

Summary

This reference covers the DelphiScript keywords used for the Scripting System in Altium Designer.

The scripting system supports the DelphiScript language which is very similar to Borland Delphi[™]'s Object Pascal language. The key difference is that DelphiScript is a typeless scripting language. In this reference, DelphiScript keywords are outlined with concise information and examples.

For more detailed information on the differences between DelphiScript and Object Pascal, refer to the [DelphiScript Reference](#) document.

Navigate to the Scripting resources via **Configuring the System » Scripting in Altium Designer** from the *Knowledge Center* panel in Altium Designer.

Reserved words in DelphiScript

A, B

And, Array, Begin, Break

C,D

Case, Const, Continue, Delete, Div, Do, DownTo

E

Else, End, Except

F,G

Finally, For, Forward, Function, Goto

I, L, M

If, Interface, Label, Mod

N, O, P

Nil, Not, Of, Or, Procedure, Program

S, T

Raise, Read, ReadLn, Repeat, Shl, Shr, String, Then, To, Try, Type

U, W, X

Unit, Until, Uses, Var, While, With, Xor

And

Declaration

And operator performs a logical/bitwise and.

Description

The and operator performs a logical and if the operators are of boolean type, or a bitwise and if the operators are integers.

Example of a boolean and evaluation

```
Var
    I, J : Integer
Begin
    I := $F0;
    J := $8F;
    ShowMessage(IntToStr(I and J));
End;
```

Example of a logical and evaluation

```
Var
    S : String;
Begin
    S := '';
    If (Length(S) > 0) and (S[1] = 'X') Then Delete(S,1,1);
End;
```

See also[Or keyword](#)[Xor keyword](#)

Array

Declaration

```
Array [index range];
```

Description

DelphiScript language is a type-less language so you need to specify the size or range of an array. You can still define what type of elements the array can hold, but it is not necessary. The Open array declaration is not supported.

Example

```
var x : array [1..2];
```

Begin

Declaration

```
Begin
    statement
End;
```

Description

The `begin` keyword starts a block in the script. A block is the main body of a script and can enclose any number of statements and can be used anywhere a single statement is required, such as the body of a conditional or loop statement.

Example

```
Var
    Test : Integer;
Begin
    Test := Test + 4;
    ShowMessage(IntToStr(Test));
End;
```

See also[End keyword](#)

Break

Declaration

```
Break;
```

Description

The `Break` jumps out of a loop which is similar to a `Goto` statement.

Example`

```
While Condition Do
Begin
    DoSomething;
    Begin
        If AnotherCondition Then
```

```

        Break;

    End;

End;

```

See also[While keyword](#)[Continue keyword](#)[Do keyword](#)[Repeat keyword](#)

Case

Declaration

```

Case expression Of
    Value range : Expression;
    Else Expression;

End;

```

Description

The `case` statements select one branch out of many possible branches depending on the value of the expression.

If you have very complex `if` statements, at times you can replace them with `case` statements. A `case` statement in an expression is used to select a value, a list of possible values or a range of values. Any types can be used in a `case` statement because DelphiScript is an un-typed language. `Case` statements can have an `else` statement that is executed if none of the labels correspond to the value of the selector (within the `Case Of` condition).

Example 1

```

Case Char Of
    '+'      : Text := 'Plus sign';
    '-'      : Text := 'Minus sign';
    '*', '/' : Text := 'Multiplication or division';
    '0'..'9' : Text := 'Number';
    'a'..'z' : Text := 'Lowercase character';
    'A'..'Z' : Text := 'Uppercase character';
    Else
        Text := 'Unknown character';

End;

```

Example 2

```

Case UserName Of
    Jack, 'Joe' : IsAdministrator := true;
    'Fred' : IsAdministrator := false;
    Else
        Raise('Unknown User');

End;

```

See also[Of keyword](#)

Continue

Declaration

```
Continue
```

Description

The `Continue` statement jumps over the body of a loop, similar to the `Goto` statement;

Example

```

Var
    I := 0; s:= 1;
Begin
    While True Do
    Begin
        S := S * 2;
        I := I + 1;
        If I <> 4 then continue;
        Break;
    End;
End;

```

See also[Break keyword](#)[While keyword](#)[Continue keyword](#)[Do keyword](#)[Repeat keyword](#)

Const

Declaration

```

Const
    Name = Expression;

```

Description

The **Const** keyword specifies any constant valued expression as the value of a constant. If you try to modify the expression that is of a const type in the script, you will get an undeclared identifier error in the scripting system.

Example

```

Const
    b = 30;
Begin
    ShowMessage(IntTtStr(b));
End;

```

If you try to change the value of the b const parameter, you will get an error, for example,

```

Const
    b = 30;
Begin
    b := 40;
    ShowMessage(IntToStr(b));
End;

```

Div

Declaration

```

dividend div divisor

```

Description

The **Div** operator performs integer division which discards fractional results without rounding. If the divisor is zero, DelphiScript reports an error.

See also[Mod Operator](#)

Div Operator

Do

Declaration

- For variable := expression1 to expression2 do statement
- While expression do statement
- With expression do statement.

Description

The **Do** keyword is part of DelphiScript's **For**, **While** and **With** statements.

Example

```
For i := 0 To AnIndex - 1 Do  
    S := S + #13 + AString;
```

See also

[For keyword](#)

[To keyword](#)

[While keyword](#)

[With keyword](#)

[DownTo keyword](#)

DownTo

Declaration

For variable := expression1 DownTo expression2 Do statement.

Description

Use **DownTo** in a **For** loop to count down.

See also

[For keyword](#)

[To keyword](#)

[Do keyword](#)

Else

Declaration

- If condition then statement Else statement
- Try statement except exception Else statement end
- Case expression of Else end;

Description

The **Else** keyword introduces the catch all part of several statements. Note that the **else** part of an **if** statement is followed by a single statement, but the **else** part of the **try-exception** and **case** statements can have multiple statements.

See also

[If keyword](#)

[Then keyword](#)

[Try keyword](#)

[Case keyword](#)

End

Declaration

```
Begin statements End;
```

```
Try statements Except Exception clauses... else Statements... End;
```

```
Try statements Finally statements End;  
Case Expression of clauses Else statements... End;
```

Description

The `End` keyword ends a block or a multiple part such as declarations, `Case` statements and so on.

See also

[Begin keyword](#)

[Case keyword](#)

[Try keyword](#)

Except

Declaration

```
Try statements Except statements End;
```

Description

Use `Try-Except` blocks 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, use the `Raise` statement inside the `Except` block and only report a textual message.

Example

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

See also

[End keyword](#)

[Finally keyword](#)

[Try keyword](#)

Finally

Declaration

```
Try statements... Finally statements... End;
```

Description

The `finally` keyword starts the `finally` part of a `try-finally` block. The statements in the `finally` block always run, no matter how the control leaves the `try` block exception, exit or break. The use of `try-finally` block is recommended when dealing with creation/destruction of objects and File IO.

See also

[End keyword](#)

[Raise keyword](#)

[Try keyword](#)

For

Declaration

- `for variable := expression1 to expression2 do statement`
- `for variable := expression1 downto expression2 do statement`

Description

A `for` loop evaluates the expressions that specify the limits of this loop, then performs the loop body repeatedly via the loop control variable which is updated after each iteration.

Example

```
For i := 0 to AnIndex - 1 Do
```

```

Begin
    S := S + #13 + AString;
End;
ShowMessage(S);

```

See also[Do keyword](#)[DownTo keyword](#)[Repeat keyword](#)[To keyword](#)[While keyword](#)[With keyword](#)

Forward

Declaration

```
subroutine header; forward;
```

Description

This `Forward` directive lets you declare a function or procedure before you call it by declaring the header (name, parameters, and return type) with the `forward` directive.

Function

Declaration

```
Function name (parameters) : return type;
```

Description

A `Function` is a subroutine that returns a value. Note that pointers to `functions` are not permitted in scripts, i.e. you cannot define functional types. Variables declared inside a `function` are not accessible outside this procedure.

Example

```

Function TestFunc(Min, Max : integer) : integer;
Begin
    Result := Random(Max - Min +1);
End;

```

Goto

Declaration

```
goto label
```

Description

The `goto` statement transfer control to the given label. The label can be any identifier or a digit string with up to four digits.

Example

```

Label StartHere;
    // code

StartHere: //do anything;

Goto StartHere;

```

See also[Label keyword](#)

If

Declaration

- if condition then statement;
- if condition then statement1 else statement2;

Description

The condition for the `If` keyword must be a boolean expression. The `Else` keyword is optional.

Example

```
If X > Y Then
    If A > B Then
        ShowMessage('X>Y and A > B');
    Else
        ShowMessage('X>Y and A <=B');
End;
```

See also

[And keyword](#)

[Begin keyword](#)

[Or keyword](#)

[Then keyword](#)

[Else keyword](#)

Interface

Declaration

```
Interface
// Globally unique identifier string.
Methods
Properties
End;
```

Description

The `interface` keyword enables you to have access to an existing object in memory and invoke the object's methods. An `interface` can only consist of properties and methods - no data. Since interfaces cannot contain data, their properties must write and read to and from methods. Most importantly interfaces have no implementation as they only define a contract (to an existing object in memory).

Think of an `interface` as a contact point to an existing object in computers memory and you have the ability to read/write data through properties of the `interface`. The `interface` requests for data from its associated object.

DelphiScript is a type-less language, therefore you cannot define new types such as new records, arrays or classes and associated interfaces as well.

Beware of the other use of the `Interface` keyword which is used for the `Interface / Implementation` sections of a Borland Delphi's unit. These `Interface/Implementation` keywords can be used in scripts but they are essentially ignored when a script is being executed in Altium Designer.

Label

Declaration

```
label digits, identifier, ...;
```

Description

The `label` keyword declares one or more labels. A `label` can be digit string with up to four digits or an identifier. A `label` can be used in the same block to identify a statement as the target of a `goto` statement.

Example

```
Label StartHere;
    // code
StartHere: //do anything;
Goto StartHere;
```


See also

Goto keyword. (create hyperlinks)

Mod**Declaration**

```
Integer expression mod integer expression.
```

Description

The `mod` operator performs an integer modulus or remainder operation. The result of `A mod B` is `A - (A div B) * B`.

See also

[Div function](#)

Nil**Declaration**

```
const nil = pointer(0);
```

Description

The `nil` keyword is a special pointer value that is guaranteed to be distinct and pointing to nothing.

Not**Declaration**

- not boolean expression
- not integer expression.

Description

The `not` operator performs a negation. If the operand has type boolean, the negative is a logical negation. `Not False = True` and `not true = false`. If the operand is an integer, the `not` operator performs a bitwise negation of each bit in the integer value, ie a complement operation.

Of**Declaration**

```
case expression of
selector: expression1
...
end
```

Description

The `Of` keyword is used for the case statement.

See also

[Case statement](#)

Or**Declaration**

- boolean expression or boolean expression
- integer expression or integer expression

Description

The `or` operator performs a logical `or` if the operands are of boolean type or a bitwise `or` if the operators are integers. A logical `or` is false only if both operands are false otherwise it is true when at least one operand is true.

See also

[And keyword](#)

[Not keyword](#)

[Shl keyword](#)

[Shr keyword](#)[Xor keyword](#)

Procedure

Declaration

- Procedure name;
- Procedure Name (Parameter, ...);

Description

The `procedure` keyword declares a subroutine that does not have a return type. Variables declared inside a `procedure` are not accessible outside this `procedure`.

Note this keyword can be used but it is ignored by the scripting system.

Example

```
Procedure TestRand(Var Rand: Integer; Max : Integer);
Begin
    Rand := Random(Max);
End;
```

See also

[Function keyword](#)

Program

Declaration

```
Program Name;
declarations...
Block.
```

Description

The `program` keyword begins a script. The file extension for a script is `*.pas`. Note this `keyword` can be used but it is ignored by the scripting system.

See also

[Function keyword](#)

Raise

Declaration

```
Raise statement;
```

Description

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, therefore you cannot use Exception objects in your scripts.

Repeat

Declaration

```
repeat
    statements;
until boolean expression
```

Description

The statements inside a `Repeat Until` block are executed repeatedly until the boolean expression is true.

Example

Repeat

```
Write('Enter a value (0..9): ');
```

```
ShowMessage(IntToStr(I));
```

```
Until (I >= 0) and (I <= 9);
```

See also

[Until keyword](#)

Result

Declaration

```
Var result : Function return type;
```

Description

Every function in a script must use the `Result` keyword to return a resultant value. The variable type is the return type of the function.

See also

[Function keyword](#)

Shl

Declaration

```
value shl bits
```

Description

The `shl` operator performs a left shift of an integer value by `Bits` bit positions. The vacated bits are filled on the right with zero bits.

See also

[And keyword](#)

[Not keyword](#)

[Or keyword](#)

[Shr keyword](#)

[Xor keyword](#)

Shr

Declaration

```
value shr bits
```

Description

The `shr` operator performs a right shift of an integer value by `Bits` bit positions. The vacated bits are filled on the left with zero bits.

See also

[And keyword](#)

[Not keyword](#)

[Or keyword](#)

[Shl keyword](#)

[Xor keyword](#)

String

Declaration

- type string;
- type Name = string[Constant]

Description

The `string` keyword represents the string type.

Then

Declaration

If expression then statement.

Description

The `Then` keyword is part of an `If` statement.

See also

[If keyword](#)

To

Declaration

For variable := expression1 to expression2 do statement.

Description

The `to` keyword is part of a `for` loop that counts up.

Example

```
For i := 0 to AnIndex - 1 do
    S := S + #13 + AString;
```

See also

[Downto keyword](#)

[For keyword](#)

Try

Declaration

- Try statements finally statements end;
- Try statements except statements end;

Description

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, use the `Raise` statement inside the `Except` block.

Example

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

See also[Raise keyword](#)

Type

Declaration

```
Type Name = type declaration ...
```

Description

The `Type` keyword declares the type for a variable. Since DelphiScript is a typeless language, it is not necessary for you to declare variables of specific type. You can do so for the sake of readability in your scripts. All variables in a script are always of Variant type. The major limitation in writing scripts is that you cannot declare records or classes.

Finally, typecasting is ignored in scripts. Types in variables declaration are ignored and can be skipped, so these declarations are correct:

Example

```
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;
```

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 once you've declared a variant variable such as the following:

Example

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

See also[Var keyword](#)

Unit

Declaration

- Unit Name;

interface

declarations

implementation

declarations

statements

Initialization

statements

finalization

statements.

end.

- Unit Name;

interface

declarations

implementation

declarations

statements

begin

statements

end.

The `unit` keyword introduces a `unit` which is the basic module for a script. Note this keyword can be used but it is ignored by the scripting system.

See also

[Function keyword](#)

[Program keyword](#)

Until

Declaration

Repeat

Statements;

Until boolean expression

Description

The `until` keyword marks the end of the `Repeat - Until` block. The statements inside a `Repeat Until` block are executed repeatedly until the boolean expression is true.

Example

Repeat

Write('Enter a value (0..9): ');

ShowMessage(IntToStr(I));

Until (I >= 0) and (I <= 9);

See also

[Repeat keyword](#)

Uses

Declaration

Uses Unit Name, ...;

Description

The `uses` keyword lists the names of units that are imported into the surrounding unit. The `uses` declaration is optional because the scripting system has supported units that are imported in Altium Designer. You can include the `uses` declaration for the sake of readability.

All units stored within the same project can access global variables from any of these units. Keep this in mind when you are declaring variables in your units within the same project.

At this time of writing, Altium Designer's Client, Nexus / FPGA, PCB, Schematic and WorkSpace Manager APIs and Borland Delphi's SysUtils, Classes and other units are imported and available for use in scripts - so there is no need to declare these units in your scripts.

See also

 For more detailed information on the Altium Designer RTL, refer to the [Using the Altium Designer RTL](#) document.

Var

Declaration

Name : Type

Name : Type = Expression;

DelphiScript Variables

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;
```

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;
```

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 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;
```

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];
```

Illegal array example

```
Type
```

```
    TVertices = Array [1..50] Of TLocation;
```

```
Var
```

```
    NewVertices : TVertices;
```

Legal array example

```
Var
```

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

While

Declaration

```
while expression do statement
```

Description

The `while` statement repeatedly executes the statement while the expression is true.

See also

[Break keyword](#)

[Continue keyword](#)

[Do keyword](#)

[DownTo keyword](#)

[For keyword](#)

[Repeat keyword](#)

[To keyword](#)

[With keyword](#)

With

Declaration

```
with expression do statement
```

Description

The `With` statement adds a record, object, class or interface reference to the scope for resolving symbol names.

Normal version example

```
Form.Canvas.Pen.Width := 2;  
Form.Canvas.Pen.Color := clSilver;
```

With version example

```
With Form.Canvas.Pen do  
Begin  
    Width := 2;  
    Color := clSilver;  
End;
```

See also

[Do keyword](#)

Xor

Declaration

- boolean expression Xor boolean expression
- integer expression Xor integer expression

Description

The `xor` operator performs an exclusive or on its operands. If the operands are of boolean type, it returns a boolean result: True if the operands are different and false if they are the same.

An integer `xor` operates on each bit of its operands, setting the result bit to 1 if the corresponding bits in both operands are different and to 0 if both operands have identical bits. If one operand is smaller than the other, the smaller operand is extended with 0 in the left most bits.

See also

[And keyword](#)

[Not keyword](#)

[Or keyword](#)

[Shl keyword](#)

[Shr keyword](#)

Revision History

Date	Version No.	Revision
06-Mar-2006	1.0	New product release
17-Mar-2006	1.1	Revised for Altium Designer. Pure Keywords only.
17-Jan-2008	1.2	Updated for Altium Designer 6.9
02-Mar-2008	1.3	Page Size to A4
16-Mar-2011	-	Updated template.

Software, hardware, documentation and related materials:

Copyright © 2011 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, DXP, Innovation Station, LiveDesign, NanoBoard, NanoTalk, OpenBus, 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.