGUIManager interface

IsSysLevelHotKey method

(IGUIManager interface)

Syntax

```
Function IsSysLevelHotKey (KeyId : Integer; Alt, Shift, Ctrl : LongBool): LongBool;
```

Description

Example

See also

IGUIManager interface

LaunchCurrentHotkey method

(IGUIManager interface)

Syntax

```
Procedure LaunchCurrentHotkey;
```

Description

Example

See also

IGUIManager interface

ProcessMessage method

(IGUIManager interface)

Syntax

```
Function ProcessMessage (Var Msg : TMessage) : LongBool;
```

Description

Example

See also

IGUIManager interface

RegisterFloatingWindow method

(IGUIManager interface)

Syntax

```
Procedure RegisterFloatingWindow (Const FloatingWindow : IFloatingWindow);
```

Description

Example

See also

IGUIManager interface

ResizePanel method

(IGUIManager interface)

Syntax

```
Procedure ResizePanel (Const AViewName : Widestring; AWidth, AHeight : Integer);
```

Description

Example

See also

IGUIManager interface

SetFocusLock method

(IGUIManager interface)

Syntax

```
Procedure SetFocusLock (Locked : LongBool);
```

Description

Example

See also

IGUIManager interface

SetPanelActiveInCurrentForm method

(IGUIManager interface)

Syntax

```
Procedure SetPanelActiveInCurrentForm (Const AViewName : Widestring);
```

Description

Example

See also

IGUIManager interface

SetPanelVisibleInCurrentForm method

(IGUIManager interface)

Syntax

```
Procedure SetPanelVisibleInCurrentForm (Const AViewName : Widestring; IsVisible : LongBool);
```

Description

Example

See also

IGUIManager interface

ShowCurrentProcessLauncherHelp method

(IGUIManager interface)

Syntax

```
Function ShowCurrentProcessLauncherHelp : LongBool;
```

Description

Example

See also

IGUIManager interface

ShowTreeAsPopup method

(IGUIManager interface)

Syntax

```
Procedure ShowTreeAsPopup (Const TreeID : Widestring);
```

Description

Example

See also

IGUIManager interface

StatusBar_GetState method

(IGUIManager interface)

Syntax

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

Description

Example

See also

IGUIManager interface

StatusBar_SetState method

(IGUIManager interface)

Syntax

```
Procedure StatusBar_SetState (Index : Integer; Const S : Widestring);
```

Description

Example

See also

IGUIManager interface

UnregisterFloatingWindow method

(IGUIManager interface)

Syntax

```
Procedure UnregisterFloatingWindow (Const FloatingWindow : IFloatingWindow);
```

Description

Example

See also

IGUIManager interface

UpdateInterfaceState method

(IGUIManager interface)

Syntax

```
Procedure UpdateInterfaceState;
```

Description

Example

See also

IGUIManager interface

UpdateTransparentToolbars method

(IGUIManager interface)

Syntax

```
Procedure UpdateTransparentToolbars;
```

Description

Example

See also

IGUIManager interface

INavigationSystem Interface

Overview

The navigation system is the workhouse for the Navigation panel which is the center-piece for net connectivity for the design project. There are three ways a design can be arranged - as a list of compiled sheets, flattened hierarchy and as a structural tree.

INavigationSystem Methods and Properties Table

INavigationSystem methods	INavigationSystem properties
RegisterNavigationProvider	
UnregisterNavigationProtocol	
RegisterSpecialURLString	
UnregisterSpecialURLString	
ParseDestinationString	
NavigateTo	
ExpandTargets	
ValidatedTarget	

See also

IClient interface

INavigationSystem Methods

UnregisterNavigationProtocol method

(INavigationSystem interface)

Syntax

```
Procedure UnregisterNavigationProtocol(Const Protocol : WideString; Handle : THandle);
```

Description

Example

See also

INavigationSystem interface

RegisterSpecialURLString method

(INavigationSystem interface)

Syntax

```
Procedure RegisterSpecialURLString (Const SpecialString : WideString; SpecialStringFunc :
TSpecialStringFunc);
```

Description

Example

See also

INavigationSystem interface

RegisterNavigationProvider method

(INavigationSystem interface)

Syntax

```
Function RegisterNavigationProvider (Const ProtocolName : WideString; Const NavigationProvider
: INavigationProvider) : THandle;
```

Description

Example

See also

INavigationSystem interface

ParseDestinationString method

(INavigationSystem interface)

Syntax

```
Procedure ParseDestinationString(Const Destination : WideString; Var Protocol, Target,
Parameters : WideString);
```

Description

Example

See also

INavigationSystem interface

NavigateTo method

(INavigationSystem interface)

Syntax

```
Function NavigateTo (Const CurrentView : IExternalForm; Var Destination : WideString; Out
TargetView : IExternalForm) : LongBool;
```

Description

Example

See also

INavigationSystem interface

ExpandTargets method

(INavigationSystem interface)

Syntax

```
Procedure ExpandTargets (Var Target : WideString);
```

Description

Example

See also

INavigationSystem interface

ValidatedTarget method

(INavigationSystem interface)

Syntax

```
Function ValidatedTarget ( Target : WideString) : WideString;
```

Description

Example

See also

INavigationSystem interface

UnregisterSpecialURLString method

(INavigationSystem interface)

Syntax

```
Procedure UnregisterSpecialURLString (Const SpecialString : WideString; SpecialStringFunc :
TSpecialStringFunc);
```

Description

Example

See also

INavigationSystem interface

INotification Interface

Overview

The `INotification` interface is used by the IClient, IServerView, IServerDocument, IServerModule, INotificationHandler interfaces.

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.

Different types of notifications

1. DocumentNotification
2. ViewNotification
3. DocumentFormNotification
4. ModuleNotification
5. SystemNotification
6. MessagesNotification
7. DragDropNotification
8. FastCrossSelectNotification

Setting up notifications in a server project,

1. Override the `ReceiveNotifications` method in the `TServerModule` class to handle and process different notifications.
2. Define different notification handlers.
3. Process each handler based on the Code property of each notification.

Example

```
Procedure TNotificationModule.ReceiveNotification(Const ANotification: INotification);
Var
    DocumentNotification : IDocumentNotification;
    ViewNotification     : IViewNotification;
    FormNotification     : IDocumentFormNotification;
    ModuleNotification   : IModuleNotification;
    SystemNotification   : ISystemNotification;
Begin
    If Supports(ANotification, IDocumentNotification, DocumentNotification) Then
           HandleDocumentNotification(DocumentNotification);

    If Supports(ANotification, IViewNotification, ViewNotification) Then
           HandleViewNotification(ViewNotification);

    If Supports(ANotification, IDocumentFormNotification, FormNotification) Then
           HandleFormNotification(FormNotification);

    If Supports(ANotification, IModuleNotification, ModuleNotification) Then
           HandleModuleNotification(ModuleNotification);

    If Supports(ANotification, ISystemNotification, SystemNotification) Then
           HandleSystemNotification(SystemNotification);
```

```
End;
```

The `INotification` interface hierarchy is as follows;

INotification

IDocumentNotification

IViewNotification

IDocumentFormNotification

IModuleNotification

ISystemNotification

IMessageNotification

IDragDropNotification

IDocumentRequest

IFastCrossNotification

INotification methods	**INotification properties**

See also

IClient Interface

IServerView interface

IServerDocument interface

IServerModule interface

INotificationHandler interface

IDocumentNotification interface

IViewNotification interface

IDocumentFormNotification interface

IModuleNotification interface

ISystemNotification interface

IMessageNotification interface

IDragDropNotification interface

IDocumentRequest interface

IFastCrossNotification interface

IDocumentFormNotification Interface

(IDocumentFormNotification interface)

Overview

Description

Example

See also

IClient interface

IExternalForm interface

ISystemNotification Interface

(ISystemNotification interface)

Syntax

Description

Example

See also

IClient interface

IExternalForm interface

IMessagesNotification Interface

Overview

The IMessagesNotification interface

IMessagesNotification methods

IMessagesNotification properties

`Code`

See also

IClient interface

IExternalForm interface

IModuleNotification Interface

Overview

See also

IClient interface

IExternalForm interface

IViewNotification Interface

Overview

Description

Example

See also

IClient interface

IExternalForm interface

IDragDropNotification Interface

Overview

Notes

Inherited from INotification interface.

IDragDropNotification Methods	**IDragDropNotification Properties**
`GetCode`	
`GetDragDropObject`	

See also

IDragDropObject interface

IEventNavigated Interface

Overview

IEventNavigated Methods	**IEventNavigated Properties**
`GetCode`	Code
`GetWnd`	Wnd

See also

IDragDropObject interface

INavigationProvider Interface

Overview

INavigationProvider Methods	**INavigationProvider Properties**
`NavigateTo`	

See also

IDragDropObject interface

INavigator Interface

Overview

INavigator Methods	**INavigator Properties**
`NavigateTo`	

See also

IBackForwardNavigator Interface

Overview

IBackForwardNavigator Methods	**IBackForwardNavigator Properties**
`GetAddress : WideString;`	Address
`GetCaption : WideString;`	Caption
`GetBackwardHistoryCount`	
`GetBackwardHistoryAddress`	
`GetBackwardHistoryCaption`	

```
MoveBackward

GetForwardHistoryCount
GetForwardHistoryAddress
GetForwardHistoryCaption
MoveForward
```

See also

INavigationSystem Interface

Overview

INavigationSystem Methods	**INavigationSystem Properties**
`RegisterNavigationProvider`	
`UnregisterNavigationProtocol`	
`RegisterSpecialURLString`	
`UnregisterSpecialURLString`	
`ParseDestinationString`	
`NavigateTo`	
`ExpandTargets`	
`ValidatedTarget`	

See also

IDragDropObject interface

INavigateAttributes Interface

Overview

INavigateAttributes Methods	**INavigateAttributes Properties**
`GetAddress :`	Address
`GetCaption :`	Caption
`IsSameAddress`	

See also

IDynamicHelpManager Interface

Overview

This interface represents the Knowledge Center panel in Altium Designer. This interface is part of the IClient interface.

IDynamicHelpManager Methods	**IDynamicHelpManager Properties**
`AddCustomContent`	

`RemoveCustomContent`

`GetCustomSectionName`
`GetCustomSectionBody`
`GetCustomSectionsCount`

See also

IClient interface

IFastCrossSelectNotification Interface

Overview

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

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

Overview

Description

Example

See also

IClient interface

INotification interface

INotificationHandler Interface

Overview

The `INotificationHandler` interface handles notifications broadcasted in the Altium Designer 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.

To register a Notification handler in the server project (either in a server module object, panel view object or document view object)

1. When a object is created, the Client.RegisterNotificationHandler is invoked.

2. When a object is destroyed, the Client.UnregisterNotificationHandler is invoked.

3. To handle custom notifications, a object has a HandlerNotification method which checks if the custom notification code is intercepted then a call can be made to update for example the Panel's preferences controls.

The INotificationHandler is inherited in the TServerModule, TServerDocumentForm and TServerPanelForm classes and thus custom notifications can be registered and handled.

INotificationHandler methods

HandleNotification

See also

IClient interface

INotificationHandler Methods

HandleNotification

(INotificationHandler interface)

Syntax

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

Description

Example

See also

IClient interface

IProcessLauncher Interface

Overview

This `IProcessLauncher` interface is a mechanism that launches a server process in Altium Designer. 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 and Properties Table

IProcessLauncher methods

`PostMessage`

`SendMessage`

`GetCommandState`

See also

ICommandLauncher interface

IClient interface

IServerProcess interface

ICommandLauncher interface

IProcessLauncherInfo Interface

Overview

The `IProcessLauncherInfo` interface hierarchy is as follows;

IProcessLauncherInfo Methods and Properties Table

IProcessLauncherInfo methods	IProcessLauncherInfo properties
GetCaption	Caption
GetParameters	Parameters
GetDescription	Description
GetImageFile	ImageFile
GetKey	Key
GetShift	Shift
GetKey2	Key2
GetShift2	Shift2
GetServerCommand	ShortcutText
GetShortcutText	ServerCommand

See also

IProcessLauncherInfo Methods

GetCaption method

(IProcessLauncherInfo interface)

Syntax

```
Function GetCaption : Widestring;
```

Description

Example

See also

IProcessLauncherInfo interface

GetDescription method

(IProcessLauncherInfo interface)

Syntax

```
Function GetDescription : Widestring;
```

Description

Example

See also

IProcessLauncherInfo interface

GetImageFile method

(IProcessLauncherInfo interface)

Syntax

```
Function GetImageFile : Widestring;
```

Description

Example

See also

IProcessLauncherInfo interface

GetKey method

(IProcessLauncherInfo interface)

Syntax

```
Function GetKey : Integer;
```

Description

Example

See also

IProcessLauncherInfo interface

GetKey2 method

(IProcessLauncherInfo interface)

Syntax

```
Function GetKey2 : Integer;
```

Description

Example

See also

IProcessLauncherInfo interface

GetParameters method

(IProcessLauncherInfo interface)

Syntax

```
Function GetParameters : Widestring;
```

Description

Example

See also

IProcessLauncherInfo interface

GetServerCommand method

(IProcessLauncherInfo interface)

Syntax

```
Function GetServerCommand : Widestring;
```

Description

Example

See also

IProcessLauncherInfo interface

GetShift method

(IProcessLauncherInfo interface)

Syntax

```
Function GetShift : TShiftState;
```

Description

Example

See also

IProcessLauncherInfo interface

GetShift2 method

(IProcessLauncherInfo interface)

Syntax

```
Function GetShift2 : TShiftState;
```

Description

Example

See also

IProcessLauncherInfo interface

GetShortcutText method

(IProcessLauncherInfo interface)

Syntax

```
Function GetShortcutText : Widestring;
```

Description

Example

See also

IProcessLauncherInfo interface

IProcessLauncherInfo Properties

Caption property

(IProcessLauncherInfo interface)

Syntax

```
Property Caption : Widestring Read GetCaption ;
```

Description

Example

See also

IProcessLauncherInfo interface

Description property

(IProcessLauncherInfo interface)

Syntax

```
Property Description : Widestring Read GetDescription ;
```

Description

Example

See also

IProcessLauncherInfo interface

ImageFile property

(IProcessLauncherInfo interface)

Syntax

```
Property ImageFile : Widestring Read GetImageFile ;
```

Description

Example

See also

IProcessLauncherInfo interface

Key property

(IProcessLauncherInfo interface)

Syntax

```
Property Key : Integer Read GetKey ;
```

Description

Example

See also

IProcessLauncherInfo interface

Key2 property

(IProcessLauncherInfo interface)

Syntax

```
Property Key2 : Integer Read GetKey2 ;
```

Description

Example

See also

IProcessLauncherInfo interface

Parameters property

(IProcessLauncherInfo interface)

Syntax

```
Property Parameters : Widestring Read GetParameters ;
```

Description

Example

See also

IProcessLauncherInfo interface

ServerCommand property

(IProcessLauncherInfo interface)

Syntax

```
Property ServerCommand : Widestring Read GetServerCommand;
```

Description

Example

See also

IProcessLauncherInfo interface

Shift property

(IProcessLauncherInfo interface)

Syntax

```
Property Shift : TShiftState Read GetShift ;
```

Description

Example

See also

IProcessLauncherInfo interface

Shift2 property

(IProcessLauncherInfo interface)

Syntax

```
Property Shift2 : TShiftState Read GetShift2 ;
```

Description

Example

See also

IProcessLauncherInfo interface

ShortcutText property

(IProcessLauncherInfo interface)

Syntax

```
Property ShortcutText : Widestring Read GetShortcutText ;
```

Description

Example

See also

IProcessLauncherInfo interface

IProcessControl Interface

Overview

The `IProcessControl` interface controls the process depth for each design document in Altium Designer. 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.

Process Depths for Schematic and PCB documents

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	**IProcessControl Properties**
`PostProcess`	`ProcessDepth`
`PreProcess`	

See also

IPCB_ServerInterface for PostProcess and PreProcess methods

ISch_ServerInterface for PostProcess and PreProcess methods

IProcessControl Methods

PostProcess method

(IProcessControl interface)

Syntax

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

Description

This procedure performs a post processing within in a main server which could involve finalizing the states of the environment of the server such as the Undo system. The AContext parameter is usually the focussed document in Altium Designer 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

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 Altium Designer 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

PostProcess method

IProcessControl Properties

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));
```

ILicenseManager Interface

Overview

The **ILicenseManager** interface hierarchy is as follows;

ILicenseManager methods	ILicenseManager properties
UseLicense	
ReleaseLicense	
ChangeToNetwork	
ChangeToStandalone	
UseLicenseByName	
GetLicenses	

See also

ILicenseManager Methods

UseLicense method

(ILicenseManager interface)

Syntax

```
Procedure UseLicense (Const LicenseFileName : Widestring);
```

Description

Example

See also

ILicenseManager interface

ReleaseLicense method

(ILicenseManager interface)

Syntax

```
Procedure ReleaseLicense (Const LicenseFileName : Widestring);
```

Description

Example

See also

ILicenseManager interface

GetLicenses method

(ILicenseManager interface)

Syntax

```
Procedure GetLicenses (Licenses : TList);
```

Description

Example

See also

ILicenseManager interface

ChangeToStandalone method

(ILicenseManager interface)

Syntax

```
Procedure ChangeToStandalone;
```

Description

This procedure changes from a networked license to a standalone license for a copy of Altium Designer that's running on a computer. A standalone computer is a computer that is not connected to the internet.

Example

See also

ILicenseManager interface

ChangeToNetwork method

(ILicenseManager interface)

Syntax

```
Procedure ChangeToNetwork (Const ServerName : Widestring);
```

Description

This procedure changes from a standalone license to a networked license for a copy of Altium Designer that's running on a computer. You will need to supply the server name as a string.

A standalone computer is a computer that is not connected to the internet.

Example

See also

ILicenseManager interface

UseLicenseByName method

(ILicenseManager interface)

Syntax

```
Procedure UseLicenseByName (Const LicenseName : Widestring);
```

Description

Example

See also

ILicenseManager interface

IOptionsManager Interface

Overview

The `IOptionsManager` interface deals with the options of a system wide Preferences dialog or project centric Project Options dialog.

Notes

A server needs to register its own options pages within the Client module of Altium Designer. The `TServerModule` class from the `RT_ServerImplementation` unit within the Altium Designer RTL has a `RegisterOptionsPageClass` procedure for you to override. You need to pass in the name of the options page and the Options Form of `TOptionsForm` type. Normally this form is the same as the server panel form with the controls on it.

IOptionsManager methods	**IOptionsManager properties**
GetOptionsReader	
GetOptionsWriter	
OptionsExist	

Example

```
Procedure TGraphicPreferences.Save;
Var
    Writer : IOptionsWriter;
Begin
    Writer := Client.OptionsManager.GetOptionsWriter(CGraphicViewer);
    If Writer = Nil Then Exit;
    Writer.WriteBoolean(cGraphicPreferences, 'ScaleImage'     , FScaleImage     );
    Writer.WriteBoolean(cGraphicPreferences, 'KeepAspectRatio', FKeepAspectRatio);
End;
```

See also

IOptionsReader interface

IOptionsWriter interface

IOptionsPage interface

GraphicViewer server project from \Developer Kit\Examples\Dxp\GraphicViewer folder

IOptionsManager Methods

OptionsExist method

(IOptionsManager interface)

Syntax

```
Function OptionsExist (Const ServerName, OldSettingsPath : WideString) : LongBool;
```

Description

This function checks if the options for the specified server exist on the system wide Preference dialog.

Example

See also

IOptionsManager interface

GetOptionsWriter method

(IOptionsManager interface)

Syntax

```
Function GetOptionsWriter (Const ServerName : WideString) : IOptionsWriter;
```

Description

This function retrieves the IOptionsWriter method which enables you to write setting values for the Options of the specified server.

Example

```
Var
```

```
    Writer : IOptionsWriter;
Begin
    Writer := Client.OptionsManager.GetOptionsWriter(CGraphicViewer);
    If Writer = Nil Then Exit;

    Writer.WriteBoolean(PreferencesName, OptionName , OptionValue);
End;
```

See also

IOptionsManager interface

IOptionsWriter interface

IOptionsReader interface

GetOptionsReader method

(IOptionsManager interface)

Syntax

```
Function GetOptionsReader (Const ServerName, OldSettingsPath : WideString) : IOptionsReader;
```

Description

This function retrieves the `IOptionsReader` method which enables you to read setting values for the Options of the specified server.

Example

```
Var
    Reader : IOptionsReader;
Begin
    Reader := Client.OptionsManager.GetOptionsReader(NameOfServer,'');
    If Reader = Nil Then Exit;

    OptionValue := Reader.ReadBoolean(ServerPreferencesName,OptionName,DefaultValue);
End;
```

See also

IOptionsManager interface

IOptionsWriter interface

IOptionsReader interface

IOptionsReader Interface

Overview

The `IOptionsReader` interface reads values for options on a page in the system wide Preferences dialog or Project options dialog from the registry storage.

IOptionsReader methods	IOptionsReader properties
ReadBoolean	
ReadDouble	
ReadInteger	
ReadString	
ReadSection	
SectionExists	
ValueExists	

Example

```
Var
    Reader : IOptionsReader;
Begin
    Reader := Client.OptionsManager.GetOptionsReader(NameOfServer,'');
    If Reader = Nil Then Exit;

    AValue := Reader.ReadBoolean(NameOfServerPreferences,SettingName,DefaultValue);
End;
```

See also

IClient interface

IOptionsManager interface

IOptionsReader Methods

ValueExists method

(IOptionsReader interface)

Syntax

```
Function ValueExists (Const SectionName, ValueName : WideString) : LongBool;
```

Description

This function determines whether the value name exists for this section name. This is useful if you need to check if a value name exists in the registry storage before you commit a value to this location.

The section name is the targetted page in the system wide preferences dialog.

Example

See also

IOptionsReader interface

SectionExists method

(IOptionsReader interface)

Syntax

```
Function SectionExists(Const SectionName : WideString) : LongBool;
```

Description

This function checks whether the section (or the targetted page) exists or not.

The section name is the targetted page in the system wide preferences dialog.

Example

See also

IOptionsReader interface

ReadString method

(IOptionsReader interface)

Syntax

```
Function ReadString (Const SectionName, ValueName, DefaultValue : WideString) : WideString;
```

Description

This `ReadString` function retrieves a string value for the specified server and the setting name that are represented by the system wide Preferences dialog.

The section name is the targetted page in the system wide preferences dialog.

Example

See also

IOptionsReader interface

ReadSection method

(IOptionsReader interface)

Syntax

```
Function ReadSection (Const SectionName : WideString) : WideString;
```

Description

This function retrieves the data for the section which is the targetted page in the system wide Preferences dialog.

Note the section name is the targetted page in the system wide preferences dialog.

Example

See also

IOptionsReader interface

ReadInteger method

(IOptionsReader interface)

Syntax

```
Function ReadInteger (Const SectionName, ValueName : WideString; DefaultValue : Integer) :
Integer;
```

Description

This `ReadInteger` function retrieves an integral value for the specified server and the setting name that are represented by the system wide Preferences dialog.

The section name is the targetted page in the system wide preferences dialog.

Example

See also

IOptionsReader interface

ReadDouble method

(IOptionsReader interface)

Syntax

```
Function ReadDouble (Const SectionName, ValueName : WideString; DefaultValue : Double) :
Double;
```

Description

This `ReadDouble` function retrieves a double value for the specified server and the setting name that are represented by the system wide Preferences dialog.

The section name is the targetted page in the system wide preferences dialog.

Example

See also

IOptionsReader interface

ReadBoolean method

(IOptionsReader interface)

Syntax

```
Function ReadBoolean (Const SectionName, ValueName : WideString; DefaultValue : LongBool) :
LongBool;
```

Description

This `ReadBoolean` function retrieves a boolean value for the specified server and the setting name that are represented by the system wide Preferences dialog.

The `DefaultValue` parameter for the `ReadBoolean` method returns a default Boolean value if the specific control on the Preferences dialog is not returning a valid Boolean value.

The section name represents the target server's page in the system wide preferences dialog.

Example

```
Var
    Reader : IOptionsReader;
Begin
    Reader := Client.OptionsManager.GetOptionsReader(NameOfServer,'');
    If Reader = Nil Then Exit;

    AValue := Reader.ReadBoolean(NameOfServerPreferences,SettingName,DefaultValue);
End;
```

See also

IOptionsReader interface

IOptionsWriter Interface

Overview

The `IOptionsWriter` interface writes values for options on a page in the system wide Preferences or Project options dialog to a registry storage.

IOptionsWriter methods	IOptionsWriter properties
EraseSection	
WriteBoolean	
WriteDouble	
WriteInteger	
WriteString	

Example

```
Var
    Writer : IOptionsWriter;
Begin
    Writer := Client.OptionsManager.GetOptionsWriter(CGraphicViewer);
    If Writer = Nil Then Exit;
    Writer.WriteBoolean(cGraphicPreferences, 'ScaleImage'     , FScaleImage     );
    Writer.WriteBoolean(cGraphicPreferences, 'KeepAspectRatio', FKeepAspectRatio);
End;
```

See also

`IClient interface`

IOptionsManager interface

IOptionsWriter Methods

EraseSection method

(IOptionsWriter interface)

Syntax

```
Procedure EraseSection(Const SectionName : WideString);
```

Description

This procedure removes all the option values for the section (targetted page in the system wide preferences dialog).

Example

See also

IOptionsWriter interface

WriteInteger method

(IOptionsWriter interface)

Syntax

```
Procedure WriteInteger(Const SectionName, ValueName : WideString; Value : Integer);
```

Description

This `WriteInteger` procedure writes an integral value for the option name used by the specified server (SectionName) which is represented by the system wide Preferences dialog.

The section name is the targetted page in the system wide preferences dialog.

Example

See also

IOptionsWriter interface

WriteDouble method

(IOptionsWriter interface)

Syntax

```
Procedure WriteDouble (Const SectionName, ValueName : WideString; Value : Double);
```

Description

This `WriteDouble` procedure writes a double value for the option name used by the specified server (SectionName) which is represented by the system wide Preferences dialog.

The section name is the targetted page in the system wide preferences dialog.

Example

See also

IOptionsWriter interface

WriteBoolean method

(IOptionsWriter interface)

Syntax

```
Procedure WriteBoolean(Const SectionName, ValueName : WideString; Value : LongBool);
```

Description

This `WriteBoolean` procedure writes a boolean option value for the option name used by the specified server (SectionName) which is represented by the system wide Preferences dialog.

The section name is the targetted page in the system wide preferences dialog.

Example

```
Var
    Writer : IOptionsWriter;
Begin
    Writer := Client.OptionsManager.GetOptionsWriter(CGraphicViewer);
    If Writer = Nil Then Exit;

    Writer.WriteBoolean(cGraphicPreferences, 'ScaleImage'     , FScaleImage     );
    Writer.WriteBoolean(cGraphicPreferences, 'KeepAspectRatio', FKeepAspectRatio);
End;
```

See also

IOptionsWriter interface

WriteString method

(IOptionsWriter interface)

Syntax

```
Procedure WriteString (Const SectionName, ValueName, Value : WideString);
```

Description

This `WriteString` procedure writes a string option value for the option name used by the specified server (SectionName) which is represented by the system wide Preferences dialog.

The section name is the targetted page in the system wide preferences dialog.

Example

See also

IOptionsWriter interface

IOptionsPage Interface

Overview

The `IOptionsPage` interface represents the page of controls in the system wide Preferences dialog. For example, in Altium Designer, the controls on this page in the Preferences dialog are mapped from the controls on a server panel of this server. The controls on a page is represented by the `TOptionsForm` object and its `IOptionsPage` interface.

Note

The server module (`TServerModule` class) has the `RegisterOptionsPageClass` method which takes in the `TOptionsForm` object. The `IOptionsPage` interface represents this `TOptionsForm` object.

The TOptionsForm class has methods that you need to override to implement the OptionsPage, OptionsManager, OptionsReader and OptionsWriter interfaces.

ClearModified

GetModified

GetStateControls

GetNotificationCode

DoSetStateControls

SetDefaultState

IOptionsPage Methods and Properties table

IOptionsPage methods	**IOptionsPage properties**
GetModified	Modified
SetModified	
GetStateControls	
SetStateControls	
GetNotificationCode	
SetDefaultState	
PostEditControls	

Example

```
    TGraphicPrefsForm_General = Class(TOptionsForm)
        chxScale        : TCheckBox;
        chxProportional : TCheckBox;
    Private
        FScaleStored        : Boolean;
        FProportionalStored : Boolean;
    Protected
        Procedure ClearModified;                       Override;
        Function  GetModified : Boolean;               Override;
        Procedure GetStateControls;                    Override;
        Function  GetNotificationCode : Integer;       Override;
        Procedure DoSetStateControls;                  Override;
        Procedure SetDefaultState;                     Override;
    End;
{...........................................................................}
Function TGraphicPrefsForm_General.GetNotificationCode: Integer;
Begin
    Result := cGraphicPreferencesChanged;
End;
Procedure TGraphicPrefsForm_General.GetStateControls;
Begin
    gv_GraphicPreferences.ScaleImage      := chxScale       .Checked;
    gv_GraphicPreferences.KeepAspectRatio := chxProportional.Checked;
End;
Procedure TGraphicPrefsForm_General.DoSetStateControls;
Begin
    chxScale       .Checked := gv_GraphicPreferences.ScaleImage;
    chxProportional.Checked := gv_GraphicPreferences.KeepAspectRatio;
End;
Procedure TGraphicPrefsForm_General.SetDefaultState;
Begin
    chxScale       .Checked := False;
    chxProportional.Checked := False;
    Inherited;
End;
Procedure TGraphicPrefsForm_General.ClearModified;
Begin
    FScaleStored        := chxScale.Checked;
    FProportionalStored := chxProportional.Checked;
End;
Function TGraphicPrefsForm_General.GetModified : Boolean;
Begin
    Result := (FScaleStored <> chxScale.Checked) Or
              (FProportionalStored <> chxProportional.Checked);
End;
```

See also

IOptionsManager interface

IOptionsReader interface

`IOptionsWriter interface`

IOptionsPage GetState and SetState Methods

GetModified method

(IOptionsPage interface)

Syntax

```
Function GetModified : Boolean;
```

Description

Example

See also

IOptionsPage interface

SetModified method

(IOptionsPage interface)

Syntax

```
Procedure SetModified(Value : Boolean);
```

Description

Example

See also

IOptionsPage interface

IOptionsPage Methods

SetStateControls method

(IOptionsPage interface)

Syntax

```
Procedure SetStateControls;
```

Description

This procedure updates the controls on the form from a data structure in a server module.

Example

See also

IOptionsPage interface

SetDefaultState method

(IOptionsPage interface)

Syntax

```
Procedure SetDefaultState;
```

Description

This procedure sets the controls on a page within the system wide Preferences dialog to their default values.

Note

The `SetDefaultState` procedure is overridden in a server's `TOptionsForm` object.

Example

See also

IOptionsPage interface

PostEditControls method

(IOptionsPage interface)

Syntax

```
Procedure PostEditControls;
```

Description

Example

See also

IOptionsPage interface

GetStateControls method

(IOptionsPage interface)

Syntax

```
Procedure GetStateControls;
```

Description

This procedure

Note

Example

See also

IOptionsPage interface

GetNotificationCode method

(IOptionsPage interface)

Syntax

```
Function GetNotificationCode : Integer;
```

Description

Each server that handles Option notifications to its server panel and the system wide Preferences dialog in Altium Designer will have its own Notification code which could be a value of 100 upwards.

Note

A server module will have a `TOptionsForm` object registered and this object will have a `GetNotificationCode` function overridden. This server module will have its own notification code value. Ensure these notification codes are unique.

Example

See also

IOptionsPage interface

IOptionsPage Properties

Modified property

(IOptionsPage interface)

Syntax

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

Description

Example

See also

IOptionsPage interface

IServerProcess Interface

Overview

The `IServerProcess` interface returns information for commands (server processes) in a server installation file;

- the command name (GetOriginalID method)
- the long summary
- the number of parameters if any
- parameter names if any

The `IServerProcess` interface is an aggregate interface used within the `IServerRecord` interface.

Notes

A typical installation file structure is as follows

```
ClientInsFile 1.0
Server
    EditorName        = 'AddOn'
    EditorExePath     = 'AddOn.DLL'
    EditorDescription = 'A demonstratory AddOn module'
    Version           = 'Version 8.1.4.2763'
    Date              = '24-Dec-2004'
    HelpAboutInfo     = 'This software is protected by copyright law and international
treaties.'
    Copyright         = 'Copyright © Altium Limited 2004'
    Updates           = 'ADVPCB'
End
Command Name = 'CountPads'      LongSummary = 'Find how many pads on a PCB document' End
Command Name = 'RunAPCBProcess' LongSummary = 'Invoke a PCB process'                 End
```

IServerProcess Methods

```
GetOriginalId
GetLongSummary
GetParameter
GetParameterCount
```

IServerProcess Properties

Example

```
//ServerRecord is a IServerRecord interface
CommandCount := ServerRecord.GetCommandCount;
For J := 0 To CommandCount - 1 Do
Begin
     //ServerProcess is a IServerProcess interface
     ServerProcess := ServerRecord.GetCommand(J);
     ReportFile.Add('        Process #' + IntToStr(J + 1) + ' Name = '  +
     ServerProcess.GetOriginalId + ' LongSummary = ' + ServerProcess.GetLongSummary);

     ParameterCount := ServerProcess.GetParameterCount;
     For K := 0 To ParameterCount - 1 Do
         S := S + ServerProcess.GetParameter(K) + ', ';
     ReportFile.Add('        Parameters = ' + S);
End;
```

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), it will perform the task of zooming in on the currently active schematic sheet.

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

See also

IServerRecord interface

ServerProcessReport script in \Examples\Scripts\DXP\ folder

IServerProcess Methods

GetLongSummary method

(IServerProcess interface)

Syntax

```
Function GetLongSummary : WideString;
```

Description

The `GetLongSummary` function returns the Long Summary identifier string.

Example

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.

Example

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 `GetParameterCount` method. A server process can be parametric, and thus can have a number of parameters.

Example

See also

IClient interface

IServerProcess interface

GetParameterCount method

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.

Example

See also

IClient interface

IServerProcess interface

GetParameter method

IServerRecord Interface

Overview

This interface extracts the servers installation files information from the \System folder which has a list of server installation files. That is each server installation file (with an INS extension) correspond to a IServerRecord itnerface.

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 Altium Designer, 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
```

Example

```
PCB_SR := Client.GetServerRecordByName('PCB');
```

See also

IClient interface

IServerModule interface

CS server example in the \Developer Kit\Examples\DXP\ClientServer Interfaces\ folder.

IServerRecord 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.

Example

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

Example

See also

IServerRecord interface

GetCopyRight method

(IServerRecord interface)

Syntax

```
Function GetCopyRight : PChar;
```

Description

The method returns the copyright string.

Example

See also

IServerRecord interface

GetDescription method

(IServerRecord interface)

Syntax

```
Function GetDescription : PChar;
```

Description

The method returns the description string.

Example

See also

IServerRecord interface

GetExePath method

(IServerRecord interface)

Syntax

```
Function GetExePath : PChar;
```

Description

The method returns the path to the server file.

Example

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.

Example

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.

Example

See also

IServerRecord interface

GetInsPath method

(IServerRecord interface)

Syntax

```
Function GetInsPath : PChar;
```

Description

The method returns the path to the installation file.

Example

See also

IServerRecord interface

GetName method

(IServerRecord interface)

Syntax

```
Function GetName : PChar;
```

Description

The method returns the name of the server.

Example

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.

Example

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.

Example

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.

Example

See also

IServerRecord interface

GetRCSFilePath method

(IServerRecord interface)

Syntax

```
Function GetRCSFilePath : PChar;
```

Description

The method returns the path to the resources file.

Example

See also

IServerRecord interface

GetSystemExtension method

(IServerRecord interface)

Syntax

```
Function GetSystemExtension : LongBool;
```

Description

The method returns the file system extension string.

Example

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.

Example

```
RecordCount := Client.GetServerRecordCount;
For I := 0 to RecordCount - 1 Do
Begin
    // obtain details of the DXP.INS file
    ServerRecord := Client.GetServerRecord(I);
    If ServerRecord.GetName = 'Client' Then
    Begin
        Version := ServerRecord.GetVersion;
        Break;
    End;
End;
```

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.

Example

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.

Example

See also

IServerRecord interface

GetWindowKindCount method

(IServerRecord interface)

Syntax

```
Function GetWindowKindCount : Integer;
```

Description

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

Example

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.

Example

See also

IServerRecord interface

IServerWindowKind interface

IServerWindowKind Interface

Overview

This `IServerWindowKind` interface reports the type of a design document in Altium Designer 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 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.

Example

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.

Example

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.

Example

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.

Example

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.

Example

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.

Example

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.

Example

See also

IClient interface

IServerWindowKind interface

GetName method

(IServerWindowKind interface)

Syntax

```
Function GetName : Widestring;
```

Description

Returns the name of the window kind.

Example

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 Altium Designer.

Example

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.

Example

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.

Example

See also

IClient interface

IServerWindowKind interface

GetWindowKindClass

(IExternalForm interface)

Syntax

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

Description

The method returns the indexed window kind class.

Example

See also

IClient interface

IServerWindowKind interface

GetWIndowKindClassCount

(IServerWindowKind interface)

Syntax

```
Function GetWindowKindClassCount : Integer;
```

Description

The method returns the number of window kind classes.

Example

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 Altium Designer.

Example

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.

Example

See also

IClient interface

IServerWindowKind interface

IServerSecurity Interface

Overview

The `IServerSecurity` interface hierarchy is as follows;

IServerSecurity methods	IServerSecurity properties
IsTechnologySetSupported	

See also

IServerSecurity Methods

IsTechnologySetSupported method

(IServerSecurity interface)

Syntax

```
Function IsTechnologySetSupported (Const ATechnologySet : Widestring) : Boolean;
```

Description

Example

See also

IServerSecurity interface

ITimerManager Interface

Overview

The `ITimerManager` interface manages the timing mechanisms efficiently in Altium Designer which registers timer objects and calls them when used. Normally a Timer object needs a window to run and responds to `WM_Timer` messages. This is for internal use.

ITimerManager methods	ITimerManager Properties
AddHandler	
RemoveHandler	
GetHandlerEnabled	
SetHandlerEnabled	
SetGlobalEnabled	

See also

ITimerHandler interface

ITimerManager Methods

AddHandler method

(ITimerManager interface)

Syntax

```
Function  AddHandler(Const AHandler : ITimerHandler; AInterval : Cardinal; AEnabled : Boolean
= True) : DWord;
```

Description

Internal Use only

Example

See also

ITimerIManager interface

GetHandlerEnabled method

(ITimerManager interface)

Syntax

```
Function GetHandlerEnabled(ID : DWord) : Boolean;
```

Description

Internal Use only

Example

See also

ITimerManager interface

RemoveHandler method

(ITimerManager interface)

Syntax

```
Procedure RemoveHandler    (ID : DWord);
```

Description

Internal Use only

Example

See also

ITimerManager interface

SetGlobalEnabled method

(ITimerManager interface)

Syntax

```
Procedure SetGlobalEnabled (AEnabled : Boolean);
```

Description

Internal Use only

Example

See also

ITimerManager interface

SetHandlerEnabled method

(ITimerManager interface)

Syntax

```
Procedure SetHandlerEnabled(ID : DWord; AEnabled : Boolean);
```

Description

Internal Use only

Example

See also

ITimerManager interface

ITimerHandler Interface

Overview

Each timer object is represented by the ITimerHandler interface and all timer objects are managed by the `ITimerManager` interface.

This is for internal use.

ITimerHandler methods	ITimerHandler properties
HandleTimerEvent	

See also

ITimerManger interface

ITimerHandler Methods

HandleTimerEvent method

(ITimerHandler interface)

Syntax

```
Procedure HandleTimerEvent(ID : DWord);
```

Description

Example

See also

ITimerHandler interface

ITranslationManager Interface

Overview

The `ITranslationManager` interface deals with the installed locale languages for Altium Designer. The installed locale languages are Simplified Chinese, Japanese, German and French. The default locale is Standard English.

ITranslationManager methods	ITranslationManager properties
GetTranslated	
SetComponentToTranslate	
HasTranslationData	

See also

ITranslationManager Methods

GetTranslatedProperty method

(ITranslationManager interface)

Syntax

```
Function  GetTranslatedProperty(Const ComponentName, PropName : WideString; Out OutValue :
WideString) : LongBool;
```

Description

Example

See also

SetComponentToTranslate method

(ITranslationManager interface)

Syntax

```
Procedure SetComponentToTranslate(Const ComponentName : WideString);
```

Description

Example

See also

HasTranslationData method

(ITranslationManager interface)

Syntax

```
Function  HasTranslationData : LongBool;
```

Description

Example

See also

Client Enumerated Types

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

TCommandProc procedure type

Syntax

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

TDocumentsBarGrouping type

```
TDocumentsBarGrouping = (dbgNone, dbgByDocKind, dbgByProject);
```

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,eHighlight_Dim,eHighligh
t_Thicken, eHighlight_ZoomCursor);
```

THighlightMethodSet type

Syntax

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

TSnippetCreationMode type

```
TSnippetCreationMode = (eSnippetCreationBySelection, eSnippetCreationByUnionIndex);
```

TServerModuleFactory function type

Syntax

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

Client Constants

General constants

```
    cDXPHomePage = 'DXP://Home';
    cDXPProcess  = 'DXPProcess';
    cDXPDocument = 'DXPDoc';
    cViewNameParam = 'ViewName';
    cContextHelpDelimiter = '.';

{$IFDEF ALTIUMINTERNAL}
   cWebUpdate_DefaultURL            =
'http://intranet.altium.com.au/rd/AltiumDesigner6/Updates/';
{$ELSE}
   cWebUpdate_DefaultURL            = 'http://www.altium.com/webupdate/';
{$ENDIF}
   cWebUpdate_DefaultNetworkPath    = '';
   cWebUpdate_DefaultUseNetworkPath = False;
   cWebUpdate_DefaultCheckFrequency = wucfEveryDay;

    cWebUpdate_CheckFrequencyNames : Array[TWebUpdate_CheckFrequency] Of AnsiString =
    (
        'Never',
        'On Altium Designer startup',
        'Every day',
        'Every 3 days',
        'Every week',
        'Every 2 weeks',
        'Every month');
```

DocumentNotification 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;

cDocumentOwnershipChanged    = 300;
```

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;
cSchPreferencesChanged_UpdateStringsFont = 10;
cCustomDynamicHelpUpdated                = 11;
```

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;
```

Low Level Routines Reference

The section has run time library information derived from `ClientAPIReg`, `RT_Util` and `RT_Param` units from the Altium Designer RTL that can be used for scripts and for server development.

Scale Factor Table

T 10^{12}

G 10^{9}

M, Meg = 10^{6}

K 10^{3}

U 10^{-6}

N 10^{-9}

P 10^{-12}

F 10^{-15}

Constants

```
cMeasureUnitSuffixes : Array[TMeasureUnit] Of TDynamicString = ('mil', 'mm', 'in', 'cm',
'dxp', 'm');

cMeasureUnitConvert  : Array[TMeasureUnit, TMeasureUnit] Of Double =
(// to  mil           mm         in        cm          dxp         m
(1          , 2.54/100 , 1/1000  , 2.54/1000 , 1/10      , 2.54/100000), // from mils
(100/2.54   , 1        , 1/25.4  , 1/10      , 10/2.54   , 1/1000     ), // from mm
(1000       , 25.4     , 1       , 2.54      , 100       , 0.0254     ), // from in
(1000/2.54  , 10       , 1/2.54  , 1         , 100/2.54  , 1/100      ), // from cm
(10         , 2.54/10  , 1/100   , 2.54/100  , 1         , 2.54/10000 ), // from dxp
(100000/2.54, 1000     , 100/2.54, 100       , 10000/2.54, 1          )  // from m
);

cPaintColorModes : Array[TPaintColorMode] Of TDynamicString = ('FullColor', 'GrayScale',
'Monochrome');

  CaseSensitive   = True;
  CaseInSensitive = False;
  OrdNumOfZero    = 48;
  cDefThumbnailSizeX = 96;
  cDefThumbnailSizeY = 72;

   Delimiter       : Set of char = [#0,#39,',',' ',#10,#13,#9,'(',')'];
   StringDelimiter = #39;

  cm_Share_Compat      = $0;
  cm_Share_DenyRW      = $10;
  cm_Share_DenyW       = $20;
  cm_Share_DenyR       = $30;
  cm_Share_DenyN       = $40;
  cm_Access_ReadOnly  = $0;
  cm_Access_WriteOnly = $1;
  cm_Access_ReadWrite = $2;
  cm_NoInheritance    = $80; {A child process would not inherit file handle and mode}

  fe_NoAccessError                 = $0;
  fe_FunctionInvalid               = $1;
  fe_FileNotFound                  = $2;
  fe_PathNotFoundOrFileDoesntExist = $3;
  fe_NoHandleIsAvalible            = $4;
  fe_AccessIsDenied                = $5;
  fe_FileAccessCodeInvalid         = $C;

   FileExtension_Temp         = '$$$';

   cPathSeparator         = '\';

    cBooleanStrings : Array[False..True] Of TString = ('False','True');
```

Conversion Routines

```
Function GetPrevSettings_Count : Integer;
Function GetPrevSettings_Name                                (AIndex : Integer) :
TDynamicString;
Function GetPrevSettings_SpecialKey_SoftwareAltiumApp        (AIndex : Integer) :
TDynamicString;
Function GetPrevSettings_SpecialKey_SoftwareAltiumAppDXP     (AIndex : Integer) :
TDynamicString;
Function GetPrevSettings_SpecialFolder_AltiumApplicationData (AIndex : Integer) :
TDynamicString;

Function ConvertMeasureUnits(Const AValue : Double; FromUnit, ToUnit : TMeasureUnit) : Double;

Function StripMeasureUnits(Var S : TDynamicString; Var Value : Double; Var UsedUnits :
TMeasureUnit) : Boolean;
```

Enumerated Types

TAltShiftCtrlCombination

```
TAltShiftCtrlCombination = TShiftState;
```

TBoolean

```
TBoolean       = Boolean;
```

TBusKind

```
TBusKind           =
(eBusKindUndefined,eBusKindLowValueFirst,eBusKindHighValueFirst,eBusKindGeneric);
```

TByte

```
TByte          = Byte;
```

TChar

```
TChar  = Array[0..256] of Char;
```

The Char type is equivalent to AnsiChar. Because the implementation of Char is subject to change, it's a good idea to use the standard function SizeOf rather than a hard-coded constant when writing programs that may need to handle characters of different sizes. The TChar type can be used instead of a PChar.

Example

```
Var
  P : TChar;
Begin
    lResult := GetModuleFileName(HInstance,P,255)
....
End;
```

TDate

```
TDate = Record
    Year   : Word;
    Month  : Word;
    Day    : Word;
End;
```

TDouble

```
TDouble = Double;
```

TDynamicString

```
TDynamicString = AnsiString;
```

TExtended

```
TExtended      = Extended;
```

TFileFunction

(RT_FileUtil in Altium Designer RTL)

```
TFileFunction = Function(Path : TDynamicString) : Boolean Of Object;
```

THugeInt

```
THugeInt       = Comp;
```

TMatchFileNameKind

```
TMatchFileNameKind = (eMatchByPath,eMatchByFileName);
```

TPaintColorMode

```
TPaintColorMode    = (ePaintColorMode_FullColor, ePaintColorMode_GrayScale,
ePaintColorMode_Monochrome);
```

TMeasureUnit

```
TMeasureUnit = (cUnitMil, cUnitMM, cUnitIN, cUnitCM, cUnitAltium Designer, cUnitM);
```

TPaintScaleMode

```
TPaintScaleMode = (psmScreen, psmDefault, psmPrint);
```

TReal

```
TReal          = Single;
```

TString

```
TString = ShortString;
```

TTime

```
TTime = Record
    Hours        : Word;
    Minutes      : Word;
    Seconds      : Word;
    MilliSeconds : Word;
End;
```

TNonRefCountedInterfaceObject

```
TNonRefCountedInterfacedObject = Class(TObject, IInterface)
  Protected
    FRefCount : Integer;
    Function    QueryInterface(Const IID: TGUID; Out Obj): HResult; StdCall;
    Function    _AddRef: Integer;                                   StdCall;
    Function    _Release: Integer;                                  StdCall;
End;
```

Dialogs

ConfirmOkCancel

(RT_Util unit)

Declaration

```
Function  ConfirmOKCancel (S : TDynamicString) : Boolean;
```

Description

The ConfirmOkCancel function displays a dialog with the S parameter for the message body of the dialog. This function returns a Boolean value. Since there are ‘OK’ and ‘Cancel’ buttons, if you pressed the OK button, the functions returns a true value, otherwise the function returns a false value

See also

ConfirmNoYes, ShowError, ShowInfo, ShowWarning procedures.

ConfirmOkCancelWithCaption

(RT_Util unit)

Declaration

```
Function  ConfirmOKCancelWithCaption   (Caption, S : TDynamicString) : Boolean;
```

Description

The ConfirmOkCancelWithCaption 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. This function returns a Boolean value. Since there are ‘OK’ and ‘Cancel’ buttons, if you pressed the OK button, the functions returns a true value, otherwise the function returns a false value

See also

ConfirmNoYes, ShowError, ShowInfo, ShowWarning procedures.

ConfirmNoYes

(ClientAPIReg unit)

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

ConfirmNoYesCancel

(ClientAPIReg)

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.

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.

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 a False value is returned, or when the user chose the Yes button, a True value is returned

See also

ConfirmNoYes, ShowError, ShowInfo, ShowWarning procedures.

SortedListBoxCompare

(IRT_Util unit from Altium Designer RTL)

Declaration

```
Function SortedListBoxCompare(Const S1, S2 : AnsiString) : Integer;
```

Description

This function has its internal sorting routine that sorts lists alphanumerically. Delphi’s sort can only provide digital or alphabet sorting only. You will need to invoke the CustomSort routine for a TStringList or other Delphi equivalent string lists and pass this function into this CustomSort routine.

Example

See also

DisplayNotImplementedMessage

(RT_Util unit in Altium Designer RTL)

Declaration

```
Procedure DisplayNotImplementedMessage(ProcessId,ProcessDescription : TDynamicString);
```

Description

This procedure displays a dialog with the Server Process not Implemented Message for server projects. This is used in the commands unit of a server project.

See also

ShowInfo and ShowWarning procedures.

RunNetworkConnectionDialog

(Rt_Util from Altium Designer RTL)

Syntax

Procedure RunNetWorkPrintersDialog(HWindow : Hwnd);

Description

This procedure invokes the Network Printers dialog with the handle of the current dialog or window in Altium Designer.

Example

See also

RunNetworkPrintersDialog

(Rt_Util from Altium Designer RTL)

Syntax

Procedure RunNetWorkConnectionDialog(HWindow : Hwnd);

Description

This procedure invokes the Network Connection dialog with the handle of the current dialog or window in ALTIUM DESIGNER.

Example

See also

RunOpenDocumentDialog

(RT_Util from Altium Designer RTL)

Syntax

```
Function  RunOpenDocumentDialog (Caption : TDynamicString; MultiSelect : Boolean; Var Path,
SelectedType, Editor : TDynamicString;  Const FileTypes, Files : TStrings) : Boolean;
```

Description

This function is based on the Client's `RunCommonDialog` process. The `Caption` parameter is used for the Title of the dialog. The `MultiSelect` parameter allows you to select files from the dialog if True. If you want to only select one file use the `False` value. The `Path`, `SelectedType` and `Editor` parameters are returned after the dialog has closed. FileTypes and Files parameters determine which file types and files can be opened by the Common Dialog.

Example

See also

ShowError

(ClientAPIReg unit in Altium Designer RTL)

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.

ShowError_SystemModal

(RT_Util unit from Altium Designer RTL)

Syntax

```
Procedure ShowError_SystemModal(Const S : TDynamicString);
```

Description

The `ShowError_SystemModal` procedure displays an independent dialog with an error symbol and string, S, for the text. This dialog does not have the Altium Designer's window handle and thus appears on the taskbar of the Windows Desktop.

Example

See also

ShowInfo

(ClientAPIReg unit in Altium Designer RTL)

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.

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.

ShowWarning

(ClientAPIReg unit in Altium Designer RTL)

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.

File IO

AddBackSlashToFrontAndBack

(RT_Util unit)

Declaration

```
Function  AddBackSlashToFrontAndBack(S: TDynamicString) : TDynamicString;
```

Description

The `AddBackSlashToFrontAndBack` function adds a path separator character to the front and to the back of a string. For example if the S string is empty, only one back slash is added to the string. Otherwise the S string has a back slash added to the front and to the end of this string.

See also

CheckAgainstWildCard_CaseSensitive

(RT_Util unit)

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.

See also

CheckAgainstWildCard

(RT_Util unit)

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.

See also

ComputerName

(RT_Util unit)

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.

See also

ConvertDiskSizeToString

(RT_Util unit)

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.

See also

ConvertFIleNameToExeSystemFileName

(RT_FileUtil in Altium Designer RTL)

Declaration

```
Function ConvertFileNameToExeSystemFileName(S : TString) : TString;
```

Description

The `ConvertFileNameToExeSystemFileName` routine updates the file name to include the full path to Altium\System folder along with the filename parameter. An example is ‘C:\Program Files\Altium\System\ServerA.exe’

Example

See also

ConvertPartialPathToExeFileName

(RT_FileUnit from Altium Designer RTL)

Delaration

```
Function ConvertPartialPathToExeFileName(S : TString) : TString;
```

Description

The `ConvertPartialPathToExeFileName` routine updates the file name to include the full path to Altium\System folder along with the filename parameter. An example is ‘C:\Program Files\Altium\System\ServerA.exe’

Example

See also

CurrentModuleName

(RT_FileUtil)

Syntax

```
Function CurrentModuleName : TString;
```

Description

The `CurrentModuleName` function retrieves the full path and filename for the executable/dynamic library linking file containing the specified module.

Example

See also

DocumentIsReadOnly

(RT_Util unit)

Declaration

```
Function DocumentIsReadOnly      (FullPath : TDynamicString) : Boolean;
```

Description

The `DocumentIsReadOnly` function returns True if a design document file has a read only property set true.

Example

```
If DocumentIsReadOnly(Document.FileName) Then
Begin
    ShowError(ExtractFileName(Document.FileName) + ' is read-only.');
    Exit;
```

```
End;
```

See also

ExtractFilename function

ExistAnyWhere

(RT_FileUtil)

Declaration

```
Function ExistAnyWhere(Var S : TDynamicString) : TBoolean; Overload;
Function ExistAnyWhere(Var S : TString       ) : TBoolean; Overload;
```

Description

The `ExistAnyWhere` function returns a TBoolean value denoting whether the file exists or not. Note that the S parameter is of `TDynamicString` type.

Example

```
// Remove the .SchLib file because it is no longer needed
SchLibFileName := GetProjectLibraryPath;
If ExistAnyWhere(SchLibFileName) Then
Begin
    Project.DM_RemoveSourceDocument(SchLibFileName);
    Document := Client.GetDocumentByPath(SchLibFileName);
    If Document <> Nil Then Document.ReleaseFileOwnership;
    DeleteFile(SchLibFileName);
End;
```

See also

ExistAnyWhereAsTemplate function

ExistAnyWhereAsTemplate

(RT_FileUtil in Altium Designer RTL)

Declaration

```
Function ExistAnyWhereAsTemplate(Var S : TDynamicString) : TBoolean;
```

Description

Checks if the S parameter containing the filename exists in the following folders:

SpecialFolder_DesignTemplates,

SpecialFolder_AltiumSystemTemplates,

SpecialFolder_TemplatesForAllUsers, or

SpecialFolder_CommonDocumentTemplates.

Example

```
If Not ExistAnywhere(MacroFileName) then Exit;
```

See also

ExistAnyWhere function.

ExpandFile

(RT_Util unit)

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 sub-directories in addition to the file name and extension.

The `ExpandFileName` function does not verify that the resulting fully qualified path name refers to an existing file, or even that the resulting path exists.

Example

ShowMessage(ExpandFileName('autoexec.bat'));

See also

ExtractFilename function

FileExists function

FindFileAnyWhere

(RT_FileUtil)

Declaration

```
Function FindFileAnyWhere(Var Path : TDynamicString) : TBoolean; Overload;
```

Description

This `FindFileAnywhere` checks if the file exists in the path or anywhere else. If a file is found, a 'True' value is returned, otherwise, 'False'

Example

See also

FileExists

(RT_Util unit)

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;
```

See also

GetFreeDiskSpaceString

(RT_Util unit)

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.

See also

GetDiskSizeString

(RT_Util)

Declaration

```
Function  GetDiskSizeString     (DiskNumber : Integer) : TDynamicString;
```

Description

The `GetDiskSizeString` function returns a TDynamicString value which represents the size, in bytes, of the specified drive.

See also

GetDiskFree

(RT_Util)

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.

See also

GetMacroDescription

(RT_FileUtil)

Declaration

Function `GetMacroDescription`(MacroFileName : TString) : TString;

Description

This GetMacroDescription returns a string if the function finds the '$SUMMARY' or '$Description' identifier in a macro script.

Example

See also

HasExtension

(RT_Util)

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.

See also

IsFullPathToExistingFile

(RT_Util)

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.

See also

IsFullPathToExistingStructuredStorage function

IsFullPathToExistingStructuredStorage Function

(RT_Util)

Declaration

```
Function IsFullPathToExistingStructuredStorage(Const FullPath : TDynamicString) : Boolean;
```

Description

This function indicates whether a particular disk file contains a storage object. This function returns True if the path including the filename to an existing structured storage exists.

Example

```
If IsFullPathToExistingStructuredStorage(GetFileName) Then
    Result := fmShareDenyNone
Else
    Result := Inherited GetFileShareMode;
```

See also

IsFullPathToExistingFile function

IsPathRelative

(RT_FileUtil)

Declaration

```
Function IsPathRelative(Path : TString) : Boolean;
```

Description

This `IsPathRelative` function checks if the string contains a relative path not a full absolute path.

Example

```
    If IsPathRelative(FileName) Then
    Begin
        If Not DirectoryExists(FRootPath) Then Exit;

        S := GetCurrentDir;
        If Not SetCurrentDir(FRootPath) Then Exit;
        Try
            AbsolutePath := ExpandFileName(FileName);
        Finally
            SetCurrentDir(S);
        End;
    End
    Else
        AbsolutePath := FileName;
```

See also

ExpandFilename function

LowLevelRunTextEditorWithFile

(RT_Util unit)

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.

See also

RunCommand procedure.

ProcessAllFilesOnPath

(Rt_FileUtil)

Declaration

```
Procedure ProcessAllFilesOnPath(Filter            : TDynamicString;
                                FileFunction      : TFileFunction;
                                AbsolutePath      : TDynamicString;
                                IncludeSubFolders : Boolean = True);
```

Description

This function returns all files on the specified `AbsolutePath` and `Filter` parameters. Normally to fetch all files on the Absolute path, use this `‘*’ Filter` String. Note only one asterisk for the `Filter` parameter. Otherwise you can use the following filters for example, ‘*.*’ and ‘*.Schlib’. The FileFunction parameter outputs strings in a `TStringList` object.

Example

```
ProcessAllFilesOnPath('*',ArchiveItems_CreateAnyDirectoryFile,AFullPath,True);
```

See also

TFileFunction type

ValidDosFileName

(RT_FileUtil)

Declaration

```
Function ValidDosFileName(FileName : TSTring) : TBoolean;
```

Description

The `ValidDosFileName` returns a TBoolean value denoting whether the filename string is a valid DOS filename. A valid dos filename must not have the following characters (‘*’,’?’,’ ‘,’”’,‘/‘,’;’ ,‘|’,‘,’, ‘=’) and only have one ‘.’ fullstop character in the entire filename string.

Example

```
Filename := ForceFileNameExtension(Board.FileName, ReportFileExtension);
If GetState_ParameterUpperCaseString(Parameters, 'Filename', S) Then
    If (ValidDosFileName(S)) then Filename := S;
```

See also

ForceFileNameExtension function

Number Manipulation Routines

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.

See also

ExtendedToEng

(RT_Util unit)

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

(RT_Util unit)

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

GetHexStringFromInteger

(RT_Util unit)

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

HexToInteger

(RT_Util unit)

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

IntegerToHex

(RT_Util unit)

Declaration

```
Function IntegerToHex(L : Integer) : TDynamicString;
```

Description

Convert an integer value to an hexadecimal value.

See also

Number Manipulation routines

IntMax

(RT_Util unit)

Declaration

```
Function  IntMax(x,y : Integer) : Integer;
```

Description

The `IntMax` function returns the maximum value of X and Y integer types.

See also

Number Manipulation routines

IntMin

(RT_Util unit)

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

IntSwap

(RT_Util unit)

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

Other Routines

AltKeyDown

(ClientAPIReg unit)

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

BeginHourGlass

(ClientAPIReg unit)

Declaration

```
Procedure BeginHourGlass(ACursor : TCursor = crHourGlass);
```

Description

The `BeginHourGlass` procedure changes the cursor to a Hour Glass that denotes that the system is busy.

See also

EndHourGlass procedure

SetCursorBusy procedure

Other Routines

CheckActiveServer

(ClientAPIReg unit in Altium Designer RTL)

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

ControlKeyDown

(ClientAPIReg unit)

Syntax

```
Function ControlKeyDown: Integer;
```

Description

The `ControlKeyDown` function returns a value that indicates the state of the CONTROL key, that is, the function returns 1 if the CONTROL key is down, otherwise it returns 0.

See also

AltKeyDown and ShiftKeyDown functions.

Other Routines

BeginHourGlass

(ClientAPIReg unit)

Declaration

```
Procedure BeginHourGlass(ACursor : TCursor = crHourGlass);
```

Description

The `EndHourGlass` procedure changes the cursor from a Hour Glass cursor back to the default pointing cursor.

See also

BeginHourGlass procedure

SetCursorBusy procedure

Other Routines

EscKeyDown

(ClientAPIReg unit)

Syntax

```
Function EscKeyDown: Integer;
```

Description

The `EscKeyDown` function returns a value that indicates the state of the ESCAPE key, that is, the function returns 1 if the ESCAPE key is down, otherwise it returns 0.

See also

AltKeyDown and ShiftKeyDown functions.

Other Routines

GetActiveServerName function

(ClientAPIReg unit)

Syntax

```
Function GetActiveServerName:String;
```

Description

The `GetActiveServerName` function returns the name of the server module that is currently active in Altium Designer.

Example

See also

Other Routines

GetCurrentWindowHandle

(ClientAPIReg unit)

Declaration

```
Procedure GetCurrentWindowHandle(Var Value: HWND);
```

Description

The procedure returns an HWND value which represent the window handle of the currently active window in Altium Designer.

See also

Other Routines

GetCurrentDocumentFileName

(ClientAPIReg unit)

Declaration

```
Function GetCurrentDocumentFileName : String;
```

Description

The `GetCurrentDocumentFileName` obtains the filename of the currently focussed document in DXP.

See also

SaveCurrentDocument function.

Other Routines

GetErrorMessage

(ClientAPIReg unit)

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

(ClientAPIReg unit)

Declaration

```
Function RunApplication(Const CommandLine : String) : Integer;
```

Description

The `RunApplication` function executes an application program outside the Altium Designer 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

(ClientAPIReg unit in Altium Designer RTL)

Declaration

```
Procedure ResetCursor;
```

Description

The `ResetCursor` resets the cursor to the default arrow cursor.

See also

SetCursorBusy

Other Routines

RunCommand

(RT_API unit and RT_Util)

Syntax

```
Procedure RunCommand (Const IdString : TDynamicString; Const SpecialParameter :
TDynamicString);
```

Description

This procedure executes a server process with parameters. The `IdString` parameter denotes the `servername:serverprocessname`. The `SpecialParameter` parameter denotes the `parametername=parametervalue` blocks separated by the | pipe symbol.

This RunCommand function is not properly supported by the scripting system in Altium Designer.

Examples

```
RunCommand('Client:SetupPreferences', 'Server=PCB|PageName=Models');
RunCommand('WorkspaceManager:Configure','ObjectKind=MessageView|Action=ClearAll');
RunCommand('PCB:BoardInformation','');
RunCommand('PCB:Zoom','Action=Redraw');
```

See also

RunSystemCommand

RunSystemCommand

(RT_Util unit)

Syntax

```
Function RunSystemCommand(Const S : TDynamicString) : TBoolean;
```

Description

The `RunSystemCommand` function runs the specified application denoted by the parameter string, S.

Example

```
RunSystemCommand('NotePad.Exe ' + S);
```

See also

RunCommand procedure.

RunSystemCommandInSystemDirectory

(RT_Util unit)

Syntax

```
Function  RunSystemCommandInSystemDirectory(Const S : TDynamicString) : TBoolean;
```

Description

The `RunSystemCommandInSystemDirectory` function runs the specified application in the Windows directory and the application's filename is denoted by the string, S.

Example

```
RunSystemCommandInSystemDirectory(‘Notepad.Exe’);
```

See also

RunCommand procedure

RunSystemCommand procedure

SaveCurrentDocument

(ClientAPIReg unit)

Syntax

```
Function SaveCurrentDocument : Boolean;
```

Description

The `SaveCurrentDocument` function determines whether the current document can be saved or not.

See also

Other Routines

SetCursorBusy

(ClientAPIReg unit)

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

(ClientAPIReg unit)

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

Special Folder Path Strings

The Special Folder Paths section is defined in the RT_Util unit from the Altium Designer RTL.

ClientAPI_SpecialFolder_AltiumAllUserApplicationData

(ClientProcs unit)

Syntax

```
Function ClientAPI_SpecialFolder_AltiumAllUserApplicationData : WideString;
```

Description

This function returns the full path to the special folder.

Example

```
ShowMessage(ClientAPI_SpecialFolder_AltiumAllUserApplicationData);
//C:\Documents and Settings\All Users\Application Data\AltiumDesigner
```

See also

Special Folder Paths

ClientAPI_SpecialFolder_AltiumApplicationData

(ClientProcs unit)

Syntax

```
Function ClientAPI_SpecialFolder_AltiumApplicationData : WideString;
```

Description

This function returns the full path to the special folder.

Example

```
ShowMessage(ClientAPI_SpecialFolder_AltiumApplicationData);
//C:\Documents and Settings\*UserName*\Application Data\AltiumDesigner
```

See also

Special Folder Paths

ClientAPI_SpecialFolder_AltiumLocalApplicationData

(ClientProcs unit in Altium Designer RTL)

Syntax

```
Function ClientAPI_SpecialFolder_AltiumLocalApplicationData : WideString;
```

Description

This function returns the full path to the special folder.

Example

```
ShowMessage(ClientAPI_SpecialFolder_AltiumLocalApplicationData);
//C:\Documents and Settings\*UserName*\Local settings\Application Data\AltiumDesigner
```

See also

Special Folder Paths

SpecialFolder_AdminTools

(RT_Util unit)

Declaration

```
Function SpecialFolder_AdminTools : TDynamicString;
```

Description

This function returns the path to the All User Application Data folder.

See also

Special Folder Paths

SpecialFolder_AllUserAdminTools

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_AllUserDesktop

(RT_Util unit)

Declaration

```
Function SpecialFolder_AllUserDesktop : TDynamicString;
```

Description

This function returns the path to the `C:\Documents and Settings\All Users\Desktop` folder.

See also

Special Folder Paths

SpecialFolder_AllUserDocuments

(RT_Util unit)

Declaration

`Function SpecialFolder_AllUserDocuments : TDynamicString;`

Description

This function returns the path to the `C:\Documents and Settings\All Users\Desktop` folder.

See also

Special Folder Paths

SpecialFolder_AltiumLibraryIntegrated

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_AltiumLibraryPld

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_AltiumLibrary

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLibrary : TDynamicString;`

Description

This function returns the path to the Altium Library folder. Example `C:\Program Files\Altium Designer\Library\`

See also

Special Folder Paths

SpecialFolder_AltiumApplicationData

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_AltiumAllUserApplicationData

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_AltiumDesignExplorer

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumDesignExplorer : TDynamicString;`

Description

This function returns the path to the Altium folder. Example `C:\Program Files\Altium\`

See also

Special Folder Paths

SpecialFolder_AltiumLocalApplicationData

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_AltiumSystem

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_AltiumSystemTasksPages

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumSystemTasksPages : TDynamicString;`

Description

This function returns the path to the Altium's system tasks pages folder. Example `C:\Program Files\Altium\System\`

See also

Special Folder Paths

SpecialFolder_AltiumSystemTemplates

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_AllApplicationData

(RT_Util unit)

Declaration

```
Function SpecialFolder_AllUserApplicationData : TDynamicString;
```

Description

This function returns the path to the `C:\Documents and settings\All Users\Application Data` folder.

See also

Special Folder Paths

SpecialFolder_AltiumTaskingApplicationData

(RT_Util unit)

Declaration

```
Function SpecialFolder_AltiumTaskingApplicationData : TDynamicString;
```

Description

This function returns the path to the Altium Tasking application data folder for example `C:\Documents and Settings\UserName\Application Data\Altium Designer\Tasking`.

See also

Special Folder Paths

SpecialFolder_AltiumSecurityAllUserApplicationData

(RT_Util unit)

Declaration

```
Function SpecialFolder_AltiumSecurityAllUserApplicationData : TDynamicString;
```

Description

This function returns the path to the Altium Security All User Application Data folder for example `C:\Documents and Settings\UserName\Application Data\AltiumDesignerSecurity\`.

See also

Special Folder Paths

SpecialFolder_AltiumSystemResources

(RT_Util unit)

Declaration

```
Function SpecialFolder_AltiumSystemResources : TDynamicString;
```

Description

This function returns the path to the Altium System Resources folder for example `C:\Program Files\Altium Designer\System\Resources`.

See also

Special Folder Paths

SpecialFolder_AltiumSystemDesktopLayouts

(RT_Util unit)

Declaration

```
Function SpecialFolder_AltiumSystemDesktopsLayouts : TDynamicString;
```

Description

This function returns the path to the Altium Device Images folder.

See also

Special Folder Paths

SpecialFolder_AltiumHelp

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumHelp : TDynamicString;`

Description

This function returns the path to the Altium Help folder for example `C:\Program Files\Altium Designer\System\Help\`

See also

Special Folder Paths

SpecialFolder_AltiumLocalResources

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLocalResources : TDynamicString;`

Description

This function returns the path to the Altium Local resources folder for example `C:\Program Files\Altium Designer \System\`.

See also

Special Folder Paths

SpecialFolder_AltiumLocalHelp

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLocalHelp : TDynamicString;`

Description

This function returns the path to the Altium Local help folder for example `C:\Program Files\Altium Designer\System\Help\`.

See also

Special Folder Paths

SpecialFolder_AltiumScripts

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumScripts : TDynamicString;`

Description

This function returns the path to the Altium Scripts folder for example `C:\Program Files\Altium Designer\Scripts\`.

See also

Special Folder Paths

SpecialFolder_AltiumSystemButtons

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumSystemButtons : TDynamicString;`

Description

This function returns the path to the Altium System Buttons folder for example `C:\Program Files\Altium Designer\System\Buttons\`.

See also

Special Folder Paths

SpecialFolder_AltiumSystemDocumentImages

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumSystemDocumentImages : TDynamicString;`

Description

This function returns the path to the Altium System Document Images folder for example `C:\Program Files\Altium Designer\System\DocumentImages\`.

See also

Special Folder Paths

SpecialFolder_AltiumSystemNavImages

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumSystemNavImages : TDynamicString;`

Description

This function returns the path to the Altium System Nav Images folder for example `C:\Program Files\Altium Designer\System\NavImages\`.

See also

Special Folder Paths

SpecialFolder_AltiumSystemNavPages

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumSystemNavPages : TDynamicString;`

Description

This function returns the path to the Altium System Nav Pages folder for example `C:\Program Files\Altium Designer\System\NavPages`.

See also

Special Folder Paths

SpecialFolder_AltiumLibraryVHDL87

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLibraryVHDL87 : TDynamicString;`

Description

This function returns the path to the Altium Library VHDL 87 folder for example `C:\Program Files\Altium Designer\Library\VHDL\IEEE87\`.

See also

Special Folder Paths

SpecialFolder_AltiumLibraryVHDL93

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLibraryVHDL93 : TDynamicString;`

Description

This function returns the path to the Altium Library VHDL93 folder for example `C:\program files\Altium Designer\library\VHDL\IEEE93\`.

See also

Special Folder Paths

SpecialFolder_AltiumLibraryVerificVHDL87

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLibraryVerificVHDL87 : TDynamicString;`

Description

This function returns the path to the Altium Library Verific VHDL87 folder for example `c:\program files\Altium Designer\library\VHDL\VHDL87\.`

See also

Special Folder Paths

SpecialFolder_AltiumLibraryVerificVHDL93

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLibraryVerificVHDL93 : TDynamicString;`

Description

This function returns the path to the Altium Library Verific VHDL93 folder for example `c:\program files\Altium Designer\library\VHDL\VHDL93\.`

See also

Special Folder Paths

SpecialFolder_AltiumSynthesis

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumSynthesis : TDynamicString;`

Description

This function returns the path to the Altium Synthesis folder, for example `c:\program files\Altium Designer\library\VHDL_LIB\`

See also

Special Folder Paths

SpecialFolder_AltiumLibraryEDIF

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLibraryEDIF : TDynamicString;`

Description

This function returns the path to the Altium Library EDIF folder for example `c:\program files\Altium Designer\library\EDIF\`.

See also

Special Folder Paths

SpecialFolder_AltiumLibraryVHDL

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLibraryVHDL : TDynamicString;`

Description

This function returns the path to the Altium Library VHDL folder for example `c:\program files\Altium Designer\library\VHDL\.`

See also

Special Folder Paths

SpecialFolder_AltiumLibraryVHDLModels

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLibraryVHDLModels : TDynamicString;`

Description

This function returns the path to the Altium Library VHDL Models folder for example `c:\program files\Altium Designer\library\VHDL\Models\`.

See also

Special Folder Paths

AltiumLibraryLMF

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumLibraryLMF : TDynamicString;`

Description

This function returns the path to the Altium Library LMF folder for example `c:\program files\Altium Designer\library\EDIF\`.

See also

Special Folder Paths

SpecialFolder_AltiumConstraintFiles

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumConstraintFiles : TDynamicString;`

Description

This function returns the path to the Altium Constraint Files folder for example `c:\program files\Altium Designer\library\FPGA\`.

See also

Special Folder Paths

SpecialFolder_AltiumDeviceConstraintFiles

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumDeviceConstraintFiles : TDynamicString;`

Description

This function returns the path to the FPGA Device Constraint Files folder for example `c:\program files\Altium Designer\library\FPGA\DeviceConstraintFiles`.

See also

Special Folder Paths

SpecialFolder_AltiumDeviceImages

(RT_Util unit)

Declaration

`Function SpecialFolder_AltiumDeviceImages : TDynamicString;`

Description

This function returns the path to the Altium Device Images folder for example `c:\program files\Altium Designer\library\deviceimages\`.

See also

Special Folder Paths

SpecialFolder_ApplicationData

(RT_Util unit)

Declaration

`Function SpecialFolder_ApplicationData : TDynamicString;`

Description

This function returns the path to the `C:\Documents and settings\UserName\Application Data` folder.

See also

Special Folder Paths

SpecialFolder_CommonAllUserApplicationData

(RT_Util unit)

Declaration

`Function SpecialFolder_CommonAllUserApplicationData : TDynamicString;`

Description

This function returns the path to the Common All User Application Data folder for example `C:\Documents and Settings\All Users\Application Data\Altium Designer\`.

See also

Special Folder Paths

SpecialFolder_CommonApplicationData

(RT_Util unit)

Declaration

`Function SpecialFolder_CommonApplicationData : TDynamicString;`

Description

This function returns the path to the Common Application data folder for example `C:\Documents and Settings\User Name\Application Data\Altium Designer\`.

See also

Special Folder Paths

SpecialFolder_CommonDocumnetTemplates

(RT_Util unit)

Declaration

`Function SpecialFolder_CommonDocumnetTemplates : TDynamicString;`

Description

This function returns the path to the `C:\Documents and Settings\UserName\Templates` folder.

See also

Special Folder Paths

SpecialFolder_CommonLocalApplicationData

(RT_Util unit)

Declaration

`Function SpecialFolder_CommonLocalApplicationData : TDynamicString;`

Description

This function returns the path to the Common Local Application data folder for example `C:\Documents and Settings\User Name\Application Data\Altium Designer\`.

See also

Special Folder Paths

SpecialFolder_CommonProgramFiles

(RT_Util unit)

Declaration

`Function SpecialFolder_CommonProgramFiles : TDynamicString;`

Description

This function returns the path to the `C:\Program Files\Common Files` folder.

See also

Special Folder Paths

SpecialFolder_CommonStartup

(RT_Util unit)

Declaration

`Function SpecialFolder_CommonStartup : TDynamicString;`

Description

This function returns the path to the `C:\Documents and Settings\All Users\Start Menu` folder.

See also

Special Folder Paths

SpecialFolder_CommonStartupPrograms

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_CommonFavorites

(RT_Util unit)

Declaration

`Function SpecialFolder_CommonFavorites : TDynamicString;`

Description

This function returns the path to the `C:\Documents and Settings\All Users\Favorites` folder.

See also

Special Folder Paths

SpecialFolder_ControlPanel

(RT_Util unit)

Declaration

`Function SpecialFolder_ControlPanel : TDynamicString;`

Description

This function returns the path to the Control Panel folder.

See also

Special Folder Paths

SpecialFolder_DesignExamples

(RT_Util unit)

Declaration

`Function SpecialFolder_DesignExamples : TDynamicString;`

Description

This function returns the path to the Design Examples folder. Example `C:\Program Files\Altium\Examples\`

See also

Special Folder Paths

SpecialFolder_DesignTemplates

(RT_Util unit)

Declaration

```
Function SpecialFolder_DesignTemplates : TDynamicString;
```

Description

This function returns the path to the DesignTemplates folder. Example `C:\Program Files\Altium\Templates\`

See also

Special Folder Paths

SpecialFolder_Desktop

(RT_Util unit)

Declaration

```
Function SpecialFolder_Desktop : TDynamicString;
```

Description

This function returns the path to the `C:\Documents and Settings\UserName\Desktop` folder.

See also

Special Folder Paths

SpecialFolder_DesktopLocation

(RT_Util unit)

Declaration

```
Function SpecialFolder_DesktopLocation : TDynamicString;
```

Description

This function returns the path to the `C:\Documents and Settings\UserName\Desktop` folder.

See also

Special Folder Paths

SpecialFolder_Favorites

(RT_Util unit)

Declaration

```
Function SpecialFolder_Favorites : TDynamicString;
```

Description

This function returns the path to the `C:\Documents and Settings\UserName\Cookies` folder.

See also

Special Folder Paths

SpecialFolder_Fonts

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_InstalledPrinters

(RT_Util unit)

Declaration

`Function SpecialFolder_InstalledPrinters : TDynamicString;`

Description

This function returns the path to the `C:\Documents and Settings\UserName\PrintHood` folder.

See also

Special Folder Paths

SpecialFolder_Internet

(RT_Util unit)

Declaration

`Function SpecialFolder_Internet : TDynamicString;`

Description

This function returns the path to the folder where the internet browser software is located in.

See also

Special Folder Paths

SpecialFolder_InternetCookies

(RT_Util unit)

Declaration

`Function SpecialFolder_InternetCookies : TDynamicString;`

Description

This function returns the path to the `C:\Documents and Settings\UserName\Cookies` folder.

See also

Special Folder Paths

SpecialFolder_InternetHistory

(RT_Util unit)

Declaration

`Function SpecialFolder_InternetHistory : TDynamicString;`

Description

This function returns the path to the `C:\Documents and Settings\UserName\Local Settings\History` folder.

See also

Special Folder Paths

SpecialFolder_InternetTemporaryFiles

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_LocalApplicationdata

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_MyComputer

(RT_Util unit)

Declaration

```
Function SpecialFolder_MyComputer : TDynamicString;
```

Description

This function returns the path to the MyComputer folder.

See also

Special Folder Paths

SpecialFolder_MyDesigns

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_MyDocuments

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_MyMusic

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_MyNetworkPlaces

(RT_Util unit)

Declaration

```
Function SpecialFolder_MyNetworkPlaces : TDynamicString;
```

Description

This function returns the path to the C:\Documents and Settings\UserName\NetHood folder.

See also

Special Folder Paths

SpecialFolder_MyPictures

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_NetWorkRoot

(RT_Util unit)

Declaration

```
Function SpecialFolder_NetworkRoot : TDynamicString;
```

Description

This function returns the path to the Network Root directory.

See also

Special Folder Paths

SpecialFolder_NonlocalizedStartupPrograms

(RT_Util unit)

Declaration

```
Function SpecialFolder_NonLocalizedStartupPrograms : TDynamicString;
```

Description

This function returns the path to the Non Localized Startup Programs folder.

See also

Special Folder Paths

SpecialFolder_Printers

(RT_Util unit)

Declaration

```
Function SpecialFolder_Printers : TDynamicString;
```

Description

This function returns the path to the Printers folder.

See also

Special Folder Paths

SpecialFolder_Profile

(RT_Util unit)

Declaration

```
Function SpecialFolder_Profile : TDynamicString;
```

Description

This function returns the path to the C:\Program Files\UserName.

See also

Special Folder Paths

SpecialFolder_Programs

(RT_Util unit)

Declaration

```
Function SpecialFolder_Programs : TDynamicString;
```

Description

This function returns the path to the C:\Documents and Settings\UserName\Start Menu\Programs folder.

See also

Special Folder Paths

SpecialFolder_ProgramFiles

(RT_Util unit)

Declaration

```
Function SpecialFolder_ProgramFiles : TDynamicString;
```

Description

This function returns the path to the C:\Program Files folder

See also

Special Folder Paths

SpecialFolder_Recent

(RT_Util unit)

Declaration

```
Function SpecialFolder_Recent : TDynamicString;
```

Description

This function returns the path to the C:\Documents and Settings\UserName\Recent folder.

See also

Special Folder Paths

SpecialFolder_Recovery

(RT_Util unit)

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

Special Folder Paths

SpecialFolder_RecycleBin

(RT_Util unit)

Declaration

```
Function SpecialFolder_RecycleBin : TDynamicString;
```

Description

This function returns the path to the Recycle Bin.

See also

Special Folder Paths

SpecialFolder_SendTo

(RT_Util unit)

Declaration

```
Function SpecialFolder_SendTo : TDynamicString;
```

Description

This function returns the path to the C:\Documents and Settings\UserName\SendTo folder.

See also

Special Folder Paths

SpecialFolder_StartMenuItems

(RT_Util unit)

Declaration

```
Function SpecialFolder_StartMenuItems : TDynamicString;
```

Description

This function returns the path to the C:\Documents and Settings\UserName\Recent folder.

See also

Special Folder Paths

SpecialFolder_SystemFolder

(RT_Util unit)

Declaration

```
Function SpecialFolder_SystemFolder : TDynamicString;
```

Description

This function returns the path to the C:\WINNT\System32 folder.

See also

Special Folder Paths

SpecialFolder_TemplatesForAllUsers

(RT_Util unit)

Declaration

```
Function SpecialFolder_TemplatesForAllUsers : TDynamicString;
```

Description

This function returns the path to the C:\Documents and Settings\All Users\Templates folder.

See also

Special Folder Paths

SpecialFolder_Temporary

(RT_Util unit)

Declaration

```
Function SpecialFolder_Temporary : TDynamicString;
```

Description

This function returns the path to the C:\DOCUME~1\UserName\LOCALS~1\Temp\ folder.

See also

Special Folder Paths

SpecialFolder_TemporarySlash

(RT_Util unit)

Declaration

```
Function SpecialFolder_TemporarySlash : TDynamicString;
```

Description

This function returns the path to the C:\Documents and settings\UserName\Local Settings\Temp\ folder.

See also

Special Folder Paths

SpecialFolder_UserStartMenuItems

(RT_Util unit)

Declaration

```
Function SpecialFolder_UserStartMenuItems : TDynamicString;
```

Description

This function returns the path to the C:\Documents and Settings\UserName\Recent folder.

See also

Special Folder Paths

SpecialFolder_WindowsFolder

(RT_Util unit)

Declaration

```
Function SpecialFolder_WindowsFolder : TDynamicString;
```

Description

This function returns the path to the C:\WINNT folder.

See also

Special Folder Paths

String Routines

Center

(RT_Util unit)

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 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

StringReplace

(SysUtils unit)

Syntax

```
Function StringReplace(const S, OldPattern, NewPattern: string; Flags: TReplaceFlags): string;
```

Description

Basically this function returns a string with occurrences of one substring replaced by another substring. The `StringReplace` replaces occurrences of the substring specified by `OldPattern` with the substring specified by `NewPattern`.

Parameters

S is the source string, whose substrings are changed.

`OldPattern` is the substring to locate and replace with `NewPattern`.

`NewPattern` is the substring to substitute for occurrences of `OldPattern`.

`Flags` is a set of flags that govern how `StringReplace` locates and replaces occurrences of `OldPattern`. If `Flags` does not include `rfReplaceAll`, `StringReplace` only replaces the first occurrence of OldPattern in S. Otherwise, `StringReplace` replaces all instances of `OldPattern` with `NewPattern`. If the `Flags` parameter includes `rfIgnoreCase`, the comparison operation is case insensitive.

Notes

```
Type
  TReplaceFlags = set of (rfReplaceAll, rfIgnoreCase);
```

Example

```
Result := StringReplace(AKeys, ADelimiter, cDatabase_KeyFieldDelimiter, [rfReplaceAll]);
```

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

DateString

(RT_Util unit)

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

(RT_Util unit)

Declaration

```
Procedure GetCurrentDate     (Var   DateRecord   : TDate);
```

Description

The `GetCurrentDate` procedure is based on the Window API’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

(RT_Util unit)

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

(RT_Util unit)

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

(RT_Util unit)

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

(RT_Util unit)

Declaration

```
Function  GetDateAndTimeStamp : TDynamicString;
```

Description

This function returns the string containing the current date and the time.

See also

Time and Date Routines

GetElapsedTime

(RT_Util unit)

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

(RT_Util unit)

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 `TimeString_Elapsed` function.

See also

Time and Date Routines

GetFileDateString

Declaration

```
Function  GetFileDateString(Const AFileName : TDynamicString) : TDynamicString;
```

Description

The GetCurrentDateString function returns a String representing date in '12-Jan-1985' format for example.

See also

Time and Date Routines

GetMilliSecondTime

(RT_Util unit)

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

(RT_Util unit)

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

(RT_Util unit)

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

(RT_Util unit)

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 `GetElapsedTimeDate` 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

(RT_Util unit)

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

(RT_Util unit)

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

(RT_Util unit)

Declaration

```
Procedure WaitMilliSecondDelay(N : Integer);
```

Description

The `WaitMilliSecondDelay` function provides a delay in the code in milli-seconds as specified by the `N` integer value. This is useful if a function in the software needs delaying for a while before doing something else giving the software a chance to catch up. This function uses the `GetMilliSecondTime` function.

Example

```
WaitMilliSecondDelay(1000); // waits for 1 second. 1000 milliseconds = 1 second.
```

See also

Time and Date Routines

Functions from ClientProcs unit

```
Function  ClientAPI_GetPrefAnimatedPanels                                 : Boolean;
Function  ClientAPI_GetPrefSaveToolsLayout                                : Boolean;
Function  ClientAPI_GetPrefAutoTransparency                               : Boolean;
Function  ClientAPI_GetPrefDynamicAutoTransparency                        : Boolean;
Function  ClientAPI_GetPrefSuppressStartupScreen                          : Boolean;
Function  ClientAPI_GetPrefTransparencyHighest                            : Integer;
Function  ClientAPI_GetPrefTransparencyLowest                             : Integer;
Function  ClientAPI_GetPrefTransparencyForce                              : Integer;
Function  ClientAPI_GetPrefPopupPanelDelay                                : Integer;
Function  ClientAPI_GetPrefHidePanelDelay                                 : Integer;
Function  ClientAPI_GetPrefAnimatedPanelSpeed                             : Integer;
Function  ClientAPI_GetPrefPathInTitleBar                                 : Boolean;
Function  ClientAPI_GetPrefUseShadow                                      : Boolean;
Function  ClientAPI_GetPrefUseLuna                                        : Boolean;
Function  ClientAPI_GetPrefHideFloatingPanels                             : Boolean;
Function  ClientAPI_GetPrefRestoreOpenDocuments                           : Boolean;
Function  ClientAPI_GetPrefOpenTasksIfNothingOpen                         : Boolean;
Function  ClientAPI_GetPrefHideBinderViewTabs                             : Boolean;
Function  ClientAPI_GetPrefNoRestoreKindCount                             : Integer;
Procedure ClientAPI_GetPrefNoRestoreKind                    (Index        : Integer; Buffer
: PChar);

Procedure ClientAPI_SetPrefAnimatedPanels                   (Value        : Boolean);
Procedure ClientAPI_SetPrefSaveToolsLayout                  (Value        : Boolean);
Procedure ClientAPI_SetPrefAutoTransparency                 (Value        : Boolean);
Procedure ClientAPI_SetPrefDynamicAutoTransparency          (Value        : Boolean);
Procedure ClientAPI_SetPrefSuppressStartupScreen            (Value        : Boolean);
Procedure ClientAPI_SetPrefTransparencyHighest              (Value        : Integer);
Procedure ClientAPI_SetPrefTransparencyLowest               (Value        : Integer);
Procedure ClientAPI_SetPrefTransparencyForce                (Value        : Integer);
Procedure ClientAPI_SetPrefPopupPanelDelay                  (Value        : Integer);
Procedure ClientAPI_SetPrefHidePanelDelay                   (Value        : Integer);
Procedure ClientAPI_SetPrefAnimatedPanelSpeed               (Value        : Integer);
Procedure ClientAPI_SetPrefPathInTitleBar                   (Value        : Boolean);
Procedure ClientAPI_SetPrefUseShadow                        (Value        : Boolean);
Procedure ClientAPI_SetPrefUseLuna                          (Value        : Boolean);
Procedure ClientAPI_SetPrefHideFloatingPanels               (Value        : Boolean);
Procedure ClientAPI_SetPrefRestoreOpenDocuments             (Value        : Boolean);
Procedure ClientAPI_SetPrefOpenTasksIfNothingOpen           (Value        : Boolean);
Procedure ClientAPI_SetPrefHideBinderViewTabs               (Value        : Boolean);
Procedure ClientAPI_SetPrefNoRestoreKindClear;
Procedure ClientAPI_SetPrefNoRestoreKindAdd                 (Value        : PChar);
Function  ClientAPI_GetPrefRememberFormForDocKind                         : Boolean;
Procedure ClientAPI_SetPrefRememberFormForDocKind           (Value        : Boolean);
Procedure ClientAPI_SetAutoShowComponentSymbols             (Value        : Boolean);
```

```
Function  ClientAPI_GetAutoShowComponentSymbols                              : Boolean;



Procedure ClientAPI_ShowProductStartup (Bitmap       : TDynamicString);
Procedure ClientAPI_HideProductStartup;
Procedure ClientAPI_AddStartupMessage  (S            : TDynamicString);
Procedure ClientAPI_AddShutdownMessage (S            : TDynamicString);

Procedure ClientAPI_Synchronize (Const ASync : IThreadSynchronize);
Procedure ClientAPI_CheckSynchronize;

Function ClientAPI_GetCurrentOutputGenerator : IUnknown;
Procedure ClientAPI_SetCurrentOutputGenerator(Const Generator : IUnknown);

Function  ClientAPI_GetBuiltInNavigationBar          : Boolean;
Procedure ClientAPI_SetBuiltInNavigationBar   (Value : Boolean);
Function  ClientAPI_GetAlwaysShowNavBarInTasks       : Boolean;
Procedure ClientAPI_SetAlwaysShowNavBarInTasks(Value : Boolean);
{..............................................................................}
{..............................................................................}
Function  ClientAPI_GetFavoritesThumbnailSize       : TSize;
Procedure ClientAPI_SetFavoritesThumbnailSize(Value : TSize);
{..............................................................................}
{..............................................................................}
Function  ClientAPI_GetGroupingInDocumentsBar            : TDocumentsBarGrouping;
Procedure ClientAPI_SetGroupingInDocumentsBar    (Value : TDocumentsBarGrouping);
Function  ClientAPI_GetEqualButtonsInDocumentsBar       : Boolean;
Procedure ClientAPI_SetEqualButtonsInDocumentsBar(Value : Boolean);
Function  ClientAPI_GetAutoHideDocumentsBar             : Boolean;
Procedure ClientAPI_SetAutoHideDocumentsBar      (Value : Boolean);
Function  ClientAPI_GetMultilineDocumentsBar            : Boolean;
Procedure ClientAPI_SetMultilineDocumentsBar     (Value : Boolean);
Function  ClientAPI_GetMiddleClickClosesDocumentTab       : Boolean;
Procedure ClientAPI_SetMiddleClickClosesDocumentTab(Value : Boolean);
Function  ClientAPI_GetIntegratedHelpSystem             : Boolean;
Procedure ClientAPI_SetIntegratedHelpSystem      (Value : Boolean);
Function  ClientAPI_GetUseSystemLocaleLanguage          : Boolean;
Procedure ClientAPI_SetUseSystemLocaleLanguage   (Value : Boolean);
Function  ClientAPI_GetUseLocalizedDialogs              : Boolean;
Procedure ClientAPI_SetUseLocalizedDialogs       (Value : Boolean);
Function  ClientAPI_GetUseLocalizedResources            : Boolean;
Procedure ClientAPI_SetUseLocalizedResources     (Value : Boolean);
Function  ClientAPI_GetVSStyleCtrlTab                   : Boolean;
Procedure ClientAPI_SetVSStyleCtrlTab            (Value : Boolean);
Function  ClientAPI_GetActivateLastActiveOnClose        : Boolean;
Procedure ClientAPI_SetActivateLastActiveOnClose (Value : Boolean);
```

```
{..............................................................................}

Function ClientAPI_GetHelpFileAndTopic(Const AHelpTopicID : WideString; Out HelpFileName,
HelpTopicName : WideString) : Boolean;


Function  ClientAPI_UpdateFont(Var Font : TLogFont) : LongBool;
Procedure ClientAPI_SetErrorInfo(Const ErrorMsg, ErrorReport : WideString; ErrorAddr :
Pointer);
Procedure ClientAPI_ClearErrorInfo;
Procedure ClientAPI_HandleException(Const Message : WideString);


Procedure ClientAPI_QueryUpdatesInfo        (Var   UpdatesURL, UpdatesNetworkPath :
WideString; Var UpdatesUseNetworkPath : LongBool; Var   UpdatesPathToDownloadUpdates :
WideString;
    Var CheckFrequency : TWebUpdate_CheckFrequency); Stdcall;


Procedure ClientAPI_SetUpdatesInfo          (Const UpdatesURL, UpdatesNetworkPath :
WideString;     UpdatesUseNetworkPath : LongBool; Const UpdatesPathToDownloadUpdates :
WideString;
        CheckFrequency : TWebUpdate_CheckFrequency); Stdcall;
```

Server Process Routines

Servers

A server provides its services in the Altium Designer environment. The Client module within the Altium Designer interprets the tasks in terms of server processes and then delegates these processes to the appropriate servers.

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

Generally a process is executed by selecting a command which is a packaged process launcher (such as clicking on a toolbar button, or pressing a hot key or selecting a menu item) in Altium Designer. Up to three different types of process launchers can be used to launch the same process.

You can manually run a process by going to the Run Process menu item in the System menu within

Server Processes

Each server 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 the Schematic Editor 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 Altium Designer), it will perform the task of zooming in on the currently active schematic sheet.

A process is implemented as a **server name:server process name** string. Processes are stored in a command launcher table maintained by the server. Every time you execute a process via the user interface, it consults the appropriate server’s command table to fetch the process string and then sends this string over to the server for the server to determine which process to execute. These processes are stored in corresponding server installation text files with an INS extension.

Parametric Processes

A parametric server process allows the information, a process needs, to be passed when the process is called. This ability to be able to pass process parameters allows direct control over the operation of a process. For parametric processes, each parameter has a value assigned and this parameter / value block is represented as Parameter = Name.

For example `FileName = C:\Program Files\TestFile.Txt.`

To concatenate several parameters as a whole string, each parameter / value block is separated by the pipe | symbol.

For example `Parameter1 = Name1 | Parameter2 = Name 2` etc.

Manipulating Server Processes

There are server process functions and a `TParameterList` class from the `RT_Param` unit part of the Altium Designer RTL that do the manipulation of process strings much more easily.

TParameterList Class

(RT_Param unit)

Overview

The `TParameterList` class stores parameter name = value blocks separated by the Pipe symbols in a single null terminated string easily. For example, `Orientation=1|Location.X=10000000|Location.Y=20000000` is a typical parameter string.

To add parameters in the `TParameterlist` object, you use one of the following `SetState_AddParameterX` methods. Normally the `SetState_AddParameterAsString` method is used in this case.

To retrieve a specially formatted null terminated string from the `TParameterList` object, you can invoke one of the `GetState_ParameterX` methods. The `GetState_ToString` or `GetState_ParameterAsPChar` methods are used in this case.

You create an instance of the `TParameterList` class and invoke the `ClearAllParameters` method to reset it.

TParameterList Methods

```
Constructor Create;
Destructor  Destroy; Override;
```

SetState_FromString and GetState_ToString methods

```
Procedure   SetState_FromString (Const S : TDynamicString);
Function    GetState_ToString            : TDynamicString;
```

SetState_AddParameterX methods

```
Procedure   SetState_AddParameter           (Const AName, AValue : TDynamicString);
Procedure   SetState_AddParameterAsString   (Const AName : TDynamicString; Const Value :
TDynamicString);
Procedure   SetState_AddParameterAsBoolean  (Const AName : TDynamicString; Value : Boolean);
Procedure   SetState_AddParameterAsInteger  (Const AName : TDynamicString; Value : Integer);
Procedure   SetState_AddParameterAsInt64    (Const AName : TDynamicString; Value : Int64);
Procedure   SetState_AddParameterAsDouble   (Const AName : TDynamicString; Const Value :
Double);
```

GetState_AddParameterX methods

```
Function    GetState_ParameterAsString      (Const Name : TDynamicString; Var Value :
TDynamicString ) : Boolean; Overload;
Function    GetState_ParameterAsString      (Const Name : TDynamicString; Var Value : TString
) : Boolean;        Overload;
Function    GetState_ParameterAsPChar       (Const Name : TDynamicString; Var Value : PChar   )
: Boolean;
Function    GetState_ParameterAsLongInt     (Const Name : TDynamicString; Var Value : LongInt)
: Boolean;
Function    GetState_ParameterAsInteger     (Const Name : TDynamicString; Var Value : Integer)
: Boolean;
Function    GetState_ParameterAsInt64       (Const Name : TDynamicString; Var Value : Int64   )
: Boolean;
Function    GetState_ParameterAsSmallInt    (Const Name : TDynamicString; Var Value :
SmallInt) : Boolean;
Function    GetState_ParameterAsWord        (Const Name : TDynamicString; Var Value : Word    )
: Boolean;
```

```
Function    GetState_ParameterAsBoolean     (Const Name : TDynamicString; Var Value : Boolean)
: Boolean;
Function    GetState_ParameterAsWordBool    (Const Name : TDynamicString; Var Value :
WordBool) : Boolean;
Function    GetState_ParameterAsReal        (Const Name : TDynamicString; Var Value : Single
) : Boolean;
Function    GetState_ParameterAsDouble      (Const Name : TDynamicString; Var Value : Double)
: Boolean;
```

Other methods

```
Function    GetState_ParameterByName  (Const AName : TDynamicString) : TParameter;
Function    SetState_RemoveByName     (Const AName : TDynamicString) : Boolean;
Procedure   ClearAllParameters;
Procedure   SetState(P : PChar);
Procedure   GetState(P : PChar);
```

Scripting Notes

In Scripting, we can only use the following methods `SetState_FromString (Const S : TDynamicString);` and `GetState_ToString` to process strings. The `SetState` and `GetState` methods cause problems in the scripting engine.

Example in DelphiScript

```
    //Parameters = Orientation=1|Location.X=10000000|Location.Y=20000000';
    P := TParameterList.Create; // P is of TParameterList type.
    P.ClearAllParameters;
    P.SetState_FromString(Parameters);
    P.SetState_AddParameterAsString ('Orientation','1');
    P.SetState_AddParameterAsString ('Location.X' ,'10000000');
    P.SetState_AddParameterAsString ('Location.Y' ,'20000000');
    P.SetState_AddParameterAsString ('Designator' ,'dB1');
    P.SetState_AddParameterAsString ('Comment'    ,'50pF');
    Parameters := P.GetState_ToString;

    IntegratedLibraryManager.PlaceLibraryComponent(SchLibRef,SchLibpath,Parameters);
    P.Free;
```

Process Parameter Functions

```
Function  GetState_Parameter      (P : PChar; Const Name : TString; Var Value : TString) :
Boolean; Overload;
Function  GetState_Parameter      (P : PChar; Const Name : TDynamicString; Var Value :
TDynamicString) : Boolean; Overload;


Procedure SetState_RemoveParameter(P : PChar; Const Name : TDynamicString); Overload;
Function  GetState_ParameterPChar (P : PChar; Const Name : TDynamicString;     Value : PChar)
: Boolean;
Procedure SetState_ParameterPChar (P : PChar; Const Name : TDynamicString;     Value : PChar);
Procedure SetState_Parameter      (P : PChar; Const Name : TDynamicString; Const Value :
TDynamicString); Overload;


Function  GetState_Parameter      (Const S : TDynamicString; Const Name : TDynamicString; Var
Value : TDynamicString) : Boolean; Overload;
Procedure SetState_Parameter      (Var   S : TDynamicString; Const Name : TDynamicString;
Const Value : TDynamicString); Overload;
```

```
Procedure SetState_RemoveParameter(Var   S : TDynamicString; Const Name : TDynamicString);
Overload;
```

Server Routines from ClientApiReg Unit

The server process routines are defined in the ClientApiReg unit as part of the Altium Designer RTL.

There are two ways you can execute a process in a script

To execute a server process in a script, you need to use commands such as **ResetParameters** and **RunProcess** procedures or invoke the **Client.SendMessage** function.

RunProcess Example

```
ResetParameters;
AddStringParameter('OpenMode','NewFromTemplate');
AddStringParameter('ObjectKind,'Project');
RunProcess('WorkSpaceManager:OpenObject);
```

Client.SendMessage Example

```
Client.SendMessage('WorkspaceManager:OpenObject','OpenMode=NewFromTemplate |
ObjectKind=Project',1024,Nil);
```

See also

Process Parameters Reference online help

Process Examples in \Examples\Scripts\Delphiscript Scripts\Processes\ folder.

AddWordParameter

(ClientAPIReg unit in Altium Designer RTL)

Declaration

```
Procedure AddWordParameter(Const Name: String; Value: Word);
```

Description

The **AddWordParameter** procedure defines a parameter with a Word data type to the parameter buffer for use by a server process.

Example

```
Begin
    ResetParameters;
    AddWordParameter('WordValue',5);
    // code here
End;
```

See also

Server Process routines

AddColorParameter

(ClientAPIReg unit in Altium Designer RTL)

Declaration

```
Procedure AddColorParameter(Const Name: String; Red: Integer; Green: Integer; Blue: Integer);
```

Description

This procedure adds a color value parameter to the parameter buffer in Altium Designer. This procedure is used to define a color for use by a process that requires a color parameter.

The Color is a value where value = RedVal + 256*(GreenVal + 256*BlueVal) and Name is the name representing this color value.

See also

Server Process routines

AddIntegerParameter

(ClientAPIReg unit in Altium Designer RTL)

Declaration

```
Procedure AddIntegerParameter(Const Name: String; Value: Integer);
```

Description

The AddIntegerParameter procedure defines a parameter with an Integer data type to the parameter buffer for use by a server process.

Example

```
Begin
    ResetParameters;
    AddStringParameter('ObjectKind','Netlist');
    AddIntegerParameter('Index',5);
    AddStringParameter('ReturnGeneratedDocuments', 'True');
    RunProcess('WorkspaceManager:GenerateReport');
End;
```

See also

Server Process routines

AddLongIntParameter

(ClientAPIReg unit)

Declaration

```
Procedure AddLongIntParameter(Const Name: String; Value: LongInt);
```

Description

The AddLongIntParameter procedure defines a parameter with a longint data type to the parameter buffer for use by a server process.

Example

```
Begin
    ResetParameters;
    AddLongIntParameter('LongIntValue',5);
    // code here
End;
```

See also

Server Process routines

AddSingleParameter

(ClientAPIReg unit)

Declaration

```
Procedure AddSingleParameter(Const Name: String; Value: Single);
```

Description

The AddLongIntParameter procedure defines a parameter with a single data type to the parameter buffer for use by a server process.

Example

```
Begin
    ResetParameters;
    AddSingleParameter('SingleValue',5);
    // code here
End;
```

See also

Server Process routines

AddStringParameter

(ClientAPIReg unit)

Declaration

```
Procedure AddStringParameter(Const Name, Value: String);
```

Description

This procedure adds a parameter with a string value to the parameter buffer. The Name parameter represents the name of the process parameter and the Value parameter represents the value of the process parameter.

Example

```
    ResetParameters
    Call AddStringParameter("Object","JumpToLocation10")
    Call RunProcess("PCB:Jump")
    ResetParameters
    Call AddStringParameter("ZoomLevel","2.0")
    Call RunProcess("PCB:Zoom")
```

See also

Server Process routines

GetColorParameter

(ClientAPIReg unit)

Declaration

```
Procedure GetColorParameter(Const Name: String; Var Red: Integer; Var Green: Integer; Var
Blue: Integer);
```

Description

The GetColorParameter procedure retrieves the values of a color parameter as RGB values from the parameter buffer after running a process that returns a color value.

See also

Server Process routines

GetIntegerParameter

(ClientAPIReg unit)

Declaration

```
Procedure GetIntegerParameter(Const Name: String; Var Value: Integer);
```

Description

The GetIntegerParameter procedure retrieves the value of an integer type parameter from the parameter buffer. This procedure after a process has been executed can return a resultant word value.

Example

```
Var
    ErrorCode : Integer;
    CommandLine : String;
    Result : Integer;
    NetlistName : String
Begin
    ResetParameters;
    AddStringParameter('ObjectKind','Netlist');
    AddIntegerParameter('Index',5);
    AddStringParameter('ReturnGeneratedDocuments', 'True');
    RunProcess('WorkspaceManager:GenerateReport');
    GetIntegerParameter('Result', Result);
    If Result = 0 Then Exit;
    NetListName := GetStringParameter('File1', Result);
End;
```

See also

Server Process routines

GetLongIntParameter

(ClientAPIReg unit)

Declaration

```
Procedure GetLongIntParameter(Const Name: String; Var Value: LongInt);
```

Description

The GetLongIntParameter procedure retrieves the value of a long int type parameter from the parameter buffer. This procedure after a process has been executed can return a resultant long int type value.

See also

Server Process routines

GetSingleParameter

(ClientAPIReg unit)

Declaration

```
Procedure GetSingleParameter(Const Name: String; Var Value: Single);
```

Description

The GetSingleParameter procedure retrieves the value of a single type parameter from the parameter buffer. This procedure after a process has been executed can return a resultant single type value.

See also

Server Process routines

GetStringParameter

(ClientAPIReg unit)

Declaration

```
Procedure GetStringParameter(Const Name: String; Var Value: String);
```

Description

The GetSingleParameter procedure retrieves the value of a string type parameter from the parameter buffer. This procedure after a process has been executed can return a resultant string type value.

Example

```
Var
    ErrorCode : Integer;
    CommandLine : String;
    Result : Integer;
    NetlistName : String
Begin
    ResetParameters;
    AddStringParameter('ObjectKind','Netlist');
    AddIntegerParameter('Index',5);
    AddStringParameter('ReturnGeneratedDocuments', 'True');
    RunProcess('WorkspaceManager:GenerateReport');
    GetIntegerParameter('Result', Result);
    If Result = 0 Then
        Exit;
    NetListName := GetStringParameter('File1', Result);
End;
```

See also

Server Process routines

GetWordParameter

(ClientAPIReg unit)

Declaration

```
Procedure GetWordParameter(Const Name: String; Var Value: Word);
```

Description

The GetWordParameter procedure retrieves the value of a word type parameter from the parameter buffer. This procedure after a process has been executed can return a resultant integer value.

See also

Server Process routines

ResetParameters

(ClientAPIReg unit)

Declaration

```
Procedure ResetParameters;
```

Description

The **ResetParameters** procedure clears the parameter buffer. Execute the procedure to reset the parameter buffer before setting parameters used by a process in your script or server project.

When you use any of the Add...Parameter procedures, the parameter declared is appended to the parameter buffer. When you run a process, any parameters that need to be passed to the process are read from the parameter buffer.

Running a process, however, DOES NOT clear the parameter buffer. Therefore, it is important to use the **ResetParameters** procedure to clear the buffer of old values before placing a new series of parameters into the buffer.

Example

```
Var
    ErrorCode : Integer;
    CommandLine : String;
    Result : Integer;
    NetlistName : String
Begin
    ResetParameters;
    AddStringParameter('ObjectKind','Netlist');
    AddIntegerParameter('Index',5);
    AddStringParameter('ReturnGeneratedDocuments', 'True');
    RunProcess('WorkspaceManager:GenerateReport');
    GetIntegerParameter('Result', Result);
    If Result = 0 Then
        Exit;
    NetListName := GetStringParameter('File1', Result);
End;
```

See also

Server Process routines

RunProcess

(ClientAPIReg unit in Altium Designer RTL)

Declaration

```
Procedure RunProcess(Const Command: String);
```

Description

The **RunProcess** procedure allows you to execute a server process. If the process invoked by this extension requires parameters to be passed to it, you must add the parameters to the parameter buffer using the AddXXXParameter functions before running the process.

If the process returns values, these will be placed in the return buffer and can be read using the GetXXXParameter functions.

Server: Process format

The Command string takes on the following form: Server:Process

where Server is the name of the server the process is supplied by, and Process is the command name of the process. An example is PCB:Zoom.

Client Process example

```
// available parameters for Dialog: Color or FileOpenSave Names
ResetParameters;
AddStringParameter('Dialog','Color'); // color dialog
AddStringParameter('Color', '0');     // black color
RunProcess('Client:RunCommonDialog');

//Result value obtained from the RunCommonDialog's Ok or Cancel buttons.
GetStringParameter('Result',S);
If (S = 'True') Then
Begin
    GetStringParameter('Color',S);
    ShowInfo('New color is ' + S);
End;
```

PCB Process example

```
// Refresh PCB workspace.
ResetParameters;
AddStringParameter('Action', 'Redraw');
RunProcess('PCB:Zoom');
```

Schematic Process example

```
// Refresh Schematic workspace
ResetParameters;
AddStringParameter('Action', 'All');
RunProcess('Sch:Zoom');
```

Workspace Manager Process example

```
Var
    ErrorCode : Integer;
    CommandLine : String;
    Result : Integer;
    NetlistName : String
Begin
    ResetParameters;
    AddStringParameter('ObjectKind','Netlist');
    AddIntegerParameter('Index',5);
    AddStringParameter('ReturnGeneratedDocuments', 'True');
    RunProcess('WorkspaceManager:GenerateReport');
End;
```

See also

Server Process routines

Helper Functions and Objects for 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.

Few useful functions are:

- `CopyFile`

Few useful classes are:

- `TStringList`
- `TList`
- `TIniFile`

Many routines and objects from the Borland Delphi's 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 typeless.

CopyFile function

Declaration

The `CopyFile` function copies a file specified by the original filename to a new file with the new filename. The function returns a true value if the CopyFile fuhnction is successful otherwise a false value is returned.

The `FailIfExists` parameter controls how an existing target file can be overrwritten or not with the new source file by the CopyFile function.

- If this parameter is TRUE and the new file already exists, the function fails.
- If this parameter is FALSE and the new file already exists, the function overwrites the existing file and succeeds.

Syntax

```
Function CopyFile(SourceFileName, TargetFilename : PChar; FailIfExists : Boolean) : Boolean;
```

DelphiScript Example

```
Procedure CopyFromTo;
Var
    Project    : String;
    PathSource : String;
    PathTarget : String;
Begin
    PathSource := 'C:\3M Footprints.PcbLib';
    PathTarget := 'C:\Temp\3M Footprints.PcbLib';
    CopyFile(PathSource, PathTarget, False);
End;
```

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 in Altium Designer for example.

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 of the Altium Designer installation.

See also

Helper Classes and Functions

Revision History

Date	**Version No.**	**Revision**
23-Nov-2005	1.0	New product release
15-Dec-2005	1.1	Updated for Altium Designer 6
23-Feb-2006	1.2	Revised for Altium Designer 6
29-Jun-2006	1.3	Updated for Altium Designer 6.3
7-Jul-2006	1.4	Updated page numbering and removed blank pages
28-Feb-2008	1.5	Updated Page Size to A4 and updated information.
20-Apr-2008	1.6	Updated path references.
5-Jun-2008	1.7	Updated information for the CopyFile function and some formatting updates.
24-Jun-2008	1.8	Updated information for the WaitMilliSecondDelay function. Some formatting updates.
4-Aug-2008	1.9	Added information from RT_Param unit of Altium Designer RTL.