VB Script Reference

Summary

This reference manual describes the VB Script language used in Altium Designer.

This reference covers the following topics:

- *Exploring the VB Script Language*
- *VB Script source files*
- *About VB Script examples*
- *Using Altium Designer RTL in VBScript Scripts*
- *Writing VB Script scripts*
- *VB Script keywords, statements and functions*
- *Using Components in Script Forms*

Exploring the VB Script Language

This Reference details each of the VisualBasic Scripting statements, functions and extensions that are supported in the scripting system. The Visual Basic Scripting or VB Script for short can deal with Altium Designer Object Models and Visual Components. It is assumed that you are familiar with basic programming concepts and the basic operation of your Altium Designer-based software.

The scripting system supports the VB Script language (along with other scripting languages) which is derived from the Microsoft ActiveX Scripting system, for instance you should be able to use CScripts or WScripts which are based on the same ActiveX scripting engine that Altium Designer uses.

All scripting languages supported in Altium Designer are typeless or untyped, meaning you cannot define records or classes and pass pointers as parameters to functions.

VB Script script example

```
Sub DisplayName (sName)
    MsgBox "My Name is " & sName
End Sub
```

For detailed information on VB Script and its keywords, operators and statements, please refer to Microsoft Developers Network website, *http://msdn2.microsoft.com/en-us/library/d1wf56tt(VS.71).aspx*.

Altium Designer and Borland Delphi Run Time Libraries

The Scripting system supports a subset of Borland Delphi Run Time Library (RTL) and a subset of Altium Designer RTL.

There are several Object Models in Altium Designer; such as the PCB Object Model which you can use to deal with PCB objects on a PCB document, and the WorkSpaceManager Object Model, which allows you to work with Projects and their documents and extract netlist data for example.

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

The Scripting Online Reference Help contains information on interfaces with respect to Altium Designer Object Models, components, global routines, types, and variables that make up this scripting language. You can consult the Visual Basic documentation by Microsoft for more information on VB Script functions.

Server Processes

A script can execute server processes (which represent commands in Altium Designer). The server processes and parameters are covered in the *Server Process Reference* document.

VB Script Source Files

. 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 DXP menu.

PRJSCR, VBS and DFM files

Scripts are organized into projects with a `*.PRJSCR` file extension. Each VB Script project consists of files with a `*.vbs` extension. Files can be either script units or script forms. Note that a script form has a VB Script file with `*.vbs` extension and a corresponding script form with a `*.dfm` extension). A script form is a graphical window that hosts different controls that run on top of Altium Designer.

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

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.

About VB Script Examples

The examples that follow illustrate the basic features of VB Script programming in Altium Designer. The VB Scripts can use script forms, script units, functions and objects from the Altium Designer Run Time Library and a subset of functions and objects from the Borland Delphi that is exposed in the scripting system.

The example scripts are organized into the `\Examples\Scripts\VB Scripts\` folder.

Writing VB Script Scripts

In this section:

- *VB Script Naming Conventions*
- *Local and Global Variables*
- *Subroutines and Functions*
- *Splitting a Line of Script.*

VB Script naming Conventions

VB Script variables are case insensitive, that is, variables in upper and lower case have the same meaning:

Example

The variables b and B are the same.

```
b = 60
B = 60
```

Local and Global Variables

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

If variables are defined inside a routine, then these local variables are not accessible outside these routines. Since scripts are typeless, you do not initialize variables with their types.

Variable Initialization

The local variables inside a procedure are automatically initialized.

```
Sub Example
    Dim X
    Dim s
    ' x set to 0
    x = 0
    ' s set to empty
    s = ""
End Sub
```

Subroutines and Functions

VB Script allows two kinds of procedures; subroutines and functions. A function returns a value only. The syntax of calling a subroutine or a function in a script is as follows:

```
Call SubRoutineA(parameters)
```

or

```
SubRoutine parameters
```

Subroutine Example

```
Sub SetTheHeight AHeight
    Set Component.Height = AHeight
End Sub
```

Function Example

```
Function Addone(value)
    AddOne = Value + 1
End Function
```

Another Function Example

```
Function Test(s)
    Test = S + " rules.."
End Function


Sub DisplayName (sName)
    MsgBox sName
End Sub


Sub Main
Dim S
    S = "Altium Designer"
    DisplayName Test(s)
End Sub
```

Parameters and Arguments

The procedure declaration normally has a list of parameters (remember variables are considered typeless and the scripting system works out automatically what the variable types are). The value used in place of the parameter when you make a procedure call is called an argument.

Example of a subroutine with a parameter

```
Sub DisplayName (sName)
    MsgBox "My Name is " & sName
End Sub
```

Example of calling a subroutine

```
Sub Main
    DisplayName "Altium Designer Rules"
End Sub
```

Notes

The use of the **Call** keyword to invoke a subroutine or a function is optional (maintained for backward compatibility).

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 ' are ignored by VB Script.

Any text following Rem are ignored by VB Script.

Example

```
' This whole line is a comment
Rem this whole line is also a comment
DocName = Document.Name ' Get name of active document
```

Splitting a Line of Script

Each code statement is terminated on each line to indicate the end of this statement. VB Script allows you to write a statement on several lines of code, splitting a long instruction on two or more lines using the underscore character (_).

VB Script 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 VB 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 EndIf
```

Formatted code example

```
If Not (PcbApi_ChooseRectangleByCorners(BoardHandle,_
                                    "Choose first corner",_
                                    "Choose final corner",_
                                    x1,y1,x2,y2)) Then EndIf
```

Using Altium Designer Run Time Library and Object Models in VB Scripts

The biggest feature of the scripting system is that the Object Interfaces of Altium Designer objects are available to use in VB scripts. For example, you can update design objects on Schematic and PCB documents through the use of Schematic Interfaces and PCB interfaces respectively.

The Altium Designer Object Interfaces are available for use in any script. Normally in scripts, there is no need to instantiate an interface, you extract the interface representing an existing object and from this interface, extract embedded or aggregate interface objects and from them, you can get or set property values.

To access a schematic document and its data objects, you invoke the `SchServer function`.

Example

```
' Checks if the current document is a Schematic document
If SchServer Is Nothing Then Exit Sub
Set CurrentSheet = SchServer.GetCurrentSchDocument
If CurrentSheet Is Nothing Then Exit Sub
```

To have access to a PCB document, you invoke the PCBServer.

Creation of a PCB Object Using the PCB Object Model

```
Sub ViaCreation
    Dim Board
    Dim Via
    Set Board = PCBServer.GetCurrentPCBBoard
    If Board is Nothing Then Exit Sub
    ' Create a Via object
    Via = PCBServer.PCBObjectFactory(eViaObject, eNoDimension, eCreate_Default)
    Via.X = MilsToCoord(7500)
    Via.Y = MilsToCoord(7500)
    Via.Size     = MilsToCoord(50)
    Via.HoleSize = MilsToCoord(20)
    Via.LowLayer  = eTopLayer
    Via.HighLayer = eBottomLayer
    ' Put this via in the Board object
    Board.AddPCBObject(Via)
End Sub
```

Different Objects, Interfaces Functions in your scripts

Objects, Interfaces Functions in your scripts can be used from the following:

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

📖 Refer to *Getting Started With Scripting* and *Building Script Projects* tutorials.

📖 Refer to the *Using the Altium Designer RTL* guide for details on how to use design objects and their interfaces in your scripts.

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

TR0125 (v1.6) Feb 27, 2008

VB Script Keywords

The scripting system supports the VB Script language which is derived from the Microsoft Active Scripting language technology. This section covers the VB Script keywords.

Reserved Words and Functions in VB Script

A, B

Abs, Array, Asc, Atn

C

Call, Case, CBool, CByte, CCur, CDate, CDbl, Chr, CInt, Class, CLng, Const, Conversions, Cos, CreateObject, CSng, CStr

D,E

DateAdd, DateDiff, DatePart, DateSerial, DateValue, Day, Derived Math, Dim, Do, Each, Erase, Escape, Empty, Eval, Execute, Exit, Exp

F,G, H

False, Filter, For, FormatCurrency, FormatDateTime, FormatNumber, FormatPercent, Function GetLocale, GetObject, GetRef, Hex, Hour

I, L, M

If, Is, InputBox, Instr, InStrRev, Int, IsArray, IsDate, IsEmpty, IsNull, IsNumeric, IsObject, Join, LBound, LCase, Left, Len, LoadPicture, Log, LTrim, Maths, Mid, MInute, Month, MonthName, MsgBox

N, O

Next, Nothing, Now, Null, Oct, On Error

P,R

Private, Property, Public, Randomize, ReDim, Rem, RTrim, Replace, RGB, Right, Rnd, Round

S, T

ScriptEngine, ScriptEngineBuildVersion, ScriptEngineMajorVersion, ScriptEngineMinorVersion, Second, Select, Set, SetLocale, Sgn, Sin, Space, Split, Sqr, Stop, StrComp, String, StrReverse, Sub, Tan, Then, Time, Timer, Timeserial, TimeValue, Trim, True, TypeName

U, V, W, X, Y

UCase, Unescape, While, Wend, With, VarType, Weekday, WeekdayName, Year

VB Script Statements

In this section:

- *Conditional Statements*
- *Expressions and Operators*

Conditional Statements

The main conditional statements supported by the VB Script:

- *If Then*
- *For Next Loop*
- *Exit For*
- *For Each Next*
- *Do Loop*
- *While WEnd*
- *Select Case*

You have to be careful to code your scripts to avoid infinite loops, ie the conditions will eventually be met.

The If.. Then Statement

The syntax is:

```
If Condition Then

Else If AnotherCondition Then

Else

End If
```

The For Loop

The For Next statement repeatedly loops through a block of code. The basic syntax is:

```
For counter = start to end
    ' block of code here
Next
```

The Exit For

The Exit For statement exits a For loop prematurely.

```
For counter = start to end
    if condition then Exit For
Next
```

The For Each Loop

The For Each loop is a variation on the For loop which is designed to iterate through a collection of objects as well as elements in an array. The basic syntax is:

```
For Each ObjectVar in Collection
    ' block of code here
Next
```

The Do Loop

The Do Loop has several loop variations.

1.

```
Do while until condition
    ' code block
Loop
```

2.

```
Do
    ' code block
Loop while until condition
```

3.

```
Do
    ' code block
Loop
```

The While WEnd Loop

The While WEnd statement repeatedly loops through a block of code. The basic syntax is;

```
While until condition
    ' code block
WEnd
```

The Select Case Statement

You can use the `SELECT` statement if you want to select one of many blocks of code to execute:

```
Select case payment
 case "Cash"
    msgbox "pay cash"
 case "MasterCard"
    msgbox "pay by Mastercard"
 case Else
    msgbox "Unknown payment method"
end select
```

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.

VB Script 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.

Arithmetic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Division with integer result
^	Exponentiation
Mod	Modulo

Comparison Operators (lowest precedence)

=	Test whether equal or not.
<>	Test whether not equal or not.
<	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.
Is	Compares two object reference variables.

String Operators

&	Concatenation

Logical Operators

Not	Logical NOT
And	Logical AND
Or	Logical OR
XOR	
Eqv	
Imp	
&	

VB Script Sub Routines and Functions

In this section:

- *Passing parameters to procedures*
- *Dates Times*
- *File IO Routines*
- *Math Routines*
- *String Routines*
- *Server Process Routines*.

Passing Parameters to Sub Routines and Functions

When you define a function or sub routine in a script that can accept parameters, you can pass variables to the function or sub routine in two ways: by reference or by value.

To declare the method that parameters are passed, use the `ByRef` or `ByVal` keywords in the parameter list when defining the function or sub routine in a `Sub` or `Function` statement. For example, the following code fragment defines a sub routine that accepts two parameters. The first is passed by value and the second by reference:

```
Sub Test (ByVal Param1 As Integer , ByRef B As String)
```

The difference between the two methods is that `ByRef` passes a reference to the variable passed and allows the sub routine or function to make changes to the actual variables that are passed in as parameters (this is the default method of passing parameters and is used if the method is not explicitly declared).

The `ByVal` passes the value of the variable only. The sub routine or function can use this value, but the original variable passed is not altered.

The following examples illustrate the differences between methods. The main procedure is as follows:

```
Sub Main
    Dim X, Y
    X = 45 : Y = "Number"
    Test X, Y  ' Call to a subprocedure called Test.
    MsgBox X
    MsgBox Y
```

```
End Sub
```

The above procedure includes a call to a subprocedure, `Test`.

If the subroutine is defined as follows:

```
Sub Test (ByRef A, ByRef B)
    B = B & " = " & A : A = 10*A
End Sub
```

Then the variables X and Y in the main procedure are referenced directly by the sub routine. The result is that the values of X and Y are altered by the sub routine so that after the Test is executed X = 450 and Y = "Number = 45".

If, however, the sub routine is defined as follows:

```
Sub Test (ByVal A, ByVal B)
    B = B & " = " & A : A = 10*A
End Sub
```

Then after Test is executed X = 45 and Y = "Number", i.e. they remain unchanged.

If the sub routine is defined as follows:

```
Sub Test (ByRef A, ByVal B)
    B = B & " = " & A : A = 10*A
End Sub
```

Then after Test is executed, X = 450 and Y = "Number" since Y was passed by value, it remains unchanged.

You can override the `ByRef` setting of a function or sub routine by putting parentheses around a variable name in the calling statement.

Calling `Test` with the following statement:

```
Test (X), Y
```

would pass the variable X by value, regardless of the method defined for that parameter in the procedure definition.

Dates and Times routines

The VB Script language set supports a set of Date/Time routines and a few routines outlined below:

Date	Day	Hour
IsDate	Minute	Month
Now	Second	Time
Year		

File IO Routines

The VB Script language set supports a set of File IO routines:

Dir	FileLen	FileTimeDate
FileCopy	Kill	Name
RmDir	MkDir	

Math Routines

The VB Script language set supports a set of Math routines:

Abs	Atn	Cos
Exp	Log	Not
Oct	Rnd	Sin
Sqn	Tan	

String Routines

The VB Script language set supports a set of String routines and a few string routines outlined below:

Asc	Chr	Format
InStr	InStrRev	LCase
Len	Left	Mid
Right	Str	Trim
LTrim	RTrim	UCase

Server Process Routines

The server process 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.

To execute processes and parameters in scripts, use the 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 Components in Script Forms

Although Forms and Components are based on Borland Delphi's Visual Component Library, you still use the *Tool Palette* to drop controls on a form and generate VB Script based event handlers and write code in VB Script language.

In this section:

- *Components*
- *Designing Script Forms*
- *Writing Event Handlers.*

Components

The scripting system handles two types of components: Visual and Non-visual components. The visual components are the ones you use to build the user interface and the non-visual components are used for different tasks such as **Timer**, **OpenDialog** and **MainMenu** components. The Timer non-visual component can be used to activate specific code at scheduled intervals which is never seen by the user. The Button, Edit and Memo components are examples of visual components

Both types of components appear at design time but non visual components are not visible at runtime. Components from the *Tool Palette* panel are object oriented and have the three following items:

- Properties
- Events
- Methods

A **property** is a characteristic of an object that influences 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 defines 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 and 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 *Tool Palette* panel.

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 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 Altium Designer environment. Designing script forms is the core of visual development in the Altium Designer. 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 `*.VBS` file). For every script form, there is the *.VBS file and the corresponding *.DFM file.

When you are working with a script form and its components, you can customize 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. A script form has a title which corresponds to the **Caption** property on the *Object Inspector* panel.

Creating a New Script Form

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 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 routine that displays the form when the script form is executed in Altium Designer. Within this routine, 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

```
Sub RunDialog
    DialogForm.ShowModal
End Sub
```

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

```
sub bOKButtonClick(Sender)
  ModalResult := mrOK
end sub


sub bCancelButtonClick(Sender)
  ModalResult := mrCancel
end sub


sub RunShowModalExample
    'Form 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 sub
```

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 form 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 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.

```
sub TScriptForm.ButtonClick(Sender)
    If Edit1.Text = "" Then
        ShowMessage("Warning - empty input!")
        Exit
    End
    ' do something else for the input
End sub
```

Note a user can move the input focus by using the Tab key or by clicking with the house 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.

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 that respond to events is contained in 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 on a component.. All such components have an event for getting and losing the focus. 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 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. 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 to, select a component and in the *Object Inspector* activate the **Events** tab. To create an event handling procedure, decide which event you want your component to react to and double click the event name.

For example, select the **Button1** component from the *Tool Palette* 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.

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, select a button component, either on the form or by using the *Object Inspector* panel, select the **Events** tab and double click on the right side of the `OnClick` event. A new event handler will appear on the script. Alternatively, 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, refer to the *Component Reference* document.

Dropping Components on a Script Form

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

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

Creating Components from a Script

You can 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. You 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).

Revision History

Date	Version No.	Revision
01-Dec-2004	1.0	New product release
26-Apr-2005	1.1	Altium Designer
15-Dec-2005	1.2	Updated for Altium Designer 6
6-Jul-2006	1.3	Updated for Altium Designer 6.3 DXP references changed to Altium Designer. For To Next syntax corrected and While WEnd loop added.
1-Sept-2006	1.4	Updated for Altium Designer 6.4 Typing of variables corrected. Variables are typeless and functions declaration clarified.
18-Feb-2008	1.5	Updated for Altium Designer 6.9
27-Feb-2008	1.6	Updated Page Size to A4

Software, hardware, documentation and related materials: