



# DelphiScript Reference

---

## Summary

Technical Reference  
TR0120 (v1.4) Mar 17, 2006

This reference manual describes the DelphiScript language.

---

This reference includes the following topics:

- Exploring the DelphiScript language
- DelphiScript source files
- Creating new scripts
- Adding scripts to a project
- Executing a script in Altium Designer
- Assigning a script to a process launcher
- About DelphiScript examples
- Writing DelphiScript scripts
- DelphiScript Error Codes
- Expressions and Operators
- DelphiScript Functions
- Forms and Components.

## Exploring the DelphiScript Language

---

### Introduction

This DelphiScript reference details each of the statements, functions and extensions that are supported. These are special procedures that are used to control and communicate directly with the Design Explorer. It is assumed that you are familiar with basic programming concepts as well as the basic operation of your Design Explorer-based software.

The scripting system supports the DelphiScript language which is very similar to Borland Delphi (TM). The key difference is that, DelphiScript is a typeless or untyped scripting language which means you cannot define records or classes and pass pointers as parameters to functions for example. You can still declare variables within scripts for readability.

The DelphiScript Reference Help contains reference material on interfaces, components, global routines, types, and variables that make up the DelphiScript scripting language.

### Objects

## **DelphiScript Reference**

An object consists of methods, and in many cases, properties, and events. Properties represent the data contained in the object. Methods are the actions the object can perform. Events are conditions the object can react to. All objects descend from the ancestor object TObject.

### **Interfaces**

An interface consists of methods, and in many cases, properties but cannot have data fields. An interface represents an existing object and each interface has a GUID which marks it unique. Properties represent the data contained in the object that the interface is associated with. Methods are the actions the object (in which the interface is associated with) can perform.

### **Components**

Components are visual objects that you can manipulate at design time from the Tool Palette panel. All components descend from the TComponent class in the Borland Delphi Visual Component Library.

### **Routines**

Global routines are the procedures and functions from the scripting system. These routines are not part of a class, but can be called either directly or from within class methods on your scripts.

### **Types**

The types described in the Help are used as return types and parameter types for interface methods and properties and object's methods, properties and events, and for global routines. In many cases, types are documented in the Enumerated Types sections in each DXP Object Model reference help.

### **Altium Designer and Borland Delphi Run Time Libraries**

The Scripting system also supports a subset of Borland Delphi Run Time Library (RTL) and a subset of Altium Designer RTL which is covered in the Altium Designer RTL Reference.

### **Server Processes**

A script can execute server processes and thus server processes and parameters are covered in the Server Process Reference.

## **DelphiScript source files**

You open a script project in Altium Designer and you can edit the contents of a script inside the Altium Designer. A script project is organized to store script documents (script units and script forms). You can execute the script from a menu item, toolbar button or from the Run Script dialog from the Altium Designer's system menu.

### **PRJSCR, PAS and DFM files**

The scripts are organized into projects with a PRJSCR extension. Each project consists of files with a pas extension. Files can be either script units or script forms (each form has a script file with pas extension and a corresponding form with a dfm extension). A script form is a graphical window that hosts different controls that run on top of Altium Designer.

However it is possible to attach scripts to different projects and it is highly recommended to organize scripts into different projects to manage the number of scripts and their procedures / functions.

Scripts (script units and script forms) consist of functions/procedures that you can call within Altium Designer.

## Creating new scripts

You can add existing or new scripts into the specified project in the Projects panel in Altium Designer. There are two types of scripts; Script Units and Script Forms. With a project open in Altium Designer, right click on a project in the Projects panel, and a pop up menu appears, click on the Add New to Project item, and choose Script Unit. A new script appears.

A script can have at least one procedure which defines the main program code. You can, however, define other procedures and functions that can be called by your code. Functions and procedures are defined within a Begin End statement block as well as functions.

It is possible to have no procedures or functions within a script but at least it is necessary to have a Begin End. (with the full stop at end of the End keyword) block so that the script can have a chance to get executed.

### Example of a procedure

```
Procedure CreateSchObjects;
Begin
    If SchServer = Nil Then Exit;
    SchDoc := SchServer.GetCurrentSchDocument;

    If SchDoc = Nil Then Exit;
    PlaceSchematicObjects;
    SchDoc.GraphicallyInvalidate;
End;
```

### Example of Begin End. block

```
Begin
    ShowMessage('The dialog will appear when you run this script');
End.
```

If you have a statement in your script but there is no Begin End. block or no procedures, then the script will not be executed.

## Adding scripts to a project

You can add existing scripts to a specified project in the Projects panel in Altium Designer. With a project open in Altium Designer, right click on this project in the Projects panel, and a pop up menu appears, click on the Add Existing to Project... item.

A Choose Documents To Add to Project dialog appears. You can multi-select as many scripts you want to add into the specified project.

## Executing a script in Altium Designer

### In Text Editor workspace

You can configure the Run command when you are in the text editor to point to a script and execute it. Every time you click on the Run icon from the Text Editor menu or press F5, the scripting system executes the script pointed to by the Set Project Startup Procedure item. You can change the start up

procedure by clicking on the Set Project Start Up Procedure item in the Run menu which invokes the Select Item to Run dialog. You can then select which procedure of a script to be set.

### Executing a script on a design document

To execute a script in Altium Designer, there are two methods and there are two different dialogs for each method. These methods are necessary if you wish to run a script on a server specific document such as PCB or Schematic documents.

#### 1. Using the Select Item To Run dialog to execute a script

Click on the Run Script from the system menu and the Select Item to Run dialog appears with a list of procedures (those parameter-less procedures/functions only appear) within each script in a opened project.

Note that you can also click on a script unit filename within this Select Item to Run dialog and the functionless/procedureless Begin End. block within the script gets executed. See code example here

#### Script unit

```
Var
Begin
    //script here with no function/procedure
    A := 50;
    A := A + 1;
    ShowMessage(IntToStr(A));
End.
```

Now, only parameter-less functions and procedures for each script of an opened project only appear on the Select Item to Run dialog. It is a good idea for script writers to write the functions in scripts so that they will appear in this dialog and the other functions with parameters not to appear in this same dialog.

When you are working in a different editor such as PCB editor, you can assign the script to a process launcher and use it to run a specified script easily. See the Assigning a script to a process launcher.

You can add a list of installed script projects so that, every time you invoke the Select item to Run dialog, the installed script projects will appear along with other script projects currently open in the Projects panel. Invoke Scripting System Settings item from Tools » Scripting Preferences menu in the TextEditor workspace and the Scripting System Settings dialog appears.

#### 2. Using the Run Process dialog to execute a script

Invoke the Run Process dialog from the System menu and execute the ScriptingSystem:RunScript process in the Process: field and specify the script parameters, the ProjectName parameter which is the path to the project name and the ProcName parameter to execute the specified procedure from a specified script in the Parameters: field.

You need the following parameters for the ScriptingSystem:RunScript process to execute a specified script.

#### Process:

```
ScriptingSystem:RunScript
```

#### Parameters:

ProjectName (string - full path to a script project)

ProcName (string - script name > procedure name)

### Example

```
Process: ScriptingSystem:RunScript
```

```
Parameters : ProjectName = C:\Program Files\Altium Designer
6\Examples\Scripts\Delphiscript Scripts\DXP\DXP_Scripts.PrjScr | ProcName =
OpenADoc>OpenAndShowATextDocument .
```

To run a script repeatedly in the text editor, assign the script to the Set Project Startup Procedure item from the Run menu of the Text editor server. you can then click on the Run button. or press F5 to execute this script. To run a different script, you will need to re-invoke the Set Project Startup Procedure from the Run menu and assign a new script to it.

You can click on the Run Script item from the system menu and the Select Item to Run dialog appears with a list of procedures (those parameterless procedures/functions only appear) within each script in a project. This may be needed if you wish to run a script on a specific document type such as PCB or Schematic documents.

You can also use the Run Process dialog and specify the scriptingsystem server process and specify the parameters for this scripting system server to execute a script, this may also be needed if you wish to run a script on a specific document type such as PCB or Schematic documents.

## Assigning a script to a menu, toolbar or key

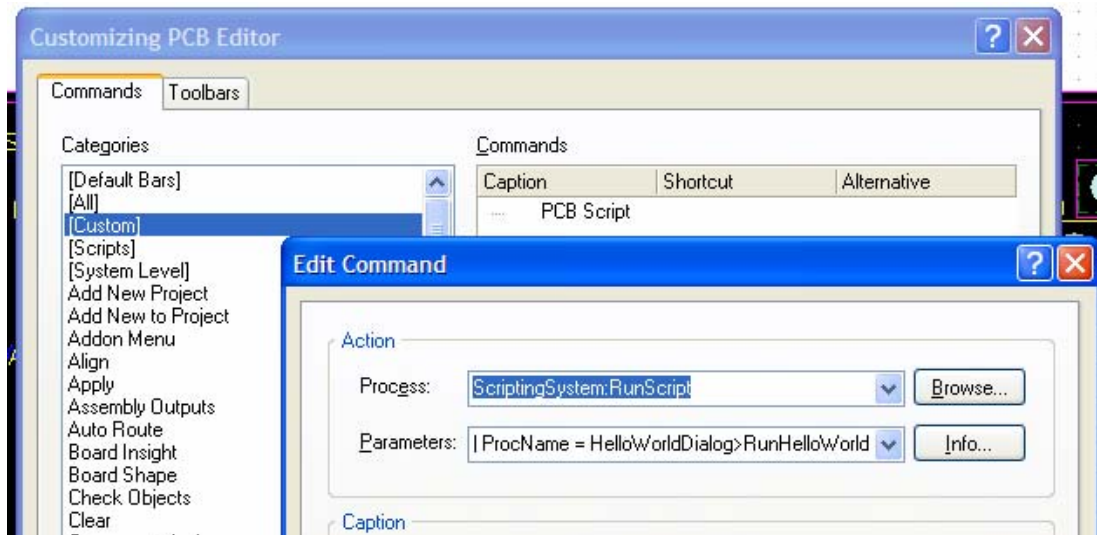
You have the ability to assign a script to a server menu, toolbar or hot key in Altium Designer which makes it possible for you to run the script over a current PCB document for example. You will need to specify the full path to a project where the script resides in and specify which unit and procedure to execute the script.

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

### Assigning to a process launcher example

To illustrate this ability to assign a script to a resource, we will open a PCB document in Altium Designer and use the HelloWorld script example from the \Scripts\General\ folder.

1. Double click on the PCB menu and the *Customizing PCB Editor* dialog appears.
2. Click on the **New** button from the *Customizing PCB Editor* dialog.
3. Choose **ScriptingSystem:RunScript** process in the **Process:** field of the *Customizing PCB Editor* dialog.
4. Enter ProjectName = C:\Program Files\Altium Designer  
6\Examples\Scripts\General\HelloWorld.PrjScr | ProcName =  
HelloWorldDialog>RunHelloWorld text in the Parameters: field for example.



5. You will need to give a name to this new command and assign a new icon if you wish. In this case, the name is **PCBScript** in the Caption: field of this dialog. The new commands appear in the **[Custom]** category of the **Categories** list. Click on the **[Custom]** entry from the **Categories** list. The **PCBScript** command appears in the **Commands** list of this dialog.
6. You then need to drag the new **PCBScript** command onto the PCB menu from the *Customizing PCB Editor* dialog. The command appears on the menu. You can then click on this new command and the HelloWorldDialog form appears.

## About Example Scripts

The examples that follow illustrate the basic features of DelphiScript programming. The examples show simple scripts for the Altium Designer application. The example scripts are organized into Scripts folder and its sub folders; General, Processes, PCB, Schematic and WSM folders accordingly.

### DelphiScript Scripts\General sub folder - Demonstrate DelphiScript keywords

- Hello World script - introductory script
- IniFileEg - demo the use of the TINIFile object
- Mandelbrot script - showing off the graphical capabilities of DelphiScript
- ShowModalEg script - demo the use of ModalResult of a script form.
- Sinewave script - demo the graphical capabilities
- TextFileConvert script - demo File I/O capabilities
- TicTac script - demo various DelphiScript keywords and to have fun!
- UpdateTime - demo the use of a Timer component.

### DelphiScript Scripts\Processes sub folder - Demonstrate server processes

Various scripts which use different server processes and parameters.

### DelphiScript Scripts\DXP sub folder - Demonstrate Client and system API

Various scripts that demonstrate several aspects of the Client API

**\DelphiScript Scripts\Sch subfolder - Demonstrate SCH API**

Various scripts that demonstrate several aspects of the Schematic API

**\DelphiScript Scripts\PCB subfolder - Demonstrate PCB API**

Various scripts that demonstrate several aspects of the PCB API

**\DelphiScript Scripts\WSM subfolder - Demonstrate WSM API**

Various scripts that demonstrate several aspects of the Work Space Manager API.

## Writing DelphiScript Scripts

---

In this Writing DelphiScript Scripts section:

- DelphiScript naming conventions
- Local and Global Variables
- Using named variables in a script
- Functions and procedures in a script
- Including comments in scripts
- Splitting a line of script
- Case sensitivity
- The space character
- Calculating expressions with the evaluate function
- Exiting from a procedure
- Passing parameters to functions and procedures
- Using DXP objects in scripts
- Tips on writing scripts
- Differences between DelphiScript and Object Pascal
- DelphiScript Error Codes.

### DelphiScript naming conventions

In general, there is no restriction to the names you can give to procedures, functions, variables and constants as long as they adhere to the following rules:

- The name can contain the letters A to Z, a to z, the underscore character "\_" and the digits 0 to 9.
- The name must begin with a letter.
- The name cannot be a DelphiScript keyword, directives or reserved word.
- Names are case insensitive when interpreted. You may use both upper and lower case when naming a function, subroutine, variable or constant, however the interpreter will not distinguish between upper and lower case characters. Names which are identical in all but case will be treated as the same name in DelphiScript.

In a DelphiScript file, the functions and procedures are declared using the Procedure Begin End or Function Begin End block or Var Begin End block. Both of these statement blocks require a name to be given to the procedure or function. DelphiScript allows you to create named variables and constants to hold values used in the current script.

### Including comments in scripts

In a script, comments are non-executed lines of code which are included for the benefit of the programmer. Comments can be included virtually anywhere in a script. Any text following //, or enclosed with (\*\*) or {} are ignored by DelphiScript.

#### // Comment type example

```
//This whole line is a comment
```

### **{}** Comment type example

```
{This whole line is a comment}
```

### **(\* \*)** comment type example

```
(*
This whole line is a comment
This whole line is a comment
This whole line is a comment
*)
```

Comments can also be included on the same line as executed code. For example, everything after the semi colon in the following code line is treated as a comment.

```
ShowMessage ('Hello World'); //Display Message
```

## Local and Global Variables

### Local and Global variables

Since all scripts have local and global variables, its very important to have unique variable names in your scripts within a script project. If the variables are defined outside any procedures and functions, they are global and can be accessed by any unit in the same project.

If variables are defined inside a procedure or function, then these local variables are not accessible outside these procedures/functions.

### Example of local and global variables in a script

```
// The Uses keyword is not needed.
// Variables from UnitA script are available to this unit script,
// as long UnitA is in the same project as this Unit script.
Const
    GlobalVariableFromThisUnit='Global Variable from this unit';

Procedure TestLocal;
var
    Local;
Begin
    // can we access a variable from UnitA without the Uses
    Local := 'Local Variable';
    ShowMessage(Local);
End;

Procedure TestGlobal;
```

Begin

```
//ShowMessage(Local); // produces an error.  
ShowMessage(GlobalVariableFromThisUnit);  
ShowMessage(GlobalVariableFromUnitA);
```

End;

### Unit A script

Const

```
GlobalVariableFromUnitA = 'Global Variable from Unit A';
```

## Using named variables in a script

In a script, you use named variables or constants to store values to be used during program execution. All variables in a script are always of Variant type. Typecasting is ignored. Types in variables declaration are ignored and can be skipped, so these declarations are correct:

```
var a : integer;  
var b : integer;  
var c, d;
```

## Splitting a line of script

Each code statement is terminated with the semi-colon ";" character to indicate the end of this statement. DelphiScript allows you to write a statement on several lines of code, splitting a long instruction on two or more lines. The only restriction in splitting programming statements on different lines is that a string literal may not span several lines.

For example:

```
X.AddPoint( 25, 100);  
X.AddPoint( 0, 75);  
// is equivalent to:  
X.AddPoint( 25, 100); X.AddPoint( 0, 75);
```

But

```
'Hello World!'
```

is not equivalent to

```
'Hello  
World!'
```

DelphiScript does not put any practical limit on the length of a single line of code in a script, however, for the sake of readability and ease of debugging it is good practice to limit the length of code lines so that they can easily be read on screen or in printed form.

If a line of code is very long, you can break this line into multiple lines and this code will be treated by the DelphiScript interpreter as if it were written on a single line.

### Unformatted code example

```
If Not (PcbApi_ChooseRectangleByCorners(BoardHandle,'Choose first
corner','Choose final corner',x1,y1,x2,y2)) Then Exit;
```

### Formatted code example

```
If Not (PcbApi_ChooseRectangleByCorners(BoardHandle,
                                         'Choose first corner',
                                         'Choose final corner',
                                         x1,y1,x2,y2)) Then Exit;
```

## Case sensitivity

The DelphiScript language used in writing scripts is not case sensitive, i.e. all keywords, statements, variable names, function and procedure names can be written without regard to using capital or lower case letters. Both upper and lower case characters are considered equivalent. For example, the variable name `myVar` is equivalent to `myvar` and `MYVAR`. DelphiScript treats all of these names as the same variable.

The only exception to this is in literal strings, such as the title string of a dialog definition, or the value of a string variable. These strings retain case differences.

## The space character

A space is used to separate keywords in a script statement. However, DelphiScript ignores any additional white spaces in a statement.

For example:

```
x = 5
```

is equivalent to

```
x      =          5
```

You may use white spaces to make your script more readable.

## Functions and procedures in a script

The DelphiScript interpreter allows two kinds of procedures: Procedures and Functions. The only difference between a function and a procedure is that a function returns a value.

A script can have at least one procedure which defines the main program code. You can, however, define other procedures and functions that can be called by your code. As with Borland Delphi, procedures are defined within a `Begin End` statement block as well as functions are defined within a `Begin.End` statement block too.

To invoke or call a function or procedure, simply include the name of the function or procedure in a statement in the same way that you would use the built-in DelphiScript functions and procedures. If the function or procedure requires parameters, then you must include these in the calling statement. Both functions and procedures can be defined to accept parameters, but only functions can be defined to return a value to the calling statement.

You may assign any name to functions and procedures that you define, as long as it conforms to the standard DelphiScript naming conventions.

### Typical DelphiScript procedure

## **DelphiScript Reference**

```
Procedure CreateSchObjects;  
Begin  
    If SchServer = Nil Then Exit;  
    SchDoc := SchServer.GetCurrentSchDocument;  
  
    If SchDoc = Nil Then Exit;  
    PlaceSchematicObjects;  
    SchDoc.GraphicallyInvalidate;  
End;
```

### **Typical DelphiScript function**

```
Function BooleanToString(AValue : Boolean) : String;  
Begin  
    If (AValue) Then Result := 'True'  
    Else          Result := 'False';  
End;
```

The name of a function can not be used to set its return value. The Result keyword must be used instead.

### **Var Begin End global block**

```
Var  
    A, B, C;  
Begin  
    B := 10;  
    C := 20;  
    A := B + C;  
    ShowMessage(IntToStr(A));  
End.
```

## **Using DXP Objects in scripts**

You cannot create your own records or classes types and instantiate them in a script, however you can use DXP run time libraries and DelphiScript's pre-defined classes for example TStringList and TList classes and instantiate them to act as containers of data storage for your scripting needs. The list of supported Delphi RTL and Design Explorer, PCB and Schematic classes are covered in the Altium Designer RTL reference document.

The biggest feature of DelphiScript, is that the Interfaces of Altium Designer objects are available to use in scripts. For example you have the ability to message design objects on Schematic and PCB documents through the use of Schematic Interfaces and PCB interfaces.

Beware that the Interface keyword has two uses - the Implementation / Interface sections and the Interface / Class declaration / implementation. In this case, the interfaces are used as declarations of the classes they are associated with. Interfaces are not local to a function or to a script. Therefore DXP Interfaces are available for use on any script.

Normally in scripts, there is no need to instantiate an interface, you just extract the interface representing an existing object in Altium Designer and from this interface you can extract embedded or aggregate interface objects and from them you can get or set property values.

Interface names as a convention have an I added in front of the name for example IPCB\_Board represents an interface for an existing PCB document. An example of PCB interfaces in use is shown next.

### Interface example

```

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

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

    (* Put this via in the Board object*)
    Board.AddPCBObject(Via);
End;

```

Objects, Interfaces, functions and types in your scripts can be used from the following units:

- Supported Borland Delphi™ functions and classes and DelphiScript extensions in the DelphiScript reference document.
- Client API
- PCB Server API

## DelphiScript Reference

- Schematic Server API
- Work Space Manager Server API
- Nexus API
- Altium Designer RTL functions
- Parametric processes

Check out the scripts in the \Altium Designer 6\Examples\Scripts\ folder to see DXP Interfaces, Delphi objects and functions being used in scripts.

## Tips on writing scripts

### Referencing scripts in a script project

You can have code in one script to call another procedure in another script in the same script project and access to any global variable in any script within the same project.

### Local and Global variables

Since all scripts have local and global variables, its very important to have unique variable names in your scripts within a script project. If the variables are defined outside any procedures and functions, they are global and can be accessed by any unit in the same project.

If variables are defined inside a procedure or function, then these local variables are not accessible outside these procedures/functions.

It is recommended to put scripts of similar nature in a project and that there are not too many scripts in a project. Keeping track of global variables in many scripts becomes an issue. It is not mandatory to store scripts in a script project, you can put scripts in other project types.

### Unique identifiers and variables

With script forms, ensure that all script forms have unique form names, for example it is possible (although wrong) to have all script forms be named form1 in the same script project. The scripting system gets confused when trying to display which form when a script form is executed.

Change the script form name by using the Object Inspector, and the name gets changed in the script unit and in the script form DFM files automatically.

### Parameter-less Procedures and Functions

Try and design your scripts so that those procedures that need to be invoked to run the script will only appear in the Select Items to Run dialog. To prevent other procedures/functions from appearing in the Select Items to Run dialog, you can for example insert a (Dummy : Integer) parameter next to the method name. See the example below.

### Example

```
Function TSineWaveform.CreateShape(Dummy : Integer) : TShape;
Begin
    // do something
End;
{.....}
{.....}
```

```

Procedure TSineWaveform.bCloseClick(Sender: TObject);
var
    I : integer;
Begin
    // doing something
    Close;
End;
{.....}
{.....}
procedure DrawSine;
Begin
    SineWaveform.showmodal;
End;

```

## Differences between DelphiScript and Object Pascal

In this section, the differences between DelphiScript and Object Pascal of Borland Delphi 32 bit versions will be covered in detail. The scripting system uses untyped DelphiScript language therefore there are no data types in scripts.

Although you can declare variables and their types and to specify the types for functions/procedures or methods' parameters for readability, DelphiScript converts undeclared variables on the fly (when a script is being executed).

You cannot define records or classes for example.

### Delphiscript Variables

All variables in a script are always of Variant type, thus, typecasting of variables is ignored. Types in variables declaration are ignored and can be skipped, so these declarations are correct:

```

Var
    a : Integer;
Var
    b : Integer;
Var
    c, d;

```

Types of parameters in procedure/function declaration are ignored and can be skipped. For example, this code is correct:

```

Function Sum(a, b) : Integer;
Begin
    Result := a + b;
End;

```

## ***DelphiScript Reference***

In general, you can use variants to store any data type and perform numerous operations and type conversions. A variant is type-checked and computed at run time. The compiler won't warn you of possible errors in the code, which can be caught only with extensive testing. On the whole, you can consider the code portions that use variants to be interpreted code, because, many operations cannot be resolved until run time. This affects in particular the speed of the code.

Now that you are aware of the use of the Variant type, it is time to look at what it can do. Basically, once you've declared a variant variable such as the following:

```
Var
    V;
Begin
    // you can assign to it values of several different types:
    V := 10;
    V := 'Hello, World';
    V := 45.55;
End;
```

Once you have the variant value, you can copy it to any compatible-or incompatible-data type. If you assign a value to an incompatible data type, Delphiscript interpreter performs a conversion, if it can. Otherwise it issues a run-time error. In fact, a variant stores type information along with the data, and thus a Delphiscript is slower than a Borland Delphi compiled code.

### **Functions / procedures inside a function or procedure**

It is recommended that you write standalone functions or procedures (recursive procedures/functions are permitted although).

This function snippet is not recommended.

```
Function A
    Function B
    Begin
        // blah
    End;
Begin
    B;
End;
```

### **Recommended function structure**

```
Function B
Begin
    // blah
End;
```

```
Function A
Begin
    B;
End;
```

### Result keyword

You can't use the function name to set the return value within a function block, use Result to do so.

#### Example

```
Function Foo : String;
Begin
    Result := 'Foo Foo';
End;
```

### Nested Routines

Nested routines are supported but you can't use variables of top level function from the nested one.

### Array elements

Type of array elements is ignored and can be skipped so these declarations are equal:

```
Var
    x : array [1..2] of double;
Var
    x : array [1..2];
```

You cannot declare array types but you can declare arrays to variables

#### Illegal example

```
Type
    TVertices = Array [1..50] Of TLocation;
Var
    NewVertices : TVertices;
```

#### Legal example

```
Var
    NewVertices : Array [1..50] of TLocation;
```

### Open array declaration

The Open array declaration is not supported.

### Case keyword

## ***DelphiScript Reference***

The case keyword can be used for any type. So you can write

```
Case UserName of
    'Alex', 'John' : IsAdministrator := true;
    'Peter' : IsAdministrator := false;
Else
    Raise('Unknown user');
End;
```

### **Class declarations**

You cannot define new classes, but you can use existing DelphiScript classes and instantiate them. For example TList and TStringList classes can be created and used in your scripts.

### **See also**

TList

TStringList

### **CreateObject function**

The CreateObject function can be used to create objects that will be implicitly freed when no longer used. So instead of

```
Procedure Proc;
Var
    l;
Begin
    l := TList.Create;
    Try
        // do something with l
    Finally
        L.Free;
    End;
End;
```

you can write

```
Procedure Proc;
Var
    l;
Begin
    l := CreateObject(TList);
    // Do something with l
End;
```

**See also**

CreateObject keyword

**Raise exceptions**

Raise can be used without parameters to re-raise the last exception. You can also use Raise with string parameter to raise the exception with the specified message string. The Exception objects are not supported, because the On keyword is not supported.

**Example**

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

**See also**

Try keyword

Finally keyword

Raise keyword

**ThreadVar**

The Threadvar keyword is treated as Var. Note that in Object Pascal, the variables declared using Threadvar have distinct values in each thread.

**Set operator**

The Set operator In is not supported. You can use InSet to check whether a value is a member of set. For example,

```
If InSet(fsBold, Font.Style) then
    ShowMessage('Bold');
```

Note, that set operators '+', '-', '\*', '<=', '>=' don't work correctly. You have to use logical operators.

**Example**

```
ASet := BSet + CSet; should be changed to
ASet := BSet or CSet;
```

The [...] set constructors are not supported. You can use MkSet to create a set.

**Example**

```
Font.Style := MkSet(fsBold, fsItalic);
```

**See also**

MkSet keyword

InSet keyword

**Operators**

^ and @ operators are not supported.

**Directives**

The following directives are not supported (note that some of them are obsolete and aren't supported by Delphi too): absolute, abstract, assembler, automated, cdecl, contains, default, dispid, dynamic,

## DelphiScript Reference

export, external, far, implements, index, message, name, near, nodefault, overload, override, package, pascal, private protected, public, published, read, readonly, register, reintroduce, requires, resident, safecall, stdcall, stored, virtual, write, writeonly.

Note, the "in" directive in uses clause is ignored.

### Ignored Keywords

The interface, implementation, program and unit keywords are ignored in Delphiscript. The scripts can have them but they have no effect but these keywords can enhance the readability of scripts.

### Unsupported Keywords

The following keywords are not supported in Delphiscript:

- as, asm, class, dispinterface, exports, finalization, inherited, initialization, inline, interface, is, library, object, out, property, record, resourcestring, set, supports, type.

The following Delphi RTL functions aren't supported in DelphiScript:

- Abort, Addr, Assert, Dec, FillChar, Finalize, Hi, High, Inc, Initialize, Lo, Low, New, Ptr, SetString, SizeOf, Str, UniqueString, VarArrayRedim, VarArrayRef, VarCast, VarClear, VarCopy.

The functions from the Borland Delphi's Windows unit (windows.pas file) are not supported (for example the RGB function is not supported).

## DelphiScript Error Codes

---

Error	Description
%s expected but %s found	Wrong string used in the script..
%s or %s expected	Wrong string used in the script.
Function %s is already defined	Multiple instances of functions with same name in the code are not permitted. Rename the other functions that have the same name.
Unknown identifier: %s	Unknown identifier. Need to declare this identifier first before using this identifier.
Unknown variable type during writing program	The script has a Variable type which is not valid or unknown.
Unit %s already defined	Multiple instances of same unit names are not permitted. Ensure script unit names are unique.
Unit declaration error	The unit declaration is not properly defined.
Function %s not found	Missing function in the script.
Link Error	DelphiScript is unable to link the script to the required internal components.
Label <%s> already	Multiple instances of the same label exist in the script. Ensure labels are

Error	Description
defined	unique in the script.
Error in declaration block	The declaration block is not defined correctly.
Label <%s> not defined	The Goto label is not defined.
Variable <%s> already defined	Multiple instances of the same variables exist in the script. Ensure variables are unique.
Error in variable declaration block	Error exists in the variable declaration block. Wrong declarations or declarations not recognized by the scripting system.
Variable <%s> not defined	Variable was not defined, so the scripting system cannot define this variable.
Method declaration error	Method signature is illegal.
Method parameters declaration error	Wrong parameters used for the method.
Properties are not supported	Properties of an object not recognized by the scripting system.
Only class declarations allowed	Declarations other than classes were attempted to be declared.
%s declaration error	Declaration error exists in the script.
Syntax error at Line: %d Char: %d'#13#10'%s	A syntax error has occurred on the script - illegal statement, missing character or unrecognized keyword.
Bad identifier name <%s>	Invalid identifier name such as duplicated identifier name. Redefine the identifier name.
Bad identifier <%s>	Invalid identifier. Redefine a new identifier
Invalid function usage	Function not used correctly in the script - such as invalid parameters.
Invalid procedure usage	Procedure not used correctly in the script - such as invalid parameters.
Hex constant declaration error	Hex constant value not declared correctly.
Compile before run	The script needs to be compiled first before it can be executed. An internal error.
Real constant declaration error	Real type constant declaration error.
String constant	String type constant declaration error.

Error	Description
declaration error	
Unsupported parameter type	Unknown parameter type as reported by the scripting system.
Variable Result not found for %s	Variable value result not found for the specified string in the script.
Procedure %s not found	Missing procedure in the script.
Parameter %S not found	Missing parameter in the script.
Unknown reader type	An internal error.
Wrong number of params	The same procedure or function declared and implemented differently. Check the parameters between the two.
One of the params is not a valid variant type	One of the parameters of a method, function or procedure does not have a correct variant type.
Property does not exist or is readonly	An attempt to set a value to the read only property or a property does not exist.
Named arguments are not supported	Arguments used for the procedure or function not valid for the script.
Parameter not found	Missing parameter value.
Parameter type mismatch	Wrong parameter type used.
Unknown interface	This interface is not declared or defined.
A required parameter was omitted	Missing parameter required for the method, function or procedure.
Unknown error	DelphiScript has detected an unknown script error that is not defined in the internal errors table.
Invalid operation code	DelphiScript has detected an invalid operation code.

## Expressions and Operators

An expression is a valid combination of constants, variables, literal values, operators and function results. Expressions are used to determine the value to assign to a variable, to compute the parameter of a function, or to test for a condition. Expressions can include function calls.

DelphiScript has a number of logical, arithmetic, Boolean and relational operators. Since these operators are grouped by the order of precedence which is different to the precedence orders used by Basic, C etc. For example, the AND and OR operators have precedence compared to the relational one.

For example if you write `a<b and c<d`, the DelphiScript will do the AND operation first, resulting in an error. To fix this problem, you have to enclose each of the `<` expression in parentheses: `(a<b) and (c<d)`;

These operators listed below are the operators supported by DelphiScript.

### Operators grouped by precedence

Unary operators have the highest precedence

Not	Boolean or bitwise NOT.
-----	-------------------------

### Multiplicative and Bitwise Operators

*	Arithmetic multiplication.
/	Floating point division.
div	Integer division.
mod	modulus (remainder of integer division).
and	Boolean or bitwise AND.
shl	Bitwise left shift.
shr	Bitwise right shift.

### Additive Operators

+	Arithmetic addition, string concatenation.
-	Arithmetic subtraction.
or	Boolean or bitwise OR
xor	Boolean or bitwise EXCLUSIVE OR.

### Relational and Comparison Operators (lowest precedence)

=	Test whether equal or not.
<>	Test whether not equal or not.

## ***DelphiScript Reference***

<	Test whether less than or not.
>	Test whether greater than or not.
<=	Test whether less than or equal to or not.
>=	Test whether greater than or equal to or not.

### **Note**

The ^ and @ operators are not supported by DelphiScript.

## DelphiScript Functions

---

A few statements used by the DelphiScript language are covered here. The range of functions are covered in the FileIO routines, Math Routines, String Routines and Extension routines.

In this DelphiScript Functions Section:

- Calculating expressions with the evaluate function
- Passing parameters to functions and procedures
- Exiting from a procedure
- File IO routines
- Math routines
- String routines
- Extension routines
- Using sets in DelphiScript
- Using exception handlers
- DelphiScript Language.

### Calculating expressions with the evaluate function

The built in function Evaluate can be used to interpret a string as a script code during runtime of a script and execute the code contained in the string. For example, you can write script like `Evaluate(ProcNames[Proclndex]);` and the procedure which name is specified in `ProcNames[Proclndex]` will be called.

To calculate such an expression you can use Evaluate method, where expression is specified by Expr parameter. For example, you can calculate expressions like the following:

```
Evaluate('2+5');  
Evaluate('((10+15)-5)/2*5');  
Evaluate('sin(3.1415926/2)*10');  
Evaluate('2.5*log(3)');
```

### Passing parameters to functions and procedures

Both functions and procedures you define in a script can be declared to accept parameters. Additionally, functions are defined to return a value.

Types of parameters in procedure/function declaration are ignored and can be skipped. For example, this code is correct:

```
Function sum(a, b) : integer;  
Begin  
    result := a + b;  
End;
```

## Exiting from a procedure

DelphiScript provides Exit and Break statements, should you want you exit from a procedure before the procedure would terminate naturally. For example; if the value of a parameter is not suitable, you might want to issue a warning to the user and exit, as example below shows.

```
Procedure DisplayName (s);
Begin
    If s = '' Then
        Begin
            ShowMessage('Please enter a name');
            Exit;
        End;
    ShowMessage(S + ' is shown');
End;
```

## File IO routines

DelphiScript has the following IO routines

- Append
- AssignFile
- ChDir
- CloseFile
- Eof
- Eoln
- Erase
- GetDir
- Mkdir
- Read
- Readln
- Reset
- Rewrite
- Rmdir
- Write
- Writeln

DelphiScript gives you the ability to write information out to a text file, since DelphiScript is an untyped language, you can only deal with strings. Thus Read/ReadLn routines are equivalent. They read a line up to but not including the next line. A Writeln(String) routine is equivalent to a Write(S) and a Write(LineFeed + CarriageReturn) routine.

To be able to write out a text file, you need to employ AssignFile, ReWrite, Writeln and CloseFile procedures. To read in a text file, you need to employ the AssignFile, Reset, Readln and CloseFile procedures. This example writes to a text file and adds an end-of-line marker.

Use of Try / Finally / End block is recommended to make scripts secure in the event of an IO failure.

**Example**

```
Var
    InputFile   : TextFile;
    OutputFile  : TextFile;
    I           : Integer;
    Line        : String;
Begin
    AssignFile(OutputFile,eConvertedFile.Text);
    Rewrite(OutputFile);

    AssignFile(InputFile,eOriginalFile.Text);
    Reset(InputFile);
    Try
        While not EOF(InputFile) do
            Begin
                Readln(InputFile,Line);
                For I := 1 to Length(Line) Do
                    Line[I] := UpperCase(Line[I]);

                Writeln(Outputfile, Line);
            End;
        Finally
            CloseFile(InputFile);
            CloseFile(OutputFile);
        End;
    End;
End;
```

## Math routines

DelphiScript has the following math routines;

- Abs
- ArcTan
- Cos
- Exp
- Frac
- Int
- Random
- Randomize
- Round
- Sin
- Sqr
- Sqrt
- Trunc

## String Routines

DelphiScript has the following string routines which can manipulate strings of characters. Only a select few routines are shown here.

- Copy
- Format
- Length
- Ord
- Pos
- Pred
- Round
- SetLength
- Succ
- Trunc
- UpCase

## Extension routines

The extension routines are used when you are dealing with processes in your scripts especially if you need to extract or set strings for the parameters of processes. Some of the routines are listed below, refer to the Altium Designer RTL and Process references for more details.

**Executing parameters of processes in your script, you might need following functions:**

- AddColorParameter
- AddIntegerParameter
- AddLongIntParameter
- AddSingleParameter
- AddWordParameter
- GetIntegerParameter
- GetStringParameter
- ResetParameters
- RunProcess

### Useful functions

- SetCursorBusy
- ResetCursor
- CheckActiveServer
- GetActiveServerName
- GetCurrentDocumentFileName
- RunApplication
- SaveCurrentDocument

### Useful Dialogs

- ConfirmNoYes
- ConfirmNoYesCancel
- ShowError
- ShowInfo
- ShowWarning

## Using sets

Since Object Pascal's Set and In keywords are not supported in DelphiScript, to overcome this limitation, you can use these two functions: MkSet and InSet. You should use logical operations to include or exclude elements to set.

**MkSet - this is a set constructor. It has variable number of arguments. You can write like**

```
Font.Style = MkSet(fsBold,fsItalic).
```

**InSet - this function is used as substitution of Object Pascal's In operator. A in B is equal to InSet(A, B).**

## ***DelphiScript Reference***

```
If InSet(A,B) then
    ShowMessage('A is in B set')
Else
    ShowMessage('A not in B set');
```

The set operators '+', '-', '\*', '<=' and '>=' don't work correctly in DelphiScript. You should use logical operators instead. For example,

```
ASet := BSet + CSet;
```

should be changed to

```
ASet := BSet or CSet;
```

in order to achieve the desired result in your script.

## **Using exception handlers**

The try keyword introduces a try-except statement or a try-finally statement. These two statements are related but serve different purposes.

### **Try Finally**

The statements in the finally block are always executed no matter how control leaves the try block exception, Exit or break. Use try-finally block to free temporary objects and other resources and to perform clean up activities. Typically you do not need more than one try-finally statement in a subroutine.

### **Example**

```
Reset(F);
Try
    ... // process file F
Finally
    CloseFile(F);
End;
```

### **Try Except**

Use try-except to handle exceptional cases for example to catch specific exceptions and do something useful with them, such as log them in an error log or create a friendly dialog box. Since the On keyword is not supported in DelphiScript, so you can use the Raise statement inside the Except block.

### **Example**

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

End;

### **Raise**

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

### **Example**

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

Note, the On keyword is not supported, thus you cannot use Exception objects as in Borland Delphi.

## Forms and Components

---

In this section:

- DelphiScript Components
- Designing script forms
- Writing Event Handlers.

### Introduction to Components

The scripting system handles two types of components: Visual and Nonvisual components. The visual components are the ones you use to build the user interface, and the nonvisual components are used for different tasks such as these Timer, OpenFileDialog and MainMenu components. You use the Timer nonvisual component to activate specific code at scheduled intervals and it is never seen by the user. The Button, Edit and Memo components are visual components for example.

Both types of components appear at design time, but non visual components are not visible at runtime. Basically components from the Tool Palette panel are object orientated and all these components have the three following items:

- Properties
- Events
- Methods

A property is a characteristic of an object that influence either the visible behavior or the operations of this object. For example the Visible property determines whether this object can be seen or not on a script form.

An event is an action or occurrence detected by the script. In a script the programmer writes code for each event handler which is designed to capture a specific event such as a mouse click.

A method is a procedure that is always associated with an object and define the behavior of an object.

All script forms have one or more components. Components usually display information or allow the user to perform an action. For example a Label is used to display static text, an Edit box is used to allow user to input some data, a Button can be used to initiate actions.

Any combination of components can be placed on a form, and while your script is running a user can interact with any component on a form, it is your task, as a programmer, to decide what happens when a user clicks a button or changes a text in an Edit box.

The Scripting system supplies a number of components for you to create complex user interfaces for your scripts. You can find all the components you can place on a form from the Toolbox palette.

To place a component on a form, locate its icon on the Tool Palette panel and double-click it. This action places a component on the active form. Visual representation of most components is set with their set of properties. When you first place a component on a form, it is placed in a default position, with default width and height however you can resize or re-position this component. You can also change the size and position later, by using the Object Inspector.

When you drop a component onto a form, the Scripting system automatically generates code necessary to use the component and updates the script form. You only need to set properties, put code in event handlers and use methods as necessary to get the component on the form working.

## Designing script forms

A script form is designed to interact with the user within the environment. Designing script forms is the core of visual development. Every component you place on a script form and every property you set is stored in a file describing the form (a DFM file) and has a relationship with the associated script code (the PAS file). Thus for every script form, there is the PAS file and the corresponding DFM file.

When you are working with a script form and its components, you can operate on its properties using the Object Inspector panel. You can select more than one component by shift clicking on the components or by dragging a selection rectangle around the components on this script form. A script form has a title (the Caption property on the Object Inspector panel).

### Creating a new script form

With a project open, right click on a project in the Projects panel, and a pop up menu appears, click on the Add New to Project item, and choose Script Form item. A new script form appears with the Form1 name as the default name.

### Displaying a script form

In a script, you will need to have a procedure that displays the form when the script form is executed. Within this procedure, you invoke the ShowModal method for the form. The Visible property of the form needs to be false if the ShowModal method of the script form is to work properly.

#### ShowModal example

```
Procedure RunDialog;
Begin
    DialogForm.ShowModal;
End;
```

The ShowModal example is a very simple example of displaying the script form when the RunDialog from the script is invoked. Note, you can assign values to the components of the DialogForm object before the DialogForm.ShowModal is invoked.

#### ModalResult example

```
procedure TForm.OKButtonClick(Sender: TObject);
begin
    ModalResult := mrOK;
end;

procedure TForm.CancelButtonClick(Sender: TObject);
begin
    ModalResult := mrCancel;
end;

Procedure RunShowModalExample;
```

Begin

```
// Form's Visible property must be false for ShowModal to work properly.  
If Form.ShowModal = mrOk      Then ShowMessage('mrOk');  
If Form.ShowModal = mrCancel Then ShowMessage('mrCancel');
```

End;

The ModalResult property example here is a bit more complex. The following methods are used for buttons in a script form. The methods cause the dialog to terminate when the user clicks either the OK or Cancel button, returning mrOk or mrCancel from the ShowModal method respectively.

You could also set the ModalResult value to mrOk for the OK button and mrCancel for the Cancel button in their event handlers to accomplish the same thing. When the user clicks either button, the dialog box closes. There is no need to call the Close method, because when you set the ModalResult method, the script engine closes the script form for you automatically.

Note, if you wish to set the form's ModalResult to cancel, when user presses the Escape key, simply enable the Cancel property to True for the Cancel button in the Object Inspector panel or insert `Sender.Cancel := True` in the form's button cancel click event handler.

### Accepting input from the user

One of the common components that can accept input from the user is the EditBox component. This EditBox component has a field where the user can type in a string of characters. There are other components such as masked edit component which is an edit component with an input mask stored in a string. This controls or filters the input.

The example below illustrates what is happening, when user clicks on the button after typing something in the edit box. That is, if the user did not type anything in the edit component, the event handler responds with a warning message.

```
Procedure TScriptForm.ButtonClick(Sender : TObject);
```

Begin

```
  If Edit1.Text = '' Then  
  Begin  
    ShowMessage('Warning - empty input!');  
    Exit;
```

```
  End;
```

```
  // do something else for the input
```

End;

Note, A user can move the input focus by using the Tab key or by clicking with the mouse on another control on the form.

### Responding to events

When you press the mouse button on a form or a component, Altium Designer sends a message and the Scripting System responds by receiving an event notification and calling the appropriate event handler method.

### See also

Writing Scripts

Writing Event Handlers

Using DXP Objects in scripts

DelphiScript Language

HelloWorld project from the \Examples\Scripts\DelphiScript Scripts\General\ folder.

ShowModalEg script within the General\_Scripts project from the \Examples\Scripts\DelphiScript Scripts\General\ folder.

## Writing Event Handlers

Each component, beside its properties, has a set of event names. You as the programmer decide how a script will react on user actions in Altium Designer. For instance, when a user clicks a button on a form, Altium Designer sends a message to the script and the script reacts to this new event. If the OnClick event for a button is specified it gets executed.

The code to respond to events is contained in DelphiScript event handlers. All components have a set of events that they can react on. For example, all clickable components have an OnClick event that gets fired if a user clicks a component with a mouse. All such components have an event for getting and loosing the focus, too. However if you do not specify the code for OnEnter and OnExit (OnEnter - the control has focus; OnExit - the control loses focus) the event will be ignored by your script.

Your script may need to respond to events that might occur to a component at run time. An event is a link between an occurrence in Altium Designer such as clicking a button, and a piece of code that responds to that occurrence. The responding code is an event handler. This code modifies property values and calls methods.

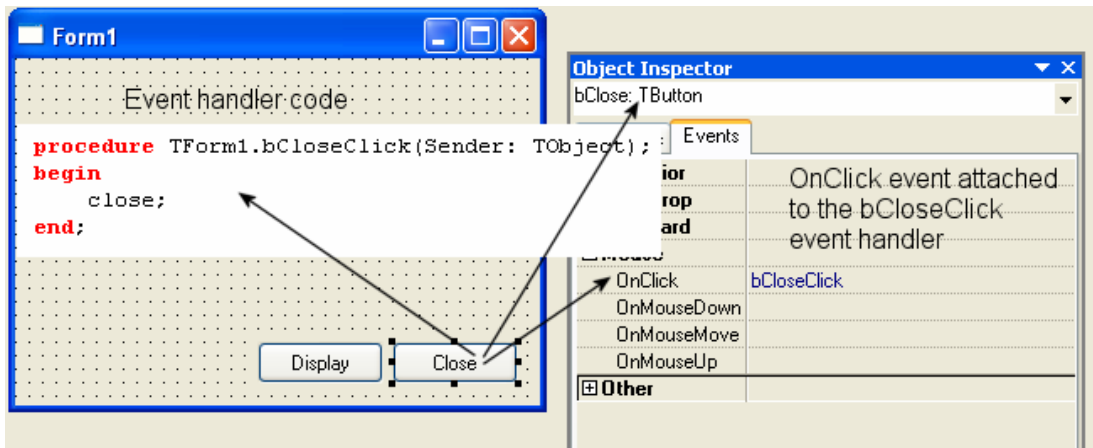
### List of properties for a component

To see a list of properties for a component, select a component and in the Object Inspector, activate the Properties tab.

### List of events for a component

To see a list of events a component can react on, select a component, and in the Object Inspector activate the Events tab. To create an event handling procedure, decide on what event you want your component to react, and double click the event name.

For example, select the Button1 component from the Toolbox panel and drop it on the script form, and double click next to the OnClick event name. The scripting system will bring the Code Editor to the top of the Altium Designer and the skeleton code for the OnClick event will be created.



For example, a button has a Close method in the CloseClick event handler. When the button is clicked, the button event handler captures the on click event, and the code inside the event handler gets executed. That is, the Close method closes the script form.

In a nutshell, you just select a button component, either on the form or by using the Object Inspector panel, select the Events page, and double click on the right side of the OnClick event, a new event handler will appear on the script. OR double click on the button and the scripting system will add a handler for this OnClick event. Other types of components will have completely different default actions.

### List of methods for a component

To see a list of methods for a component, see the DelphiScript Component Reference.

## Using Components in your scripts

### Dropping components on a script form

To use components from the Tool Palette panel in your scripts, you need to have a script form first before you can drop components on the form. Normally when you drop components on a script form, you do not need to create or destroy these objects, the script form does them for you automatically.

The scripting system automatically generates code necessary to use the component and updates the script form. You then only need to set properties, put code in event handlers and use methods as necessary to get the script form working.

### Creating components from a script

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

For example, you can create and destroy Open and Save Dialogs (TOpenDialog and TSaveDialog classes as part of Borland Delphi Run Time Library).

## Index

---

### A

About Example Scripts .....	6
Adding scripts to a project .....	3
Assigning script to a resource in Altium Designer ....	5

### C

Calculating expressions with the evaluate function	25
Case sensitivity .....	11
Components .....	32
Creating new scripts .....	3

### D

DelphiScript Error Codes .....	20
DelphiScript naming conventions .....	8
DelphiScript source files .....	2
DelphiScript's Built in Functions .....	25
Designing script forms .....	33
Differences between DelphiScript and Object Pascal .....	15

### E

Exiting a procedure .....	26
Exploring DelphiScript .....	1
Expressions and Operators .....	23
Extension routines .....	29

### F

File IO routines .....	26
Forms and Components .....	32
Functions and procedures in a script .....	11

### I

Including comments in scripts .....	8
Introduction .....	1

### L

Local and Global Variables .....	9
----------------------------------	---

### M

Math routines .....	28
---------------------	----

### P

Passing parameters to functions and procedures .	25
--	----

### R

Running a script in Altium Designer .....	3
---	---

### S

Splitting a line of script .....	10
String Routines .....	28

### T

The space character .....	11
Tips on writing scripts .....	14
Try Keyword .....	30

### U

Using DXP Objects in scripts .....	12
Using named variables in a script .....	10
Using sets in DelphiScript .....	29

### W

Writing DelphiScript Scripts .....	8
Writing Event Handlers .....	35

## Revision History

---

Date	Version No.	Revision
01-Dec-2004	1.0	New product release
26-Apr-2005	1.1	Updated for Altium Designer
09-Dec-2005	1.2	Updated for Altium Designer 6
03-Mar-2006	1.3	Formatting and Images revised for Altium Designer 6
17-Mar-2006	1.4	Removed DelphiScript keywords and statements.

Software, hardware, documentation and related materials:

Copyright © 2007 Altium Limited.

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