@ALT

Syntax:

@ALT(<string>)

Description:

This function is only useful when used within the string argument of a WINDOW SEND command to send keystrokes to another window. It generates keystrokes that are the equivalent of holding down the Alt key while the keys in <string> are being sent.

OK:

Unchanged.

Example:

```
rem close down an application

window send,Test - WordPad,@alt(F)X
```

See also:

@CR @CTRL @KEY @SHIFT @TAB

@ASC

Syntax:

@ASC(<string>)

Description:

This function returns the ASCII code of the first (or only) character of <string>..

OK:

Unchanged.

Example:

%A = @asc(A)

See also:

@CHR

@ASK

Syntax:

@ASK(<string>)

Description:

This function displays a dialog box containing a question mark icon, the message <string>, and buttons for Yes and No. The value 1 (true) is returned (and OK is set to true) if the user responds Yes; otherwise the function returns null (false) and OK is set to false.

OK:

Unchanged.

Example:

```
IF @ASK(Do you want to continue?)
  REM Do something useful
END
```

See also:

INFO WARN @INPUT @MSGBOX @QUERY

@BOTH

Syntax:

@BOTH(<string1>,<string2>)

Description:

This function returns the value 1 (true) only if both <string1> and <string2> are non-null (true); otherwise the function returns null (false).

OK:

Unchanged.

Example:

```
REPEAT
   WAIT 1
UNTIL @BOTH(@file(a.txt),@file(b.txt))
```

See also:

@EQUAL @NOT @NULL @ZERO

@CHR

Syntax:

@CHR(<value>)

Description:

This function returns the character whose ASCII code is <value>. It can be used to generate characters that cannot easily be entered from the keyboard.

OK:

Unchanged.

Example:

%A = Copyright @CHR(169) 1995, 1996 Bloggs Inc.

See also:

@ASC @CR @ESC @TAB

@CLICK

Syntax:

@CLICK(<flags> {, <top>, <left>, <bottom>, <right>})

Description:

This function should be used after a CLICK event has been generated by a BITMAP dialog element, to find out where on the bitmap the mouse was clicked, and which button was used. The information is returned as a string, consisting of one or more items as specified by the flags. If more than one item is specified then each item is separated by the field separator character in a form that can be processed by the PARSE command.

If the B flag is included, the function returns LEFT, RIGHT or CENTER depending on which mouse button was pressed.

If the N flag is included, the function returns the name of the bitmap control over which the mouse was clicked.

If the flags X or Y are included, the function returns the X or Y co-ordinates of the pointer when the mouse was clicked.

If the R flag is included then the function returns the value 1 (true) if the mouse was clicked within the region bounded by the co-ordinates <top>, <left>, <bottom> and <right>, otherwise it returns null (false).

Note: this function does not return meaningful information after CLICK events generated by other types of dialog element.

OK:

Set to false if one of the REGION parameters is invalid. Otherwise unchanged.

Example:

```
:evloop
wait event
if @equal(@event(),click)
  parse "%B;%X;%Y",@click(BXY)
  info You clicked the %B Button@CR()Co-ords X: %X Y: %Y
  goto evloop
end
```

See also:

Dialog Programming Events

@COUNT

Syntax:

@COUNT(<list>)

Description:

This function returns the number of items in the string list <list>. The parameter <list> must be either a list number or the name of the dialog list control to which the function will apply.

OK:

Unchanged.

Example:

```
LIST 1,CREATE,SORTED
LIST 1,LOADFILE,NAMES.TXT
INFO There are @COUNT(1) names in the list
LIST 1,CLOSE
```

See also:

@INDEX @ITEM @MATCH @NEXT LIST

@CR

Syntax:

@CR()

Description:

This function returns a carriage return character. You use it to create output on multiple lines, or to send an Enter keystroke in a WINDOW SEND command.

OK:

Unchanged.

Example:

%A = This is the first line of output

%B = This is the second line

INFO %A@CR()%B

See also:

@CHR @ESC @TAB WINDOW

@CTRL

Syntax:

@CTRL(<string>)

Description:

This function is only useful when used within the string argument of a WINDOW SEND command to send keystrokes to another window. It generates keystrokes that are the equivalent of holding down the Ctrl key while the keys in <string> are being sent.

OK:

Unchanged.

Example:

```
rem make text bold

window send,Microsoft Word,@ctrl(B)
```

See also:

@ALT @CR @KEY @SHIFT @TAB

@CURDIR

Syntax:

@CURDIR(<drive letter>)

Description:

This function returns the current directory of the specified drive. If no drive letter is specified, the function returns the current directory of the current default drive.

OK:

Unchanged.

Example:

```
%C = @CURDIR()

REM %C now holds the current directory

%D = @CURDIR(D)

REM %D now holds the current directory of drive D:
```

See also:

@WINDIR

@DATETIME

Syntax:

@DATETIME(<format-string> {, <time-value>)

Description:

This function returns the current date and/or time (or if <time-value> is given, the date and/or time corresponding to <time-value>) formatted in accordance with <format-string>. If no format is specified, the value returned is a floating point number. If a <time value> is specified that is not a formatted date, it must also be in this format.

Also, two digit dates are "windowed" so that years 00 - 79 are assumed to be in the next millennium. This window date can be changed using OPTION CENTURYWINDOW.

Format strings may be comprised of:

d	returns day number without leading zero
dd	returns day number with leading zero
ddd	returns day as an abbreviation e.g. Mon
dddd	returns full name of day e.g. Monday
ddddd	returns date using Windows' Short Date style (set in Control Panel)
dddddd	returns date using Windows' Long Date style (set in Control Panel)
m	returns month number without leading zero
mm	returns month number with leading zero
mmm	retuns month as an abbreviation e.g. Jan
mmmm	returns full name of month e.g. January
yy	returns year as two digit number
yyyy	returns year as four digit number
h	returns hour without leading zero
hh	returns hour with leading zero
nn	returns minute (note: **not** mm)
ss	returns seconds
t	returns time using Windows' Short Time style (set in Control Panel)
tt	returns time using Windows' Long Time style (set in Control Panel)
am/pm	uses 12 hour clock and displays am or pm as appropriate
a/p	uses 12 hour clock and displays a or p as approprate
/	returns date separator as set in Control Panel
:	returns time separator as set in Control Panel
ampm	returns AM symbol or PM symbol as set in Control Panel

Spaces and other separator characters (e.g. the current field separator) can be included in the format string.

OK:

Unchanged.

Example:

```
%D = @DATETIME(t   dddd dddddd,%F)

PARSE "%H;%M;%S",@datetime(hh|mm|ss)
```

See also:

@DDEITEM

Syntax:

@DDEITEM(<item>)

Description:

This function requests the data from a DDE server which is identified by <item>. For information on what items are supported see the DDE server documentation.

OK:

True if DDE request is successful, false if it fails.

Example:

DDE LINK,QPW,SYSTEM

%I = @DDEITEM(SYSITEMS)

DDE TERMINATE

INFO DDE Topics supported by QPW:@CR()%I

See also:

DDE

@DIFF

Syntax:

@DIFF(<value1>,<value2>)

Description:

This function returns the difference of its two arguments: <value1> - <value2>.

OK:

Set to false if either of the arguments is null or non-numeric, or if overflow occurs.

Example:

%C = @DIFF(%A,%B)

See also:

@DIV	@FADD	@FDIV	@FMUL	@FORMAT	
@FSUB	@HEX	@NUMERIC	@PRED	@PROD	@SUCC
@SUM	@ZERO				

@DIRDLG

Syntax:

@DIRDLG(<caption> {, <path> })

Description:

This function displays a standard browser dialog which allows the user to choose a directory, returning the selected directory name when the dialog is closed. The optional caption specifies the text to appear above the browser window.

The directory browser will open with the specified directory selected (<path>). If no argument is specified the default is "My Computer".

OK:

True if dialog was closed with the OK button, otherwise false.

Example:

%D = @dirdlg()

See also:

@FILEDLG

@DIV

Syntax:

@DIV(<value1>,<value2>)

Description:

This function returns the quotient of its two arguments: <value1> / <value2>.

OK:

Set to false if either of the arguments is null or non-numeric, or if overflow occurs.

Example:

%C = @DIV(%A,%B)

See also:

@DIFF	@FADD	@FDIV	@FMUL	@FORMAT
@FSUB	@HEX	@NUMERIC	@PRED	@PROD
@SUCC	@SUM	@ZERO		

@DLGTEXT

Syntax:

@DLGTEXT(<control name>)

Description:

This function is used to read the text that appears in dialog window controls. The string <control name> identifies the control that is the target of the function.

In the case of a list box the contents of the selected item is returned, or null if no item is selected.

OK:

Unchanged.

Example:

%A = @dlgtext(Name)

%B = @dlgtext(Addr1)

See also:

DIALOG Dialog Programming

@ENV

Syntax:

@ENV(<string>)

Description:

This function returns the value of the DOS environment variable named <string>.

OK:

Unchanged.

Example:

%P = @ENV(PATH)

See also:

@EQUAL

Syntax:

@EQUAL(<string1>,<string2> {, EXACT})

Description:

This function compares two string values. If the values are both valid numbers then a numeric comparison is performed, otherwise a string comparison is performed based on ASCII character values. The value 1 (true) is returned if they are identical; otherwise the function returns null (false).

Unless the optional EXACT parameter is specified the string comparison is not case sensitive, so "Visual DialogScript" and "VISUAL Dialogscript" would be considered equal.

OK:

Unchanged.

Example:

```
IF @EQUAL(%F,WIN.INI)
   WARN You must not delete this file!
END
```

See also:

@BOTH @NOT @NULL @ZERO

@ERROR

Syntax:

@ERROR(<flags>)

Description:

This function can be called after an error trap has occurred (see OPTION ERRORTRAP) and returns information about the error according to the flags used.

The flags are:

E - error code, from which you can determine the type of error;

N - the line number containing the error.

Note that, because the compiler does not include in the EXE lines that do nothing, the line number reported may not be the same as the line number in the original script.

If more than one flag is specified then each value in the returned string is separated by a field separator character, in a form that can be split up into separate variables using the PARSE command.

OK:

Unchanged.

Example:

INFO Error @ERROR(E) at line @ERROR(N)

See also:

OPTION PARSE Error trapping

@ESC

Syntax:

@ESC()

Description:

This function generates an Escape character (ASCII 27) which can be used in a WINDOW SEND command.

OK:

Unchanged.

Example:

WINDOW SEND,Error,@ESC()

See also:

@CHR @CR @TAB WINDOW

@EVENT

Syntax:

@EVENT({D})

Description:

This function returns the name of the last event to have occurred. It returns a null string if no event has occurred. After the function has been called the event is cleared, so it should be stored in a variable if you need to test it more than once.

This function has an optional parameter D, for use in programs that have more than one dialog. If present, the event name will be followed by a field separator and a number indicating the dialog that issued the event. The main window has the value 0. A child window will have the value 1. There may be more than one child dialog in which case the number will reflect the order they were created but the number will change if earlier dialogs are closed, because the number is actually the position in a list.

OK:

Unchanged.

Example:

```
:LOOP
  WAIT EVENT
  GOTO @EVENT()
:OKBUTTON
  INFO You pressed OK
  GOTO LOOP
:CANCELBUTTON
  INFO You pressed Cancel
```

See also:

Dialog Programming Events

@EXT

Syntax:

@EXT(<string>)

Description:

This function returns the extension portion of a file path specified in <string>.

OK:

Unchanged.

Example:

%X = @EXT(C:\WINDOWS\WIN.INI)

REM %X now contains INI

See also:

@NAME @PATH @SHORTNAME

@FADD

Syntax:

@FADD(<value1>,<value2>)

Description:

This function returns the sum of its two arguments: <value1> + <value2>, which may be floating-point numbers.

OK:

Not changed. Error 25 will occur if an argument is non-numeric. Error 26 will occur if underflow or overflow occurs.

Example:

%F = @FADD(%A,3.14159)

See also:

@DIFF	@DIV	@NUMERIC	@PRED
@PROD	@SUCC	@SUM	@ZERO

@FATN

Syntax:

@FATN(<value1>)

Description:

This function returns the arctangent of its argument <value1>.

OK:

Not changed. Error 25 will occur if an argument is non-numeric. Error 26 will occur if underflow or overflow occurs.

Example:

%T = @FDIV(@FSIN(%X),@FCOS(%X))

See also:

@DIFF	@DIV	@NUMERIC	@PRED
@PROD	@SUCC	@SUM	@ZERO

@FCOS

Syntax:

@FCOS(<value1>)

Description:

This function returns the cosine of its argument <value1>.

OK:

Not changed. Error 25 will occur if an argument is non-numeric. Error 26 will occur if underflow or overflow occurs.

Example:

%F = @FCOS(2)

See also:

@DIFF	@DIV	@NUMERIC	@PRED
@PROD	@SUCC	@SUM	@ZERO

@FDIV

Syntax:

@FDIV(<value1>,<value2>)

Description:

This function returns the result of: <value1> / <value2>, where the two values may be floating-point numbers.

OK:

Not changed. Error 25 will occur if an argument is non-numeric. Error 26 will occur if underflow or division by zero occurs.

Example:

%F = @FDIV(%A,%B)

See also:

@DIFF	@DIV	@NUMERIC	@PRED
@PROD	@SUCC	@SUM	@ZERO

@FILE

Syntax:

@FILE(<file description>, <attributes>)

Description:

This function returns the name of the first file that matches <file description> if it exists. It returns a null string if the file does not exist.

The attribute string can be used to filter the filenames that will be checked. The attributes can be:

A - archive

D - directory

H - hidden

R - read only

S - system

The 'pseudo-attributes' F, L, T or Z may also be present. These pseudo-attributes are used to specify what information is returned by the function. If none of these pseudo-attributes is present, F is assumed.

The information returned if these attributes are specified is as follows:

F - file path

L - path of the file a shortcut points to (this is only valid if the file is a shortcut)

Z - file size in bytes

T - file date/time as a packed binary value

If more than one of these pseudo-attributes is specified the information is returned in the order shown in the above list with each item delimited by the current field separator character in a format that can be split into separate variables using the PARSE command.

The fact that a file path is returned when the L pseudo-attribute is used is no guarantee that the file pointed to by the shortcut actually exists.

The packed binary value returned by the T pseudo-attribute can be converted to a normal date/time format using the @DATETIME function.

OK:

Set to false if the function fails.

Example:

```
parse "%F;%S;%D",@file(c:\autoexec.bat,FTZ)
if @file(%F,D)
  INFO Directory %F does not exist
end
```

See also:

@EXT @NAME @PATH @SHORTNAME @VERINFO @VOLINFO

@FILEDLG

Syntax:

@FILEDLG(<file description> {title>, <initial filename>,<parameter>})

Description:

This function displays a Windows common file dialog, and returns the name of the file that was selected (or a null string if no file was selected.) The string <file description> contains a filter for the file type to be displayed, for example: *.txt. The string <title> contains an optional title which will be used for the dialog. The string <initial filename> specifies an optional default filename.

Optional fourth parameters are:

MULTI allows multiple selections to be made. In this event, the files are returned as a string of names separated by carriage return characters. The resulting string can be assigned to a list using LIST ASSIGN, so each filename will then be a separate list item.

SAVE creates a dialog with a Save button instead of an Open button. These options are mutually exclusive: you cannot select multiple files to save.

NOTE: if the third parameter (default filename) is specified it must be a valid filename. If it is not the file dialog box is not displayed and the value of OK is set to false.

Multiple choice filters and descriptions of the file types can be specified if <file description> follows the format:

"<file description 1>|<*.ext1> {|<file description2>|<*.ext2>} ..."

This format must be followed exactly or the description will be ignored. Quotes must surround the entire file description string.

OK:

True if dialog was closed with the OK button, otherwise false.

Example:

%F = @filedlg(*.txt)

%F = @filedlg("Text file (*.txt)|*.txt|Document file (*.doc)|*.doc",Open file)

See also:

@DIRDLG

@FMUL

Syntax:

@FMUL(<value1>,<value2>)

Description:

This function returns the product of its two arguments: <value1> x <value2>, which may be floating-point numbers.

OK:

Not changed. Error 25 will occur if an argument is non-numeric. Error 26 will occur if underflow or overflow occurs.

Example:

%F = @FMUL(%A,%B)

See also:

@DIFF	@DIV	@NUMERIC	@PRED
@PROD	@SUCC	@SUM	@ZERO

@FORMAT

Syntax:

@FORMAT(<value>,<format>)

Description:

This function returns <value> formatted in accordance with <format>, which must be in the form **n.d** where n is the total number of digits and d is the number of decimal places. If only n is specified then two decimal places are assumed by default.

OK:

Set to false if either of the arguments is null or non-numeric.

Example:

DIALOG SET,AM1,£@FORMAT(%T,5.2)

See also:

@DIFF	@DIV	@HEX	@NUMERIC
@PRED	@PROD	@SUCC	@SUM
@ZERO	Floating point functions		

@FSIN

Syntax:

@FSIN(<value1>)

Description:

This function returns the sine of its argument <value1>.

OK:

Not changed. Error 25 will occur if an argument is non-numeric. Error 26 will occur if underflow or overflow occurs.

Example:

%F = @FSIN(2)

See also:

@DIFF	@DIV	@NUMERIC	@PRED
@PROD	@SUCC	@SUM	@ZERO

@FSQT

Syntax:

@FSQT(<value1>)

Description:

This function returns the square root of its argument <value1>.

OK:

Not changed. Error 25 will occur if an argument is non-numeric. Error 26 will occur if underflow or overflow occurs.

Example:

%F = @FSQT(2)

See also:

@DIFF	@DIV	@NUMERIC	@PRED
@PROD	@SUCC	@SUM	@ZERO

@FSUB

Syntax:

@FSUB(<value1>,<value2>)

Description:

This function returns the difference of its two arguments: <value1> - <value2>, which may be floating-point numbers.

OK:

Not changed. Error 25 will occur if an argument is non-numeric. Error 26 will occur if underflow or overflow occurs.

Example:

%F = @FSUB(%A,%B)

See also:

@DIFF	@DIV	@NUMERIC	@PRED
@PROD	@SUCC	@SUM	@ZERO

@GREATER

Syntax:

@GREATER(<value1>, <value2>)

Description:

This function compares two string values. If the values are both valid numbers then a numeric comparison is performed, otherwise a string comparison is performed based on ASCII character values. The value 1 (true) is returned if <value1> is greater than <value2>; otherwise the function returns null (false).

OK:

Unchanged.

Example:

```
IF @GREATER(%1,%2)
   INFO Larger value is %1
END
```

See also:

@EQUAL @NOT @ZERO @NULL

@HEX

Syntax:

@HEX(<value>{,<number>})

Description:

This function returns <value> formatted as a hexadecimal number.

The optional second argument specifies the number of digits (the result will be padded out with leading zeros.)

OK:

Set to false if the argument is null or non-numeric.

Example:

DIALOG SET,Hexval,@HEX(%N)

See also:

@DIFF	@DIV	@FADD	@FDIV	@FMUL
@FORMAT	@FSUB	@PRED	@PROD	@SUCC
@SUM	@ZERO			

@INDEX

Syntax:

@INDEX(<list>)

Description:

This function returns the current index (pointer) value for the string list <list>. The parameter <list> must be either a list number or the name of the dialog list control to which the function will apply.

If <list> is a LIST or COMBO element and no list item is selected the value returned by this function is -1.

OK:

Unchanged.

Example:

```
LIST 1,CREATE,SORTED
LIST 1,LOADFILE,NAMES.TXT
IF @MATCH(1,John)
  INFO John is at position @INDEX(1) in the list
END
LIST 1,CLOSE
```

See also:

@COUNT @ITEM @MATCH @NEXT LIST

@INIREAD

Syntax:

@INIREAD(<section name>,<key name> {, <default>})

Description:

This function returns a string containing the value of <key name> in the <section name> section of the INI file that was specified in the preceding INIFILE command (or the default INI file if no INIFILE OPEN command has been executed.) If <section name> is null the section [Default] is used. The returned string is empty if the section or key do not exist unless the optional <default> value has been supplied.

The maximum length of text that can be read in using this function is set by OPTION REGBUF.

OK:

Unchanged.

Example:

%A = @INIREAD(Extensions,.txt)

See also:

INIFILE

@INPUT

Syntax:

@INPUT(<prompt text>,<default result> {, PASSWORD})

Description:

This function prompts the user to enter some text. The optional default result is returned if the user presses Cancel.

If the optional PASSWORD parameter is supplied then the text in the input box is shown as asterisks.

OK:

Set to false if the user presses Cancel.

Example:

%P = @INPUT(Enter password:,,PASSWORD))

See also:

@ASK @MSGBOX @QUERY

@ITEM

Syntax:

@ITEM(<list>{,<number>})

Description:

This function returns the contents of the string at the current index position in the string list <list>. The index can be set using the LIST SEEK command.

The optional second parameter (@item(i,x)) allows you to retrieve an item from a list randomly without having to seek to that item first. The index value afterwards has changed just as if a LIST SEEK command had been carried out.

The parameter <list> must be either a list number or the name of the dialog list control to which the command will apply. An error will occur if the list does not already exist.

OK:

Set to false if <item number> is out of range..

Example:

```
LIST 1,CREATE
LIST 1,LOADFILE,NAMES.TXT
LIST 1,SEEK,9
%I = @ITEM(1)
INFO The tenth name is %I
LIST 1,CLOSE
```

See also:

@COUNT @INDEX @MATCH @NEXT LIST

@KEY

Syntax:

@KEY(<key name>)

Description:

This function is only useful when used within the string argument of a WINDOW SEND command to send keystrokes to another window. It is used to generate keystrokes that are equivalent to the key named in <key name>.

Valid key namesaare: HOME, END, UP, DOWN, LEFT, RIGHT, PGUP, PGDN, INS, DEL, plus F1 to F12..

OK:

Unchanged.

Example:

rem move cursor

window send,Test - WordPad,@key(HOME)@key(PGDN)@key(PGDN)

See also:

@ALT @CR @CTRL @SHIFT @TAB

@LEN

Syntax:

@LEN(<string>)

Description:

This function returns the length of <string> in characters. The value 0 is returned if the string is empty.

OK:

Unchanged.

Example:

%L = @LEN(%A)

See also:

@POS @SUBSTR @UPPER

@LOWER

Syntax:

@LOWER(<string>)

Description:

This function returns the string <string> converted entirely to lower case characters.

OK:

Unchanged.

Example:

%S = @LOWER(The Quick Brown Fox)

REM %S now contains 'the quick brown fox'

See also:

@UPPER

@MATCH

Syntax:

@MATCH(<list>, <string>)

Description:

This function returns 1 (true) if a string in the string list <list>, starting from the current pointer position, contains text matching <string>. The match is not case-sensitive. The pointer is advanced to the matching item number, so you can obtain the contents of the string using @ITEM or @NEXT, rewrite it using LIST PUT and obtain the index value using @INDEX. If no match is found, null (false) is returned and the index value is unchanged.

The parameter <list> must be either a list number or the name of the dialog list control to which the command will apply.

OK:

Set to false if no match is found.

Example:

```
LIST 1,CREATE
LIST 1,LOADFILE,NAMES.TXT
%M = @MATCH(1,Jim)
INFO %M
LIST 1,CLOSE
```

See also:

@COUNT @INDEX @ITEM @NEXT LIST

@MCI

Syntax:

@MCI(< MCI command string >)

Description:

This function is used to control multimedia devices using the Multimedia Control Interface (MCI). An MCI command is supplied as the parameter to the function. The function returns the result of the command. It returns the text of the MCI error message if the command fails..

Note: MCI is a feature of Windows. For a full description of how to use it you will need documentation such as the *Microsoft Multimedia Development Kit Programmer's Workbook* or the *Microsoft Windows Software Development Kit Multimedia Programmer's Reference*.

OK:

Set to false if the command fails.

Example:

```
%R = @MCI(open C:\WINDOWS\MEDIA\THEMIC~1.WAV alias sound)
if @ok()
  %R = @MCI(play sound)
else
  warn MCI error: %R
end
```

See also:

PLAY Using MCI

@MSGBOX

Syntax:

@MSGBOX(<message>, <title>, <icon/button flags>)

Description:

This function displays a standard Windows message dialog box with the message, title, buttons and icons specified, and returns a value which indicates which button was pressed.

The <icon/button flags> argument is a value which is best expressed as a hexadecimal number. It is built up by adding one number from each of the following three sections:

Default button:

First button	$000
Second button	$100
Third button	$200

Icon:

None	$000
No entry	$010
Question mark	$020
Exclamation mark	$030
Information	$040

Buttons:

OK	$000
OK, Cancel	$001
Abort, Retry, Ignore	$002
Yes, No, Cancel	$003
Yes, No	$004
Retry, Cancel	$005

The return value is one of the following:

1 OK

2 Cancel

3 Abort

4 Retry

5 Ignore

6 Yes

7 No

The INFO and WARN commands, and the @ASK and @QUERY functions, are a simpler way to create certain message dialogs.

OK:

Unchanged.

Example:

```
IF @EQUAL(@MSGBOX(Cannot read from drive A:,Backup,$35),2)
  STOP
END
```

See also:

INFO WARN @INPUT @QUERY

@NAME

Syntax:

@NAME(<string>)

Description:

This function returns the filename portion (not including the extension) of a file path specified in <string>.

OK:

Unchanged.

Example:

%N = @NAME(C:\WINDOWS\WIN.INI)

REM %N now contains WIN

See also:

@EXT @PATH @SHORTNAME

@NEXT

Syntax:

@NEXT(<list>)

Description:

This function returns the contents of the next item in the string list <list>. that the pointer currently points to, and then advances the pointer by 1. The pointer may be set using the LIST SEEK command. For LIST and COMBO elements, if no item is selected the index value is -1 and the value returned by @next() will be null.

The parameter <list> must be either a list number or the name of the dialog list control to which the command will apply.

You use the @NEXT function if you want to read sequentially through the items in a list.

OK:

Set to false when the end of the list is reached.

Example:

```
LIST 1,CREATE
LIST 1,LOADFILE,NAMES.TXT
REPEAT
  INFO @NEXT(1)
UNTIL @NOT(@OK())
LIST 1,CLOSE
```

See also:

@COUNT @INDEX @ITEM @MATCH LIST

@NOT

Syntax:

@NOT(<string>)

Description:

This function returns the value 1 (true) if the string is empty (null); otherwise the function returns null (false).

This function is equivalent to @NULL.

OK:

Unchanged.

Example:

```
IF @NOT(@ZERO(%2))
   GOTO loop
END
```

See also:

@BOTH @EQUAL @NULL @ZERO

@NULL

Syntax:

@NULL(<string>)

Description:

This function tests whether the enclosed string is empty. The value 1 (true) is returned if it is empty (null), otherwise the function returns null (false).

This function is equivalent to @NOT.

OK:

Unchanged.

Example:

```
IF @NULL(%2)
   WARN No command line parameter supplied!
END
```

See also:

@BOTH @EQUAL @NOT @ZERO

@NUMERIC

Syntax:

@NUMERIC(<string>)

Description:

This function returns a value of true (1) if the string is a valid number, otherwise it returns a value of false (null)..

OK:

Unchanged.

Example:

```
IF @NUMERIC(%A)
  %B = @SUM(%A,2)
END
```

See also:

@DIFF	@DIV	@FADD	@FDIV	@FMUL
@FORMAT	@FSUB	@HEX	@PRED	@PROD
@SUCC	@SUM	@ZERO		

@OK

Syntax:

@OK()

Description:

This function is used to test the status of various I/O commands which set the DialogScript OK flag. It returns 1 (true) if OK is true, otherwise the function returns null (false).

OK:

Unchanged.

Example:

```
IF @OK()
  rem Do something useful
ELSE
  WARN Operation failed!
END
```

See also:

@PATH

Syntax:

@PATH(<string>)

Description:

This function returns the path portion (up to and including the final colon or backslash) of the file path specified in <string>.

OK:

Unchanged.

Example:

```
%P = @PATH(C:\WINDOWS\WIN.INI)

REM %P now contains C:\WINDOWS\
```

See also:

@EXT @NAME @SHORTNAME

@POS

Syntax:

@POS(<string1>,<string2>)

Description:

This function returns the starting character position of the first occurrence of <string1> in <string2>. The characters in the string are counted from 1. The value 0 is returned if either string is empty or <string1> does not occur in <string2>. Note that the comparison is not case sensitive.

OK:

Unchanged.

Example:

%P = @pos(a,%A)

info Position of a in %A is: %P

See also:

@LEN @SUBSTR @UPPER

@PRED

Syntax:

@PRED(<value>)

Description:

This function returns the predecessor of <value>, i.e. <value> - 1. It is more efficient than using @diff(<value>,1)..

OK:

Set to false if <value> is null or non-numeric, or if overflow occurs.

Example:

%P = @PRED(%P)

See also:

@DIFF	@DIV	@FADD	@FDIV	@FMUL
@FORMAT	@FSUB	@HEX	@NUMERIC	@PROD
@SUCC	@SUM	@ZERO		

@PROD

Syntax:

@PROD(<value1>,<value2>)

Description:

This function returns the product of its two arguments: <value1> x <value2>.

OK:

Set to false if either of the arguments is null or non-numeric, or if overflow occurs.

Example:

%C = @PROD(%A,%B)

See also:

@DIFF	@DIV	@FADD	@FDIV	@FMUL
@FORMAT	@FSUB	@HEX	@NUMERIC	@PRED
@SUCC	@SUM	@ZERO		

@QUERY

Syntax:

@QUERY(<string>)

Description:

This function displays a dialog box containing a question mark icon, the message <string>, and buttons for OK and Cancel. The value 1 (true) is returned (and OK is set to true) if the user responds OK; otherwise the function returns null (false) and OK is set to false.

OK:

Unchanged.

Example:

```
IF @QUERY(Insert disk in drive A: and press OK)
  REM Do something with disk in drive A:
END
```

See also:

INFO WARN @ASK @MSGBOX @INPUT

@REGREAD

Syntax:

@REGREAD(<root key>, <subkey>, <name>, <default>)

Description:

This function returns the value of the specified key from the Windows registry.

Binary key values are returned as a string of numbers separated by fieldsep.

<root key> specifies the root key to search from. The permissible values are:

ROOT specifies HKEY_CLASSES_ROOT

CURUSER specifies HKEY_CURRENT_USER

LOCAL specifies HKEY_LOCAL_MACHINE

USERS specifies HKEY_USERS

STATS specifies HKEY_DYN_DATA

DEFAULT specifies the key Software\SADE\VDS\3.0\UserScripts in HKEY_CURRENT_USER.

<subkey> specifies the key value to retrieve. Use backslashes to specify keys several levels deep.

<name> specifies the named value to retrieve. If omitted, the value of the default value of the key is retrieved.

<default> specifies the value to be returned if the specified key does not exist.

It is recommended that the DEFAULT root key is used for any registry keys created for the use of DialogScript programs.

OK:

Set to false if the key does not exist, or if the contents exceed the buffer size. This can be changed using the OPTION REGBUF command.

Example:

```
%T = @regread(CURUSER,Software\SADE\VDS\3.0\SourceWin,Top)
```

See also:

REGISTRY OPTION DEFAULT OPTION REGBUF Tip

@RETCODE

Syntax:

@RETCODE()

Description:

This function returns the exit code (e.g. DOS errorlevel) of the last program executed using a RUN or SHELL command with a WAIT parameter.

OK:

Unchanged.

Example:

```
RUN PKZIP.EXE

info Return code was @retcode()
```

See also:

RUN SHELL

@SENDMSG

Syntax:

@SENDMSG(<window>, <message number>, <wparam>, <lparam>)

Description:

This function is for use only by very advanced users. You will need knowledge of Windows messaging and access to Windows API documentation to make full use of this function.

This function lets you use the Windows API SendMessage function to send a system message to another window. Messages are used to control windows (which include individual controls on a window) and to send information to them or retrieve information from them.

All arguments to the function must be supplied. The <window> argument is the identifier of the window that is to be the target of the message, and will normally be obtained using @WINEXISTS or @WINATPOINT. The <message number> argument is the number which identifies the Windows message.

The arguments <wparam> and <lparam> are parameters to the message. Their exact contents are dependent on the message type, and will be determined from the Windows API documentation. If an argument is a string starting with the characters $, - or 0 to 9, it is interpreted as a word or integer value (in hexadecimal if starting with $.). If the argument is the special function @STRBUF, it is replaced by the address of a buffer in which Windows will return a string value. If it is anything else, it is considered to be a string, and a pointer to the string is passed to the SendMessage API function. (The last two options are only valid in the <lparam> position.)

The function will return the numeric value returned by the SendMessage function, unless @STRBUF was used for the <lparam> value, in which case the return value is the string that was placed in the buffer.

Note: you can find the window identifier of a dialog element on the VDS program's dialog using @WINEXISTS with the argument of the dialog element name prefixed by a tilde '~'. This may be useful to perform operations that cannot be achieved in VDS any other way. Use of @SENDMSG can cause unpredictable effects, and may crash VDS or, indeed, Windows, so this command should only be used by those who know what they are doing.

OK:

Unchanged.

Example:

```
rem - make listbox LST display a horizontal scroll bar
%P = @sendmsg(@winexists(~LB),1045,1000,0)


rem - return index of item in list box that starts with "Line 5"
%P = @sendmsg(@win(~LB),$018F,0,Line 5)
rem - use $01A2 for match on whole item


rem - set EDIT dialog element to use a password character of X
%P = @sendmsg(@win(~EDIT1),$00CC,@asc(X),0)
```

See also:

WINDOW @WINACTIVE @WINATPOINT @WINEXISTS @WINPOS

@SHIFT

Syntax:

@SHIFT(<string>)

Description:

This function is only useful when used within the string argument of a WINDOW SEND command to send keystrokes to another window. It generates keystrokes that are the equivalent of holding down the Shift key while the keys in <string> are being sent.

OK:

Unchanged.

Example:

```
rem highlight some text

window send,Test - WordPad,@shift(@key(HOME)@key(DOWN)@key(DOWN))
```

See also:

@ALT @CR @CTRL @KEY @TAB

@SHORTNAME

Syntax:

@SHORTNAME(<file description>)

Description:

This function returns a DOS-compatible short filename version of the file path <file description> if it exists.

OK:

Set to false if the function fails.

Example:

run LIST @shortname(%1),wait

See also:

@FILE OPTION

@STRDEL

Syntax:

@STRDEL(<string>,<pos1>,<pos2>)

Description:

This function returns a string consisting of <string> with the characters in positions <pos1> to <pos2> deleted. The characters in the string are counted from 1.

If <pos1> is omitted or zero then the function returns <string> unmodified. If< pos2> is zero or omitted, just the single character at <pos1> is deleted. If <pos2> is negative, the ending position is counted from the end of the string.

OK:

Unchanged.

Example:

%S = @strdel(%A,4,8)

info Result of deleting chars 4 to 8 from %A is: %S

See also:

@LEN @POS @STRINS @SUBSTR @UPPER

@STRINS

Syntax:

@STRINS(<string>,<pos1>,<string2>)

Description:

This function returns a string consisting of <string> with <string2> inserted at position <pos1>. The characters in the string are counted from 1.

If <pos1> is omitted or zero then the function returns <string> unmodified. If <pos1> is greater than the length of <string> then <string2> is inserted at the end of the string.

OK:

Unchanged.

Example:

%S = @strins(%A,4,%B)

info Result of inserting %B into %A is: %S

See also:

@LEN @POS @STRDEL @SUBSTR @UPPER

@SUBSTR

Syntax:

@SUBSTR(<string>,<pos1>,<pos2>)

Description:

This function returns the substring of <string> starting at character position <pos1> and ending at character position <pos2>. The characters in the string are counted from 1.

If <pos1> is omitted or zero then the function returns <string> unmodified. If< pos2> is zero or omitted, just the single character at <pos1> is returned. If <pos2> is negative, the ending position is counted from the end of the string.

OK:

Unchanged.

Example:

%S = @substr(%A,4,8)

info Substring of %A from 4 to 8 is: %S

See also:

@LEN @POS @STRDEL @STRINS @UPPER

@SUCC

Syntax:

@SUCC(<value>)

Description:

This function returns the successor of <value>, i.e. <value> + 1. It is more efficient than using @sum(<value>,1)..

OK:

Set to false if <value> is null or non-numeric, or if overflow occurs.

Example:

%S = @SUCC(%S)

See also:

@DIFF	@DIV	@FADD	@FDIV	@FMUL
@FORMAT	@FSUB	@HEX	@NUMERIC	@PRED
@PROD	@SUM	@ZERO		

@SUM

Syntax:

@SUM(<value1>,<value2> {, ...})

Description:

This function returns the sum of its two or more arguments, which are treated as integers.

OK:

Set to false if either of the arguments is null or non-numeric, or if overflow occurs.

Example:

%S = @SUM(%S,10)

See also:

@DIFF	@DIV	@FADD	@FDIV	@FMUL
@FORMAT	@FSUB	@HEX	@NUMERIC	@PRED
@PROD	@SUCC	@ZERO		

@SYSINFO

Syntax:

@SYSINFO(<string>)

Description:

This function returns various system information dependent on the value of <string>.

The possible values are:

FREEMEM	Returns the amount of memory currently free, in Kb
PIXPERIN	Returns the number of pixels per inch of screen resolution
SCREENHEIGHT	Returns the height of the screen in pixels
SCREENWIDTH	Returns the width of the screen in pixels
WINVER	Returns the Windows version numbe
WIN32	Returns the value 1 (true) if running in a 32 bit version of Windows.
DSVER	Returns the DialogScript version number

OK:

Unchanged.

Example:

```
title System Information
%W = Windows version@tab()= @SYSINFO(WINVER)
%M = Free memory@tab()= @SYSINFO(FREEMEM)Kb
%S = Screen width@tab()= @SYSINFO(SCREENWIDTH)
%T = Screen height@tab()= @SYSINFO(SCREENHEIGHT)
%P = Pixels per inch@tab()= @SYSINFO(PIXPERIN)
info %W@CR()%M@CR()%S@CR()%T@CR()%P
```

See also:

Screen metrics

@TAB

Syntax:

@TAB()

Description:

This function returns a tab character. You use it to insert a tab in text which is being output, or to send a tab keystroke in a WINDOW SEND command.

OK:

Unchanged.

Example:

WINDOW SEND,Report - WordPad,Column 1@TAB()Column 2

See also:

@CHR @ESC @CR WINDOW

@TRIM

Syntax:

@TRIM(<string>)

Description:

This function returns a string which is the same as <string> but with leading and trailing spaces and other control characters removed.

OK:

Unchanged.

Example:

%T = @trim(@next(1))

See also:

@UPPER

Syntax:

@UPPER(<string>)

Description:

This function returns the string <string> converted entirely to upper case characters.

OK:

Unchanged.

Example:

%S = @UPPER(The quick brown fox)

REM %S now contains THE QUICK BROWN FOX

See also:

@LOWER

@VERINFO

Syntax:

@VERINFO(<filename>, <information type>)

Description:

This function returns the embedded version informaton (if present) about the specified file, which must be an executable file type.

The information type contains flags, which can be:

C - company name

D - file description

N - original file name

P - product name

T - type of executable file

V - version

X - product version

Y - copyright message

If no <information type> is specified, the default V is used.

If more than one flag is specified, each item of information is separated by the field separator character. The data can be stored in separate variables using the PARSE command.

The type of executable can be either 'NE', a Windows 16-bit New Executable file, or 'PE', a Windows 32-bit Portable Executable file. Other file types return the value '??'. Version information is not present in all Windows executable files.

OK:

Not changed.

Example:

%V = @verinfo(MYPROG.EXE,V)

See also:

@FILE

@VOLINFO

Syntax:

@VOLINFO(<drive>, <information type>)

Description:

This function returns informaton about the specified volume.

The information type contains flags, which can be:

F - return amount of space free (Kb)

N - return volume name

S - return total space on drive (Kb)

T - return type of volume:removable, Fixed, Network, CD-ROM or RAM.

Y - returns a text string describing the file system type (for example, FAT32, NTFS.)

Z - returns the hard disk serial number.

If no <information type> is specified, the default N is used.

The F and S flags which return the free space and total size of a drive return an incorrect result for volumes larger than 2GB in earlier versions of Windows 95. This is because of a bug in Windows 95. The function has been modified to use a new function present only in Windows 95 OSR2 or later and Windows NT 4.0 so that it reports these values correctly when run on those systems.

If more than one flag is specified, each item of information is separated by the field separator character. The data can be stored in separate variables using the PARSE command.

OK:

Set to false if function fails

Example:

```
rem find free space on D:

%E = @volinfo(D,F)
```

See also:

@FILE

@WINACTIVE

Syntax:

@WINACTIVE(<flags>)

Description:

This function is used to obtain information about the currently active window. This information is needed when using certain window control commands, such as WINDOW SEND.

The flags are:

C - returns the class name;

I - returns the window identifier;

N - causes the name (title bar text) to be returned. This is the default.

OK:

Unchanged.

Example:

%W = @winactive()

See also:

WINDOW @WINATPOINT @WINEXISTS @WINPOS

@WINATPOINT

Syntax:

@WINATPOINT(<x pos>, <y pos>)

Description:

This function can be used to obtain the window identifier of the window or control (such as a button or edit box) at absolute position <x pos>, <y pos> on the screen. This value can be used in commands like WINDOW SEND.or WINDOW SETTEXT.

OK:

Unchanged.

Example:

WINDOW SETTEXT,@winatpoint(225,304),This is the new text

See also:

WINDOW @WINPOS

@WINCLASS

Syntax:

@WINCLASS(<window>)

Description:

This function returns the window class name of the window specified in <window>. The class name is one of the ways that a Visual DialogScript program can identify a window.

OK:

Unchanged.

Example:

%C = @winclass(@winatpoint(256,72))

See also:

WINDOW @WINATPOINT @WINPOS @WINTEXT

@WINDIR

Syntax:

@WINDIR(<parameter>)

Description:

This function returns the full path of the Windows directory. The optional <parameter> may be either W or S. If S, the full path of the Windows system directory is returned instead.

OK:

Unchanged.

Example:

%W = @WINDIR()

REM On most systems %W will now contain C:\WINDOWS

See also:

@CURDIR

@WINEXISTS

Syntax:

@WINEXISTS(<window> {, <child window> })

Description:

This function is used to determine whether the window <window> is present or not. It returns the window identifier if a main window or dialog box with a title bar of <string> exists, and null (false) if not. The <window> is specified using its full title bar text or its class name.

To determine whether an MDI child window exists, or obtain its window identifier, the optional <child window> argument must be supplied, giving the full title bar text of the required window which is a child of <window>.

OK:

Unchanged.

Example:

```
if @not(@WinExists(Untitled - Notepad))
  run NOTEPAD.EXE
end
```

See also:

WINDOW @WINACTIVE @WINATPOINT @WINPOS

@WINPOS

Syntax:

@WINPOS(<window>, <flags>)

Description:

This function returns information about the position of the window <window> according to the value of <flags>.

Valid flags are:

T return the top co-ordinate

L return the left co-ordinate

W return the width

H return the height

S return the window status (1 = normal; 2 = iconized; 3 = maximized)

Where more than one flag is specified the information returned is separated using the current field separator in a format that can be processed by the PARSE command.

OK:

Set to false if the specified window cannot be found.

Example:

```
PARSE "%T;%L",@winpos(Explorer,TL)
```

See also:

WINDOW @WINACTIVE @WINATPOINT @WINCLASS @WINEXISTS @WINTEXT

@WINTEXT

Syntax:

@WINTEXT(<window>)

Description:

This function returns the text contents of the window specified in <window>. When the @WINATPOINT function is used to identify the window, this function can retrieve the text from controls such as buttons and edit fields.

OK:

Unchanged.

Example:

%T = @wintext(@winatpoint(256,72))

See also:

WINDOW @WINATPOINT @WINCLASS @WINPOS

@ZERO

Syntax:

@ZERO(<string>)

Description:

This function returns the value 1 (true) if the value of <string> is zero; otherwise the function returns null (false).

OK:

Unchanged.

Example:

```
IF @ZERO(%V)
  INFO Result is zero.
END
```

See also:

@DIFF	@DIV	@HEX	@NUMERIC
@PRED	@PROD	@SUCC	@SUM

Applications for Visual DialogScript

Examples of tasks for which you could use Visual DialogScript are:

- create an 'intelligent' start-up script to load applications or perform housekeeping tasks at start-up, dependent on time, date etc.;
- create a start-up menu with buttons to select which applications are loaded;
- create an 'Agent' script that runs in the background and performs actions at certain times;
- control other Windows applications by sending keystrokes or mouse clicks, using DDE and setting the size and position of their windows;
- create software installation programs;
- create simple utilities such as databases or system resource monitors;
- create front-ends for DOS programs, which can run invisibly in the background so that it looks as if a Windows application is running;
- produce interactive multimedia applications that display bitmaps and play sounds.

Assignments

Like commands, assignments need not start in the first character position, so they may be indented using spaces for readability.

An assignment consists of a variable name, an equals symbol and a string, (which may contain variable or function references) each separated by spaces.

Here are some examples of assignments:

```
%S = @VOLINFO(D:,S)

%T = Backing up drive %D
```

@Automating Applications

Visual DialogScript has several features which enable you to control other Windows applications. One method is to use Dynamic Data Exchange (DDE); however, this requires that the application to be automated is a DDE server, which few are. The most commonly used method is therefore to simulate key presses and mouse clicks. There are no hard and fast rules for achieving this and with some applications it can be quite difficult to do. This section explains the basic principles.

The key to most application automation operations is identifying the window that is the target of your key presses and mouse clicks. In Visual DialogScript you can use the text in the title bar of the main window, or you can use the window class name. However, neither of these will identify a specific instance of a window if two or more copies are running, since in many cases the title and class name will be the same for each instance.

If you launch the program you wish to automate from within your script then you can get the window identifier that is allocated by Windows to the instance of the application's window that has just been created. You can usually do this using the @WINACTIVE function, along the lines of:

```
%H = @winactive(I)

run myapp.exe

repeat

  wait 1

  %I = @winactive(I)

until @not(@equal(%H,%I))
```

Once you have got a way to identify your target window, automating it is simply a matter of using WINDOW SEND and WINDOW CLICK commands in your script.

You use WINDOW SEND to send keystrokes to the application. The command activates the named window, and then sends the keystrokes, which will be directed to the control that has input focus at that time. The main difficulty is in timing the arrival of the keystrokes, since your program can send them much faster than a user seated at the keyboard would. You can use OPTION SKDELAY to add a fixed delay between each keystroke, which can make this more reliable.

Mouse clicks can be sent using WINDOW CLICK (or WINDOW RCLICK to send a right-button click.) As with WINDOW SEND, the command activates the named window to make sure it is fully visible before the mouse click is sent. To double-click you just use two WINDOW CLICK commands. To make things easy the X and Y co-ordinates of the pointer are expressed relative to the top left corner of the named window.

The biggest problem with automating applications is the inability of your program to get the visual feedback that a real user would get that an operation has finished and it is alright to continue. The usual solution is simply to use WAIT commands to allow a long enough period for any lengthy operation to finish.

Any operation that causes another window or dialog box to appear should ensure that it has appeared by testing for it, for example using the @WINEXISTS function. This function returns the identifier of the window, which will normally be needed if you want to direct any keystrokes to it.

Sometimes you can get information from a field displayed on the window, for example, the status line. You could do this using the instruction:

```
%T = @wintext(@winatpoint(%X,%Y))
```

The @WINTEXT function returns the contents of the text property of the window whose identifier is passed as its argument. This can be a button caption, the contents of an edit field or label text. (To understand this it is necessary to realise that in Windows terms almost all the items you see on a screen are in fact windows.) The @WINATPOINT function returns the handle of the window (or control) at the point specified by variables %X and %Y. Note that the co-ordinates X and Y in this case are relative to the top left corner of the screen: since the position of a window control will usually be defined relative to the top left co-ordinate of its main window you will need to use the @WINPOS function to find this and then do some arithmetic.

You cannot extract any item of text that you see using @WINTEXT. It is entirely application dependent. Be aware, too, that features like user-selectable toolbars or a display mode that uses large fonts can alter the position of the controls you want which can make calculating the X and Y co-ordinates of each control quite involved. But then nobody said this was supposed to be easy!

The WINDOW SETTEXT command can sometimes be used to set the contents of edit fields, instead of WINDOW SEND. If you use this, the window identifier must be that of the exact control that is to receive the text, as obtained from @WINATPOINT. This method bypasses the normal method of data entry and so may cause problems, depending on how the target application has been written. It is a case of try it and see. You can also

use this command to change button captions and other normally inaccessible text, though it is difficult to think of a useful application for this.

BEEP

Syntax:

BEEP

Description:

Sounds a beep. This command can optionally accept an argument, an integer value corresponding to the constants permitted in the MessageBeep API.

OK:

Unchanged.

Example:

BEEP

See also:

DIALOG ADD BITMAP,<name>,<top>,<left>,<width>,<height>,<filename>,<style>{,<style>}

This dialog element creates a bitmap at the position and size specified, containing the image <filename>.

The BITMAP dialog element supports only BMP and ICO files. In addition, it can load a bitmap from a BMP file that has been compiled into a resource file (special VDS resource format.) In this case, the filename must be followed by a vertical bar and the offset, in bytes, of the BMP file within the resource. No spaces are permitted between the filename, the vertical bar and the offset.

If the width or height are omitted then the bitmap control is automatically sized to the image.

If the STRETCH style is specified then the image is sized to fit the bitmap control (note that this has no effect on icons.)

If CLICK is specified, the bitmap will generate a <name>CLICK event when clicked with the mouse. You can test whereabouts on the bitmap the mouse was clicked and which button was used with the @CLICK function.

The HAND style is the same as CLICK, but a hand cursor will be shown when over the bitmap. The CROSS style is the same as CLICK, but a cross cursor will be shown when over the bitmap.

DIALOG ADD,BUTTON,<name>,<top>,<left>,<width>,<height>,<caption>,{ styles...}

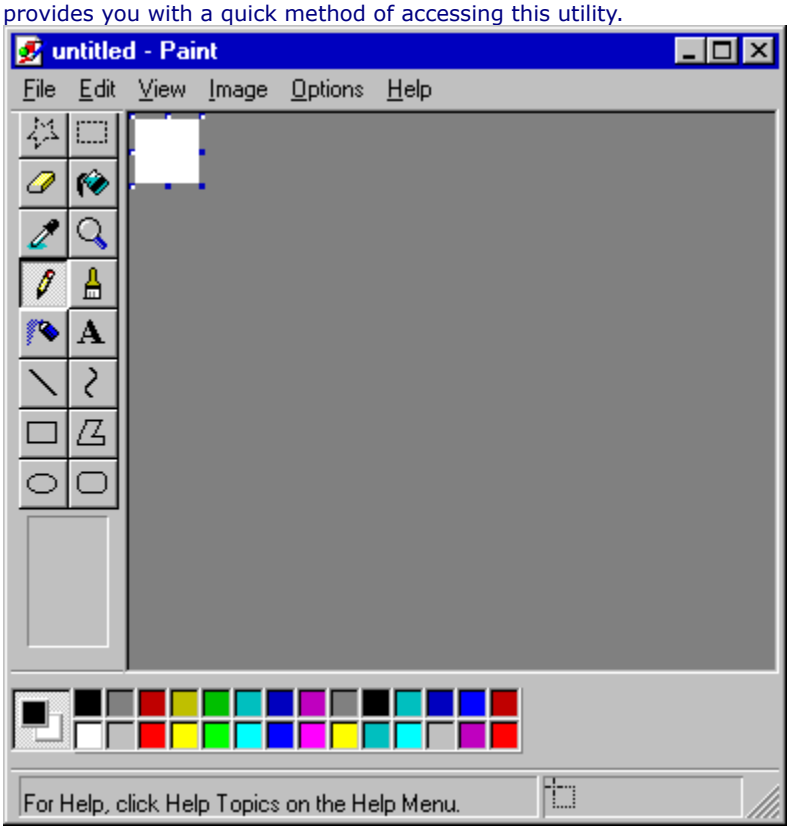
This dialog element creates a button at the position and size specified. The name is used as the caption for the button. If you want to use characters in the button caption that are invalid in a name, such as the & symbol which makes the following character a keyboard shortcut, you can specify a separate caption. When the button is pressed, it will cause a<name>BUTTON event.

If button is called CANCEL, a CANCELBUTTON event will occur when the Esc key is pressed, as well as when the button is clicked.
If button is called HELP, a HELPBUTTON event will occur when the F1 key is pressed, as well as when the button is clicked.

A keyboard shortcut, shown by an underline, can be created by putting an & before a letter in the button caption. This is a Windows feature.

Bitmap Editor

Windows already includes a useful bitmap editor - Windows Paint. The Bitmap Editor option of the Tools menu provides you with a quick method of accessing this utility.



DIALOG ADD,CHECK,<name>,<top>,<left>,<width>,<height>,<caption>,<value>,{ styles.. }

This dialog element creates a check box at the size and position specified, with a text caption of <caption> and an initial state of <value> which may be unchecked (0) or checked (1).

If <value> is null for the box to be unchecked, and non-null for it to be checked. This is consistent with the treatment of other logical operations, such as the evaluation of expressions.

If the CLICK style is specified a <name>CLICK event will be generated whenever the check box is clicked.

CLIPBOARD

Syntax:

CLIPBOARD APPEND, <string>

CLIPBOARD CLEAR

CLIPBOARD SET, <string>

Description:

The CLIPBOARD command is used to put data into the Windows clipboard.

CLIPBOARD APPEND adds the contents of <string> to whatever is already in the clipboard. Successive appends add the text on a new line.

CLIPBOARD CLEAR empties the clipboard of any data that was in it.

CLIPBOARD SET sets the contents of the clipboard to the text <string>.

OK:

Unchanged.

Example:

```
CLIPBOARD SET,Hello Clipboard!
```

See also:

LIST

DIALOG ADD,COMBO,<name>,<top>,<left>,<width>,<height>,<value>, {style...}

This dialog element creates a combo box at the position and size specified. A combo box is a combination list box and edit control. To get data into and out of the edit control you use the DIALOG SET command and @DLGTEXT function. To get data into and out of the list box you must use the LIST command.

The SORTED style specifies whether the list items are to be maintained in ASCII order or not. Data entered in the edit control is automatically inserted into the drop-down list when the focus moves away from the control.

The LIST style causes the combo box to work like a LIST dialog element, so you cannot enter data into the control, only choose values from the drop down list.

The CLICK style causes a <name>CLICK event to be generated when an item is chosen from the drop-down list.

The EXIT style causes a <name>EXIT event to be generated when the combo box loses the input focus.

Command Reference

Commands

Unlike labels, commands need not start in the first character position. It is recommended that they are indented using spaces for readability.

A command consists of the command name (see Command Reference) followed optionally by a string. The string is used as the argument (or parameters) to the command. Most commands have only a single argument, but some have more than one, in which case commas are used to separate the parameters. A space must separate the command from the string. Commands are not case-sensitive.

Here are some examples of commands:

```
TITLE My first script

INIFILE WRITE,Reg_Info,UserName,Fred Bloggs
```

Strings may include variable and function references, which are evaluated before the command is carried out. Here is an example of commands containing variable and function references:

```
IF @FILE(%F)
  INFO File %F exists
END
```

Contents

Creating Dialogs

The DIALOG command is used to create a dialog window for the program and manage its controls. The dialog becomes the program's main window, and will exist until the program terminates.

The dialog definition takes the basic form of:

```
DIALOG CREATE, ...      define dialog window, title, size and properties
DIALOG ADD, ...         define the dialog elements
DIALOG SHOW             show the dialog.
```

Dialog elements begin with the element type appended to the DIALOG ADD command, which is optionally followed by details such as the control name, size, position and initial value. The details are contained within brackets and separated by semicolons. See Dialog Elements for more information.

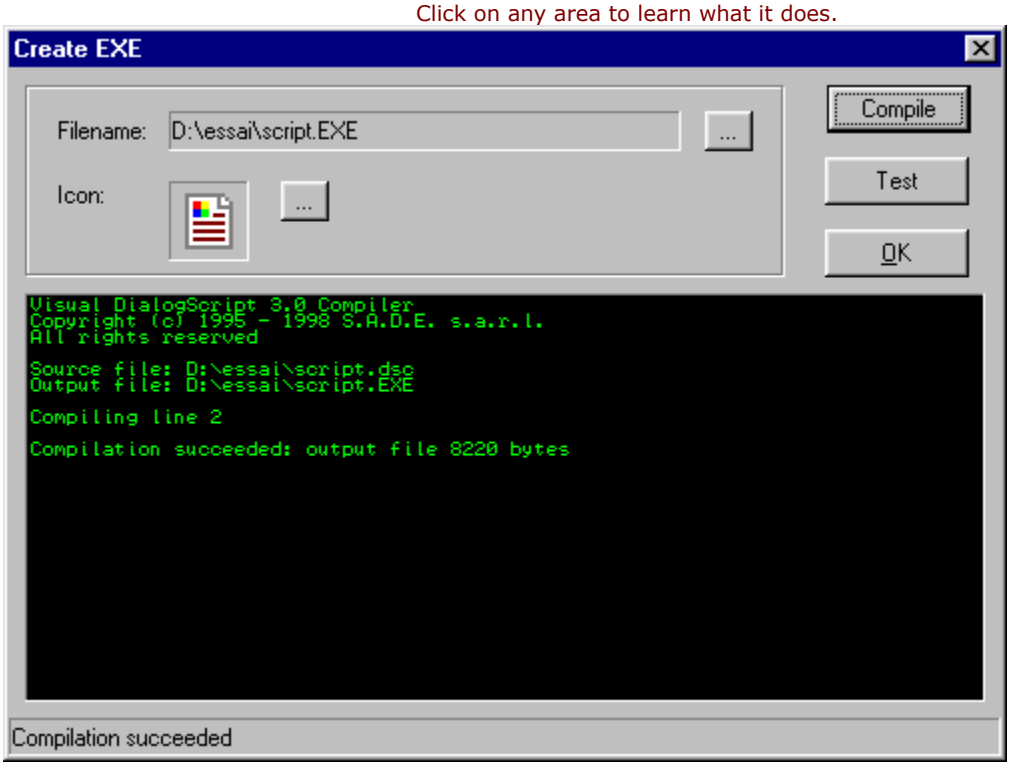
Visual DialogScript's dialog designer allows you to design a dialog box interactively, and then generates the DIALOG CREATE command for you.

Creating Executable Files

Once your DialogScript programs are finished and fully debugged you will probably want to run them on their own, as standalone programs. To do this you must create an executable file.

From the File menu select Compile to EXE, and this dialog appears:

Click on any area to learn what it does.



The settings in this dialog are stored in the in the .DSP file which is created for each script. So any changes made do not have to be recreated the next time you create an executable from the same script.

DDE

Syntax:

DDE LINK, <servername>, <topicname>

DDE EXECUTE, <macro>

DDE POKE, <itemname>, <data>

DDE TERMINATE

Description:

The DDE commands are used to establish and terminate a link using DDE between a script program acting as a DDE client and another application acting as a DDE server. Note that Windows requires programs using DDE communication to have a window, so this command will generate an error unless a dialog box has been created.

The DDE LINK command is used to initiate the DDE link before any communication can take place. The DDE TERMINATE command is used to terminate the link once communication has finished.

The DDE EXECUTE command is used to send to the server application a command (or DDE macro) to be executed. The commands that are valid are dependent on the application. Tip

The DDE POKE command is used to send data to a named item in the DDE server. Many applications do not accept poked data.

The DDE TERMINATE command closes the DDE link. It is good practice to do this once DDE communication has finished.

OK:

Set to false if the DDE command fails.

Example:

```
DDE LINK,Progman,Progman
if @ok()
  DDE EXECUTE,[CreateGroup(%U)]
end
if @ok()
  DDE EXECUTE,[ShowGroup(%U",1")]
end
if @ok()
  DDE EXECUTE,[AddItem(@shortname(%D\ds.exe)",Visual DialogScript")]
  DDE EXECUTE,[AddItem(@shortname(%D\ds.hlp)",Visual DialogScript 2 Help,winhelp.exe")]
  DDE EXECUTE,[AddItem(@shortname(%D\tutorial.hlp)",Visual DialogScript 2 Tutorial,winhelp.exe")]
end
if @not(@ok())
  warn Setup failed to create start menu shortcuts
end
DDE TERMINATE
```

See also:

@DDEITEM LINK

DIALOG

Syntax:

DIALOG ADD,<element_type>,<element_name>, element description

DIALOG CLEAR, <control name>

DIALOG CLEARSEL, <control name>

DIALOG CLOSE

DIALOG CREATE,<title>,<top>,<left>,<width>,<height>,{ <styles> }

DIALOG CURSOR {, WAIT}

DIALOG DISABLE, <control name>

DIALOG ENABLE, <control name>

DIALOG FOCUS, <control name>

DIALOG HIDE, <control name>

DIALOG POPUP, <items> {<x>,<y>}

DIALOG SELECT, <id>

DIALOG SET, <control name>, <text>

DIALOG SETPOS, <name>,<top>,<left>,<width>,<height>

DIALOG SHOW, <element name>

DIALOG SHOWMODAL

DIALOG TITLE, <title>

Description:

DIALOG ADD adds a new dialog element to the currently selected dialog. The element types, and the valid description for each of them, see "Dialog Elements".

DIALOG CLEAR clears the text in the control named <control name>.

DIALOG CLEARSEL clears the selected item in a list box control named <control name>.

DIALOG CLOSE causes a CLOSE event to be issued for a main window. However the window will not close until the application terminates. For a child window, this command will also issue a CLOSE event. (If @event(d) is used, the dialog ID will ne > 0.) The effect is the same as clicking the close (X) button. The script can respond to this by reading information from any dialog elements, if required. A second DIALOG CLOSE must then be executed (which will generate a second CLOSE event, which can be thrown away by calling @event().) This time the child window will close.

DIALOG CURSOR, WAIT sets the cursor for the dialog to an hourglass. The command DIALOG CURSOR sets it back to the default cursor. To provide feedback to the user your script program should show the hourglass cursor whenever it will take some time to respond to a user action.

DIALOG CREATE function is just to create a (hidden) dialog on which elements are added using DIALOG ADD, and which is then displayed using DIALOG SHOW. It therefore contains just the title, size and position, and any styles that relate to the dialog. This command has many different parameters which form the <description>. It is described in more detail in the topics Creating Dialogs and Dialog Elements. You would normally use the Dialog Designer to generate this command. Note that the <width> and <height> values are the width and height of the client area of the dialog window, in other words the area in which it is possible to place dialog elements. The total width of the dialog is slightly larger than this

value. The total height is the height value plus a small amount for the border, plus the title bar height, plus the menu height if a menu is defined. It means that you don't have to worry about calculating the size to take account of non-standard desktop settings (e.g. large fonts, desktop themes etc.)

DIALOG DISABLE and DIALOG ENABLE disable and enable the control named <control name>.

DIALOG FOCUS sets the input focus to the control named <control name>.

DIALOG HIDE and DIALOG SHOW hide and show the control named <control name>.

DIALOG POPUP This command is completely changed. There is no longer a POPUP dialog element. Instead, popup menus are defined when processing the CLICK event that triggers them. This means that the items on the popup menu no longer have to be fixed at design time so a popup menu can be a true context menu.
DIALOG POPUP,<items> {<x>,<y>} <items> is a list of menu items, separated by vertical bars. For example: Cut|Copy|Paste. Each item, when clicked, generates an event of type <name>MENU, for example: CutMENU if you click the Cut item on the popup. The optional <x> and <y> are numbers that specify the screen co-ordinates where the menu is to pop up. If omitted it will pop up where the mouse pointer is.

DIALOG SELECT is needed for scripts that use more than one dialog. The DIALOG commands, and functions that operate on dialog elements such as @dlgtext, search only the currently selected dialog for the named dialog element. This command is used to ensure that the correct dialog is selected. DIALOG SELECT,<id> The <id> is a number that refers to the dialog's position relative to the main window dialog 0 (zero) in a list. <id> 1 would be an only child dialog. Multiple dialogs can be created, then closed in a different order, in which case the <id> to refer to a particular dialog will change. Only advanced users should make use of this feature!

DIALOG SET sets the text in the control named <control name> to <text>. When applied to LIST dialog elements, this command sets the contents of the list box <name> to <text>. To replace the selected item you should use LIST PUT,<name>,<text> instead. To clear a check box use DIALOG SET,<name>. (In other words, the third parameter is omitted, not zero). To set it use DIALOG SET,<name>,<anything>. This is a change which was made for consistency: a check box is a boolean item and VDS treats a null value as false, not zero. Anything else (non-null) is treated as true.

DIALOG SETPOS,<name>,<top>,<left>,<width>, <height>
DIALOG SETPOS lets you alter the size or position of a dialog element (or the dialog itself) at run-time. <name> is the name of the dialog element, such as EDIT1. The <top>,<left>,<width> and <height> are the values for those respective properties. For each parameter value that is left null, the existing value is left untouched. If <name> is null, the command refers to the active dialog. In this case, <width> and <height> specify the size of the client area of the dialog, not including any border, menu bar or title bar.

DIALOG SHOW ,<element name>
Un-hides an element that has been hidden. If <element_name> is omitted, the dialog that has just been created (or the selected dialog which is currently hidden) is shown.

DIALOG SHOWMODAL This command is similar to DIALOG SHOW, except that the script freezes on this command until a button is pressed. When a button is pressed, the <button_name>BUTTON event is generated, but the dialog behaves as if it has been closed. The script can use @event() to test which button was pressed, and if it was not a Cancel button get information from the dialog. Then it must use DIALOG CLOSE to close finally the dialog.

DIALOG TITLE sets the text in the title bar of the dialog window to <title>. This is different from setting the title of the script program, which is done using the TITLE command.

Dialog styles are:

CLICK: generate a CLICK event if user clicks on the surface of the dialog;

DRAGDROP: generate a DRAGDROP event if user drops file(s) on the dialog;

NOSYS: don't have a system menu or close (X) box;

NOMIN: din't have a minimise button;

NOTITLE: creates non-movable window with no title bar;

ONTOP: keep the dialog on top of other windows;

SAVEPOS: save the dialog position (and size) and restore from the saved information next time it is created. This style can have an optional text ID to distinguish between different dialogs in a multi-dialog script, for example: SAVEPOS Options.

SMALLCAP: have a small caption bar; (if used, this style must come first.)

RESIZABLE: allow the window to be resized. If this style is used, RESIZE events will occur. The script must respond to them and use @dlgpos to obtain the new width and height, then use DIALOG SETPOS to reposition or resize any dialog elements that are affected by this.

CLASS: This style must be followed by a name, which will be used as the window class name for the window or dialog instead of the default which is TVDSDialog. For example: CLASS TOptionsDlg.

PAINT: This style will cause PAINT events to be generated whenever the dialog is redrawn. This occurs when it is resized (if RESIZABLE), restored or when any part of the dialog surface that has formerly been obscured by another window now becomes visible. This style is **only** likely to be of use to writers of extension DLLs that draw directly on to the surface of the dialog. It serves no useful purpose in any other context, since VDS itself takes care of redrawing all dialog elements that need to be redrawn. Because VDS events are not queued, it is recommended that resizable windows that also use PAINT events should treat both RESIZE and PAINT events the same, i.e.: the two labels are placed one after the other so the same event processing code is executed. Otherwise one or other event may be lost and the corresponding code not executed.

OK:

Unchanged.

Example:

```
dialog title,Address Book
dialog clear,Name
dialog set,NM,1
dialog focus,Name
```

See also:

@DLGTEXT TITLE Dialog Programming

DIRECTORY

Syntax:

DIRECTORY CHANGE, <path>

DIRECTORY CREATE, <path>

DIRECTORY DELETE, <path>

DIRECTORY RENAME, <path1>, <path2>

Description:

The DIRECTORY command is used to change, create, delete or rename directories.

DIRECTORY CHANGE changes the current directory to the one named in <path>. If <path> is on a different drive to the current drive then this is changed as well. This command is similar in operation to the MS-DOS CHDIR command.

DIRECTORY CREATE creates a new directory named <path>. If necessary, it will recursively create all the subdirectories in the path. This command is similar in operation to the MS-DOS MKDIR command.

DIRECTORY DELETE removes the directory named in <path>. The directory must be empty or it will not be removed.

DIRECTORY RENAME renames the directory named in <path1> to the directory named in <path2>.

Both DIRECTORY DELETE and DIRECTORY RENAME now use the Windows 95 Shell API, as does the FILE command. This means that the ALLOWUNDO, CONFIRM and SHOWERRORS options are supported with these two operations. See the FILE command for more information.

OK:

True if the operation is successful; false if not.

Example:

directory delete,f:\tmp1

directory create c:\test\subdir1\subdir2

See also:

Data Lists

Syntax:

LIST DROPFILES,<list>

LIST FILELIST,<list>, <filespec>, {<attributes>}

LIST LOADFILE,<list><filename>

LIST LOADTEXT,<list>

LIST REGKEYS,<list>,<root key>, <subkey>

LIST REGVALS, <list>,<root key>, <subkey>

LIST SAVEFILE,<list>,<filename>

LIST WINLIST,<list>,{<flags>}

Description:

These LIST commands are used to get data into string lists. The parameter <list> must be either a list number or the name of the dialog list control to which the command will apply. An error will occur if the list does not already exist.

DROPFILES is used to add to the list <list> the names of files that have been dropped on to a dialog window that has the DRAGDROP property.

FILELIST is used to add to the list <list> the names of files that match a particular specification, which may include wildcards. Note that whether just the name and extension or the full path is returned depends on whether a full path is given in <filespec>. The file specification may optionally be followed by a list of attributes which will be used to filter the list of files selected. The attributes may be specified as: A - archive; D - directory; H - hidden; R - read only; S - system; V - volume label. The attributes are additive, so if you specify HS for example the list will contain files that have the hidden attribute, the system attribute or both..

LOADFILE is used to create a list holding the contents of a named text file.

LOADTEXT loads text from the script file into the list. The text to be loaded should immediately follow the command, and each line should begin with a double-quote (") in column 1. (There is no need for a closing quote. The quote is just an indication to the interpreter that the line is to be added to the string list not treated as a command.)

REGKEYS is used to obtain a list of subkeys of the named subkey (see the REGISTRY command for more details).

REGVALS is used to obtain a list of values of the named subkey.

SAVEFILE is used to save the contents of a list to a named text file.

WINLIST is used to obtain a list of all the windows (including hidden windows) present on the system. The flags may be specified as: C - class name; I - window identifier; N - window name or title. If more than one flag is specified then the values are concatenated in the list with each one separated by the current field separator character in a form suitable for splitting up with the PARSE command.

OK:

Set to true if the command is successful or false if it fails.

Example:

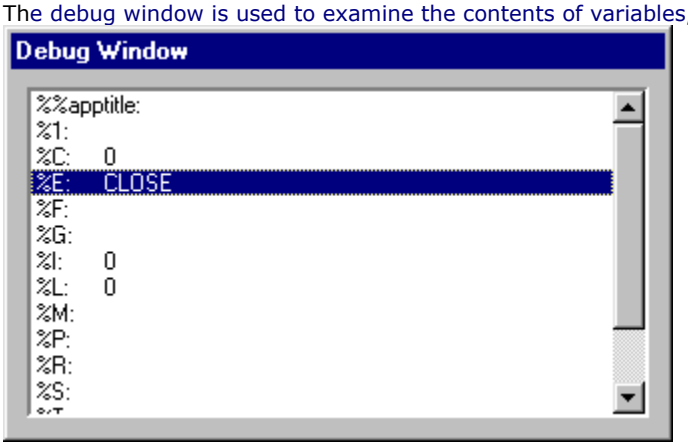
```
LIST LOADFILE,1,C:\CONFIG.SYS

LIST FILELIST,LB1,*.tmp
```

See also:

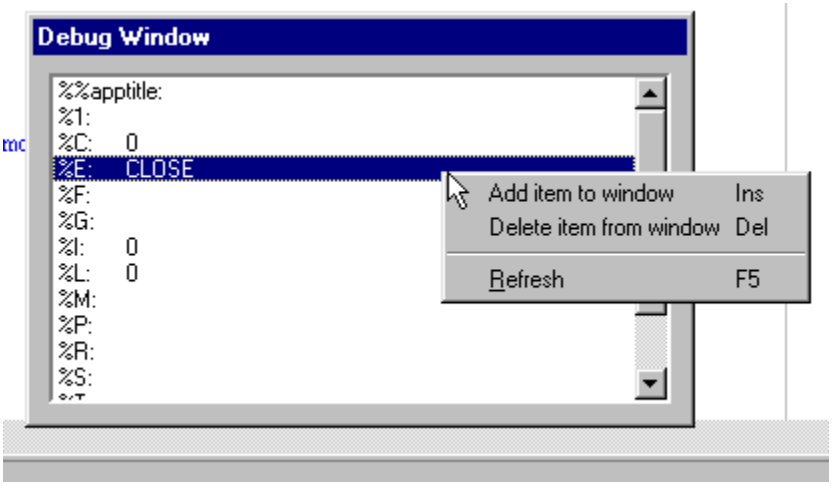
@COUNT @INDEX @ITEM @MATCH @NEXT LIST Using Lists

Debug Window

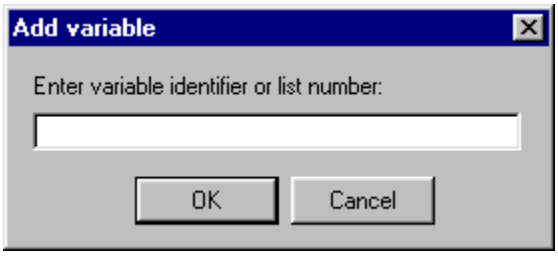
The debug window is used to examine the contents of variables, lists and the OK status variable.



This window is most useful when single-stepping through a script. To watch a variable value, you have to add it by a double click in the variable name itself in the script window or by a right click in the debug window to get a popup menu:



then select "Add item to window" to get the dialog below:



Debugging Scripts

When a script does not behave as expected it can sometimes be difficult to work out why. Writing programs is never easy, but Visual DialogScript comes with a set of tools that makes it as easy as possible to work out what is going wrong.

Start by resetting the program (menu Run / Reset) so that you start running from the beginning. Then step through your script a line at a time using the Single Step button or the F8 key. Open the Debug Window so that you can see the result of all the variables used by your script after each line has been executed. This is usually enough to work out what the problem is, if you think after each line about what the correct values are supposed to be.

Sometimes a command will not work, and will report that it does not work by setting the OK indicator to false, rather than by halting with an error message. The script language does this to give you a chance to cope with the error within your script, rather than have a user presented with a cryptic error code. However, if you don't use the @OK function to test the result of OK, but just assume the command or function will work, a script will not work correctly if the command fails. You can check the status of OK at any time during debugging as it is shown in the status bar of the debug window.

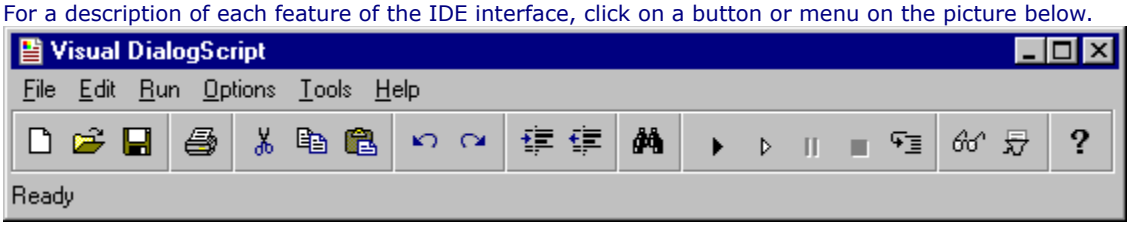
A common source of problems is failure to get information from files. This is usually caused by path problems. If you only specify a filename and not a full path, the script will look in the current directory for the file. The current directory is not necessarily the directory in which the script program resides. You should always specify a full path when referencing any file. If the file belongs to the script and will always be kept in the same directory use @PATH(%0) to get the directory of the script program.

If you have a complicated script and it would be too long-winded to step through the whole thing line by line then you can set breakpoints to halt the script at a particular point. For more complex problems you can insert BREAK commands with an @EQUAL, @GREATER, @NULL or @ZERO function in the first parameter, which will cause a breakpoint to occur when the conditional function evaluates to True.

Development Environment

Visual DialogScript has an interactive development environment (IDE) which makes it easy to develop and debug your Windows scripts. The environment is run from the main window, shown below, which is normally positioned at the top of the screen.

For a description of each feature of the IDE interface, click on a button or menu on the picture below.



Scripts are edited in the script window, which is an advanced text editor with syntax highlighting. Scripts can be run from the main window, using either the menu options or the toolbar buttons.

Visual DialogScript provides several aids to debugging. You can set breakpoints at any point in the script throught the script window, and then check the value of each variable using the debug window. You can also use single-step mode to step through the script a line at a time.

The Options menu lets you set your preferences for various options in the development environment.

The Tools menu provides access to tools which you may find helpful when creating your script program. You can add your own tools to the menu. Check our Web site for add-in tools you can download.

Dialog Elements

Dialog elements are parameters to the DIALOG CREATE command which specify the characteristics of the dialog window you want to create and the controls that should appear on it.

Most dialog elements have parameters, which are appended to element name. The parameters are separated by commas. The name parameter, where required, is mandatory and is used to address the control when you want to write text to it or read text from it. Most of the remaining parameters are optional, and may be left as null or omitted;. When omitted, DialogScript will use suitable defaults. With controls you will usually want to specify at least the top and left position co-ordinates. Position co-ordinates are relative to the client area of the dialog window.

The following dialog elements are available:

BITMAP,<name>,<top>,<left>,<width>,<height>,<filename>,<style>{,<style>}

BUTTON,<name>,<top>,<left>,<width>,<height>,<caption>,<style>

BITBTN,<name>,<top>,<left>,<width>,<height>,<filename>,{<caption>,<hint>}
CHECK,<name>,<top>,<left>,<width>,<height>,<caption>,<value>,<style>{,<style>}

COMBO,<name>,<top>,<left>,<width>,<height>,<value>,<style>{,<style>}

EDIT,<name>,<top>,<left>,<width>,<height>,<value>,<style>{, <style>}

GROUP,<name>,<top>,<left>,<width>,<height>,<caption>{,style...}

LIST,<name>,<top>,<left>,<width>,<height>,<style>{,style>}

MENU,<menu_name>,<item_1>,<item_2>...

PROGRESS,<name>,<top>,<left>,<width>,<height>,<value>

RADIO,<name>,<top>,<left>,<width>,<height>,<caption>,<value list>,<value>,<style>{, <style>}

STATUS,<name>,<value>,<style>

STYLE,<name>

TAB,<name>,<top>,<left>,<width>,<height>,<tabs>{,<style>}

TASKICON,<name>,<filename>,<tooltip text>

TEXT,<name>,<top>,<left>,<width>,<height>,<value>,<style>

The order of specifying the dialog elements can be important. The first button to be specified will be the default button which is executed if the user presses Enter. The tab order of controls that can accept input will be the order of specification, and the first such control to be specified will be the one that has the input focus when the dialog window is created.

An easier way to create a dialog than by working out a list of dialog elements is to use the dialog designer .

Dialog Programming

DialogScript allows you to create multiple dialog windows which function as window for your script application. Dialog windows may contain a number of controls, such as text controls and a status panel for displaying captions and other information, and edit controls, check boxes and list boxes which can not only display information but allow interaction with the user, plus buttons and menus which tell you when to process information by generating events.

You create a dialog using the DIALOG CREATE command. You can design the dialog interactively using the dialog designer , which will then generate the correct DialogScript code to create the dialog. The dialog must include buttons which users can press when they want the script to do something with the information in the dialog.

Note that the language makes it possible to write more sophisticated programs in which conditional statements and/or calculations are performed between the DIALOG CREATE and the DIALOG SHOW to vary the appearance of the dialog in real-time.

HOWEVER (** important!! **) the Dialog Designer can only create simple dialogs consisting of DIALOG CREATE, a number of DIALOG ADD lines, and DIALOG SHOW, where all the arguments to these commands are constants. More important still, the Dialog Designer can only edit such dialogs. So, if you use the full flexibility of the language to do calculations or execute conditional statements within the definition of a dialog, you will lose the ability to edit the dialog with the Dialog Designer. If you try to do so, the Dialog Designer might lose some of the other commands, or crash.

You can create more than one dialog. The first one, ID 0 (zero), is the application's main window. Once created, it will not close until the program terminates. When you process the CLOSE event for this dialog you should save any information and go to the EXIT or STOP command. Other dialogs we will call child dialogs.

Subsequent dialogs can be created and closed at will. DIALOG commands (and also LIST commands and functions that refer to LIST or COMBO dialog elements) refer to the active dialog. If the name of a dialog element on a different dialog is used, you will get a fatal error. To allow a particular dialog to be specified, you can use the DIALOG SELECT command.

When you close a child dialog, either by clicking its Close button or by executing a DIALOG CLOSE command, it does not close straight away. A CLOSE event is generated. A script can respond to this event by saving information in the dialog, then issuing another DIALOG CLOSE which this time closes the dialog.

Because events can have the same name, no matter which dialog generated them, the @EVENT function has been enhanced so you can obtain the dialog ID. Programming with multiple dialogs is quite difficult so it is best to examine the simple examples that show how it is done.

When a button is pressed it generates an event. For a user-defined button the name of the event is the name of the button followed by BUTTON; for example, when the OK button is pressed an OKBUTTON event occurs. The dialog close button (and selecting Close from the system menu) generates a CLOSE event. Other examples of events are the DRAGDROP event, which occurs if the dialog window is drag and drop enabled and files are dragged to the window, and the CLICK event which occur when the mouse is clicked over certain controls. See Events for more information.

There are two ways to process events. You can use WAIT EVENT. This halts the script entirely until an event occurs. When it does, you can test it using the @EVENT function, carry out whatever processing is required, and if appropriate loop back to the WAIT EVENT command to wait for the next event.

If you require your script to do other work while the dialog is displayed then you can simply test @EVENT regularly: it will return null if no event has occurred. If your script needs to respond to events as well as doing some processing on a regular basis you can use WAIT EVENT,<n>, which in addition to dialog events will generate a TIMER event every n seconds.

The dialog will remain until the program terminates, when the final EXIT command is executed.

The simplest way to write a dialog-based DialogScript program is to use the Application Wizard. This lets you design the dialog using the dialog designer and then generates a skeleton program with labels for all the possible events. All you need do is write the code to respond to each event.

Application Wizard

The Application Wizard helps you to write a dialog-based DialogScript program.

When you run the Wizard, you enter a title for your program, design the dialog using the dialog designer, and then the Wizard generates all the code for the program, including dummy handlers for each of the possible events. You then simply replace the dummy event handlers with your own code.

DialogScript Language

The DialogScript programming language has been designed to be simple, flexible and easy to use. The language has three main elements, labels, commands and assignments.

DIALOG ADD,EDIT,<name>,<top>,<left>,<width>,<height>,<value>,{,styles}

This dialog element creates an edit (input) box at the position and size specified, containing the text <text>.

Available styles:

PASSWORD style, specific to edit controls, causes asterisks to be displayed for every character typed.

EXIT style causes a <name>EXIT event to be generated whenever the input focus leaves this window control. One use of this would be to invoke a validation procedure.

MULTI: This makes the element into a multi-line edit control, similar to Notepad (and with similar restrictions, such as an approximately 32KB limit on the size of text.)

WRAP: This style applies if the dialog element is multi-line, and causes text to be word-wrapped within the boundaries of the element.

SCROLL: This style applies if the dialog element is multi-line, and causes scroll bars to appear so that non word-wrapped text (or longer text than can fit in the control) can be viewed and edited.

READONLY: makes the text in the control read only.

TABS: Allows tab characters to be entered when editing text. If this style is not present, hitting tab will cause focus to go to the next control, instead.

ELSE

See: IF

END

See: IF

EXIT

Syntax:

EXIT

Description:

When obeyed after a GOSUB command, causes execution to continue at the line following the GOSUB. Otherwise, EXIT causes execution of the script to terminate.

OK:

Unchanged.

Example:

EXIT

See also:

GOSUB STOP

EXITWIN

Syntax:

EXITWIN <exit option>

Description:

This command lets you shut down Windows under script control.

The valid options are:

SHUTDOWN	A normal shutdown. This is the default.
REBOOT	Shuts down Windows and reboots the system.
LOGOFF	Logs the user off the system (the program that issued the command appears to remain running)
FORCE	Forcibly shuts down Windows without allowing programs to display any "OK to close" messages or similar.

OK:

Unchanged.

Example:

EXITWIN REBOOT

See also:

EXTERNAL

Syntax:

EXTERNAL <DLL path>, <string>

Description:

This command is used to install a Visual DialogScript extension. This is a dynamic link library which adds a new command and function to the DialogScript language, which may then be used within the script.

Developers wishing to create Visual DialogScript extensions can obtain documentation describing the extension API on request.

OK:

Unchanged.

Example:

EXTERNAL VDSOLE.DLL,100

See also:

Error Messages

1 Invalid Command

An invalid DialogScript command has been encountered. Usually this means it has been misspelt.

2 Missing parameter(s)

The command executed expects more parameters than the number given.

3 Style already defined

An OPTION STYLE command defining this style has previously been executed.

4 Invalid list operation

The string list or list box control referenced in this command does not exist..

5 Invalid variable name

Valid variable names are %1 .. %9 and %A .. %Z.

6 "=" symbol expected

A command started with a variable name but no equals symbol was found.

7 Invalid @ function

An invalid DialogScript function has been encountered. Usually this means it has been misspelt.

8 Syntax error in @ function

Something is wrong with a function call.

9 Missing END or ELSE

An IF or ELSE command has been executed but the corresponding ELSE or END could not be found.

10 Command nested too deeply

REPEAT or GOSUB commands have been nested more than 9 deep. This can be caused by incorrect use of the GOTO command.

11 Missing argument(s) to @ function

The function executed expects more parameters than the number given.

12 Label not found

The label named in a GOTO or GOSUB command could not be found.

13 Invalid argument to @function

One of the parameters to a function is not valid.

14 Invalid parameter to command

One of the parameters to a command is not valid.

15 UNTIL without REPEAT

An UNTIL command was encountered but no previous REPEAT has been executed.

16 Invalid style

A style parameter of a dialog element does not exist. This could be because it has been mis-spelt. Note that if text has been specified in the dialog element, either as fixed text or in a variable, and that text includes semicolons, then the semicolons will be treated as parameter separators and part of the text interpreted as parameters, which will probably give rise to this error message.

17 Dialog already exists

A DIALOG CREATE command cannot be executed when a dialog window already exists.

18 Dialog control does not exist

The dialog control referenced in the DIALOG or LIST command or @DLGTEXT function does not exist.

19 List index out of range

The item number in a list index operation is less than 0 or greater than the number of items in the list.

20 File or path does not exist

The file or directory referenced in a command does not exist.

21 Cannot create control

Visual DialogScript cannot create a control specified in a DIALOG CREATE command. A dialog element may contain an invalid parameter, for example, the control name may be duplicated or invalid.

22 Operation invalid when no dialog showing

Function or command can only be used when a dialog is being displayed.

23 Control name not valid

The name specified for a dialog control contains an invalid character: names should contain only alphanumerics, and begin with a letter.

24 Mismatched brackets

The parser has reached the end of a line and a closing bracket is expected.

25 Non-numeric value in arithmetic function

An invalid character (such as a letter) appears in a string which is being treated as a numberic value.

26 Arithmetic error

An error such as overflow, underflow or division by zero has occurred.

27 Untrapped error in an external command or function

28 External library not available

Either the DLL specified in an EXTERNAL command could not be located on the search path, or a condition for its use was not met (refer to the documentation for using the extension.)

29 Insufficient memory for operation

30 Breakpoint reached

The script was stopped at a breakpoint.

31 Stopped by user

The script was paused by the user.

Error trapping

Run-time error messages may not be much of a problem in scripts you write for your own use, but they can be baffling if encountered by somebody else. To solve that problem, DialogScript allows you to trap run-time errors and write your own code for processing them.

To create an error trap you include the command:

OPTION ERRORTRAP, <label>

somewhere near the start of the script. The effect of this command is that, if a run-time error occurs, execution will jump immediately to the liine containing the label <label>. Obviously, it is a good idea to make sure the label exists, otherwise you will just get another run-time error.

Once in the error trap code, it is up to you how you process the error. It would be a good idea to turn the error trap off in case an error in the error trap causes the script to loop. You could then use WARN to display a friendly message to the user telling them what has happened and what to do. To terminate the script use the STOP command, since if the error occurred during execution of a GOSUB, the EXIT command will cause execution to carry on at the line after the GOSUB.

The @ERROR function lets you determine the error code, the number of the line that caused it, and the actual line of script that caused the error. You could display this information, or write it to an error log using a string list.

More advanced users could use error traps to improve the robustness of scripts that could be affected by user input or differences in the systems on which they are run. For example, if a user enters a value which causes a run-time error and it is difficult to validate the vaue in code, you could set an error trap, check for the error code and the line number it occurred at, and if they match this particular case, display a warning message and ask the user to enter the value again.

Events

Events occur when the user interacts with a dialog which you have created using the DIALOG CREATE command. Some events occur by default, such as those generated by buttons. Others only occur if you specify a style in a dialog element definition.

Because events can have the same name, no matter which dialog generated them, the @EVENT function has an extra parameter D to obtain the dialog ID: @EVENT(D). Programming with multiple dialogs is quite difficult so it is best to examine the simple examples that show how it is done.

You can halt your script and wait for an event to occur using the WAIT EVENT command. You can find out the type of event using the @EVENT function.

BUTTON events occur when a button or a bitbtn is pressed. The type of event is <name>BUTTON, where <name> is the name of the button. So when the user presses a button labelled OK an OKBUTTON event occurs.

CLICK events are optional. They cause a <name>CLICK event to occur when the dialog element is clicked with the mouse. In the case of elements such as LIST or RADIO you can use the event to perform some action dependent on the new setting of the control. In the case of BITMAP dialog elements you can use the @CLICK function to find out which button was pressed and the position of the mouse at the time it was clicked..

DBLCLICK events are optional. They cause a <name>DBLCLICK event to occur when the dialog element (a list) is double clicked with the mouse.

CLOSE events occur when the user closes the dialog from the system menu or using the close button, or when the user shuts down Windows. The script should respond to this event by saving any unsaved data and terminating.

To receive **DRAGDROP** events DRAGDROP style must be used with the DIALOG CREATE command. A DRAGDROP event occurs when a file or files are dragged to the dialog window. You can find out the filenames by using the LIST DROPFILES command to get them into a string list.

EXIT events are optional. They cause a <name>EXIT event to occur when the dialog element loses the input focus. This event is available for EDIT and COMBO dialog elements. It can be used to trigger a procedure to validate the data that has been entered in the control's input field.

An **ICON** event occurs when the user clicks on a task bar icon created when a TASKICON dialog element is used.

MENU events occur if you have defined a menu for the dialog window. The type of event is <name>MENU where <name> is the name of the menu item that was selected. They also occur for a DIALOG POPUP command.

A **TIMER** event occurs when you use a command of the form WAIT EVENT, <interval>. It occurs <interval> seconds after the command was issued.

FAQ

Frequently Asked Questions

(For an updated FAQ check http://www.dialogscript.com)

I want to put some quotes '"' in a string, but they are always removed. Why?

Visual DialogScript uses double-quotes as a delimiter. Within double-quotes, a % character is not treated as the start of a variable name, the @ character is not treated as the start of a function name, and commas are not treated as parameter separators. Quotes act as toggles: they can appear anywhere in a string and turn the delimiter effect on and off as the string is parsed from left to right. In the process the quotes are removed. To get double-quote characters into a string you must use @CHR(34) which is converted to the ASCII character with code 34, which is a double-quote.

When I run a compiled script from another program I cannot get the program to wait for the script to finish. Why?

This is because non-integrated VDS EXE files just call the runtime engine and then terminate. It is the runtime engine that actually runs the compiled script. The solution is either to create an integrated EXE, or to run the runtime engine from your program, passing it the path to the compiled script EXE as the first parameter (and any runtime parameters as the second and subsequent parameters.)

I am trying to copy a file from one directory to another and it is not working. Why?

The FILE COPY command is not like the DOS COPY command in that it must have a full path (including filename) as the target, not just a directory, even if the name of the copy is to remain the same.

How can I copy a list of files, such as *.TXT, using FILE COPY?

The FILE COPY command does not accept wildcards. Instead you must create a list of the files to be copied using LIST FILELIST (which does let you use wildcards) and then iterate through the list copying each file one by one.

Why does not a horizontal scroll bar appear in a list box if an item in the list is too wide to fit?

The LIST dialog element does not have a style option for displaying a horizontal scroll bar. However, you can get Windows to display a horizontal scroll bar using the line:

%P = @sendmsg(@winexists(~LIST1),$0194,1000,0)

after the dialog has been created (where LIST1 is the name of the LIST dialog element).

When I try to run a compiled script I get an error dialog that says "Cannot run script." Why?

You have created or installed a non-integrated EXE in a different directory to the one the VDS runtime (DSRUN.EXE / DSRUN16.EXE) is installed in. If you do this, you must add the directory containing the runtime to the DOS PATH string, or else move the runtime to the Windows directory. It must be possible for the EXE you create to execute the run-time engine DSRUN.EXE from wherever it is run, and if it is not in the same directory then it must be on the search path.

Why does my script not halt at WAIT EVENT command?

There may be another event waiting to be processed. You must clear the event by reading the event type using the @EVENT function, even if your script does not care what the event is. If not, the event will still be active at the next WAIT EVENT command.

I used a bitmap or icon in my dialog. When another user runs the program the image does not appear. Why?

The path to the bitmap or icon file is a full path which is not valid on the user's system. It is best to put resources such as bitmaps into the same directory as the EXE, and specify only the filename. The VDS runtime will look in the EXE's directory as well as the current directory when trying to locate a resource file.

FILE

Syntax:

FILE COPY, <file path 1>, <file path 2> {, <ALLOWUNDO>, <CONFIRM>,<SHOWERRORS> }

FILE DELETE, <file path>, {, <ALLOWUNDO>, <CONFIRM>,<SHOWERRORS> }

FILE RENAME, <file path 1>, <file path 2>, {, <ALLOWUNDO>, <CONFIRM>,<SHOWERRORS> }

FILE SETDATE, <file path>, <time>, <date>

FILE SETATTR, <file path>, <attributes>

Description:

COPY, DELETE and RENAME operations permits the use of wildcards ? and *, and the use of just a destination path instead of a full file path in COPY operations.

These three operations accept the additional optional flags ALLOWUNDO, CONFIRM and SHOWERRORS. In FILE DELETE, ALLOWUNDO deletes the file(s) to the Recycle Bin, so they can be undeleted later if required. CONFIRM causes the display of the dialogs familiar when you use Explorer, to confirm that files should be deleted or that one file should be overwritten with another of the same name. If SHOWERRORS is specified, the Windows shell puts up a modal message box with a warning message whenever it is unable to complete an operation (such as delete a file because the file is held open by another application.) OK is set to false if the FILE operation fails for any reason.

The first filename parameter can be a list of filenames, such as might be obtained from a string list using the **@text** function.

In FILE COPY, if no directory is specified for the target file, the file is copied to the *current directory*. In FILE RENAME, if no directory is specified for the target file, the file is renamed in its present location. If a directory is specified the file is, in effect, moved.

IMPORTANT: The ALLOWUNDO flag **ONLY** takes effect if the filename(s) passed to the FILE DELETE command are *fully qualified* pathnames. If only the filename is passed and the file exists in the current directory it *will* be deleted, but *not* to the Recycle Bin.

FILE COPY copies the file named in <file path 1> to <file path 2>. The original date and time are preserved. The WARNOVERWRITE flag of FILE COPY in VDS 2.x is no longer supported. Instead you should use CONFIRM and let Windows display the warning dialog message.

FILE DELETE erases the file named in <file path>. This command is similar in operation to the MS-DOS DEL command.

FILE RENAME renames the file named in <file path 1> to <file path 2>. A file can be renamed from one directory to another (in other words, moved) only if both directories reside on the same logical drive. This command is similar in operation to the MS-DOS REN command.

FILE SETDATE changes the time and optionally the date of the file <file path>. The <time> and <date> should be in the short time and short date styles set in the Windows control panel.

FILE SETATTR changes the attributes of the file <file path>. The string <attributes> consists of one or more of the following characters:

+	turns on the attributes following (default)
-	turns off the attributes following
A	archive attribute
H	hidden attribute
R	read only attribute
S	system attribute

OK:

True if the operation is successful; false if not.

Example:

```
file delete,TEST.TXT
file copy A:\DS.EXE,%C\DS.EXE
file setdate,DS.HLP,2:00,1/3/96
file setattr,C:\MSDOS.SYS,-SRH
```

See also:

@FILE

Floating point math functions

Visual DialogScript supports the following functions which can be used to perform floating point calculations:

@FADD addition

@FATN arctangent

@FCOS cosine

@FDIV division

@FEXP exp

@FLN ln

@FMUL multiplication

@FSIN sine

@FSQT square root

@FSUB subtraction

The @FORMAT function can be used to convert floating point values to a fixed number of decimal places for display or comparison purposes.

Note that Visual DialogScript does not support the use of commas as decimal separators, as used in some European countries. This would be incompatible with the use of commas as parameter separators in DialogScript commands and functions.

Function Reference

@ALT	@ASC	@ASK
@BOTH	@CHR	@CLICK
@COUNT	@CR	@CTRL
@CURDIR	@DATETIME	@DDEITEM
@DIFF	@DIRDLG	@DIV
@DLGTEXT	@DLGPOS	@ENV
@EQUAL	@ERROR	@ESC
@EVENT	@EXT	

Floating point functions

@FILE	@FILEDLG	@FORMAT
@GREATER	@HEX	@INDEX
@INIREAD	@INPUT	@ITEM
@KEY	@LEN	@LOWER
@MATCH	@MCI	@MOD
@MOUSEPOS	@MSGBOX	@NAME
@NEXT	@NOT	@NULL
@NUMERIC	@OK	@PATH
@POS	@PRED	@PROD
@QUERY	@REGREAD	@RETCODE
@SENDMSG	@SHIFT	@SHORTNAME
@STRDEL	@STRINS	@SUBSTR
@SUCC	@SUM	@SYSINFO
@TAB	@TEXT	@TRIM
@UPPER	@VERINFO	@VOLINFO
@WINACTIVE	@WINATPOINT	@WINCLASS
@WINDIR	@WINDOW	@WINEXISTS
@WINPOS	@WINTEXT	@ZERO

Functions

DialogScript contains a range of functions (see Function Reference) which are evaluated at run time and return a string containing information.

Functions start with an @ symbol followed by the function name. The argument(s) to the function are in the form of a string enclosed in parentheses. The parentheses must be present even if the function takes no arguments. For functions that take more than one argument the arguments are separated by commas.

Here are some examples of functions:

```
@ASK(Do you want to continue?)

@EQUAL(%F,WIN.INI)
```

Note that because the @ symbol is used to identify functions you cannot use it for any other purpose unless it is enclosed within double quotes..

GOSUB

Syntax:

GOSUB <string>

Description:

Causes script execution to continue at the command following the label :<string>. When an EXIT command is encountered, execution will jump back to the command following the GOSUB.

OK:

Leaves unchanged.

Example:

```
GOSUB sayhello
INFO Goodbye
EXIT
:sayhello
  INFO Hello
  EXIT
```

See also:

EXIT STOP

GOTO

Syntax:

GOTO <string>

Description:

Causes script execution to continue at the command following the label :<string>.

Note that using the GOTO command to branch to a label that is within an IF ... END or REPEAT ... UNTIL group of commands will result in a "Missing END or ELSE" or "UNTIL without REPEAT" error message, unless the GOTO command and the label are both within the same group of commands.

Using a GOTO command to branch out of a REPEAT ... UNTIL group of commands will eventually result in a "Command nested too deeply" error. The reason is that the DialogScript interpreter does not know that the REPEAT command has finished until it has executed the UNTIL command with the terminating condition as true.

OK:

Leaves unchanged.

Example:

```
GOTO label

WARN This message box will not be displayed

:label

INFO This message box will be displayed.
```

See also:

GOSUB

IF

Syntax:

IF <string>

... commands executed if string not null (true)

ELSE

... commands executed if string is null (false)

END

Description:

The IF command is used to allow conditional execution of commands in a script. The <string> is evaluated and if the resulting string is non-null this is treated as true. If the result is null this is treated as false.

If <string> evaluates to true, the commands between the IF command and the ELSE (or the END, if ELSE is omitted) are executed. If <string> is null (false) and an ELSE is present the commands between the ELSE and END are executed. Otherwise execution skips to the line following the END.

IF commands may be nested. If this is done then it is important to ensure that the correct ELSE and END commands are not omitted. For clarity it is a good idea to indent the nested IF commands as shown in the example.

Note that unlike many other languages there is no need for a THEN at the end of the IF command line. Because DialogScript treats a non-null result for the condition string as true, if you *do* mistakenly put a THEN at the end of an IF command line this will be treated as part of the string which will therefore always be non-null and the condition will always be true. A similar problem will occur if you omit the @ from a function name, causing it to be treated simply as text instead of being evaluated as a function.

OK:

Leaves unchanged.

Example:

```
IF @ask(Do you want to continue?)
  info You answered YES
ELSE
  IF @ask(Are you sure?)
    info You answered NO
  ELSE
    info Make your mind up
  END
END
```

See also:

@NOT @NULL REPEAT Tip

INFO

Syntax:

INFO <string>

Description:

Displays a dialog box containing an information symbol icon and the message <string>. Execution of the script continues when the OK button is pressed.

OK:

Set to true.

Example:

INFO There is %SKb of free space on drive D:

See also:

WARN @ASK @MSGBOX @QUERY

INIFILE

Syntax:

INIFILE OPEN, <inifile name>

INIFILE WRITE, <section name>, <key name>, <string>

Description:

The INIFILE OPEN command sets the name of the INI file which will be used by any succeeding INI file read and write commands to <inifile name>. If no INI file is specifically opened, then scripts will use a file DEFAULT.INI.

The INIFILE WRITE command writes a line <key name>=<string> under the section header <section name> in the currently open INI file. A section name of [Default] is used if the <section name> parameter is null.

Note that there is no INIFILE CLOSE command as Windows only keeps an INI file open for the duration of each read or write.

OK:

Unchanged..

Example:

INIFILE OPEN,MYSCR.INI

INIFILE WRITE,Data,Name,Fred Bloggs

See also:

@INIREAD

LINK

Syntax:

LINK CREATE, <filename>, <link path>, <link name> {, <icon path> {, <start directory> {, <arguments> }}}

Description:

The LINK CREATE command is used to create a shortcut to a program or file.

The <filename> is the name of the program or file that the shortcut will be a link to. The <link path> is the directory or folder in which the shortcut is to be created. The Windows desktop can usually be found at the path @WINDIR()\Desktop. The Start menu is usually found at the path @WINDIR()\Start Menu. The <link name> is the description that will appear beneath the shortcut.

The <icon path> is optional. If present it gives the location of the icon that will be used for the shortcut.

The <start directory> is optional. If present it gives the starting directory that will be used.

The <arguments> parameter is also optional. If present, it specifies arguments on the command line of program <filename>. If commas are part of the arguments they should be enclosed in quotes.

If successful, a shortcut will be created with the path <link path>\<link name>.LNK.

OK:

Set to false if the LINK command fails.

Example:

```
rem create a shortcut on the desktop
link create,c:\prog\readme.txt,@windir()\desktop,Shortcut to readme.txt
if @ok()
  info Link created successfully
else
  warn Link create failed
end
```

See also:

@DDEITEM DDE

LIST

Syntax:

LIST <command>, <list>, <parameters>

Description:

The LIST command is used to create, manipulate and dispose of string lists. DialogScript allows you to have several string lists (currently 9) each identified by a number. List boxes and combo boxes (list controls) which appear in a dialog window can also be treated as string lists. The parameter <list> must be either a list number or the name of the dialog list control to which the command will apply.
Lists can contain any number of strings holding up to 255 characters each. A list can either be sorted, or will retain the order in which the information was entered.
For more information see Using Lists and Data Lists.

This command has changed in one major respect which is that the order of parameters of this command has been changed. Parameter 1 and parameter 2 are swapped round. This means that the command now takes the format:

LIST <operation>, <list_id> {other params ...}

which is consistent with all the other commands. The previous form, with the <list_id> as the first parameter, was inconsistent.

In addition to this, the following changes have been made:

LIST ASSIGN now assigns text to the list if the third argument is not a valid list name. In other words:

LIST ASSIGN,1,2

would copy contents of list 2 to list 1, but:

LIST ASSIGN,1,Here is some text@cr()Here is some more text

would assign two lines of text to the list.

LIST FILELIST supports a pseudo-attribute * which works in conjunction with a root path (not a file spec) to produce a recursed list of directories starting at that path. For example:

LIST FILELIST,1,C:\Windows,*

would create a list of all the subdirectories of Windows.

LIST LOADFILE can load text from a file, or from a text file that has been compiled into a special VDS resource file. In the latter case, the filename must be followed by a vertical bar and the offset, in bytes, of the text file within the resource. No spaces are permitted between the filename, the vertical bar and the offset.

LIST SORT sorts the contents of a list. (It works with string lists 1 .. 9 only, not LIST or COMBO elements)

LIST PRINT, <list_id> {, <font_name>, <font_size>} prints the contents of a list.

OK:

Set to true if the command is successful, false if it fails.

Example:

```
LIST CREATE,1, SORTED
LIST ADD,1, Fred Jones
LIST ADD,1, John Smith
```

See also:

@COUNT @INDEX @ITEM @MATCH @NEXT

DIALOG ADD,LIST,<name>,<top>,<left>,<width>,<height>,<style>{,style>}

This dialog element creates a list box at the position and size specified. LIST dialog elements have built-in tab-stops so that information containing tabs is displayed in columns.

The CLICK style causes a <name>CLICK event to be generated when an item is chosen from the list.

The DBLCLICK style returns a <name>DBLCLICK event if user double-clicks an item in the list box.

The SORTED style specifies whether the list items are to be maintained in ASCII order or not.

To get data into the list box you must use the LIST command.

Labels

Labels are used as the target of GOTO and GOSUB commands. They start in the first character position of a line with a colon, and are followed by the label name.

This is an example of a label:

```
:LABEL
```

Labels, GOTO and GOSUB commands are used to change the order of execution of script commands.

OPTION

Syntax:

OPTION CENTURYWINDOW,nn

OPTION DECIMALSEP, <separator>

OPTION ERRORTRAP, <label>

OPTION FIELDSEP, <separator character>

OPTION FILENAMES, <SHORT|LONG>

OPTION PRIORITY, <IDLE|NORMAL|HIGH|REALTIME>

OPTION REGBUF, <buffer-size>

OPTION REGKEY,<new_default_key>

OPTION SCALE, <pixels-per-inch>

OPTION SKDELAY, <interval>

OPTION SLEEPTIME, <interval>

Description:

The OPTION command is used to set various options to be used by your script. If the value is missing the default value is set.

OPTION CENTURYWINDOW,nn

This option lets you specify a different start year for the 100 year window for two digit year numbers which will run from 19nn to 20nn-1. Numbers >= nn have 1900 added, < nn have 2000 added. So if you make nn 100, all years will be 21st century.

OPTION DECIMALSEP is used to force the decimal separator to be a period, i.e. OPTION DECIMALSEP,"." This should be used if there are any floating point numeric constants in the script. If it is not, then numeric values will be interpreted according to the Windows regional settings (for example, a separator of a comma would be expected in many European countries) so a script that worked in one country would fail in another. Note that numeric values with commas as a decimal separator should have quotes round them if they are used in a script. Commas are treated by the interpreter as parameter separators, so if a value is written with a comma as a decimal separator the interpreter will treat the number as two separate integers.

OPTION ERRORTRAP lets you define a label which execution will branch to if a run-time error occurs. If no label is present, error trapping is turned off.

OPTION FIELDSEP sets the field separator to be recognised by the PARSE command when it splits a string up into its separate fields. The default is the vertical bar (|).

OPTION FILENAMES lets you specify whether the LIST ... DROPFILE command returns short or long filenames. It is usually better to use short filenames if you will be passing filenames to a DOS program. Note that the LIST ... FILELIST command and the @FILE() function only return long filenames which are not full paths; to convert these names to short names the @SHORTNAME function must be used.

OPTION PRIORITY lets you set the priority level of the program. If IDLE, the program runs only if the system is idle, which is a good choice for scripts that are intended to run in the background. NORMAL is the priority level that all programs run at by default. HIGH and

REALTIME are higher priority levels and should be used with care, if at all, as they may cause problems by giving your program a higher priority than Windows system functions.

OPTION REGBUF lets you specify the size of the buffer used when reading and writing Registry key entries and INI keys using the REGISTRY command and @REGREAD function. The default size is 256 bytes, which is large enough for most purposes. Registry keys can be much larger, though, in which case you will need to use this option to make the buffer larger.

OPTION REGKEY,<new_default_key>

This option defines the registry key root to be used when DEFAULT is specified in the REGISTRY command and @REGREAD function. If not changed by this option the default registry key root is HKCU\Software\SADE\VDS\3.0\User Scripts\<scriptname>\.

The option prepends 'Software\' to <new_default_key>, and appends a '\', so the parameter value should

be something like 'MyCompany\MyApp'. The purpose of this option is to allow script writers to use their own registry key root without having to specify the full path in each registry command or function call. However, if scripts are for your own use the default key will suffice.

OPTION SCALE is used to make a dialog scale itself when different font sizes are used. The value in <pixels-per-inch> should be the same value given by @SYSINFO(PIXPERIN) on the system on which the dialog was designed (see Screen Metrics.) This value is determined by the font size chosen in Control Panel Display Settings. The 1standard setting is Small Fonts, which gives a value of 96 pixels per inch. If the value on the user's system is different from the value specified in this option then the size of the dialog and the position and size of dialog elements will be scaled to display the correct proportions with the font size chosen.

OPTION SKDELAY can be used to introduce a delay, specified in milliseconds, between sending characters using the WINDOW SEND command, if this is required for more reliable sending. The default is 10ms.

OPTION SLEEPTIME can be used to specify the interval, in milliseconds, that a DialogScript program suspends itself while executing a WAIT command. The default is 50ms. Increasing this value to, say, 500, would reduce the system overhead of a script that simply waits in the background, but would give poor performance when users interact with a dialog.

OK:

Unchanged.

Example:

```
OPTION FILENAMES,SHORT

OPTION SLEEPTIME,200
```

Overview

Visual DialogScript is a programming tool that enables you to quickly develop simple batch procedures or dialog-based programs for Windows as easily as writing a batch file or Basic program for DOS. It includes an interactive editor and debugger for creating programs, called scripts, which are written in the DialogScript programming language.

The package also includes tools such as a Dialog Designer for visually designing dialog boxes, an Icon Designer (Not packaged in the shareware version) to let you create icons for your scripts, a resource compiler (Not packaged in the shareware version) and a Window spy for finding out information about other applications' windows. There is also an Application Wizard which will generate all the DialogScript code for a complete dialog-based program.

DialogScript has a simple syntax, with English-like commands, spreadsheet-like formula functions and typeless variables. Script programs can be tested instantly using the development environment. You can then create an EXE which can then be run just like any other Windows application. Registered users can distribute the EXE files created by Visual DialogScript without any further payment being required.

DialogScript is not intended to be an alternative to programming languages like C, Basic or Pascal for application development. But it is a better choice when you need a simple utility or a quick solution to a problem. Most DialogScript scripts can be written and tested in a matter of minutes once you are familiar with the script language.

PARSE

Syntax:

PARSE <field list>, <string>

Description:

The PARSE command provides an easy way to split up strings which represent records in a database into their constituent data fields.

The <field list> parameter consists of a semicolon-separated list of DialogScript variables or, if a dialog is showing, dialog control names, which are to receive the data. The <string> contains the data record.

Note that because DialogScript uses commas to separate parameters on a command line, semicolons are used to separate the variable and field identifiers in the field list. Also, if a comma appears in <string> any data after the comma will be lost. This should not be a problem as the data will normally be passed to the command line using a variable, not a literal string.

Note also that to prevent variables where they appear in the field list from being substituted by their initial contents the field list itself should be enclosed in quotes. This is not necessary if the list contains only dialog control names.

If there are fewer items in the string than in the field list, the remaining fields are set to blank instead of keeping whatever they held before.

OPTION FIELDSEP sets the field separator used by the PARSE command when it splits a string up into its separate fields. By default this is the vertical bar character: |.

OK:

Unchanged.

Example:

```
%Y = @next(1)

parse "%N;%A;%B;%C", %Y
```

See also:

OPTION

PLAY

Syntax:

PLAY <file name> {, WAIT}

Description:

Plays an audio wave (*.WAV) file. If the parameter WAIT is specified the script does not proceed to the next command until the sound is finished.

OK:

Leaves unchanged.

Example:

PLAY intro.wav,WAIT

See also:

@MCI

DIALOG ADD PROGRESS,<name>,<top>,<left>,<width>,<height>,<value>

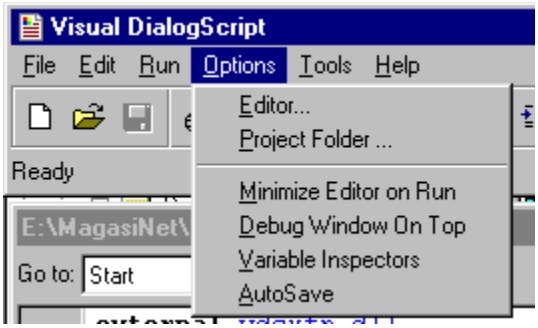
This dialog element creates a progress bar at the indicated position. The number <value> indicates its initial setting, as a percentage value (0 to 100)..

Saves a script if it has been changed every two minutes

Preferences

From the Options menu you can set your operating preferences for the development environment.

Click on a menu item to learn what it does.



DIALOG ADD,RADIO,<name>,<top>,<left>,<width>,<height>,<caption>,<value list>,<value>{,styles}

This dialog element creates a group of radio buttons at the position and size specified, with a caption of <caption> and a set of possible values shown in <value list>. The values in the value list are separated by a vertical bar delimiter, for example: Male|Female. The initial setting is <value>.

If the CLICK style is specified a <name>CLICK event is generated whenever the radio button group is clicked.

Note : The RADIO dialog element is a group dialog element. See the decription of the GROUP element for more information. However, it is not actually recommended to place other dialog elements within the group box of the RADIO dialog element. In the Dialog Designer it is difficult to do this, but it can be done by manually setting the top left corner of elements so that they fall within the boundary of the the RADIO dialog element.

REGISTRY

Syntax:

REGISTRY DELETE, <root key>, <subkey>

REGISTRY WRITE, <root key>, <subkey>, <name>, <value>{,<Type>}

Description:

The REGISTRY command modifies the value of keys in the Windows registry.

REGISTRY DELETE can delete either only a whole key and all its values or specifying a third parameter, a value name. If present, only that value is deleted, not the whole key.

Note that VDS does nothing to hide the differences between Windows 95, 98 and NT. So, although you can delete a key containing subkeys in Windows 95, this will fail in Windows NT because NT does not allow it. If you are delevoping scripts for NT you will have to enumerate the keys and delete them individually.

REGISTRY WRITE supports an optional fifth parameter, which can be BINARY or DWORD. This parameter is used to specify the type of value to be written. If omitted, the type will be a string if the value did not previously exist in the registry, otherwise it will take the same type as the existing value if possible. (Note: VDS registry commands and functions support only string, binary and dword value types: if you try to write to other types the effect is undefined.)

The second parameter to this command, DEFAULT, now uses a key root of 'Software\SADE\VDS\3.0\UserScripts\<scriptname>' unless this has been modified using OPTION REGKEY. (This affects @REGREAD as well.)

The value of the third parameter may be null. However, it only makes sense for it to be null when the second parameter is DEFAULT. When the default key is used, VDS generates a unique registry branch for that project (as described in the paragraph above, or you can specify your own using OPTION REGKEY.) In many cases there is no need for further levels of subkeys as you can use named values for all the data you need to store.

When writing to a binary key: the value is now a single parameter with byte values separated by fieldsep. Example:

REGISTRY WRITE,CURUSER,Software\SADE\Something,BinaryValue,0|1|8|0

The @regread function returns binary values in a complementary format. (With the previous syntax of passing the binary values as individual parameters separated by commas, people thought you could pass the comma separated values in a single variable. You could not, because the interpreter treated the whole contents of the variable as a single parameter.)

<root key> specifies the root key to search from. The permissible values are:

ROOT	specifies HKEY_CLASSES_ROOT
CURUSER	specifies HKEY_CURRENT_USER
LOCAL	specifies HKEY_LOCAL_MACHINE
USERS	specifies HKEY_USERS
DEFAULT	specifies the key Software\SADE\VDS\3.0\UserScripts in HKEY_CURRENT_USER.

<subkey> specifies the key value to use. Keys several levels deep can be specified using backslashes.

<name> specifies the named value to change If null, the default key value is changed

<value> is the string value to which the key is set.

Note: Registry keys may be one of a number of different types. DialogScript recognises the following types: String, DWord (32-bit integer) and Binary (byte array). Because DialogScript variables are untyped, DialogScript can only create string type keys. However, if a Registry key already exists and has a non-string type, DialogScript will attempt to convert the <value> to the data type used by the key. Note that binary keys are created using a buffer which has a default size of 256 bytes. This can be increased if necessary using the OPTION REGBUF command.

WARNING: Modifying the system registry is potentially dangerous, and can render Windows unable to run. Ensure you have a backup of the registry files before experimenting.

OK:

Set to false if the REGISTRY command fails.

Example:

```
REGISTRY WRITE,ROOT,.wsc,,wnscript
REGISTRY WRITE,DEFAULT,Test,Value,31
REGISTRY WRITE,DEFAULT,Test,Binkey,1,3,5,7,9
REGISTRY DELETE,ROOT,.tmp
```

See also:

@REGREAD Tip

REM

Syntax:

REM <text>

Description:

Use the REM command to enter a comment in your script.

OK:

Leaves unchanged.

Example:

REM This script does something interesting

See also:

REPEAT

Syntax:

REPEAT

... commands ...

UNTIL <string>

Description:

Commands between the REPEAT and UNTIL lines are repeatedly executed while the result of <string> evaluates to null (false). If the string is non-null then script execution continues on the line following UNTIL.

REPEAT commands may be nested. For clarity it is a good idea to indent the commands nested within REPEAT ... UNTIL as shown in the example.

It is not recommended to use the GOTO command to branch out of a REPEAT ... UNTIL group of commands.

OK:

Leaves unchanged.

Example:

```
REPEAT
  %A = @SUM(%A,1)
UNTIL @EQUAL(%A,6)
```

See also:

@NOT @NULL IF Tip

RUN, RUNH, RUNM, RUNZ

Syntax:

RUN <filename> <parameters>{,WAIT, <priority level>}

Description:

Runs the file or program <filename>, with the additional parameters specified. If a filename is given then the file is opened using the shell open command stored in the Windows registry.

If the WAIT parameter is included then script execution does not continue until the program has terminated.

You can optionally specify the <priority level> for the task as one of: IDLE, NORMAL, HIGH or REALTIME. If IDLE, the program runs only if the system is idle. NORMAL is the priority level that all programs run at by default. HIGH and REALTIME are higher priority levels and should be used with care, if at all, as you may experience problems by giving programs a higher priority than Windows system functions.

The variants of the command are:

RUN - run as a normal window;

RUNH - run in a hidden window;

RUNM - run as a maximized window;

RUNZ - run minimized as an icon.

If you run a DOS program and want it to close its window when it finishes make sure the program's "close window on exit" property is set.

Note: (32-bit only) The RUN command does not work with long filenames that include a space. This is because it looks for a space to determine the end of the filename and the start of any parameters. If this is a problem, either use @SHORTNAME to convert the file name to a DOS-compatible short name, or use the SHELL command instead.

OK:

Set to false if the command fails.

Example:

```
RUN winword.exe

RUNZ c:\utils\pkzip.exe c:\temp\test.zip *.txt,WAIT
```

See also:

@RETCODE SHELL

RUNH

see: RUN

RUNM

see: RUN

RUNZ

see: RUN

SHELL

Syntax:

SHELL <operation>, <filename>, <parameters>, <start-dir> {AIT, <priority level>}

Description:

Performs the Windows shell operation <operation> on the file <filename> with optional parameters <parameters> in the optional startup directory <start-dir>. If WAIT is specified the script waits until the operation is complete.

If the <operation> is null, the default operation is used. The effect is the same as when you double-click on the file in Windows Explorer.

You can optionally specify the <priority level> for the task as one of: IDLE, NORMAL, HIGH or REALTIME. If IDLE, the program runs only if the system is idle. NORMAL is the priority level that all programs run at by default. HIGH and REALTIME are higher priority levels and should be used with care,re, if at all, as you may experience problems by giving programs a higher priority than Windows system functions.

OK:

Set to false if the command fails.

Example:

SHELL "",My Data.LNK

SHELL open,http://www.swregnet.com/

SHELL print,Report.DOC

SHELL printto,Budgets.XLS,"Microsoft Fax WPSUNI.DRV FAX:",,WAIT

See also:

@RETCODE RUN

SHIFT

Syntax:

SHIFT

Description:

Shifts the command line parameter variables so that %8 takes the value of %9, %7 takes the value of %8 and so on down to %1 takes the value of %2. This is similar to the MS-DOS SHIFT batch file command.

OK:

Leaves unchanged.

Example:

SHIFT

See also:

DIALOG ADD STATUS,<name>,<value>,<style>

This dialog element creates a status panel at the bottom of the dialog window, containing the text <text>. A dialog can only have one status panel.

STOP

Syntax:

STOP

Description:

Halts execution of a script unconditionally. It is similar to EXIT but will terminate a script even from within a subroutine..

OK:

Unchanged.

Example:

STOP

See also:

EXIT GOSUB

DIALOG ADD STYLE,<name>, <fontname>, <fontsize>, <text attributes>, <background color>, <foreground color>

The STYLE dialog element defines a typeface, text attributes and colors which are associated with a style name. The style name can be used in the DLGTYPE dialog element to have it apply globally to the whole dialog, or it can be applied to individual elements. The default font is MS Sans Serif, size 8. The text attributes are B (bold), I (italic), L (left justified), C (centered) or R (right justified). The colors can be BACKGRND, FOREGRND, BLACK, DKRED, DKGREEN, BROWN, DKBLUE, MAGENTA, GRAY, SILVER, RED, LTGREEN, YELLOW, LTBLUE, CYAN, WHITE. Not all dialog controls are affected by all style attributes.

Screen metrics

Metrics is the term used by Microsoft to describe the characteristics of an output device, such as the display. One factor that may affect your scripts is the number of pixels per inch used by the display driver.. Windows displays typically use either 96ppi or 120ppi, the latter when the Large Fonts display option is selected.

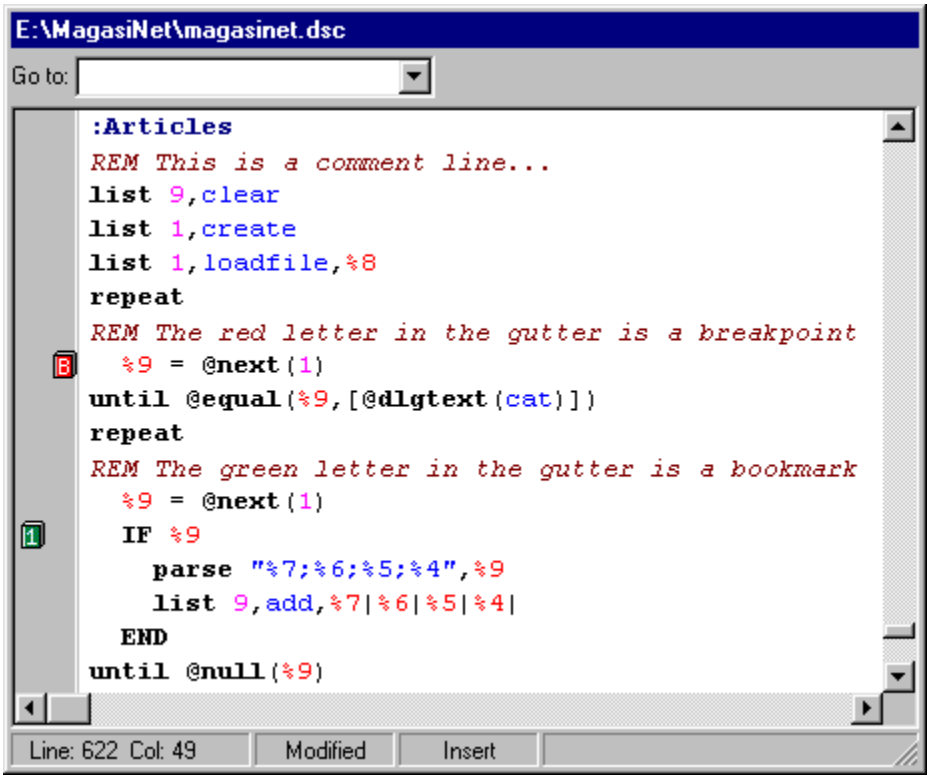
When large fonts are used, text takes up more space relative to the dialog controls it is displayed in. This can lead to text in a dialog box being truncated if it is run on a system which uses larger fonts than the system on which the dialog box was designed.

There are two solutions you can adopt if your scripts may be run on systems using either small or large fonts. One is to allow plenty of space so that text displays correctly whichever font size is chosen. The other is to use OPTION SCALE, which causes VDS to attempt to scale the dialog box to the right size for the system on which the script is being run.

Script Window

The IDE automatically starts up by opening the project that was last used.

The script window is used for editing and debugging scripts. The Editor Options from the Options menu opens a 5 page properties dialog that allows the syntax highlighting, editor behaviour and fonts to be customized. An "auto-correct" and a "code template" facility are also provided.



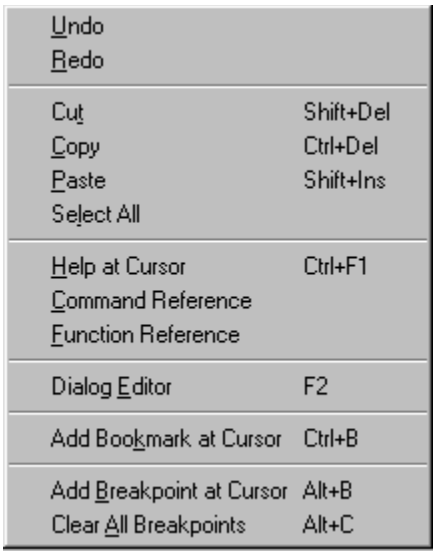
Breakpoints can be set/unset by clicking in the gutter beside the line where the breakpoint is, as well as from the context menu. Breakpoints are shown by a red graphic with the letter B.

Bookmarks are shown by a green numbered graphic.

Project options such as breakpoints, bookmarks, icon file, executable target and so on are stored in a project file (.DSP) with the same path and name as the script source.

During single-step debugging, the command which is about to be executed is shown highlighted in the script window. If an error is found during execution of a script, the line containing the error is highlighted when the DialogScript interpreter displays the error message.

The context menu provides a quick way to access many useful functions.



The menu is context sensitive. You can place the cursor in the middle of a command or function and select Help at Cursor to call up the help page for that command or function from the menu. Alternatively you can select some text with the mouse and then press Control + F1. The command reference and function reference sections of the online help can be called up directly from the menu.

You can set breakpoints to halt the execution of a script by placing the cursor in the line at which you want execution to stop, right-clicking the mouse and selecting Add Breakpoint at Cursor from the menu. If you place the cursor on a breakpoint the option changes to Clear Breakpoint at Cursor. You can also clear all breakpoints in one go.

You can also call up the dialog designer from the context menu, or by pressing F2. To edit an existing dialog description, place the cursor in the first line of the DIALOG CREATE command before opening the editor.

String Lists

String lists can be used to hold the contents of text files as well as lists of ASCII data. As the name implies, they are lists of strings. The length and number of strings in a string list are limited only by available memory.

DialogScript supports nine independent string lists, identified as 1 to 9. In addition, the list box and combo box dialog controls are also string lists, and can be manipulated using the same list commands.

String lists can be unsorted, in which case strings remain in the order they were loaded and items may be added at the end using LIST ADD, inserted using LIST INSERT or replaced using LIST PUT. Using the LIST SEEK and LIST PUT commands, and the @MATCH and @ITEM functions, lists can be used as random access files.

Alternatively lists may be sorted, in which case DialogScript maintains the order of items and new items must only be added using LIST ADD. To replace an item in a sorted list you must delete it and then add it.

For more information on how lists are used, see Using Lists.

For information on how to load data into lists see Data Lists.

Strings

In DialogScript, all variables and parameters to commands and functions are strings. Strings can contain embedded variables and functions, which are substituted or evaluated as the string is processed from left to right.

So if %D contains C: then the string:

    Drive %D has @VOLINFO(%D,F)Kb free

would be evaluated to:

    Drive C: has 32456Kb free

Note that the symbols % and @ (which are used to denote variables and functions), commas (which are used to split strings into two or more parameters for commands or functions that expect multiple parameters), and brackets (which enclose the parameters of an individual function), cannot normally appear in a string because of the special meaning attached to them. To overcome this, strings can be enclosed in double quotes (").

Text within double quotes is interpreted simply as text: no variable substitution or function evaluation will be performed. Quotes can appear anywhere in a string to prevent substitution or evaluation for a particular section, as in this example:

    %A = Order @input(Enter quantity,1) "widgets @ "%R" each (incl. "%T"% sales tax)"

DIALOG ADD,TASKICON,<name>,<filename>,<tooltip text>

This dialog element creates a task bar icon which will generate <name>ICON events when clicked with the mouse. If no <filename> is specified the script's own icon is used, otherwise the named icon (which can be an EXE or ICO file) is used (the <filename> can refer to a resource file and specify a byte offset) . The <caption> appears as a tooltip when the cursor is placed over the icon. The caption text can be changed using the DIALOG SET command. The icon can be enabled and disabled using DIALOG ENABLE and DIALOG DISABLE.

DIALOG ADD,TEXT,<name>,<top>,<left>,<width>,<height>,<value>{,styles}

This dialog element creates a text control at the position and size specified, containing the text <text>.

The CLICK style generates a <name>CLICK event if user clicks on the text.

TITLE

Syntax:

TITLE <string>

Description:

The title set here is the application title. It appears only in message and input boxes. The title of the task bar button is the title of the main window. Console scripts (in other words, those that do not have a visible window) do not appear on the task bar at all.

OK:

Leaves unchanged.

Example:

```
TITLE Daily Backup
```

See also:

DIALOG

DialogScript does not allow compound conditions. However, you can have a group of statements executed if one or more of several conditions are true by concatenating the conditions, since DialogScript considers a non-null result to be true. For example:

```
if @ext(%F,COM)@ext(%F,EXE)@ext(%F,BAT),run %F
```

To convert from using INI files to registry keys you can usually just replace the INIFILE command with REGISTRY, and insert the parameter DEFAULT at the front of the parameter list.

Using Lists

Syntax:

LIST ADD,<list>,<text>

LIST ASSIGN,<list>,<list2>

LIST CLEAR,<list>

LIST CLOSE,<list>

LIST COPY,<list>

LIST CREATE,<list>, {SORTED}

LIST DELETE,<list>

LIST INSERT,<list>

LIST PASTE,<list>

LIST PUT,<list>,<text>

LIST SEEK,<list>,<record number>

LIST WRITE,<list>,<text>

Description:

These LIST commands are used to modify, save and dispose of string lists. The parameters <list> and <list2> must be either a list number or the name of the dialog list box to which the command will apply. An error will occur if the list does not already exist or the record number (for the SEEK command) is out of range.

ADD is used to add an item to the list. You use this to add items to a sorted list (they are inserted in the correct position according to the sort order) or to append items to an unsorted list, much as you would write successive lines to a text file.

ASSIGN copies the contents of <list2> to <list>

CLEAR is used to remove all items from a list and reset the item pointer to zero.

CLOSE must be used to dispose of a list that you have CREATEd once you have finished with it. This releases the memory it used, and makes the list number available again for a new list.

COPY causes the contents of <list> to be copied to the Clipboard.

CREATE creates a new, empty string list. The option SORTED specifies whether the list is to be maintained in ASCII code order.

DELETE deletes the item at the position in the list indicated by the index (pointer). This is the first item (item 0) when the list is first opened, but can be modified by means of the SEEK command.

INSERT inserts a new item in front of the current pointer position. After the insertion, the index position is the item following the one just inserted. Note that you should not use INSERT with sorted lists.

PASTE causes <list> to be filled with the contents of the Clipboard (as text.)

PUT is used to write the specified text to the position in the list indicated by the index (pointer). PUT lets you treat a list as a random access file. Note that you cannot use PUT with sorted lists.

SEEK is used to set the index pointer to a specific item number.

OK:

Set to true if the command is successful, false if it fails.

Example:

LIST SEEK,1, 3

LIST WRITE,1,This is item 4 in the list

LIST SAVEFILE,1,LIST.TXT

LIST CLOSE,1

See also:

Using MCI

DialogScript allows you to control multimedia devices using the Windows Multimedia Control Interface (MCI). DialogScript provides the @MCI function, which can be used to send MCI command strings and obtain the response, which may consist of information or an error message.

MCI is a command language in its own right. For a complete description of it, refer to Microsoft's own multimedia documentation or third party books on Windows multimedia development. If you can obtain the help file MCISTRWH.HLP, supplied with Microsoft Visual C++, this will also be a useful reference.

MCI command strings are English-like and readily understandable. You open a device, play it, and then close it when you are finished with it. For example, the command strings:

```
open cdaudio
play cdaudio from 1 to 2
close cdaudio
```

would play the first track of an audio CD. Audio data files must be given a name, called an alias, which is used in the MCI commands. For example:

```
open C:\WINDOWS\MEDIA\THEMIC~1.WAV alias sound
play sound
close sound
```

If the above MCI command strings were sent using a DialogScript script then the result would be nothing, because the sound would be stopped by the close command before it had a chance to play. By appending the word wait to an MCI command string, control is not returned to the script until the command has completed. If the command

```
play cdaudio from 1 to 2 wait
```

was sent then execution of the script would be held up until the track had finished playing.

Variables

DialogScript allows you to have up to 99 variables.

35 standard variables. Variable names start with a percent symbol, followed by the character 1 to 9 or A to Z.

As well as the standard variables %A to %Z and %0 to %9 you can have 64 user-defined variable names. These variables begin with %%, a letter, then alphanumerics plus underscores (e.g. %%my_variable_1.) There is no limit on the length of the variable name.

The variables %1 to %9 contain the command line parameters to DialogScript, which can be used to pass information to the script program at run time. Unlike a DOS batch file you can also assign your own data to the variables %1 to %9, overwriting the original contents. The read only variable %0 can also be used in a compiled EXE script: it contains the full pathname of the program.

DialogScript variables are untyped strings of unlimited length. Because variables are untyped it is up to you to ensure that, for example, where a function requires a numeric value the variable passed to it contains a string which is a valid number.

Note that the use of the percent symbol to identify variables, and the @ symbol to identify functions, means that you cannot generally use these symbols in a string of text. Double quotes can be used to enclose text containing these symbols and prevent them from being interpreted as variables or function names, like this:

```
INFO Email address:@tab()%N"@dialogscript.com"
```

DialogScript also supports string lists. You can think of these as arrays of strings, which can be maintained in sorted or unsorted order and accessed either sequentially or by item number.

WAIT

Syntax:

WAIT <interval>

WAIT EVENT {, <interval> }

Description:

Pauses execution of the script for a period of <interval> seconds. If <interval> is not specified then the default period is 1 second.

The value for the timer <interval> may be a floating-point value. The minimum value that can be specified is 0. The minimum time that the script will actually wait for is governed by the default or specified value of OPTION SLEEPTIME.

The WAIT EVENT command is used in dialog programming. It is only valid when a dialog is being displayed. It causes the script to wait indefinitely until an event occurs such as a button being pressed.

If an interval is specified in a WAIT EVENT command, then a TIMER event will be generated when the interval has elapsed. This is useful for dialogs that must be updated at intervals, because it allows you to respond immediately to button and other events as well.

OK:

Not affected.

Example:

```
%T = @INPUT(Enter alarm time [hh:mm]:)
repeat
  wait 30
  %N = @datetime(t)
until @EQUAL(%T,%N)
beep
warn Wake up!
```

See also:

WARN

Syntax:

WARN <string>

Description:

Displays a dialog box containing an exclamation mark icon and the message <string>. Execution of the script continues when the OK button is pressed.

OK:

Set to true.

Example:

WARN There is less than 100Kb of free space on drive D:

See also:

INFO @ASK @MSGBOX @QUERY

WINDOW

Syntax:

WINDOW ACTIVATE, <window>

WINDOW CLICK, <window>, <x pos>, <y pos>

WINDOW CLOSE, <window>

WINDOW HIDE, <window>

WINDOW ICONIZE, <window>

WINDOW MAXIMIZE, <window>

WINDOW NORMAL, <window>

WINDOW ONTOP, <window>

WINDOW POSITION, <window>, <top>, <left>, <width>, <height>

WINDOW SEND, <window>, <string>

WINDOW SETTEXT, <window>, <string>

Description:

The WINDOW command is used to control other windows. The value of <window> is the window identifier, which specifies which window is the target of the command. The <window> is normally a main window or MDI child window.

WINDOW ACTIVATE activates (restores from an icon, or brings to the top) the specified window.

WINDOW CLICK simulates a mouse click at the point <x pos>, <y pos> relative to the top left corner of the specified window. To simulate a double click use the same command twice in succession. The command WINDOW RCLICK can also be used to simulate a right-button click.

WINDOW CLOSE closes the specified window.

WINDOW HIDE hides the specified window or task bar button. To un-hide the window, use WINDOW NORMAL.

WINDOW ICONIZE minimizes the specified window.

WINDOW MAXIMIZE maximizes the specified window.

WINDOW NORMAL restores to normal size the specified window.

WINDOW ONTOP sets the Topmost attribute of the specified window so that it remains in view even when not active.

WINDOW POSITION positions the specified window so that it's top left corners are at the co-ordinates specified. The width and height can also be specified. If they are omitted, the window retains its existing size.

WINDOW SEND sends the contents of <string> to the specified window as simulated keystrokes. Text can be entered as ordinary text. Functions like @TAB(), @CR() and @ESC() can be used for the Tab, Enter and Escape keys. You can also use @ALT to simulate the Alt key, @CTRL for the Ctrl key and @SHIFT for the Shift key. In addition, the @KEY function can be used to generate the keystrokes for the Home, End, Up arrow, Down arrow, Left arrow, Right arrow, PgUp, PgDn, Ins and Del. keys plus F1 to F12.

WINDOW SEND has been modified. It no longer has the limitation that you cannot send characters with an ASCII code of 228 or greater. However, now the character @chr(127) cannot be sent. To send a @chr(127) you must use two consecutive @chr(127)'s.

Note that WINDOW SEND can only be used to send characters for which there is a key on the keyboard. This varies from country to country. If there is not a key for a character in the string, that character is not sent. The WINDOW SEND command now sets OK to false if one or more characters could not be sent. (The SKTEST.DSC example illustrates this.)

WINDOW SETTEXT sends the contents of <string> to the specified window using a Windows message. Text sent to a main window using this command will replace whatever is in the title bar. To send text to a control such as an input field <window> must identify that actual control, which is typically done using the

@WINATPOINT function.

OK:

Set to false if no matching window was found.

Example:

```
if @winexists(Connected to Internet)
  window position,Connected to Internet,100,100
  window ontop,Connected to Internet
end
```

See also:

@WINACTIVE @WINATPOINT @WINCLASS @WINEXISTS @WINPOS @WINTEXT

Automating Applications

WINHELP

Syntax:

WINHELP <help file path>, <key>

Description:

Displays the specified Windows help file, at the page associated with the key <key>. The <key> can be either a keyword or a context ID string defined in the help file. If the second parameter <key> is omitted, the contents page of the help file is shown.

A context ID string must be prefixed by an "=" character, to distinguish it from a keyword. It is not usually possible to find out what the context ID strings are unless you authored the help file. The advantage of using a context ID string is that it uniquely identifies a topic in the help file.

If you specify a keyword, and more than one help topic is associated with the keyword, a list of topics is displayed. If no keyword matches the keyword exactly the keyword index is displayed with the nearest match selected.

OK:

Set to False if the command fails.

Example:

WINHELP DS.HLP,events

WINHELP ds.hlp,=key_winhelp

See also:

Window Spy

You use the Window Spy to get information about other application windows which you need to automate them using the WINDOW command.

Lauch the Window Spy Help File

Continues execution of the script from the current point

Controls are objects such as buttons, text and list boxes which may appear in a window.

If a DDE EXECUTE command isn't working, replace DDE EXECUTE by INFO. This will display the DDE command that is being sent in a message box, so you can check that the syntax is correct and it isn't being truncated.

Traces through the script. As each line is executed it is highlighted in the editor window. If the debug window is showing then its contents are updated after every line so you can watch the variables changing dynamically.

The Edit menu contains options for cutting text, copying it to and pasting it from the clipboard, and for search and replace.

Copies the text selected in the code window to the clipboard.

Cuts the text selected in the code window to the clipboard

Search for text in the script

Pastes text from the clipboard into the code window at the current cursor position

Events occur as a result of interaction with the script program, such as when a button is pressed or when the dialog window is closed. Timer events can also occur at intervals. You can halt a script and wait for an event to occur using the WAIT EVENT command, and determine the type of event using the @EVENT function..

Note:

1. When copying code from the online help into the script editor make sure that any blank lines are removed and any word-wrapped lines are restored to a single line or you may get errors when running the script.

2. Some of the example scripts use Windows 95 features which are not available if you are using a 16-bit version of Visual DialogScript.

The field separator is the character used to separate items of data on a line of text, such as a database record. DialogScript uses the vertical bar "|" by default, but this can be changed using the OPTION command. Data fields can be split up and stored in separate variables or dialog contrls using the PARSE command.

The File menu contains options for creating new scripts, opening and saving scripts, and for exiting Visual DialogScript.

Creates a new blank script.

Opens an existing script.

Saves the script.

The Help menu allows you to access Visual DialogScript's online help.

Displays the contents page of the online help.

Allows you to perform a keyword search of the online help.

A dialog control (list box, combo box) that can be treated as a string list

Creates an executable file

If this option is checked the variables in the script window will show their values as a tooltip when roll over with the mouse.

This option sets the font to be used in the debug window

This option sets the font to be used in the script editor

If this option is selected, when you press the Enter key in the editor, the new line is indented to the same level as the one above it.

If this option is checked, the development environment will minimize all its open windows when the script you are testing runs.

Check this option to have all the items in string lists displayed in the debug window. If the lists are large this option should be turned off otherwise it will slow down the debugger.

This option sets the tab interval to be used by the editor. An interval of 1 disables the tab key.

Note that tabs are implemented by inserting spaces. Tab characters must not appear in a script source file.

If this option is checked the debug window will be kept on top of other active windows.

Prints the script on the default printer

Moves a line or block of text away from the margin by as many spaces as defined in the editor options.

Runs the script.

The Run menu is used to test run the script. Other options enable you to set command line parameters for the script being tested, and make an executable file

Shell operations are defined for each file type in the Windows Registry. They are the operations like Open and Print which you see listed at the top of the context menu when you right-click on a file.

Executes the next line of the script

Stops execution of a running script. If the script is running in the debugger, you can continue or single step execution from the point at which it stops, and view the values of the variables in the debug window.

Styles are names which are associated with a set of attributes (such as color, font) that determine how a dialog or a control should appear. They are set using the STYLE dialog element. In addition, there are some predefined styles such as CLICK that apply only to dialogs or specific controls.

From the Tools menu you can access utilities which may help with the development of your script program.

Tools are placed in the Tools folder, subsidiary to the VDS main folder. The IDE looks in there at start-up, and builds the Tools menu from the names of the EXE files contained therein. So to install a tool you just place it, or a shortcut to it, in the Tools folder. A parameter is passed to the tool, of the path of the script being edited.

Shows or hides the debug window

A window can be identified by one of three things:

- its title (the text that appears in its title bar)
- its class name (an internal name that can be discovered using the Window Spy)
- its window identifier (a value obtained by using the @WINACTIVE, @WINEXISTS or @WINATPOINT functions)

The window class name is the name given to the window by the application's programmer. This name, unlike the window title, does not change. You can find out the window class name using the Window Spy, which can be selected from the Tools menu.

This is a numeric value by which Windows identifies a specific instance of a window. Microsoft calls it the *window handle*.

How to Register

This program can be registered through our sole agents RegNet - The Registration Network. RegNet can be located on the World Wide Web at the following URL:

http://www.swregnet.com/

Registration costs $99.00 US.

You can register online using your credit card. RegNet offers secure transactions for users of NetScape or Microsoft Internet Explorer 3 or higher. To go to the registration page for Visual DialogScript 3 go to:

http://www.reg.net/product.asp?ID=4240

You can reach this page by choosing Register now from the Help menu and then clicking the Register button on the Registration screen.

Alternatively, you can send a check or money order (in US dollars drawn on a US bank), made payable to Wintronix, Inc to:

RegNet
24303 Walnut Street
Suite 200
Newhall, CA 91321
U.S.A.

You can also call the toll-free number 1 800 WWW2REG (1 800 999-2734). Callers from outside the US should call (661) 288-1827.

Upon registration (allow up to 1 working day for your registration to be processed) you will receive your personal registration key. This will be sent by email if an email address was specified when registering.

When you receive your registration key, place it in the program main directory (e.g: c:\program files\Visual DialogScript). The software will then behave as a fully licensed version.

If you don't have a credit card and can't write a dollar check.

License Agreement

Visual DialogScript is not free software. The software is the copyright of S.A.D.E. s.a.r.l. and is protected under international law. You may not sell, rent or lease this software. Reverse engineering, decompiling or disassembling the software, and use of the software to create competitive products, is expressly forbidden.

This is a legal agreement between you and S.A.D.E. s.a.r.l. Subject to your acceptance of the conditions and restrictions set out below, you are granted a free-of-charge license giving limited rights to use the software.

1) This software is provided as is. All responsibility for determining fitness of purpose rests with you, the user, for which purpose the free evaluation period is provided. On purchase of a full license, you are deemed to have agreed that the software is fit for the intended purpose of use. No refund of the license fee shall be entertained thereafter, for whatever reason.

2) You agree not to hold S.A.D.E. s.a.r.l. liable for any loss or damage or consequential loss or damage arising directly or indirectly out of the use of the software or for any other reason. You accept that it is entirely your responsibility to take all necessary precautions to safeguard against any such loss, damage pr consequential loss or damage, howsoever it may be caused.

3) You agree to use the software solely and exclusively for the purposes of evaluation, for a period of not exceeding 28 days. On the expiry of this period, or when you have completed your evaluation (whichever comes first) you agree to either a) obtain a full license for the software, or b) cease using the software and any programs created using it, and to delete all the installed copies of the software that you have made.

4) You agree not to distribute programs created using the software. An unlimited license to redistribute programs created using the product is granted with every unrestricted license that you purchase.

Obtaining an Unrestricted Software License

One full license must be purchased for each non-networked system on which the software is installed and used. In the case of systems in which the program is simultaneously accessible to more than one user at a time, such as a network server, one full license must be purchased for each user who is capable of accessing the software.

Registering the software

Visual DialogScript is not free software. You are required to pay a registration fee and register the software if you wish to continue using it after an evaluation period of not exceeding 28 days.

The software itself is not time limited. However, some advanced functions as the Icon Editor or the resource compiler aren't available into the shareware package. Also, in the shareware version, scripts can't be compiled.

When you register the software you will receive a personal registration key file which you must place in the program's main directory (e.g: c:\program files\Visual DialogScript). After this key file is placed in the directory you will be able to compile your scripts.

How To Register

If you don't have a credit card and can't write a dollar check

If you don't wish to pay by credit card and you are unable to write a dollar check, then you can obtain your registration code direct from S.A.D.E. s.a.r.l. in France, **ONLY** if you can send your payment as a check written in Francs, drawn on a French bank or a Eurocheck.

Terms are strictly cash with order. S.A.D.E. s.a.r.l. can **NOT** accept credit card orders.

Product: **Visual DialogScript 3**

	Unit price	**No. copies**	**Total**
Program registration fee:	FF 699.00	_____	FF_____
Software on disk (if required):	FF 50.00	_____	FF_____
	========		
TOTAL (payable to S.A.D.E.)	FF_____		
	========		

Order number: _____

Your name: _____

Company name: _____

Your address: _____

_____

_____

Post/Zip code: _____

Country: _____

Email address: _____

Send to: S.A.D.E. s.a.r.l.
94-96, rue Victor HUGO
94200 IVRY SUR SEINE
FRANCE

Moves a line or block of text closer to the margin by as many spaces as defined in the editor options.

Undoes the last change made to the script.

Undoes the changes made by undo

Getting support

Two levels of support are offered:

Free pre- and post-sales support covers Installation problems, bug reports and the provision of workarounds (if possible) where a bug is identified,and simple questions, like "Can you do this using VDS?"

An extra-cost support plan is also offered which provides support cover for the following areas:

- Help achieving a specific task (we will write sections of script code)
- Help debugging a script that won't work
- Help writing a script (within the limit of two hours worth of work)
- The provision of interim upgrades of the VDS software by email where a bug prevents a script from working or necessitates an inconvenient workaround (other customers must wait for the fixed version to be generally released.)

The support is invoiced per hour . It has to be bought in advance by "Time Pack" :

1 hour = USD 150 *Buy through internet*

To learn more about advanced support and Time Packs, go to the web site at:

http://www.dialogscript.com/uk/suppo-direct.html

Whatever your support problem, please help us to help you by including the following information in your request:

- Your name, company (as specified on registration)
- The product name and version number (from the About box)
- The version of Windows you are using
- The sequence of events (or a section of script code) which will enable us to reproduce the problem.

Support for Visual DialogScript is provided via email to:

support@dialogscript.com

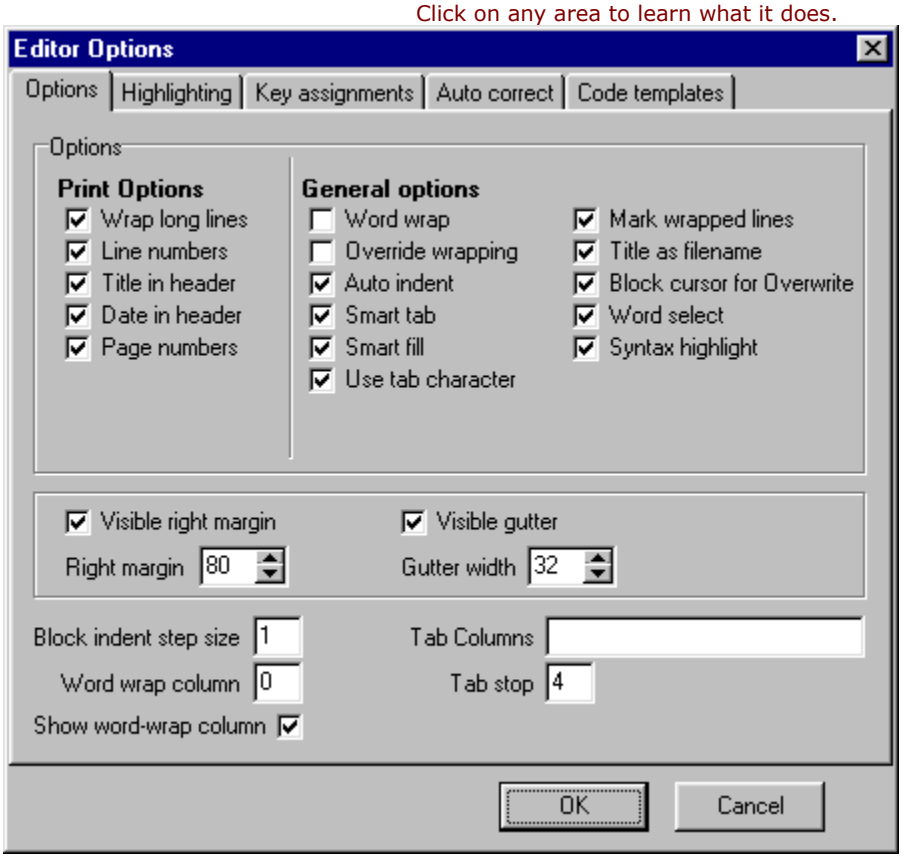
Please do NOT send large screendump files as enclosures as your message may be deleted without any notification.

Allows you to select a folder to be used as the default folder for script projects.

Editor Options

A complete property editor is supplied with which allows setting of all aspects of the display styles, fonts, options, key-definitions, code template definitions, auto-replace entries and printing option:

Click on any area to learn what it does.



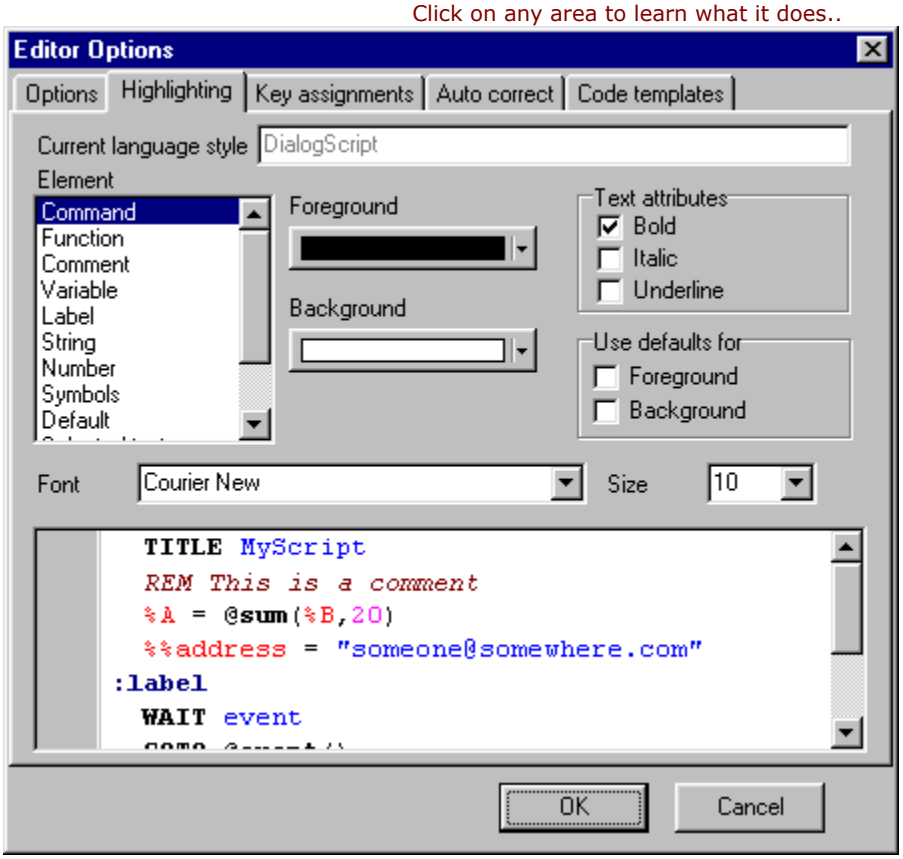
Select here the default print settings.

Set here default settings for the I.D.E.

Highlighting

With this Editor option's tab, you can set the syntax highlighting of the script window. These presets are global and are saved in a file VDSPREFS.DAT.

Click on any area to learn what it does..



Click here to select the item default background color for the current script.

Click here to select the item default text attributes for the current script.

Click here to select the language item or the script window object to be customized.

Click here to set the current values to the default values for each new script.

Select here the item default font for the current script.

Select here the item default font size for the current script.

You can see here a sample of highlighted code with your selected presets.

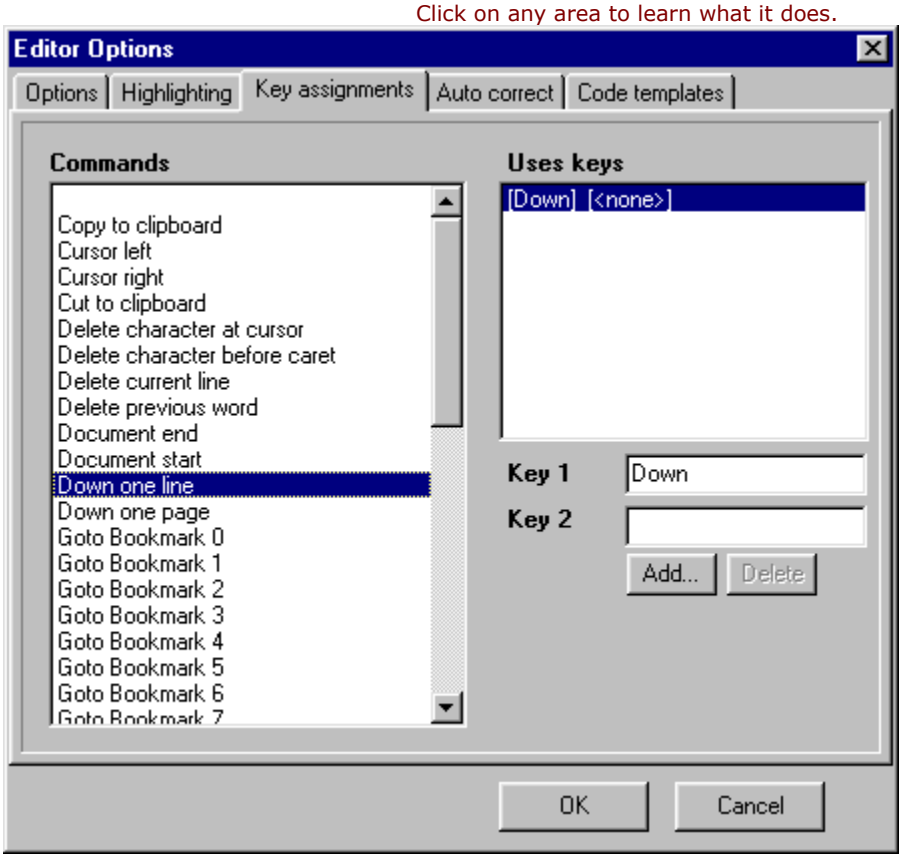
Click here to select the item default foreground color for the current script.

Show you the current language. This can't be customized.

Key assignments

With this Editor option's tab, you can set very usefull shorcuts for a more quick keyboard use:

Click on any area to learn what it does.



Hot keys combination for the selected action.

Use these fields to enter new hot keys.

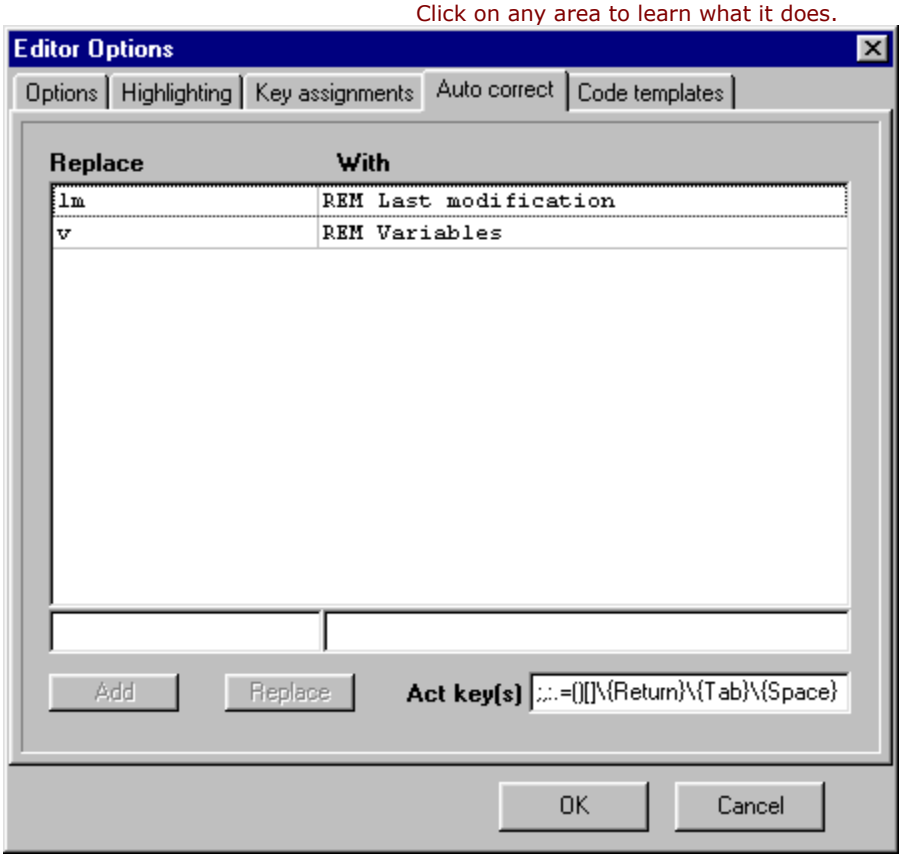
Available actions to perform with hot keys.

Add or remove a hot keys combination.

Auto Correct

With this Editor option's tab, you can "force" the script window to correct automatically some words or expressions by giving in this area those you may do:

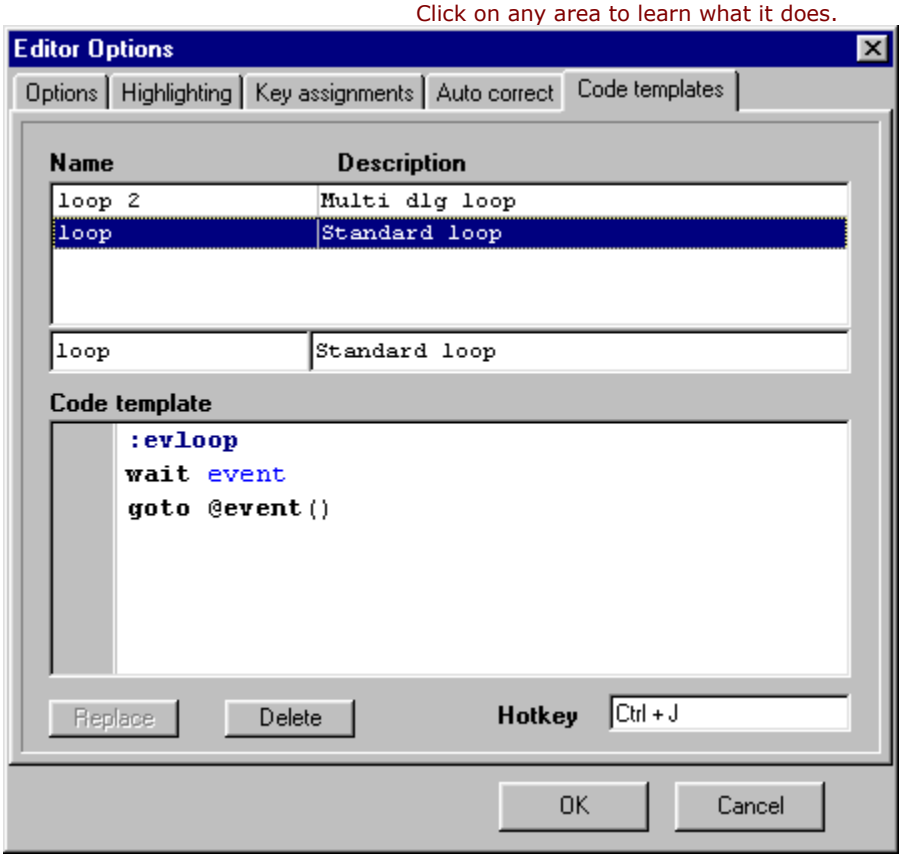
Click on any area to learn what it does.



Code templates

With this Editor option's tab, you can set the syntax highlighting of the script windows for the current script. These presets are stored in a project file (.DSP) with the same path and name as the script source.:

Click on any area to learn what it does.



Click here to select the item default background color for the current script.

You can choose a different name or location for the executable file, by pressing this Browse button. By default the EXE has the same name as the script, and will be created in the same directory.

You can change the icon from the default Visual DialogScript icon by pressing the second Browse button and choosing an icon file. Note that only 16 color 32 x 32 pixel icons can be used.

Create the executable file.

Test the created executable.

Display informations during compilation.

@DLGPOS

Syntax:

@DLGPOS(<element_name>, <flags>)

Description:

This function returns the size and position of the element <element_name> according to the value of <flags>.

Valid flags are:

T	return the top co-ordinate
L	return the left co-ordinate
W	return the width
H	return the height

Where more than one flag is specified the information returned is separated using the current field separator in a format that can be processed by the PARSE command.

If <element_name> is null, the position and size of the current selected dialog are returned. Note that the width and height returned in this case are the width and height of the client area of the dialog, excluding the border, title bar and menu (if applicable.)

OK:

Set to false if the specified window cannot be found.

@FEXP

Syntax:

@FEXP(<value>)

Description:

Floating-point function. Returns the value exp(value).

OK:

Not changed.

@FLN

Syntax:

@FLN(<value>)

Description:

Floating-point function. Returns the value ln(value).

OK:

Not changed.

@MOD

Syntax:

@mod(<value1>,<value2>)

Description:

Math function. Returns the value <value1> modulo <value2>.

OK:

Not changed.

@MOUSEPOS

Syntax:

@mousepos(<flags>)

Description:

Flags are X, Y. Default XY. Returns the X, Y co-ordinates of the mouse pointer, field separated.

OK:

Not changed.

@TEXT

Syntax:

@text(<list>)

Description:

This function gets the whole text of a string list into a variable. CR's separate each line.

OK:

Not changed.

@WINDOW

Syntax:

@WINDOW(<window>,CHILD|OWNER|FIRST|LAST|PREV|NEXT)

Description:

This function returns the window id of a window related to <window> in a manner specified by the second argument.

OK:

Unchanged.

DIALOG ADD BITBTN,<name>,<top>,<left>,<width>,<height>,<filename>, {<caption>,<hint>}

The BITBTN dialog element is a flat Explorer-type button, which shows a raised boundary when the mouse pointer is over the button. The filename is the name of the bitmap to be displayed in the button. It should ideally have 16 colours or fewer. The bitmap can be loaded from a resource file (special VDS resource format.) In this case, the filename must be followed by a vertical bar and the offset, in bytes, of the BMP file within the resource. No spaces are permitted between the filename, the vertical bar and the offset.

The caption is optional, and appears below the bitmap if present. The hint is also optional, and defines the etxt that appears in a tooltip when the pointer is over the button. If no hint text is present, the button name is used instead. There are no styles for this dialog element.

The BITBTN generates a <name>BUTTON event when it is clicked.

If the button's name is HELP, a button-click event is automatically generated if the F1 key is pressed.

DIALOG ADD,GROUP,<name>,<top>,<left>,<width>,<height>,<caption>{tyle...}

This dialog element is a group box (similar to that which surrounds a RADIO group of buttons. When a style is applied to a group box it applies automatically to all the dialog elements placed on it.

The order of creation of the GROUP dialog element is important. Dialog elements that are created subsequent to it, and whose top left corner is within its boundaries, are owned by the group element. (This is because some elements, if their owner is the dialog itself, are drawn behind the group element and are hence obscured.) One effect of this is that if a style is applied to a group element, all the elements owned by it automatically have that style also. Another effect is that in design mode, if you move the group element, all the elements owned by it are moved with it.

DIALOG ADD,MENU,<menu_name>,<item_1>,<item_2>...

This dialog element adds a pull-down menu to the window. <menu-name> is the name that appears on the menu bar. <item_x> are the individual items on the pull-down menu. Each item is a separate parameter, so they are separated by commas. When clicked, a menu item generates an event of type <item>MENU, for example: NewMENU when you click File, New. No event is generated when you click the main menu item on the menu bar (e.g. File.)

Successive pull-down menus are created by each successive DIALOG ADD command.

To specify the default keyboard shortcut, as shown by an underlined letter on the menu, you prefix that letter with an amperstand "&." For example: &File. Additional keyboard shortcuts can be specified by following the item name with a vertical bar and then the description of the shortcut. Line separators in a menu are speccified by an item name of a dash. So a simple example: DIALOG ADD,MENU,&File,&Open|F6,&Save|F10,-,E&xit|Alt+X

Note: MENUs are not implemented in the same way as other dialog elements. The SET, CLEAR, HIDE, SHOW, ENABLE, DISABLE and REMOVE verbs don't work with them. Once a menu is added to a dialog it is fixed.

DIALOG ADD,TAB,<name>,<top>,<left>,<width>,<height>,<tabs>{,<style>}

This dialog element adds a tab control to the dialog box. The names of the tabs are defined in <styles> as a string of names separated by vertical bar characters, e.g. Tab 1|Tab 2|Tab 3. When a tab is clicked, a <tab name>CLICK event is generated. The script must process this to remove dialog elements that are not required for this tab page, and recreate new ones. (In design mode, a <name>CLICK event is generated instead, since it would be a problem for dialog designers to provide handlers for user-defined tab events.)

A tab dialog element is a group element. See the description of the GROUP dialog element for more explanation of this.

List of currently setted replacements.

Type here the usual errors to be replaced.

Type here the corrected text to insert when you type the abreviations or when you make some "recursive" errors.

Add here all characters which indicate the IDE to check last word typed.

List of currently setted code templates.

Sample code associated with the selected item from the field above and to be inserted.
It's also the field to use to write a new sample.

Define here the hot key to activate the list popup which allows you to select a sample into the IDE.

Type here the name for a new sample.

Type here a short description for the new sample.

Resource Compiler

VDS 3 supports resource files which are files in which bitmaps and text have been concatenated together into one file. Resource files may be used in any command that may load a bitmap or text from a separate file. Resource files may have any file extension other than BMP, ICO, EXE, DLL or TXT. The recommended extension is DSR (DialogScript resource.) When loading from a resource file, the byte offset within the file of the particular resource you are loading is specified by following the filename with a '|' separator bar followed by the offset. Example: DIALOG SET,BITMAP1,MYRESRCE.DSR|3066 tells the VDS program to load a bitmap starting 3066 bytes into the file MYRESRCE.DSR. It is the programmer's responsibility to ensure that the byte offset is correct. If it is not, the data read in will be invalid and VDS will probably crash.

Note: icons can be included in resource files but only for use by TASKICON dialog elements. The method for loading icons into BITMAP dialog elements is different. VDS uses the file extension to determine that an icon is being loaded. If the extension is ICO, EXE or DLL VDS assumes an icon is being loaded. EXE and DLL files may contain multiple icons, in which case the same syntax as above can be used to specify the icon offset (n.b. NOT the byte offset.) For example, to load the second icon in an icon library called ICONLIB.DLL you would use: DIALOG SET,BITMAP1,ICONLIB.DLL|2.

VDS 3 does not support writing to resource files. Resource files may be created by using the VDS Resource Compiler VDSRC.EXE.

The resource compiler is a command line tool. The program displays help when run with no parameters.

Usage: VDSRC resourcescript [resourcefile] [options]
(Resourcefile is list of files to be included)

resourcescript = <filepath>.RC
resourcefile = <filepath>.DSR

Options :

/M - create map file <resourcescriptpath>.MAP
/Q - quiet operation
/G - display GUI message boxes not console output

The map file is basically the output from the console, showing the resources and their offset within the resource file, so this can be entered into the script.