@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 = Visual DialogScript @CHR(169) 1995, 1996 J M Technical Services
```

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-string> is given, a time value is returned. This is an integer value which corresponds to the DOS file date/time value.

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.

The <time-value> may be either an integer DOS file date/time value or a date in the Short Date format.

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>)

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.

Note: the 16-bit version of the directory browser uses drive and directory windows similar to the standard File Open dialogs and as with those dialogs the user must *double-click* a directory to select it.

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;

L - the line of code containing the 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): @ERROR(L)
```

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()

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.

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>})

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.

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>)

Description:

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

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.

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.

16-bit script programs will write to the configuration file DS16.INI if no INI file has been opened by name, and the text 'Program\<program name>\' will be prepended to every section name used in an attempt to make it unique. 32-bit scripts will use a default file DEFAULT.INI.

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>)

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 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 names are: 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 Example 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

Message box tester example

@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>. The first item returned will be the first item (item 0) unless the item pointer has been repositioned using the LIST SEEK command. After the function has been executed the list index is advanced by 1.

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.

<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\JM Tech\DialogScript\2.0\UserScripts in HKEY_CURRENT_USER **(32-bit only)**.

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

<name> specifies the named value to retrieve **(32-bit only)**. 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.

Note **(16-bit only)**: In the Windows 3.1 Registry only the ROOT key is supported. Named key values cannot be accessed, even when running on a 32-bit system, as the function is not supported by the 16-bit Windows API. Therefore the <name> parameter is discarded in this version.

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\JM Tech\DialogScript\2.0\SourceWin,Top)
```

See also:

REGISTRY Example Tip

@RETCODE (32-bit only)

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 (32-bit only)

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 Version Information example

@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. **(32-bit only)**

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

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:

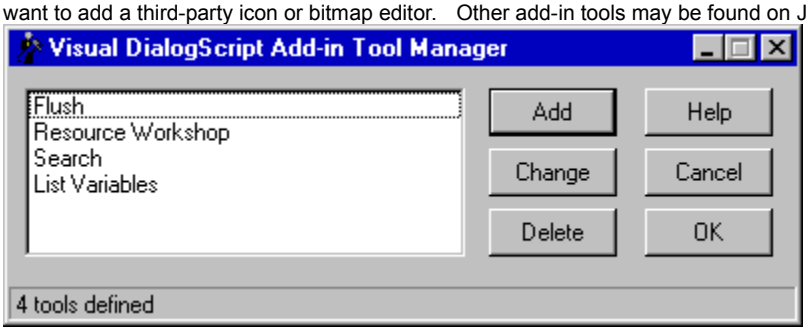
```
IF @ZERO(%V)
  INFO Result is zero.
END
```

See also:

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

Add-In Tool Manager

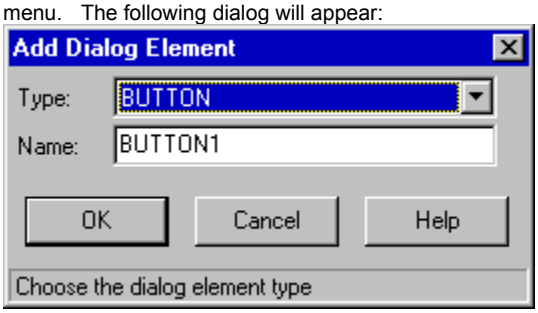
The Add-In Tool Manager lets you add other development tools to the Visual DialogScript Tools menu. For example, you might want to add a third-party icon or bitmap editor. Other add-in tools may be found on J M Technical Services' Web site.



Click Add to add a new tool to the menu. You will be asked for a name for the tool (which is what will appear on the menu) and then to specify its path.
Select a tool from the list and click Change to change either of the items of information about a tool.
Select a tool from the list and click Delete to delete a tool from the list.
Click OK to have the changes and additions you have made come into effect.

Adding Dialog Elements

To add a control or define a new style for the dialog select the dialog element list on the Dialog tab, and press **Ins**, or use the context menu. The following dialog will appear:

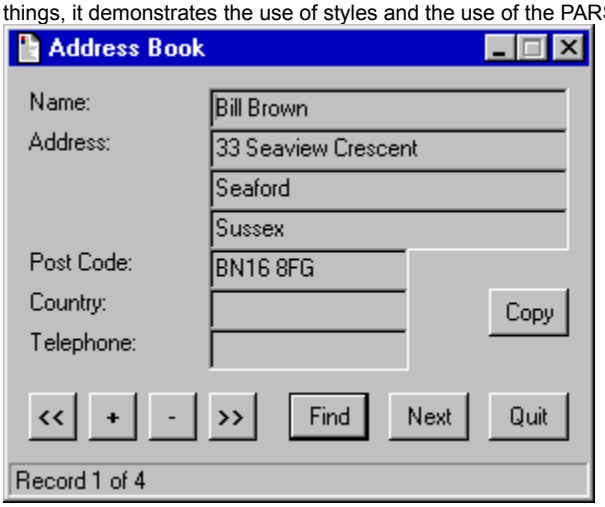


From the drop-down list you can select the type of dialog element you want to add. The Dialog Editor will choose a name for the element, which you can overwrite with a name of your choice if you prefer. The Dialog Editor will validate that the name is unique and does not contain any illegal characters when you press the OK button.

When you press OK, in the case of a control a control is added to the form with the minimum default attributes. The new control is selected and its details displayed in the Dialog Editor ready for you to edit them, as you would an existing control. For a style the editor shows a style with all fields left blank, ready for you to define the font and color attributes.

Address Book

This example shows how Visual DialogScript can be used as a simple database, storing data in ASCII text form. Amongst other things, it demonstrates the use of styles and the use of the PARSE command.




```
Title Address Book
DIALOG CREATE,Address Book,%T,%L,296,220,STYLE(Gray3D),-
  STYLE(Bold;;;B),DLGTYPE(SAVEPOS),-
  Text(T1;10;10;80;;Name:),Edit(E1;10;100;180;;;gray3d),-
  Text(T2;30;10;80;;Address:),Edit(E2;30;100;180;;;gray3d),-
  Edit(E3;50;100;180;;;gray3d),Edit(E4;70;100;180;;;gray3d),-
  Text(T3;90;10;80;;Post Code:),Edit(E5;90;100;100;;;gray3d),-
  Text(T4;110;10;80;;Country:),Edit(E6;110;100;100;;;gray3d),-
  Text(T5;130;10;80;;Telephone:),Edit(E7;130;100;100;;;gray3d),-
  Button(Find;162;140;40),Button(Next;162;190;40),
  Button(Prev;162;10;24;;<<;Bold),-
  Button(New;162;40;24;;+;Bold),Button(Del;162;70;24;;-;Bold),-
  Button(Nxt;162;100;24;;>>;Bold),Button(Copy;110;240;40),
  Button(Quit;162;240;40),Status(SP;No data)
list 1,create,sorted
%N = 1
list 1,loadfile,addrbook.txt
if @not(@zero(@count(1)))
  gosub showrec
  %N = 0
end
:evloop
wait event
goto @event()
:NewButton
  gosub updaterec
  parse E1;E2;E3;E4;E5;E6;E7,|||||||
  %N = 1
  dialog set,SP,New record
  dialog focus,E1
  goto evloop
:PrevButton
  gosub updaterec
  %S = @index(1)
  if @not(@zero(%S))
    list 1,seek,@pred(%S)
  end
  gosub showrec
  goto evloop
:NxtButton
  gosub updaterec
  %S = @succ(@index(1))
  if @greater(@count(1),%S)
    list 1,seek,%S
  end
  gosub showrec
```

```
  goto evloop
:DelButton
  if @ask(Delete record for @dlgtext(E1)?)
    list 1,delete
  end
  gosub showrec
  goto evloop
:FindButton
  %S = @index(1)
  gosub finddlg
  gosub match
  gosub showrec
  goto evloop
:NextButton
  if @null(%G)
    gosub finddlg
  else
    %S = @index(1)
    if @equal(@succ(%S),@count(1))
      info No more matches
      goto evloop
    end
    list 1,seek,@succ(%S)
  end
  gosub match
  gosub showrec
  goto evloop
:CopyButton
  clipboard clear
  %I = 1
  repeat
    clipboard append,@dlgtext(E%I)
    %I = @succ(%I)
  until @greater(%I,6)
  goto evloop
:QuitButton
:Close
  gosub updaterec
  list 1,savefile,addrbook.txt
  exit
:updaterec
  if @dlgtext(E1)
    %T = @dlg(E1)|@dlg(E2)|@dlg(E3)|@dlg(E4)|@dlg(E5)|@dlg(E6)|@dlg(E7)|
    if @not(@equal(%Y,%T))
      if @zero(%N)
        list 1,delete,%S
      end
      list 1,add,%T
    end
  end
  %N = 0
exit
:showrec
  %Y = @item(1)
  parse E1;E2;E3;E4;E5;E6;E7,%Y
  dialog set,SP,Record @succ(@index(1)) of @count(1)
  dialog focus,E1
  exit
:finddlg
  %G = @input(Enter search text:,%G)
  list 1,seek,0
  exit
:match
  if @not(@match(1,%G))
    info No match for %G found
    list 1,seek,%S
  end
  exit
```

Note

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, 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.

OK:

Unchanged.

Example:

```
BEEP
```

See also:

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 image can be a bitmap (BMP) icon (ICO) or metafile (WMF) format. If an executable (EXE) file is specified then its icon is displayed. 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.

BREAK

Syntax:

BREAK 1

BREAK <expression>

Description:

This command marks the position of a breakpoint during debugging. It is usually inserted using the Add/Clear Breakpoint at Cursor options from the script window context menu.

Execution of a script halts when a BREAK command is reached if the parameter following the command is non-null. The default use of the BREAK command, as inserted from the context menu, is BREAK 1.

You can insert a conditional breakpoint by making the parameter following the command a function, such as @EQUAL(%I,8). This example would only evaluate to non-null if variable %I was equal to 8.

OK:

Unchanged.

Example:

```
BREAK 1

BREAK @EQUAL(%I,8)
```

See also:

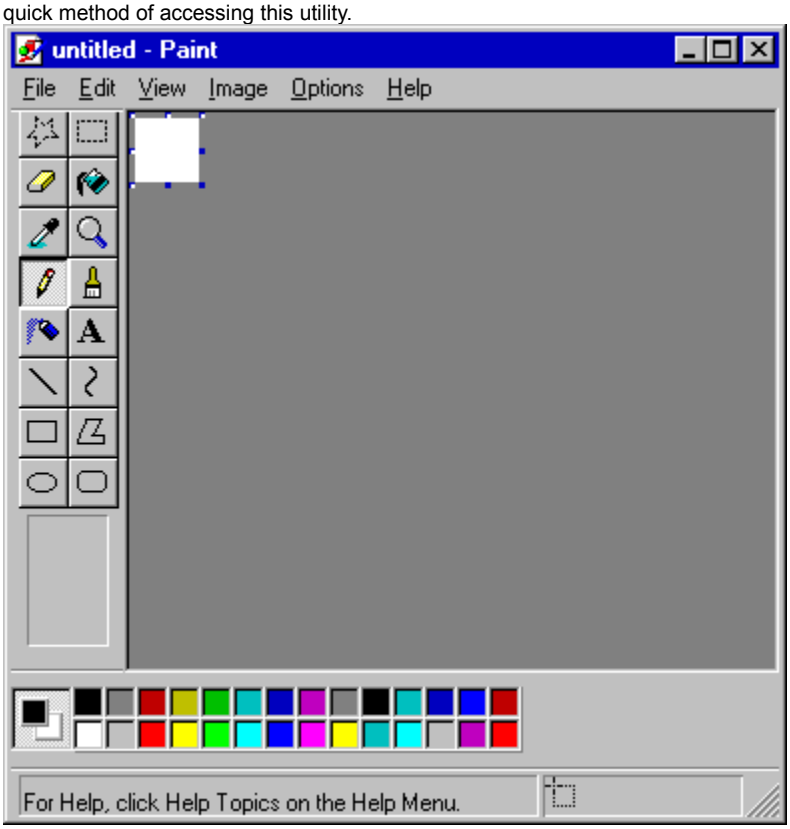
The Visual DialogScript development environment

BUTTON(<name>;<top>;<left>;<width>;<height>;<caption>;<style>)

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.

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.



CHECK(<name>;<top>;<left>;<width>;<height>;<caption>;<value>;<style>{; <style>})

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 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

COMBO(<name>;<top>;<left>;<width>;<height>;<value>;<style>{; <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

Command line compiler (Professional version only)

The command line compiler can be used to make executable files from scripts that have become too large for the integrated editor to handle. It could also be used by a "program generator" application which outputs a script tailored according to various information input to it, and then creates a custom executable.

To use the command line compiler you must execute the command:

```
DS(16) /C script-file-path
```

This will create a matching EXE in the same directory.

Additional command line options are:

/X exe-path	- specifies alternative path for EXE file
/I icon-path	- specifies path of icon to use
/INT	- specifies an integrated EXE to be created
/16	- (32-bit only) creates a 16-bit executable

DS.EXE returns an errorlevel of > 0 if the attempt to create an EXE fails. DS16.EXE gives no indication of failure as a 16-bit Windows program cannot pass an errorlevel value, so it is advisable to delete the EXE, if it exists, before running the command line compile, and then test for its existence on completion.

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

Controlling Notepad

This script gives an example of how to control another application (the Windows 95 Notepad) using Visual DialogScript. Note the use of repeat .. until @winexists() to ensure that a window has appeared before sending keystrokes to it, and the use of wait commands to give the application time to respond to commands.

```
title WinTest
%A = Notepad
%W = Untitled - Notepad
%X = Help Topics: Notepad Help
%Y = Notepad Help
%Z = Save As
if @winexists(%W)
  window activate,%W
else
  run %A.EXE
end
file delete,test.txt
wait 1
window position,%W,84,640,400
rem enter some text
window send,%W,This is a test string@CR()
window send,%W,This is some more text@CR()
window send,%W,@tab()This text is indented.@CR()
window send,%W,123+£456=£789!!  5 > 4 = true?@CR()
wait 1
rem open and close help
window send,%W,Let's look at the help file!@CR()
window send,%W,@key(F1)
rem wait for Help Topics dialog
%C = 1000
repeat
  %C = @pred(%C)
  if @zero(%C)
    warn Cannot find window %X - aborting
    exit
  end
until @winexists(%X)
wait 1
window send,%X,@key(down)@key(down)@cr()@key(down)@cr()
%C = 1000
repeat
  %C = @pred(%C)
  if @zero(%C)
    warn Cannot find window %Y - aborting
    exit
  end
until @winexists(%Y)
wait 2
window send,%Y,@alt(@key(F4))
wait 2
window send,%W,Let's copy some text to the clipboard...@CR()
wait 1
window send,%W,@ctrl(@key(home))@shift(@key(end)@key(down)@key(down)@key(down)@key(end))
wait 1
window send,%W,@ctrl(c)
wait 1
window send,%W,@ctrl(@key(end))
wait 1
window send,%W,... and insert the copied text here.@CR()
wait 1
window send,%W,@shift(@key(ins))
wait 3
window send,%W,@CR()Now let's save the file.@CR()
rem save the file as TEST.TXT
window send,%W,@alt(F)A
rem wait for Save As dialog
%C = 1000
repeat
  %C = @pred(%C)
  if @zero(%C)
```

```
    warn Cannot find window %Z - aborting
    exit
  end
until @winexists(%Z)
wait 1
window send,%Z,TEST.TXT
wait 2
window send,Save As,@chr(27)
wait 1
window send,%W,Changed my mind!@CR()
wait 2
rem quit Notepad
window send,%W,@alt(FX)
%C = 1000
repeat
  %C = @pred(%C)
  if @zero(%C)
    warn Cannot find window %A - aborting
    exit
  end
until @winexists(%A)
wait 1
rem select No option on dialog
window send,%A,@alt(n)
exit
```

Note

Creating Dialogs

DialogScript allows you to create a dialog window for your script, containing buttons and other controls. You can use the dialog to present information to (and obtain it from) the user. You create a dialog window using the DIALOG CREATE command, which has the format:

DIALOG CREATE, <window>, <top>, <left>, <width>, <height>, <description>

The text <window> appears in the title bar of the window. The parameters <top>, <left>, <width> and <height> specify the initial position and dimensions of the dialog window. If the value of <top> is negative then the window is positioned in the center of the screen. The value of <height> refers to the client area of the window: it does not include the height of the title bar.

The <description> consists of one or more dialog elements, separated by commas. Because the description can consist of many elements, DialogScript allows you to continue the description on the following line, by ending the line with a comma followed by a dash. Note that the dialog position and size parameters must be in the first line of the command.

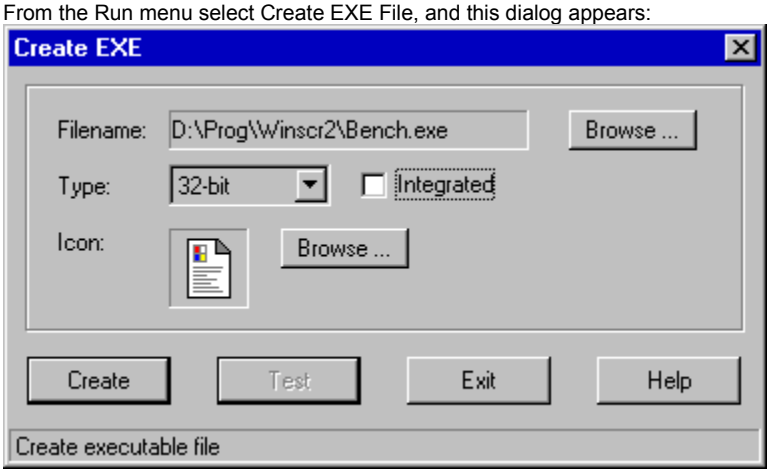
Dialog elements begin with the element type, which is optionally followed by details such as the control name, size, position and initial value. The details are contained within brackets and separated by semicolons. See Dialog Elements for more information. See also Example.

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

Making Scripts into Executable Files (Professional version only)

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 Run menu select Create EXE File, and this dialog appears:



You can choose a different name or location for the executable file, by pressing the first Browse button. By default the EXE has the same name as the script, and will be created in the same directory.

You can choose to create either a 16-bit or a 32-bit executable file. A 32-bit executable requires Windows 95 or a later version of the Windows operating system. A 16-bit executable runs under Windows 3.1 and above.

By default, the executable files created by Visual DialogScript call the runtime engine DSRUN.EXE (32-bit) or DSRUN16.EXE (16-bit) which must be present on the Windows search path (in other words, the current directory, the DOS path, the Windows directory or the Windows system directory) when the program is run. This makes the executable files very small. If you check the Integrated box then a copy of the runtime engine will be bound into the executable file, so that the EXE is completely self contained.

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.

Press Create to create the executable file.

You can test the executable by pressing the Test button.

The settings in this dialog are stored so any changes made do not have to be recreated the next time you create an executable from the same script. The settings are stored keyed on the script name, so you should not have two scripts with the same name in different directories or the settings will conflict.

Note that to test a 16-bit executable the 16-bit run time must be installed on the search path. The 16-bit runtime must also be present in order to be able to create 16-bit integrated executables.

You can also create executable files using the command line compiler.

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.

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 CLEAR, <control name>
DIALOG CLEARSEL, <control name>
DIALOG CLOSE
DIALOG CREATE, <dialog name>, <top>, <left>, <width>, <height>, <description>
DIALOG CURSOR {, WAIT}
DIALOG DISABLE, <control name>
DIALOG ENABLE, <control name>
DIALOG FOCUS, <control name>
DIALOG HIDE, <control name>
DIALOG POPUP
DIALOG SET, <control name>, <text>
DIALOG SHOW, <control name>
DIALOG TITLE, <title>

Description:

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.

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 tells the dialog window to close. It has the same effect as closing the dialog from the system menu or the close button, and will generate a CLOSE event. (Note: the dialog will not disappear until the program itself terminates. You cannot close a dialog and then create a new one within the same script.)

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 creates a dialog window to the design you specify. 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 Editor to generate this command.

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 causes the dialog's popup menu to pop up at the current cursor location. This is mainly used to display a menu when the user clicks on a task bar icon and the dialog itself is hidden.

DIALOG SET sets the text in the control named <control name> to <text>. In the case of a list box the text replaces the selected item; it is added if no item is selected. In the case of a bitmap the bitmap file <text> is loaded into the control.

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.

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>

Description:

The DIRECTORY command is used to change, create or delete 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. This command is similar in operation to the MS-DOS RMDIR command.

OK:

True if the operation is successful; false if not.

Example:

```
directory delete,f:\tmp1
directory create c:\test\subdir1\subdir2
```

See also:

DLGTYPE(<style> {, <style>})

Styles can include DRAGDROP - dialog accepts drag-and-drop operations, which will cause a DRAGDROP event to occur; ONTOP - Dialog will remain on top of other windows even when inactive; SAVEPOS - Dialog will remember its last screen position whenever the program is run; NOMIN - Dialog will not have a minimize button; NOSYS - Dialog will not have a close button or a system menu; SMALLCAP **(32-bit only)** - Dialog will have a small toolbar-style title bar; plus user defined styles that are to apply to the whole dialog.

Data Lists

Syntax:

LIST <list>, DROPFILES
LIST <list>, FILELIST, <filespec>, {<attributes>}
LIST <list>, LOADFILE, <filename>
LIST <list>, LOADTEXT
LIST <list>, REGKEYS, <root key>, <subkey>
LIST <list>, REGVALS, <root key>, <subkey>
LIST <list>, SAVEFILE, <filename>
LIST <list>, WINLIST, {<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. For more information see Dialog Programming and the WinTouch example script.

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 pseudo-attribute * (asterisk) may be specified; this will return a recursive list of directories starting from (but not including in the list) <filespec>. For an example of this in use see the File Deleter example script.

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. **(32-bit only)**

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:

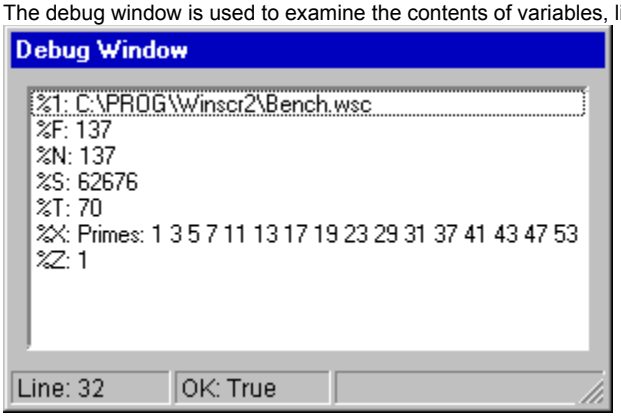
```
LIST 1,LOADFILE,C:\CONFIG.SYS

LIST LB1,FILELIST,*.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.

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.

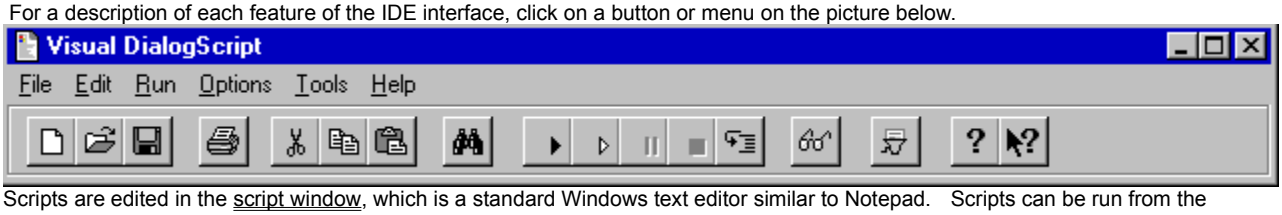
A common source of problems is failure to get information from files. This is usually caused by path problems. If you only specify a filename and not a full path, the script will look in the current directory for the file. The current directory is not necessarily the directory in which the script program 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 a standard Windows text editor similar to Notepad. 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, 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

The fields on the Dialog page of the dialog editor should be filled in as follows:

Title: This is the text that will appear in the caption bar of the dialog.

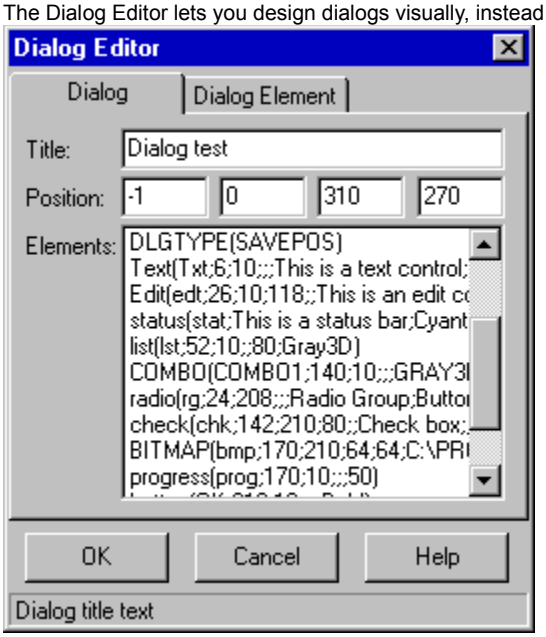
Position: These are the top position, left hand edge position, width and height, respectively, of the dialog. You can enter the value -1for the Top position if you want the dialog to appear in the center of the screen. The width and height can be set by dragging the edges of the dialog using the mouse.

Elements: This is the list of dialog elements that determines the appearance and behavior of the dialog and what controls appear on it. To edit an element you can double-click on it. To add a new element press **Ins**, or right-click the list and select **Add Item** from the context menu.

The order in which the elements appear can affect how the dialog works. When the user hits the **Tab** key the input focus will move from one control to the next in the order that the elements appear in the list. The first button to be defined will be the default button, which is the one executed when the user hits **Enter**. You can move elements up and down the list using **Ctrl+U** and **Ctrl+D**.

Dialog Editor

The Dialog Editor lets you design dialogs visually, instead of by writing a DIALOG CREATE command.



You can access the Dialog Editor in two ways:

from the code window using the context menu or the F2 keyboard shortcut. To edit an existing dialog description, place the cursor on the first line of the DIALOG CREATE command before right clicking the mouse or pressing F2. To create a dialog description where none existed before, place the cursor where you want the dialog description to go and press F2.

from the File menu, you can choose New, and then pick the Dialog Wizard. The wizard will let you run the Dialog Editor to design a new dialog from scratch, and then generate all the DialogScript code for a working (though non-functional) application.

See also:

Using the Dialog Editor

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 enclosed in brackets after the element name. The parameters are separated by semicolons. 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>)
CHECK(<name>;<top>;<left>;<width>;<height>;<caption>;<value>;<style>{; <style>})
COMBO(<name>;<top>;<left>;<width>;<height>;<value>;<style>{;<style>})
DLGTYPE(<style> {, <style>})
EDIT(<name>;<top>;<left>;<width>;<height>;<value>;<style>{; <style>})
LIST(<name>;<top>;<left>;<width>;<height>;<style>{; <style>})
POPUP(<item>{; <item>...})
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>
TASKICON(<name>;<filename>;<tooltip text>) **(32-bit only)**
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 Editor.

Dialog Programming

DialogScript allows you to create dialog windows which function as the main 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 Editor, 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. See Creating Dialogs for more information.

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. For an example of dialog processing, see the Dialog Example.

The simplest way to write a dialog-based DialogScript program is to use the Dialog Wizard. This lets you design the dialog using the Dialog Editor 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.

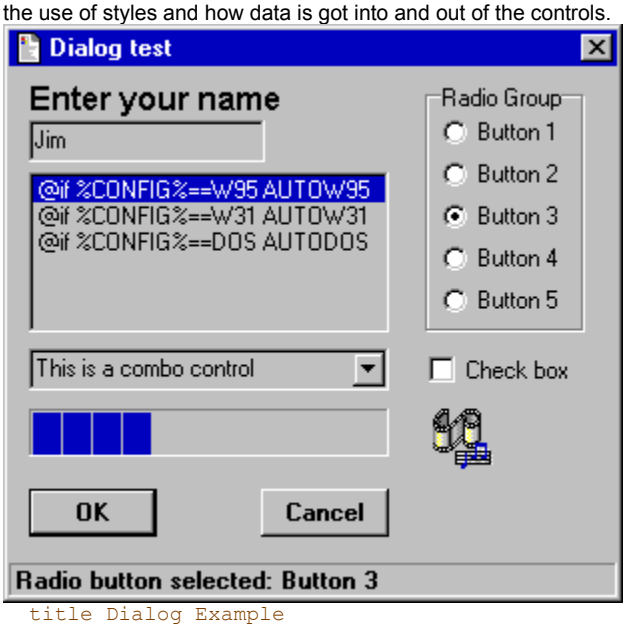
Dialog Wizard

The Dialog 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 Editor, 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.

Dialog

This script displays a dialog showing all the available control types (except the task bar icon and the popup menu.) It demonstrates the use of styles and how data is got into and out of the controls.




```
title Dialog Example
      DIALOG CREATE,Dialog test,-1,0,310,270,-
        STYLE(BigText;Arial;12;B),-
        STYLE(Gray3D),-
        STYLE(Bold;MS Sans Serif;8;B),-
        DLGTYPE(SAVEPOS;NOMIN),-
        Text(Txt;6;10;;;Enter your name;BigText),-
        Edit(edt;26;10;118;;;Gray3D;Exit),-
        status(stat;Interact with the dialog controls;Bold),-
        list(lst;52;10;;80;Gray3D;click),-
        COMBO(Combo;140;10;;;This is a combo control;GRAY3D;CLICK;EXIT),-
        radio(rg;8;208;;124;Radio Group;Button 1|Button 2|Button 3|Button 4|Button 5;Button
      1;gray3d;click),-
        check(chk;142;210;80;;Check box;;;gray3d;click),-
        BITMAP(bmp;170;210;32;32;Mmplay.ico;hand),-
        progress(prog;170;10;;;50),-
        button(OK;210;10;;;;Bold),-
        button(Cancel;210;126;;;;Bold)
        list lst,loadfile,c:\autoexec.bat
        list combo,loadtext
      "This is the combo list line 0
      "This is the combo list line 1
      "This is the combo list line 2
      "This is the combo list line 3
        %P = 0
      :evloop
        wait 1,event
        goto @event()
      :TIMER
        %P = @succ(%P)
        dialog set,prog,%P
        goto evloop
      :OKButton
        exit
      :CancelBUTTON
        if @ask(Are you sure you want to cancel?)
          exit
        end
        goto evloop
      :CLOSE
        if @ask(Are you sure you want to close?)
          exit
```

```
  end
  goto evloop
:edtEXIT
  %E = @dlgtext(edt)
  if %E
    info Hello %E
    dialog clear,edt
  end
  goto evloop
:comboEXIT
  list combo,add,@dlgtext(combo)
  goto evloop
:lstCLICK
  dialog set,stat,List item selected: @dlgtext(lst)
  goto evloop
:rgCLICK
  dialog set,stat,Radio button selected: @dlgtext(rg)
  goto evloop
:chkCLICK
  dialog set,stat,Check box state: @dlgtext(chk)
  goto evloop
:comboCLICK
  dialog set,stat,Combo list selection: @dlgtext(combo)
  goto evloop
:bmpCLICK
  dialog set,stat,Bitmap clicked at @click(x)", "@click(y)
  play sound.wav,wait
  goto evloop
```

Note

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.

EDIT(<name>;<top>;<left>;<width>;<height>;<value>;<style>{; <style>})

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

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

The 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.

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.
POWEROFF	On a system with software power control, shuts down and turns off the power. **(32-bit only)**

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.

For information about the extensions that are available, visit J M Technical Services' Web site.

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:

Editing Dialog Elements

You can change the size and position of a dialog control by dragging it on the dialog itself. To change the other attributes you must edit the dialog element which is displayed on the Dialog Element tab of the Dialog Editor.

For dialog controls the Dialog Editor looks as shown above. The type and name of the control are shown in the first two fields. They cannot be changed. To rename a control you must delete it and then recreate it.
The remaining fields are editable. Not all fields are applicable to every control type. Those which are not are disabled. When you press the Help button you will call up a page of help describing the type of control currently being edited and the fields that are applicable. After you have changed a field, press **Enter** or **Tab** and the change will be reflected in the dialog.
Note that default values are used where an applicable field is left blank. Typically, the height of a control is left blank, as controls are normally created at the appropriate height for the default size of text.

The Caption field is used for text that describes the control, such as the name of a button.

The Filename field is used to specify the name of a file, such as a bitmap file for a BITMAP control, or an icon for a TASKICON dialog element. The name can be entered manually, or selected from a standard Windows File Open dialog which is called up by pressing the button captioned ...

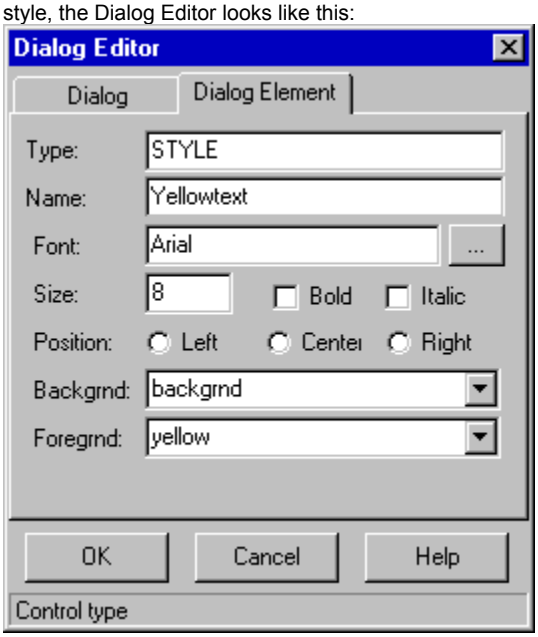
The Values field is used to define multiple values for a control, such as a group of radio buttons or the items on a popup menu. Note that values are separated from one another by vertical bar characters.

The Value field is used to set an initial value for a control which will be displayed when the dialog first appears.

In the Styles field you can select the valid styles from the drop down list. If more than one style is chosen, a semicolon must separate each one. Valid styles include those you have defined, plus special styles that apply only to a particular control, such as the PASSWORD style for an EDIT control.

Editing Styles

Styles are used to specify the colors or fonts to be used for a control, or for the entire dialog. When you are defining or editing a style, the Dialog Editor looks like this:



Again, the name of a style once defined is fixed, and can only be changed by deleting the style and redefining it.
The font name, size and style (bold, italic) can be entered manually or selected from the standard Windows font selection dialog which can be called up by pressing the button captioned ...The Position field determines whether text is left or right justified, or centered in the control.
The Backgrnd and Foregrnd fields set the background and foreground colors, respectively.
Defaults are used where fields are left blank.

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 Insufficient memory for operation

An attempt to allocate memory failed as insufficient memory was available.

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 Unexpected error: <explanation>

An untrapped error has occurred. This is usually caused by a bug in Visual DialogScript. Please report details, including the circumstances that caused it, to J M Technical Services.

30 Breakpoint reached

The script was stopped at a breakpoint.

31 Stopped by user

The script was paused by the user.

Error trap

The following is an example of an error trap that could be used to display diagnostic information in the event of a run-time error occurring in a compiled script.

Note: the example puts all the information into a single variable %9 before displaying it. This could potentially exceed 255 characters in length so this method should not be used with the 16-bit version of Visual DialogScript. Instead, it would be better to append each line to an unused string list and then save the diagnostic information to a file.

```
:errtrap
  option errortrap
  %9 = A run-time error @error(e) occurred at line number @error(n):
  %9 = %9@CR()@error(l)@CR()Variables:
  if %A
    %9 = %9@CR()"%A = "%A
  end
  if %B
    %9 = %9@CR()"%B = "%B
  end
  if %C
    %9 = %9@CR()"%C = "%C
  end
  if %D
    %9 = %9@CR()"%D = "%D
  end
  if %E
    %9 = %9@CR()"%E = "%E
  end
  if %F
    %9 = %9@CR()"%F = "%F
  end
  if %G
    %9 = %9@CR()"%G = "%G
  end
  if %H
    %9 = %9@CR()"%H = "%H
  end
  if %I
    %9 = %9@CR()"%I = "%I
  end
  if %J
    %9 = %9@CR()"%J = "%J
  end
  if %K
    %9 = %9@CR()"%K = "%K
  end
  if %L
    %9 = %9@CR()"%L = "%L
  end
  if %M
    %9 = %9@CR()"%M = "%M
  end
  if %N
    %9 = %9@CR()"%N = "%N
  end
  if %O
    %9 = %9@CR()"%O = "%O
  end
  if %P
    %9 = %9@CR()"%P = "%P
  end
  if %Q
    %9 = %9@CR()"%Q = "%Q
  end
  if %R
    %9 = %9@CR()"%R = "%R
  end
  if %S
    %9 = %9@CR()"%S = "%S
  end
  if %T
```

```
  %9 = %9@CR()"%T = "%T
end
if %U
  %9 = %9@CR()"%U = "%U
end
if %V
  %9 = %9@CR()"%V = "%V
end
if %W
  %9 = %9@CR()"%W = "%W
end
if %X
  %9 = %9@CR()"%X = "%X
end
if %Y
  %9 = %9@CR()"%Y = "%Y
end
if %Z
  %9 = %9@CR()"%Z = "%Z
end
warn %9
stop
```

Note

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.

See error trap example.

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.

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 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..

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 DLGTYPE(DRAGDROP) must be used. 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. **(32-bit only)**

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

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

Examples

These example scripts illustrate various techniques and ideas for using Visual DialogScript. To run the examples, simply copy them to a new script window,.save them to a suitable file name, modify them as necessary and then run them.

Explorer menu example

Even very simple script programs can be extremely useful. The one line script (two if you include the comment!) shown below lets you add a "DOS Prompt" item to folder context menus: when you choose the option you get a DOS window that puts you straight into the chosen folder.

```
rem Open command prompt in directory %1
shell open,command.com,,@shortname(%1)
```

Once you have made this script into an EXE (for example, DOSPrmpt.EXE) create the Registry entry for it by entering the following into a blank script window and then running it:

```
REGISTRY WRITE,ROOT,Folder\shell\DOS Prompt\command, "<insert path>\DOSPrmpt.EXE %1"
```

Note

Frequently Asked Questions

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> {, <WARNOVERWRITE> }

FILE DELETE, <file path>

FILE RENAME, <file path 1>, <file path 2>

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

FILE SETATTR, <file path>, <attributes>

Description:

FILE COPY copies the file named in <file path 1> to <file path 2>. The original date and time are preserved. The optional WARNOVERWRITE parameter **(32-bit only)** sets OK to False if the target file already exists, instead of overwriting it. **Note:** <file path 2> must be a full file path, not just a directory name (unlike the MS-DOS COPY command.) Note also that wildcards are not supported in the file paths. To copy a list of files you must use LIST FILELIST to build a list of filenames, and then copy each file one by one.

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 File Copy example

File Copy

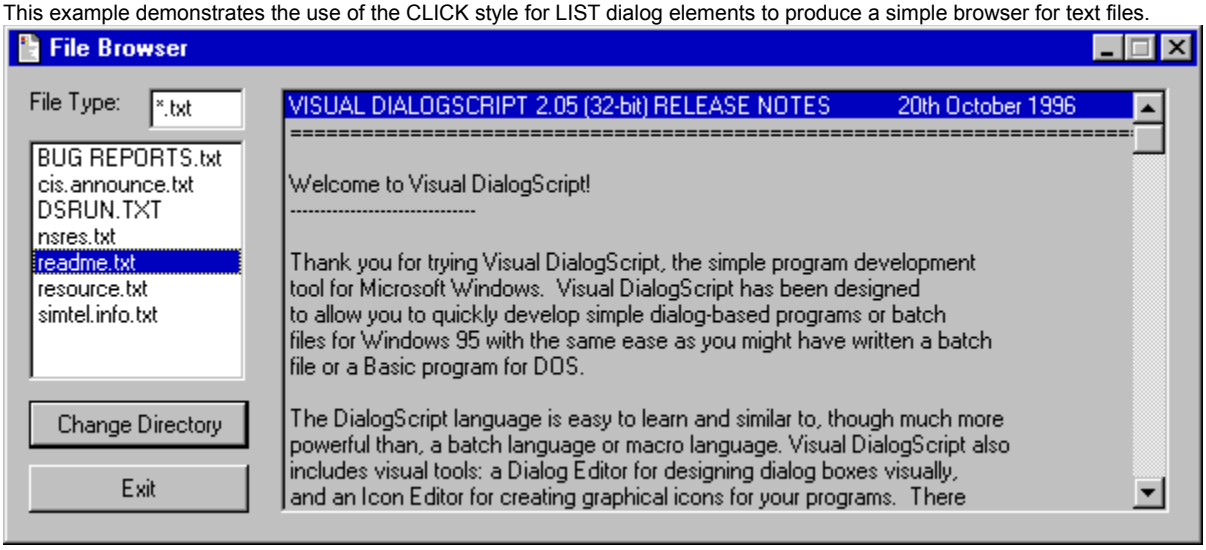
Visual DialogScript's FILE COPY command can only be used to copy a single file. This example shows how to copy a list of files generated using a wildcard, and incorporates a warning against overwriting a file in the target directory that is newer than the one in the source.

```
%S = c:\source
%T = c:\target
list 1,create
rem create list of files to be copied
list 1,filelist,%S\*.*
repeat
  %N = @file(@next(1))
  rem %N contains full path of file
  if %N
    %A = 1
    %F = @name(%N).@ext(%N)
    if @file(%T\%F)
      rem if file exists in target
      rem check if the copy is newer
      if @greater(@file(%T\%F,T),@file(%N,T))
        %A = @ask(File %F in target directory is newer. Overwrite?)
      end
    end
    if %A
      file copy,%N,%T\%F
      wait 1
    end
  end
until @null(%N)
list 1,close
exit
```

Note

File browser

This example demonstrates the use of the CLICK style for LIST dialog elements to produce a simple browser for text files.

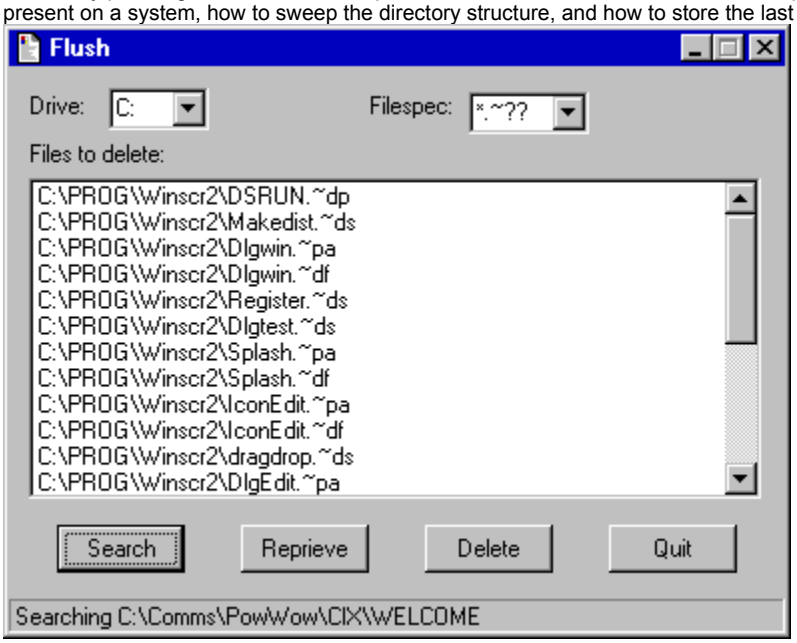



```
title File Browser
DIALOG CREATE,File Browser,-1,0,600,241,-
  STYLE(Gray;;;;BACKGRND),-
  TEXT(TEXT1;10;10;;;File Type:),-
  EDIT(Type;10;70;48;;*.txt;EXIT),-
  LIST(FileList;36;10;108;120;CLICK;SORTED),-
  BUTTON(ChDir;166;10;110;;Change Directory),-
  BUTTON(Exit;198;10;110),-
  LIST(Viewer;10;136;444;212;GRAY)
:getfiles
  list FileList,clear
  list FileList,filelist,@dlgtext(Type)
:evloop
  wait event
  goto @event()
:FileListCLICK
  list viewer,loadfile,@item(filelist)
  goto evloop
:TypeEXIT
  goto getfiles
:ChDirBUTTON
  %D = @dirdlg()
  if @ok() then
    directory change,%D
    goto getfiles
  end
  goto evloop
:ExitBUTTON
:CLOSE
  exit
```

Note

File deleter

This example script is a utility which will search through all the directories on a selected drive looking for files that match a particular file specification. Files that it finds can be removed from the list by selecting them and pressing Reprieve; those that remain are deleted by pressing Delete. The example demonstrates a number of useful techniques, including how to detect what drives are present on a system, how to sweep the directory structure, and how to store the last four combo box selections in an INI file.




```
Title Flush
rem delete all files on a drive with a specified extension
DIALOG CREATE,Flush,-1,0,394,288,DLGTYPE(SAVEPOS),TEXT(T1;12;10;;;Drive:),-
  COMBO(Drv;10;50;50;;;LIST),TEXT(T2;12;180;;;Filespec:),COMBO(Spec;12;230;60;;;Exit),-
  TEXT(T3;36;10;;;Files to delete:),LIST(LB;55;10;366;160),BUTTON(Search;228;24),-
  BUTTON(Reprieve;228;116),BUTTON(Delete;228;208),BUTTON(Quit;228;300),-
  STATUS(SP)
dialog set,Spec,@iniread(Flush,LastSpec,*.tmp)
%C = 1
repeat
  %E = @iniread(Flush,FileSpec%C)
  if %E
    list Spec,add,%E
  end
  %C = @succ(%C)
until @equal(%C,5)
dialog set,SP,Checking available drives...
gosub builddrv
%C = @match(Drv,@iniread(Flush,LastDrv,C:))
:evloop
  dialog set,SP,@count(LB) files selected
  wait event
  goto @event()
:SearchButton
  dialog cursor,wait
  list 1,create
  list 2,create
  %D = @dlgtext(Drv)
  list 1,filelist,%D,*
  repeat
    dialog set,sp,Searching %D
    list 2,filelist,%D\@dlgtext(Spec),srha
    repeat
      %F = @next(2)
      if %F
        list LB,add,%F
      end
    until @null(%F)
```

```
      list 2,clear
      %D = @next(1)
    until @null(%D)
    list 1,close
    list 2,close
    dialog cursor
    goto evloop
  :ReprieveButton
    DIALOG CLEARSEL,LB
    goto evloop
  :DeleteButton
    list LB,seek,0
    repeat
      %F = @item(LB)
      if @not(@null(%F))
        file delete,%F
        list LB,delete
      end
    until @null(%F)
    dialog clear,LB
    dialog cursor
    goto evloop
  :QuitButton
  :Close
    inifile write,Flush,LastDrv,@dlgtext(Drv)
    inifile write,Flush,LastSpec,@dlgtext(Spec)
    %C = 0
    if @not(@equal(%C,@count(Spec)))
      list Spec,seek,0
      repeat
        %C = @succ(%C)
        inifile write,Flush,FileSpec%C,@next(Spec)
      until @equal(%C,@count(Spec))@equal(%C,5)
    end
    exit
  :SpecExit
    %X = @dlgtext(spec)
    if @not(@match(spec,%X))
      if @greater(@count(spec),0)
        list spec,seek,0
      end
      list spec,insert,%X
      list spec,seek,0
    end
    goto evloop
  :builddrv
    dialog cursor,wait
    %C = 65
    repeat
      %S = @volinfo(@chr(%C),S)
      if @ok()
        list Drv,add,@chr(%C):
      end
      %C = @succ(%C)
    until @equal(%C,90)
    dialog cursor
    exit
```

Note

Floating point functions

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

@FADD addition

@FATN arctangent

@FCOS cosine

@FDIV division

@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
@CHR	@CLICK	@COUNT
@CR	@CTRL	@CURDIR
@DATETIME	@DDEITEM	@DIFF
@DIRDLG	@DIV	@DLGTEXT
@ENV	@EQUAL	@ESC
@EVENT	@EXT	
Floating point functions		
@FILE	@FILEDLG	@FORMAT
@GREATER		
@HEX		
@INDEX	@INIREAD	@INPUT
@ITEM		
@KEY		
@LEN	@LOWER	
@MATCH	@MCI	@MSGBOX
@NAME	@NEXT	@NOT
@NULL	@NUMERIC	
@OK		
@PATH	@POS	@PRED
@PROD		
@QUERY		
@REGREAD	@RETCODE	
@SENDMSG	@SHIFT	@SHORTNAME
@STRDEL	@STRINS	@SUBSTR
@SUCC	@SUM	@SYSINFO
@TAB	@TRIM	
@UPPER		
@VERINFO	@VOLINFO	
@WINACTIVE	@WINATPOINT	@WINCLASS
@WINDIR	@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 16-bit script programs will write to the configuration file DS16.INI and the text 'Program\<program name>\' will be prepended to every <section name> used in an attempt to make it unique. 32-bit 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 (32-bit only)

Syntax:

LINK CREATE, <filename>, <link path>, <link name>

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.

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 <list>, <command>, <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.

OK:

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

Example:

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

See also:

@COUNT @INDEX @ITEM @MATCH @NEXT

LIST(<name>;<top>;<left>;<width>;<height>;<style>;<style>})

This dialog element creates a list box at the position and size specified.

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

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.

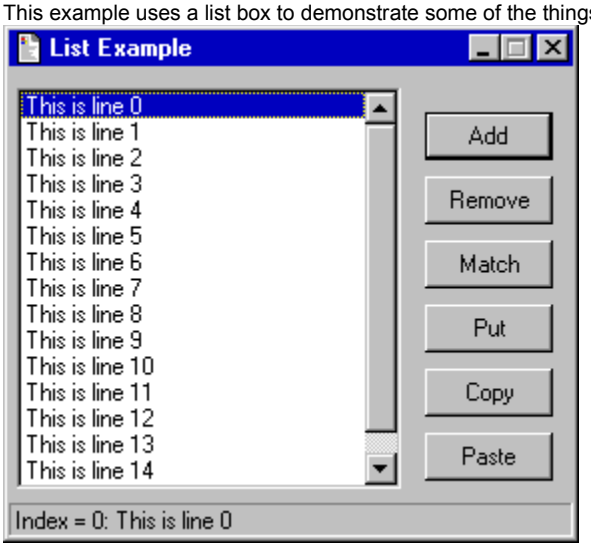
This is an example of a label:

```
:LABEL
```

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

List

This example uses a list box to demonstrate some of the things you can do with lists.




```
title List test
DIALOG CREATE,List Example,-1,0,288,240,LIST(LB;10;4;192;200;CLICK),-
  STATUS(T1),BUTTON(Add;22;208),BUTTON(Remove;54;208),-
  BUTTON(Match;86;208),BUTTON(Put;118;208),-
  BUTTON(Copy;150;208),BUTTON(Paste;182;208)
%C = 0
repeat
  list LB,add,This is line %C
  %C = @sum(%C,1)
until @equal(%C,16)
:loop
  %P = @index(LB)
  if @equal(%P,-1)
    dialog set,t1,Index = %P
  else
    dialog set,t1,Index = %P: @item(LB)
  end
wait event
goto @event()
:addbutton
%A = @input(Enter text to add:)
list LB,add,%A
goto loop
:putbutton
%A = @input(Enter text to put:)
list LB,put,%A
goto loop
:removebutton
list LB,delete
goto loop
:matchbutton
%A = @input(Enter text to match:,%A)
if @not(@match(LB,%A))
  warn No match found
end
goto loop
:copybutton
list lb,copy
goto loop
:pastebutton
list lb,paste
goto loop
:lbclick
goto loop
:close
```

Note

MCI Test

This script lets you send Multimedia Control Interface (MCI) commands, and displays any error messages that may result. It is a good tool for trying out MCI commands prior to incorporating them into your own scripts.

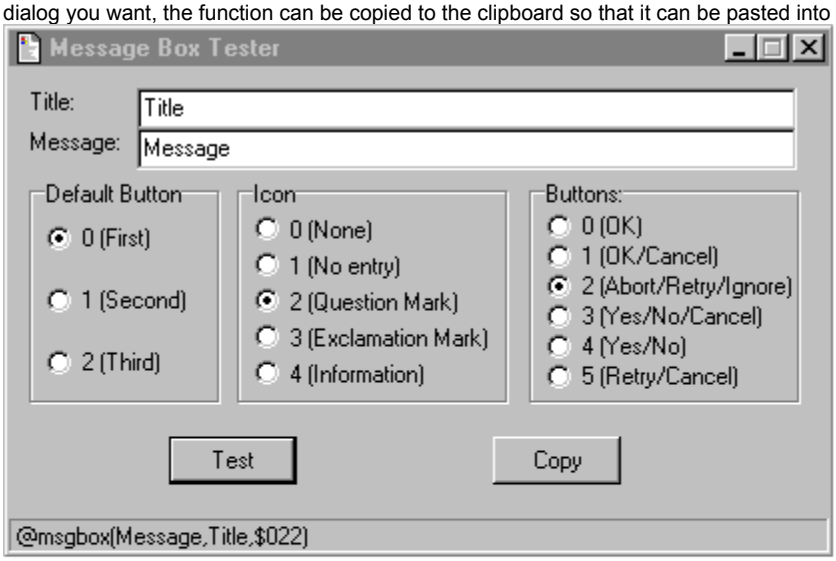
```
title Test MCI
%2 = open @windir()\MEDIA\THEMIC~1.WAV alias sound
%3 = play sound
%4 = close sound
:loop
%C = @input("Enter MCI command (Cancel quits):",%2)
if @not(@ok())
  exit
end
if @null(%C)
  exit
end
%R = @MCI(%C)
if @ok()
  info MCI Response: %R
else
  warn MCI error: %R
end
shift
goto loop
```

See also

Using MCI Note

Message Box Tester

This example script can be used to try out the different options of the @MSGBOX function. Once you have chosen the message dialog you want, the function can be copied to the clipboard so that it can be pasted into the Visual DialogScript editor.




```
title Message Box Tester
DIALOG CREATE,Message Box Tester,-1,0,416,248,-
  TEXT(TEXT1;10;10;;;Title:),-
  EDIT(Title;10;64;330;;Title),-
  TEXT(TEXT2;30;10;;;Message:),-
  EDIT(Message;30;64;330;;Message),-
  RADIO(Default;56;10;96;112;Default Button;0 (First)|1 (Second)|2 (Third);0 (First)),-
  RADIO(Icon;56;114;136;112;Icon;0 (None)|1 (No entry)|2 (Question Mark)|3 (Exclamation
Mark)|4 (Information);0 (None)),-
  RADIO(Buttons;56;260;136;112;Buttons:;0 (OK)|1 (OK/Cancel)|2 (Abort/Retry/Ignore)|3
(Yes/No/Cancel)|4 (Yes/No)|5 (Retry/Cancel);0 (OK)),-
  BUTTON(Test;184;80),-
  BUTTON(Copy;184;242),-
  STATUS(STATUS)
:evloop
  wait event
  %M = @dlg(message)
  %T = @dlg(title)
  %V = "$"@substr(@dlg(default),1,1)@substr(@dlg(icon),1,1)@substr(@dlg(buttons),1,1)
  dialog set,status,"@msgbox("%M","%T","%V")"
  goto @event()
:TestBUTTON
  %R = @msgbox(%M,%T,%V)
  dialog set,status,Result was %R
  goto evloop
:CopyBUTTON
  clipboard set,"@msgbox("%M","%T","%V")"
  dialog set,status,Function copied to clipboard
  goto evloop
:CLOSE
  exit
```

Note

Mouse click example

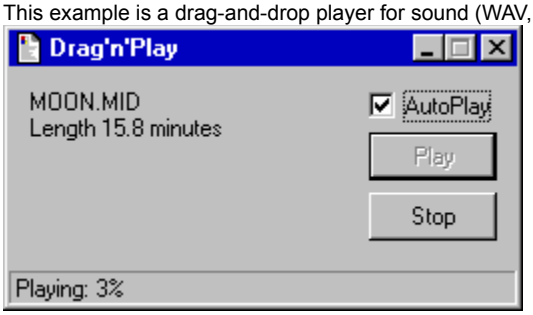
This script illustrates automating an application (in this case the Visual DialogScript Help) using mouse clicks.

```
shell open,ds.hlp
wait 1
if @winexists(##32770)
  rem open help topic by double-clicking
  window click,##32770,40,124
  window click,##32770,40,124
  wait 1
  window click,##32770,65,145
  window click,##32770,65,145
  wait 1
  window click,##32770,81,145
  window click,##32770,81,145
  wait 2
else
  rem WinHelp 3
  if @winexists(#MS_WINDOC)
    window click,#MS_WINDOC,86,130
    wait 1
    window click,#MS_WINDOC,72,112
    wait 1
  end
end
if @not(@winexists(#MS_WINDOC))
  warn Didn't open help file!
  exit
end
rem relative position of >> button
%X = 280
%Y = 53
parse "%U;%V",@winpos(#MS_WINDOC,LT)
rem calculate absolute position of >> button
%U = @sum(%U,%X)
%V = @sum(%V,%Y)
repeat
  wait 1
  window click,#MS_WINDOC,280,53
  %C = @pred(%C)
until @null(@wintext(@winatpoint(%U,%V)))
rem @wintext returns null when button disabled
exit
```

Note

Multimedia Player

This example is a drag-and-drop player for sound (WAV, MID) and video (AVI) files. You drag the files to the window to play them.




```
  title Drag'n'Play
  rem version 2 - uses MCI commands
  dialog create,Drag'n'Play,100,100,260,124,DLGTYPE(DRAGDROP),-
    TEXT(T1;10;10;160;84),CHECK(CB;10;180;;;AutoPlay;1),-
    BUTTON(Play;32;180),BUTTON(Stop;62;180),-
    STATUS(SP;Drag files to the window to play them)
  dialog disable,Play
  dialog disable,Stop
  if %2
    %F = %2
    goto file
  end
:loop
  wait event
  goto @event()
:close
  goto closemedia
:dragdrop
  gosub closemedia
  list 1,create
  list 1,dropfiles
  %F = @next(1)
  list 1,close
:file
  if @equal(@ext(%F),WAV)@equal(@ext(%F),MID)@equal(@ext(%F),AVI)
    dialog clear,SP
    %X = @name(%F).@ext(%F)@CR()
    %R = @mci(open %F alias media wait)
    if @ok()
      %R = @mci(set media time format ms))
      if @equal(@ext(%F),WAV)
        %R = @mci(status media channels)
        if @equal(%R,1)
          %X = %XMono@CR()
        else
          %X = %XStereo@CR()
        end
        %R = @mci(status media bitspersample)
        %X = %X%R bits per sample@CR()
        %R = @mci(status media samplespersec)
        %X = %X@fdiv(%R,1000)kHz sampling rate@CR()
      end
      %T = @mci(status media length)
      if @greater(60000,%T)
        %R = @format(@fdiv(%T,1000),4.1) seconds
      else
        %R = @format(@fdiv(%T,60000),4.1) minutes
      end
      %X = %XLength %R
      dialog set,T1,%X
      if @equal(@ext(%F),AVI)
         window position,#AVIWnd,100,360
        %R = @mci(window media state show wait)
      end
      if @dlgtext(CB)
```

```
        goto playmedia
      else
        dialog enable,Play
      end
    end
  end
  goto loop
:playbutton
  goto playmedia
:stopbutton
  %R = @mci(stop media)
  dialog clear,SP
  dialog enable,Play
  dialog disable,Stop
  goto loop
:playmedia
  dialog enable,Stop
  dialog disable,Play
    %R = @mci(seek media to start)
    dialog set,SP,Playing: 0%
    %R = @mci(play media)
:ploop
    %P = %R
    wait 2,event
    goto @event()
:timer
    %R = @mci(status media position)
    dialog set,SP,Playing: @div(@prod(%R,100),%T)%
    if @not(@equal(%R,%P))
      goto ploop
    end
    dialog clear,SP
  else
    warn MCI error: %R
  end
  dialog enable,Play
  dialog disable,Stop
  goto loop
:closemedia
  %R = @mci(close media)
  exit
```

Note

OPTION

Syntax:

OPTION ERRORTRAP, <label>

OPTION FIELDSEP, <separator character>

OPTION FILENAMES, <SHORT|LONG> **(32-bit only)**

OPTION PRIORITY, <IDLE|NORMAL|HIGH|REALTIME> **(32-bit only)**

OPTION REGBUF, <buffer-size> **(32-bit only)**

OPTION SCALE, <pixels-per-inch> **(not Personal Edition)**

OPTION SKDELAY, <interval>

OPTION SLEEPTIME, <interval> **(32-bit only)**

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 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 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 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 standard 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
```

See also:

What is Visual DialogScript?

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 editor for visually designing dialog boxes, and a Window Spy for finding out information about other applications' windows. There is also a Dialog 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. With the Professional version you can create an EXE file which can then be run just like any other Windows application. Registered users can distribute the EXE files created by Visual DialogScript Professional 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.

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 Address Book example

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

POPUP(<item>|{<item>;...}

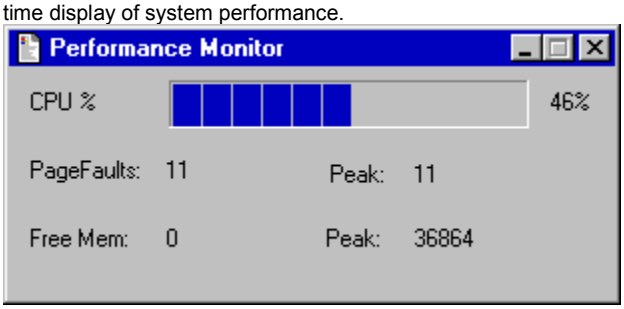
This dialog element defines a popup menu for the dialog window. When a menu item is selected, it causes an <item>MENU event. Note that the popup menu does not have a name, and the item list is a single (and the only) parameter, so the items are separated by vertical bars not semicolons.

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)..

Performance Monitor Example

This example program shows how the dynamic performance statistics in the Windows 95 registry can be accessed to provide a real-time display of system performance.

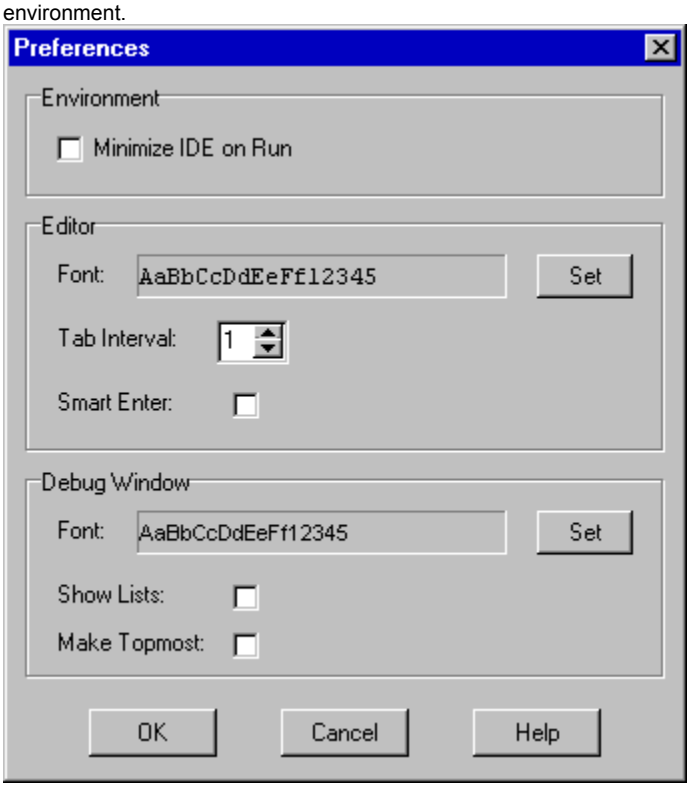



```
option fieldsep,","

rem turn on CPU monitoring
%7 = @regread(STATS,PerfStats\StartStat,KERNEL\CPUUsage)
%8 = 0
%9 = 0
%S = @regread(STATS,PerfStats\StatData,VMM\cPageFaults)
gosub compute
%P = %S
DIALOG CREATE,Performance Monitor,-1,0,309,121,DLGTYPE(ONTOP),TEXT(TEXT1;10;10;;;"CPU
%"),-
  PROGRESS(PROGRESS1;6;80),TEXT(TEXT2;44;10;;;PageFaults:),TEXT(TEXT3;44;78;;;Text),-
  TEXT(TEXT4;78;10;;;Free Mem:),TEXT(TEXT5;78;78;;;Text),TEXT(TEXT6;10;270;;;Text),-
  TEXT(TEXT7;46;160;28;;Peak:),TEXT(TEXT8;78;158;;;Peak:),TEXT(TEXT9;46;202;;15;Text),-
  TEXT(TEXT10;78;202;;;Text)
:timer
wait event,1
%S = @regread(STATS,PerfStats\StatData,KERNEL\CPUUsage)
gosub compute
dialog set,progress1,%S
dialog set,text6,%S"%"
%S = @regread(STATS,PerfStats\StatData,VMM\cPageFaults)
gosub compute
%Q = @diff(%S,%P)
dialog set,Text3,%Q
if @greater(%Q,%8)
  dialog set,Text9,%Q
  %8 = %Q
end
%P = %S
%S = @regread(STATS,PerfStats\StatData,VMM\cpgFree)
gosub compute
dialog set,Text5,%S
if @greater(%S,%9)
  dialog set,Text10,%S
  %9 = %S
end
%R = %S
goto @event()
:close
rem turn off CPU performance monitoring
%7 = @regread(STATS,PerfStats\StopStat,KERNEL\CPUUsage)
exit
:compute
  rem turn the 4 bytes read into a single integer value
  parse "%1;%2;%3;%4",%S
  %S = @sum(@sum(%1,@prod(256,%2)),@prod(65536,%3))
  exit
```

Note

Preferences

From the Options menu you can open the Preferences dialog which lets you set your operating preferences for the development environment.



Click on a dialog control to learn what it does.

RADIO(<name>;<top>;<left>;<width>;<height>;<caption>;<value list>;<value>;<style>{; <style>})

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.

REGISTRY

Syntax:

REGISTRY DELETE, <root key>, <subkey>

REGISTRY WRITE, <root key>, <subkey>, <name>, <value>

Description:

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

REGISTRY DELETE deletes the specified key and all of its subkeys.

REGISTRY WRITE sets the value of the specified key. If the subkey does not exist it is created.

<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\JM Tech\DialogScript\2.0\UserScripts in HKEY_CURRENT_USER **(32-bit only)**.

<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 **(32-bit only)**.

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

Note **(32-bit only)**: 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.

Note **(16-bit only)**: In the Windows 3.1 Registry only the ROOT key is supported. Named key values cannot be accessed, even when running on a 32-bit system, as the function is not supported by the 16-bit Windows API. Therefore the <name> parameter is discarded in this version.

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 Example 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.

(32-bit only) 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

Registry Information

This script displays information about the file associations that are set up in the Windows registry.

```
title Registry Information
%X = @input(Enter extension,.doc)
if @ok()
  %A = @regread(ROOT,%X)
  if @null(%A)
    info No registry association for %X
  else
    %B = @regread(ROOT,%A)
    %D = @regread(ROOT,%A\shell\open\command)
    %E = Registry association for %X@CR()ProgID: %A@CR()Description: %B@CR()Shell Open:%D
    info %E
  end
end
```

Note

SHELL

Syntax:

SHELL <operation>, <filename>, <parameters>, <start-dir> {, WAIT, <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.

(32-bit only) 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.

OK:

Set to false if the command fails.

Example:

```
SHELL "",My Data.LNK

SHELL open,http://www.jm-tech.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:

STATUS(<name>;<text>;<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

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

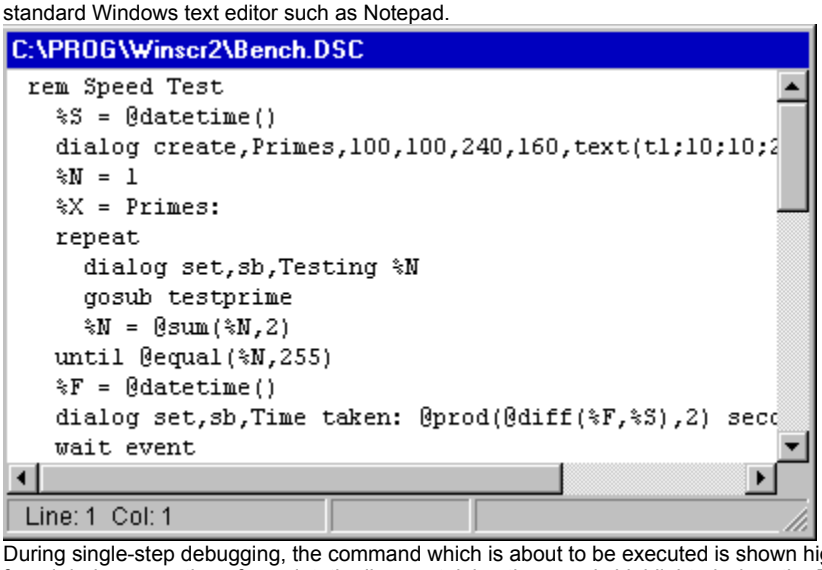
Metrics is the term used by Microsoft to describe the characteristics of an output device, such as the display. One factor that may affect your scripts is the number of pixels per inch used by the display driver.. Windows displays typically use either 96ppi or 120ppi, the latter when the Large Fonts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

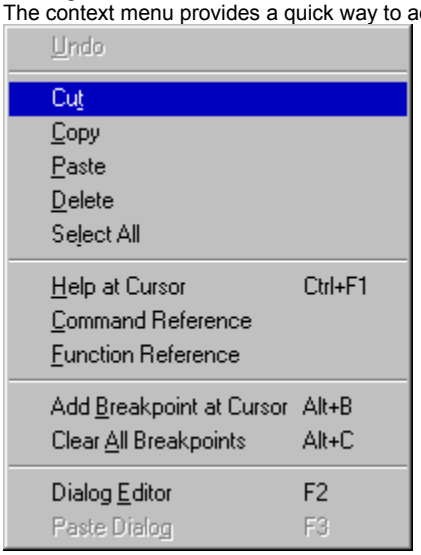
Script Window

The script window is used for editing and debugging scripts. The window supports all the editing options and shortcut keys of a standard Windows text editor such as Notepad.



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.

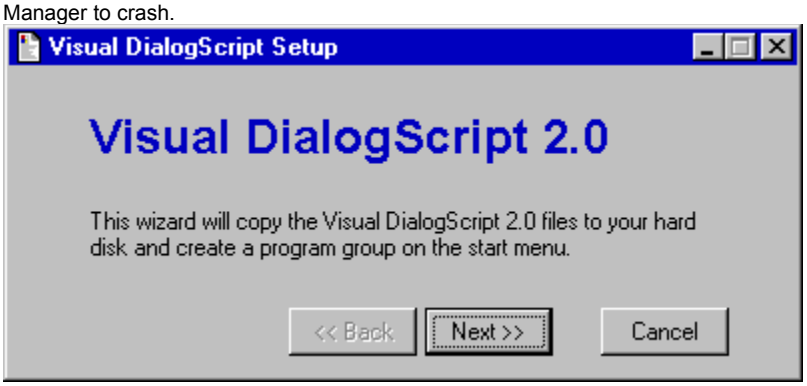
You can set breakpoints to halt the execution of a script by placing the cursor in the line at which you want execution to stop, right-clicking the mouse and selecting Add Breakpoint at Cursor from the menu. If you place the cursor on a breakpoint the option changes to Clear Breakpoint at Cursor. You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

You can also call up the Dialog Editor 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.

Setup Wizard example

The Setup Wizard which you used to install Visual DialogScript to your hard disk is a DialogScript program. This is how it works. The controls for each page lay on top of each other. A page count, %P, is maintained which is used when going from one page to the next to hide the current page's controls and show the next one. Note that the contents of all controls are accessible, even when they are not visible.

This example also shows how to create shortcut groups on the Windows 95 start menu. Note that it is important to put quotes round commas that appear in the DDE commands to Program Manager otherwise they will be interpreted as parameter separators by the DialogScript interpreter and the full command will not be sent. Invalid DDE commands can cause the Windows 3.1 Program Manager to crash.




```
  title Visual DialogScript Setup Wizard
  if @not(@file(PACKING.LST))
    warn Unable to run setup@CR()Cannot find file PACKING.LST
    exit
  end
  %T = Visual DialogScript Setup
  %U = Visual DialogScript 2.0
  %V = This wizard will copy the %U files to your hard disk and create a program group on
the start menu.
  %X = Press Finish to install %U to your hard disk.
  rem %D is default directory
  %D = C:\Program Files\Visual DialogScript
  dialog create,%T,-1,0,400,160,style(Title;Arial;18;B;;Blue),-
    text(title;20;40;340;120;%U;Title),text(text1;70;40;320;40;%V),-
    text(text2;30;40;;;Install into this directory:),-
    edit(eddir;52;40;288;;%D),button(brwse;52;332;24;21;...),-
    button(bcdir;82;236;120;22;Use current directory),-
    check(cbmenu;30;40;320;;Create start menu shortcuts?;1),-
    check(cbass;52;40;320;;Create file type association?;1),-
    text(text4;30;40;320;;%X),progress(prog;64;40;320;32;0),-
    button(Back;120;140;;;<< Back),button(Next;120;208;;;Next >>),-
    button(Finish;120;296;;;Cancel)
  dialog disable,Back
  rem %Z = no. of pages
  %Z = 4
  rem %P = current page no.
  %P = %Z
  %A = HIDE
  repeat
    rem hide all but first page controls
    gosub pagectrl
    %P = @pred(%P)
  until @equal(%P,1)
:evloop
  wait event
  goto @event()
:nextbutton
  %A = HIDE
  gosub pagectrl
  %P = @succ(%P)
  %A = show
  gosub pagectrl
  if @greater(%P,1)
    dialog enable,Back
```

```
  end
  if @equal(%P,%Z)
    dialog disable,Next
    dialog set,Finish,Finish
  end
  goto evloop
:backbutton
  %A = HIDE
  gosub pagectrl
  %P = @pred(%P)
  %A = show
  gosub pagectrl
  if @equal(%P,1)
    dialog disable,Back
  end
  if @greater(%Z,%P)
    dialog enable,Next
    dialog set,Finish,Cancel
  end
  goto evloop
:finishbutton
  if @equal(%P,%Z)
    gosub install
    exit
  else
    if @ask(Are you sure you want to cancel %T?)
      exit
    end
  end
  goto evloop
:brwsebutton
  %D = @dirdlg()
  if @ok()
    dialog set,eddir,%D
  end
  goto evloop
:bcdirbutton
  %D = @curdir()
  dialog set,eddir,%D
  goto evloop
:pagectrl
  if @equal(%P,4)
    dialog %A,text4
    dialog %A,prog
  end
  if @equal(%P,3)
    dialog %A,cbmenu
    dialog %A,cbass
  end
  if @equal(%P,2)
    dialog %A,text2
    dialog %A,eddir
    dialog %A,brwse
    dialog %A,bcdir
  end
  if @equal(%P,1)
    dialog %A,text1
    dialog %A,title
  end
  exit
:install
  rem load the list of files to install
  list 1,create
  list 1,loadfile,PACKING.LST
  %Z = @sum(@count(1),4)
  %P = 100
  %D = @dlgtext(eddir)
  if @not(@equal(%D,@curdir()))
    rem create the output directory
    gosub showprog
    dialog set,text4,Creating directory %D
```

```
    if @file(%D,D)
      if @not(@ask(Directory %D already exists.@CR()Files may be overwritten. Continue?))
        exit
      end
    else
      directory create,%D
      if @not(@file(%D,D))
        beep
        warn Could not create %D
        exit
      end
    end
    repeat
      %F = @next(1)
      if %F
        %P = @sum(%P,100)
        gosub showprog
        dialog set,text4,Copying file %F
        file copy,%F,%D\%F
      end
    until @null(%F)
    list 1,close
  end
  %P = @prod(@diff(%Z,2),100)
  gosub showprog
  wait 1
  if @dlgtext(cbmenu)
    dialog set,text4,Create the start menu shortcuts
    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")]
    end
    if @not(@ok())
      warn Setup failed to create start menu shortcuts
    end
    DDE TERMINATE
  end
  wait 1
  %P = @prod(@diff(%Z,1),100)
  gosub showprog
  wait 1
  if @dlgtext(cbass)
    dialog set,text4,Create the file type association
    %W = Visual DialogScript.Script
    registry write,root,.dsc,,%W
    registry write,root,%W,,DialogScript File
    registry write,root,%W\shell\open\command,,@chr(34)%D"\ds.exe %1"@chr(34)
  end
  wait 1
  %P = @prod(%Z,100)
  gosub showprog
  wait 1
  if @file(%D\README.TXT)
    if @ask(Would you like to view the README.TXT file?)
      run notepad README.TXT,wait
    end
  end
  info %T finished.
  exit
:showprog
  dialog set,prog,@div(%P,%Z)
  exit
```

Note

Setup script example

This simple setup script uses the LINK command to create icons in the Start Menu and on the Windows desktop.

```
title Personal VDS '97 Setup
option scale,96
DIALOG CREATE,Personal VDS Setup,-1,0,276,240,-
  STYLE(TITLE;MS Sans Serif;18;B;;LTBLUE),-
  TEXT(TEXT1;10;16;;;Personal VDS Setup;TITLE),-
  TEXT(TEXT2;42;10;248;32;This program will set up Personal VDS '97 on your hard disk.),-
  TEXT(TEXT3;74;10;250;32;"Un-check the box against any item you prefer to perform
manually, then press OK."),-
  CHECK(cbMenu;110;12;256;;Create Start Menu icon?;1),-
  CHECK(cbDsktop;130;12;256;;Create desktop shortcut?;1),-
  CHECK(cbFileAss;150;12;248;;Create file association (recommended)?;1),-
  BUTTON(OK;180;52),-
  BUTTON(Cancel;180;160),-
  STATUS(STATUS)
goto evloop
  %D = @path(%0)
  if @equal(\,@substr(%D,@len(%D)))
    %D = @strdel(%D,@len(%D))
  end
  if @not(@file(%D\PVDS.EXE))
    warn Setup cannot find PVDS.EXE. Please run Setup from the directory this file is in.
    exit
  end
:evloop
  wait event
  goto @event()
:OKBUTTON
  if @dlgtext(cbMenu)
    dialog set,status,Creating Start Menu icon
    wait 1
    rem get location of Start Menu folders
    link create,%D\PVDS.EXE,%E,Personal VDS '97
    %E = @regread(CURUSER,Software\Microsoft\Windows\CurrentVersion\Explorer\Shell
Folders,Start Menu)
    link create,%D\PVDS.EXE,%E,Personal VDS '97
    if @not(@ok())
      warn Couldn't create Start Menu icon
    end
  end
  if @dlgtext(cbDsktop)
    dialog set,status,Creating desktop shortcut
    wait 1
    rem get location of desktop folder
    %E = @regread(CURUSER,Software\Microsoft\Windows\CurrentVersion\Explorer\Shell
Folders,Desktop)
    link create,%D\PVDS.EXE,%E,Personal VDS '97
    if @not(@ok())
      warn Couldn't create desktop shortcut
    end
  end
  if @dlgtext(cbFileAss)
    dialog set,text4,Create the file type association
    wait 1
    %W = Visual DialogScript.Script
    registry write,root,.dsc,,%W
    registry write,root,%W,,Visual DialogScript Script
    registry write,root,%W\shell\open\command,,@chr(34)%D\PVDS.EXE@chr(34)
@chr(34)"%1"@chr(34)
  end
  dialog set,status,Setup completed
  beep
  wait 4
:CancelBUTTON
:CLOSE
  exit
```

Note

Startup example

This example creates a startup menu with two buttons in the center of the screen, which you can use to select which programs are run automatically at startup.


```
title Startup Menu
dialog create,Startup,-1,20,160,142,STYLE(ONTOP),-
  TEXT(T1;10;20;;;Choose a startup option:),-
  BUTTON(Standard;36;20;120),BUTTON(Quick;76;20;120),STATUS(SB)
%C = 10
repeat
  dialog set,SB,Time remaining: %C seconds
  %C = @pred(%C)
  %D = 400
  wait 1
  %E = @event()
  if @zero(%C)
    rem default
    %E = STANDARDBUTTON
  end
until %E
:startup
wait 1
:
: run stuff that should always be run
:
if @equal(%E,STANDARDBUTTON)
:
: run stuff that won't be run if you
: press the Quick button
:
end
exit
```

Note

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)"
```

TASKICON(<name>;<filename>,<caption>) **(32-bit only)**

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 <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.

TEXT(<name>;<top>;<left>;<width>;<height>;<text>;<style>)

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

TITLE

Syntax:

TITLE <string>

Description:

Sets the title of the script program. This will appear in message and input boxes and on the Windows 95 Task Bar. If no title is set, the default name of Visual DialogScript is used.

OK:

Leaves unchanged.

Example:

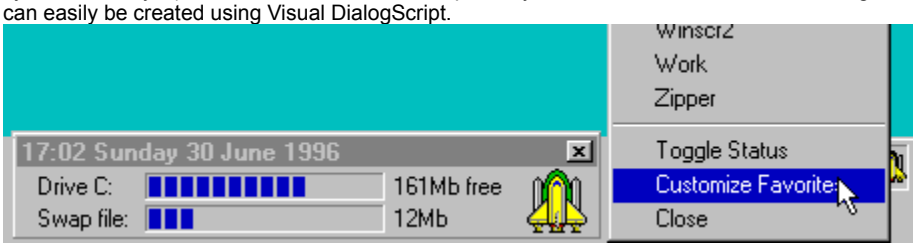
```
TITLE Daily Backup
```

See also:

DIALOG

Task Launcher

This handy little utility uses the TASKICON dialog element to create a popup menu of the shortcuts on the Windows 95 desktop, which can be used to launch the shortcuts even when they are obscured by other windows. It also contains a list of shortcuts in the Favorites folder, which can be used to add quick-access menu options to your favorite applications. The task launcher also displays the time and date, the amount of free space on C: and the swap file size (a useful indicator of how overcommitted the system memory is) in a small window that can optionally be hidden from the menu. It is a good example of the sort of utility that can easily be created using Visual DialogScript.




```
title TaskLauncher
:reload
  %D = @windir()\Desktop
  %G = @windir()\Favorites
  list 1,create,sorted
  %P = %G\*.lnk
  gosub buildmenu
  %P = %D\*.lnk
  gosub buildmenu
  if @not(@zero(@len(%M)))
    %M = @substr(%M,2,1024)|
  end
  %M = %MToggle Status|Customize Favorites|Close
  list 1,close
DIALOG CREATE,Task Launcher,-1,0,296,36,DLGTYPE(SavePos;SmallCap),-
  TASKICON(Task),BITMAP(BITMAP1;0;254;;;%0),POPUP(%M),-
  TEXT(TEXT1;2;10;;;Drive C:),TEXT(TEXT2;18;10;;;Swap file:),-
  PROGRESS(PROG1;2;64;120;14),PROGRESS(PROG2;18;64;120;14),-
  TEXT(TEXT3;2;188),TEXT(TEXT4;18;188)
  wait 1
  %M =
  %C = @volinfo(C,S)
  %4 =
  %V = 1
  %R =
  window Hide,TaskLauncher
:loop
  %T = @datetime(t dddddd)
  dialog title,%T
  dialog set,task,%T
  if %V
    gosub updatestatus
  end
  wait 10,event
  %E = @event()
if @equal(%E,TIMER)
  if %R
    if @not(@winexists(Favorites))
      rem start a new instance and close this one
      run TaskLauncher.exe
      exit
    end
  end
  goto loop
end
if @equal(%E,TaskIcon)
  dialog popup
  goto loop
end
if @equal(%E,CLOSE)
  exit
end
if @equal(%E,closemenu)
```

```
  exit
end
if @equal(%E,Toggle StatusMENU)
  if %V
    window hide,%T
    %V =
  else
    window normal,%T
    %V = 1
  end
  goto loop
end
if @equal(%E,Customize FavoritesMENU)
  shell open,@windir()\Favorites
  %R = reload
  goto loop
end
%L = @diff(@len(%E),4)
%F = @substr(%E,1,%L)
if @file(%G\%F.LNK)
  shell ,%G\%F.LNK
else
  shell ,%D\%F.LNK
end
goto loop
:buildmenu
  list 1,filelist,%P
  if @not(@zero(@count(1)))
    repeat
      %F = @next(1)
      if %F
        %M = %M|@name(%F)
      end
    until @null(%F)
    %M = %M|-
  end
  list 1,clear
  exit
:updatestatus
  %1 = @volinfo(C,F)
  %2 = @div(@sum(@file(@windir()\win386.swp,z),524288),1048576)
  dialog set,text3,@div(%1,1024)Mb free
  dialog set,prog1,@div(@prod(@diff(%C,%1),100),%C)
  dialog set,text4,%2Mb
  dialog set,prog2,@div(@prod(%2,100),64)
  exit
```

Note

TaskBar Icon Demo

This example demonstrates a dialog that has an associated task bar icon, and how to disable and enable the icon, change its tooltip caption text and respond when the icon is clicked.

```
title TaskIcon Test
dialog create,TaskIcon Test,-1,0,100,120,-
  TASKICON(Task;;Task Icon),-
  BUTTON(Enable;20;20),BUTTON(Disable;50;20),BUTTON(SetText;80;20)
:loop
wait event
%E = @event()
if @equal(%E,CLOSE)
  exit
end
if @equal(%E,EnableButton)
  dialog enable,Task
end
if @equal(%E,DisableButton)
  dialog disable,Task
end
if @equal(%E,SetTextButton)
  dialog set,Task,@input(Enter new icon text)
end
if @equal(%E,TaskIcon)
  info You clicked on the task bar icon!
end
goto loop
```

Note

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 <list>, ADD, <text>
LIST <list>, ASSIGN, <list2>
LIST <list>, CLEAR
LIST <list>, CLOSE
LIST <list>, COPY
LIST <list>, CREATE, {SORTED}
LIST <list>, DELETE
LIST <list>, INSERT
LIST <list>, PASTE
LIST <list>, PUT, <text>
LIST <list>, SEEK, <record number>
LIST <list>, WRITE, <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 1, SEEK, 3
LIST 1, WRITE, This is item 4 in the list
LIST 1, SAVEFILE, LIST.TXT
LIST 1, CLOSE
```

See also:

@COUNT @INDEX @ITEM @MATCH @NEXT LIST Data Lists Example

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

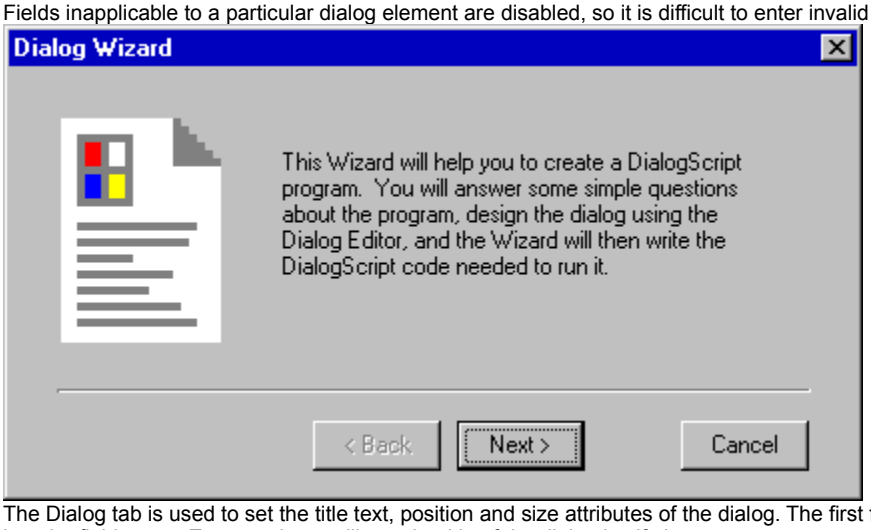
```
play cdaudio from 1 to 2 wait
```

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

Using the Dialog Editor

The Dialog Editor lets you design and edit dialogs interactively, so you can see the result of your changes straight away.

The Dialog Editor is a context sensitive, tabbed dialog which lets you create and edit dialog elements by entering values into fields. Fields inapplicable to a particular dialog element are disabled, so it is difficult to enter invalid information.



The Dialog tab is used to set the title text, position and size attributes of the dialog. The first field is for the title text. Type the text into the field, press **Enter** and you will see the title of the dialog itself change.
You can set the top, left, width and height attributes, respectively, in the next four fields. The text in the status line changes as you enter each field to tell you which attribute is being changed. A value of -1 for the top position means that the dialog will be created in the center of the screen. This attribute can be overridden using the SAVEPOS dialog type.

You can also change the size of the dialog being designed by dragging the edges using the mouse.

The dialog elements are listed next. To edit a particular dialog element you can:

double-click on it;

select it with the mouse and then click the Dialog Element tab;

select it and then choose Edit from the context menu, or press **F2**;

click on the corresponding control on the dialog itself.

You can add a new dialog element from the context menu, or by pressing **Ins.**

You can delete a dialog element from the context menu or by pressing **Del.**

You can change the position of a dialog element in the list from the context menu or by keying **Ctrl+U** (move up) or **Ctrl+D** (move down). The order that they are defined is important. When you use the dialog, the order in which controls that can accept input are visited when you press the **Tab** key will follow the order that the dialog elements appear in the list. The first button to be defined will be the default button.

You can also refresh the dialog (cause it to be recreated) from the context menu or by pressing **Ctrl+R**. It is sometimes necessary to refresh the dialog because some changes, for example changes to styles, are not immediately reflected by the dialog itself.

Variables

DialogScript allows you to have up to 35 variables. Variable names start with a percent symbol, followed by the character 1 to 9 or A to Z.

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 (in the 16-bit version the maximum length is 255 characters.) 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.

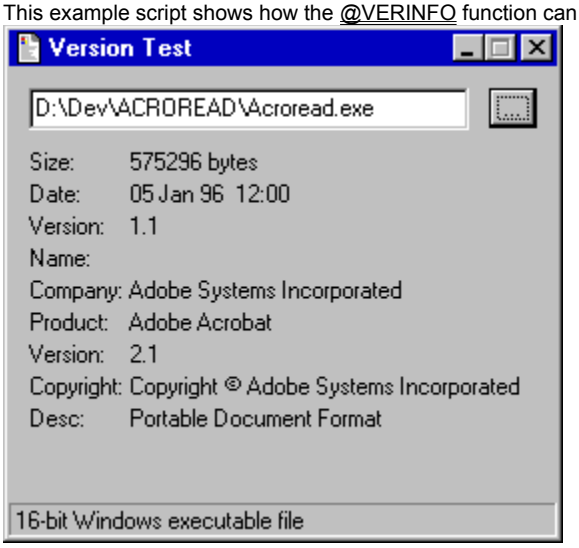
Note that the use of the percent symbol to identify variables, and the @ symbol to identify functions, means that you cannot generally use these symbols in a string of text. Double quotes can be used to enclose text containing these symbols and prevent them from being interpreted as variables or function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

```
INFO Email address:@tab()%N"@cix.compulink.co.uk"
```

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.

Version Info example

This example script shows how the @VERINFO function can be used to obtain information about executable files.




```
title Version Test
DIALOG CREATE,Version Test,-1,0,280,240,EDIT(filename;10;10;220),-
  BUTTON(browse;10;240;24;20;...),-
  STATUS(status;Click the button to choose a file),-
  TEXT(TEXT1;40;10;;;Size:),TEXT(size;40;60;80),-
  TEXT(TEXT3;56;10;;;Date:),TEXT(datetime;56;60;200),-
  TEXT(TEXT4;72;10;;;Version:),TEXT(version;72;60;200),-
  TEXT(TEXT5;88;10;;;Name:),TEXT(name;88;60;200),-
  TEXT(TEXT6;104;10;;;Company:),TEXT(company;104;60;200),-
  TEXT(TEXT7;120;10;;;Product:),TEXT(product;120;60;200),-
  TEXT(TEXT8;136;10;;;Version:),TEXT(prodver;136;60;200),-
  TEXT(TEXT9;152;10;;;Copyright:),TEXT(copr;152;60;200),-
  TEXT(TEXT10;168;10;;;Desc:),TEXT(desc;168;60;200;68)
:evloop
  wait event
  goto @event()
:browseBUTTON
  %F = @filedlg("EXE files|*.exe|DLL files|*.dll",Choose file)
  if @ok()
    dialog set,filename,%F
  end
  parse "name;company;product;version;desc;prodver;copr",|||||||||
  parse "%S;%T",@file(@dlgtext(filename),ZT)
  dialog set,size,%S bytes
  dialog set,datetime,@datetime(dd mmm yy  t,%T)
  %Z = @verinfo(@dlgtext(filename),TNCPVDXY)
  parse "%T;name;company;product;version;desc;prodver;copr",%Z
  dialog clear,status
  if @equal(%T,NE)
    dialog set,status,16-bit Windows executable file
  end
  if @equal(%T,PE)
    dialog set,status,32-bit Windows executable file
  end
  goto evloop
:CLOSE
  exit
```

Note

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 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:

Dialog example

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 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.

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:

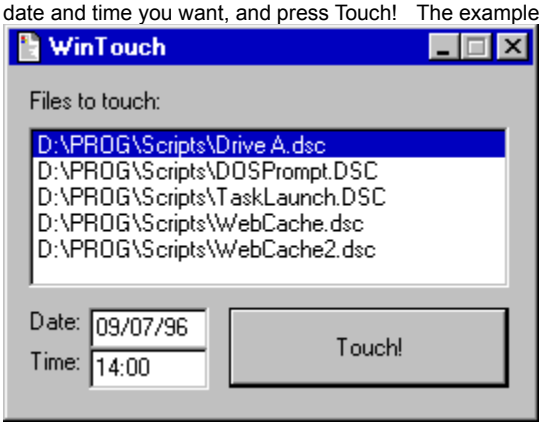
```
WINHELP DS.HLP,events
```

```
WINHELP ds.hlp,=key_winhelp
```

See also:

WinTouch

Here is another example of a useful utility that can be created using Visual DialogScript in a few lines of code. WinTouch is a handy way to change the date and time stamp of a set of files. Just drag the files to the list box from Explorer / File Manager, set the date and time you want, and press Touch! The example illustrates the use of drag and drop, and of the FILE SETDATE command.




```
title WinTouch
DIALOG CREATE,WinTouch,-1,0,268,180,DLGTYPE(DRAGDROP;SAVEPOS),-
  TEXT(TEXT1;10;10;;;Files to touch:),LIST(FileList;30;10;240;80),-
  TEXT(TEXT2;120;10;;;Date:),TEXT(TEXT3;140;10;;;Time:),-
  EDIT(Date;120;40;60;;@datetime(ddddd)),EDIT(Time;140;40;60;;@datetime(t)),-
  BUTTON(Touch;120;110;140;40;Touch!)
  dialog disable,Touch
:evloop
  wait event
  goto @event()
:TouchBUTTON
  list FileList,seek,0
  repeat
    %F = @item(FileList)
    file setdate,%F,@dlgtext(time),@dlgtext(date)
    list FileList,delete
  until @zero(@count(FileList))
  goto evloop
:DragDrop
  list FileList,DROPFILES
  dialog enable,Touch
  goto evloop
:CLOSE
  exit
```

Note

Window Spy

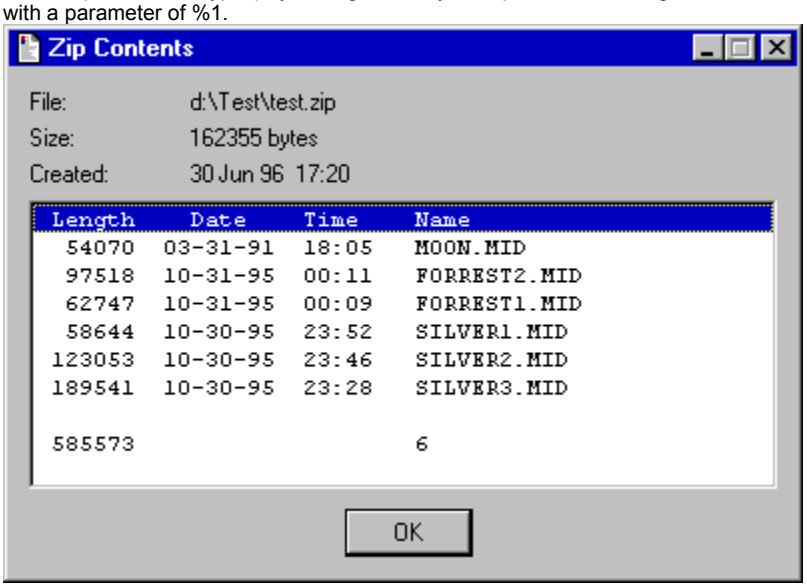
You use the Window Spy to get information about other application windows which you need to automate them using the WINDOW command.

To start the Window Spy press the Spy button. As you move the mouse pointer the status line shows its absolute X and Y co-ordinates. It also displays the text obtained from the window at that point.
When you click on a window or a control, the top two lines show the title text of the main window that became active when you clicked the mouse, and its class name. You would normally use one or other of these values to identify the window when using the WINDOW command or a related function.
The third line shows the text (if any) associated with the child window or control that you actually clicked on. If you clicked on an MDI child window then the text will be the title text of the MDI child, and the Window Spy will show [MDI Child] to confirm this. If you clicked on an ordinary window control and the control contained accessible text then this text is shown. To get this text in your script you would have to use @WINTEXT(@WINATPOINT(x, y))

The fourth line shows the X and Y co-ordinates of the point you clicked, relative to the top left corner of the main window. These are the co-ordinates you would use in a WINDOW CLICK command. To convert these to absolute co-ordinates for use with @WINATPOINT you must add the left and top position of the main window, obtained using the @WINPOS function. Since this position can vary it is not generally useful to hard code it into a script which is why the Window Spy does not calculate the absolute co-ordinates for you.

Zip List example

This example script displays a list of the contents of a Zip file in a dialog box. It uses the freeware Info-ZIP UnZip utility to generate the list, which is then displayed in a list box with a fixed pitch font. You can add this to the context menu for Zip files (My Computer: View / Options / File Types) by editing the entry for Zip files and adding a View action which runs the executable version of this script with a parameter of %1.

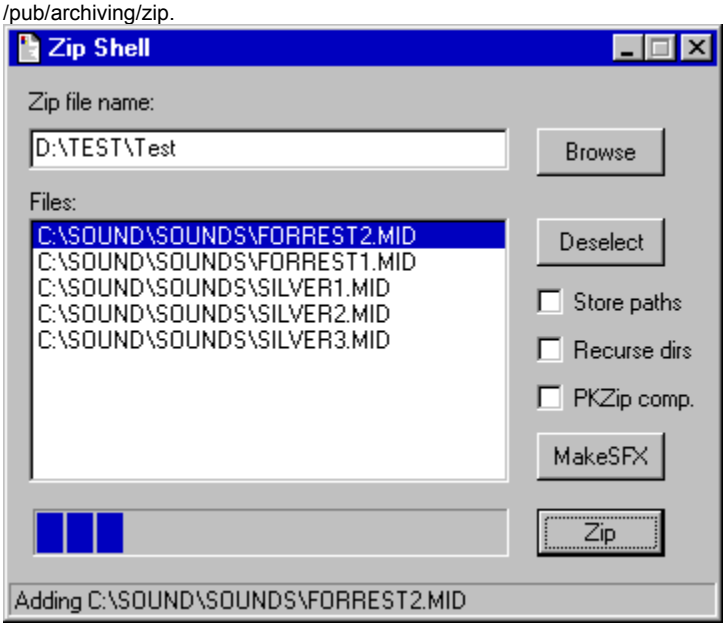



```
Title Zip List
if @equal(@ext(%1),ZIP)
  dialog create,Zip Contents,-1,0,400,260,-
    style(FixedPitch;Courier New;8;;white),-
    TEXT(T1;10;10),TEXT(T2;28;10),TEXT(T3;46;10),-
    LIST(L1;66;10;374;144;FixedPitch),BUTTON(OK;220;168)
  dialog set,T1,File:@tab()@tab()%1
  parse "%S;%T",@file(%1,TZ)
  dialog set,T2,Size:@tab()@tab()%S bytes
  dialog set,T3,Created: @tab()@datetime(dd mmm yy  t,%T)
  dialog cursor,wait
  %T = @env(TEMP)\ZS@datetime(hhnnss).TMP
  runh COMMAND /C UNZIP -l @shortname(%1) >%T,wait
  wait 1
  list 1,create
  list 1,loadfile,%T
  if @not(@zero(@count(1)))
    list 1,delete
    if @match(1,------)
      list 1,delete
    end
    if @match(1,------)
      list 1,delete
      list 1,insert,
    list L1,assign,1
    list 1,close
  end
  file delete,%T
  dialog cursor
  wait event
else
  warn Filename invalid or missing
end
```

Note

Zip Shell

This example script is a simple drag-and-drop shell for the freeware utility ZIP, which can be obtained by ftp from ftp.uu.net in /pub/archiving/zip.




```
  Title Zip Shell
  option filenames,short
  %U = @env(temp)
DIALOG CREATE,Zip Shell,-1,0,360,280,DLGTYPE(DRAGDROP;SAVEPOS),-
  TEXT(T1;10;10;;;Zip file name:),EDIT(E1;30;10;240;24),-
  BUTTON(Browse;30;264),-
  TEXT(T2;60;10;280;;Files:),LIST(L1;75;10;240;132),-
  CHECK(C1;108;264;;;Store paths),CHECK(C2;132;264;;;Recurse dirs),-
  CHECK(C3;156;264;88;;PKZip comp.),BUTTON(Zip;220;264),-
  BUTTON(Deselect;75;264),-
  BUTTON(MakeSFX;182;264),PROGRESS(P1;220;12;238;24),STATUS(SP)
  if @not(@null(%1))
    repeat
      dialog set,L1,%1
      if @null(%E)
        %E = @name(%1)
        dialog set,E1,@windir()\Desktop\%E
      end
      shift
    until @null(%1)
  else
    dialog set,E1,@windir()\Desktop\Zipfile
  end
:evloop
  dialog set,SP,@count(L1) files selected
  dialog set,P1,0
  wait event
  goto @event()
:dragdrop
  list L1,DROPFILES
  goto evloop
end
:BrowseButton
  directory change,@path(@dlgtext(E1))
  %E = @filedlg(*.Zip)
  if %E
    dialog set,E1,%E
  end
  goto evloop
:DeselectButton
  dialog clearsel,L1
```

```
  goto evloop
end
:MakeSFXButton
  %N = @name(@dlg(E1).zip)
  %P = @path(@dlg(E1))
  runh command /c copy /b c:\utils\unzipsfx.exe+%P%N.ZIP %P%N.EXE
  goto evloop
:ZipButton
  %S =
  if @not(@dlgtext(C1))
    %S = -j
  end
  if @dlgtext(C2)
    %S = -r
  end
  if @dlgtext(C3)
    %S = %S -k
  end
  %C = @count(L1)
  %N = 0
  list L1,seek,0
  repeat
    %F = @item(L1)
    dialog set,SP,Adding %F
    runh ZIP -u %S -b %U @dlg(E1) %F,wait
    if @not(@zero(@retcode()))
      warn Zip failed: error code @retcode()
      goto evloop
    end
    list L1,delete
    %N = @succ(%N)
    dialog set,P1,@div(@prod(%N,100),%C)
  until @equal(%C,%N)
  goto evloop
end
:close
  exit
```

Note

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.

BITMAP dialog element

The fields for a BITMAP dialog element should be filled in as follows:

Type: BITMAP (graphical image control).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code.

Position: These are the top position, left hand edge position, width and height, respectively, of the control. You can also position and size the control using the mouse.

Caption: Not applicable for a BITMAP.

Filename: The name of the bitmap, icon or executable file from which the bitmap image will be loaded. You can use the button captioned ... to select this. Note that if a full path is given, the program might fail when run on a different system and installed into a different directory. If the bitmap will be in the same directory as the program, just give the name.

Values: Not applicable for a BITMAP.

Value: Not applicable for a BITMAP.

Style(s): STRETCH - Bitmap will be stretched to fit the size of the control. (This does not apply to icons.)

CLICK - Bitmap will generate CLICK events when clicked with the mouse.

HAND - As CLICK, but a hand cursor will be shown when over the bitmap.

CROSS - As CLICK, but a cross cursor will be shown when over the bitmap.

User defined styles.

Multiple styles may be used, separated by semicolons.

BUTTON dialog element

The fields for a BUTTON dialog element should be filled in as follows:

Type: BUTTON (button control).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code. When pressed, the button will cause a <name>BUTTON event.

Position: These are the top position, left hand edge position, width and height, respectively, of the control. You can also position and size the control using the mouse. Note that the width and height can be left blank to take default values.

Caption: Optional. If omitted, the control name will be used. Use a caption if you want to use characters in the caption that would not be valid in a control name.

Filename: Not applicable for a BUTTON.

Values: Not applicable for a BUTTON.

Value: Not applicable for a BUTTON.

Style(s): User defined styles only.

CHECK dialog element

The fields for a CHECK dialog element should be filled in as follows:

Type: CHECK (check box control).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code.

Position: These are the top position, left hand edge position, width and height, respectively, of the control. You can also position and size the control using the mouse. Note that the width and height can be left blank to take default values.

Caption: This is the text that appears alongside the check box.

Filename: Not applicable for a CHECK.

Values: Not applicable for a CHECK.

Value: Either blank or zero (unchecked) or 1 (checked).

Style(s): User defined styles only.

COMBO dialog element

The fields for a COMBO dialog element should be filled in as follows:

Type: COMBO (edit control with drop-down list).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code.

Position: These are the top position, left hand edge position, width and height, respectively, of the control. You can also position and size the control using the mouse. Note that the width and height can be left blank to take default values.

Caption: Not applicable for a COMBO.

Filename: Not applicable for a COMBO.

Values: Not applicable for a COMBO.

Value: Initial text to appear in the control (if any).

Style(s): SORTED - items in drop-down list should be sorted into ASCII order.

User defined styles.

Multiple styles may be used, separated by semicolons.

DLGTYPE dialog element

The fields for a DLGTYPE dialog element should be filled in as follows:

Type: DLGTYPE (specifies attributes and characteristics of the dialog itself).

Name: Not applicable for a DLGTYPE.

Position: Not applicable for a DLGTYPE.

Caption: Not applicable for a DLGTYPE.

Filename: Not applicable for a DLGTYPE.

Values: Not applicable for a DLGTYPE.

Value: Not applicable for a DLGTYPE.

Style(s): DRAGDROP - Dialog accepts drag-and-drop operations, which will cause a DRAGDROP event to occur.

ONTOP - Dialog will remain on top of other windows even when inactive.

SAVEPOS - Dialog will remember its last screen position whenever the program is run.

SMALLCAP - Dialog will have a small toolbar-style title bar.

User defined styles, to apply to all controls by default.

Multiple styles may be used, separated by semicolons.

EDIT dialog element

The fields for a EDIT dialog element should be filled in as follows:

Type: EDIT (text input control).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code.

Position: These are the top position, left hand edge position, width and height, respectively, of the control. You can also position and size the control using the mouse. Note that the width and height can be left blank to take default values.

Caption: Not applicable for a EDIT.

Filename: Not applicable for a EDIT.

Values: Not applicable for a EDIT.

Value: Initial text to appear in the control (if any).

Style(s): PASSWORD - all characters in the control are displayed as asterisks.

User defined styles.

Multiple styles may be used, separated by semicolons.

LIST dialog element

The fields for a LIST dialog element should be filled in as follows:

Type: LIST (list box control).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code.

Position: These are the top position, left hand edge position, width and height, respectively, of the control. You can also position and size the control using the mouse. Note that the width and height can be left blank to take default values.

Caption: Not applicable for a LIST.

Filename: Not applicable for a LIST.

Values: Not applicable for a LIST.

Value: Not applicable for a LIST. To put data into the list use the LIST command.

Style(s): SORTED - items in drop-down list should be sorted into ASCII order.

User defined styles.

Multiple styles may be used, separated by semicolons.

POPUP dialog element

The fields for a POPUP dialog element should be filled in as follows:

Type: POPUP (popup menu control).

Name: Not applicable for a POPUP as a dialog can only have one and you cannot read from or write to it.

Position: Not applicable for a POPUP.

Caption: Not applicable for a POPUP.

Filename: Not applicable for a POPUP.

Values: The values consist of a series of items which will appear on the menu, each separated by a vertical bar. When selected, each item will generate an <item>MENU event.

Value: Not applicable for a POPUP.

Style(s): Not applicable for a POPUP.

PROGRESS dialog element

The fields for a PROGRESS dialog element should be filled in as follows:

Type: PROGRESS (progress indicator control).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code.

Position: These are the top position, left hand edge position, width and height, respectively, of the control. You can also position and size the control using the mouse. Note that the width and height can be left blank to take default values.

Caption: Not applicable for a PROGRESS.

Filename: Not applicable for a PROGRESS.

Values: Not applicable for a PROGRESS.

Value: Initial value for the progress indicator (from 0 to 100). By default, the value is zero.

Style(s): Not applicable for a PROGRESS.

RADIO dialog element

The fields for a RADIO dialog element should be filled in as follows:

Type: RADIO (group of radio buttons).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code.

Position: These are the top position, left hand edge position, width and height, respectively, of the control. You can also position and size the control using the mouse. Note that the width and height can be left blank to take default values.

Caption: This is the caption text for the whole group.

Filename: Not applicable for a RADIO.

Values: A list of values for each button, separated by vertical bars. The values will be displayed alongside each button.

Value: One of the values from the list above. By default, none of the buttons is selected.

Style(s): User defined styles only.

STATUS dialog element

The fields for a STATUS dialog element should be filled in as follows:

Type: STATUS (status panel).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code.

Position: Not applicable for a STATUS: this is always at the bottom of the dialog.

Caption: Not applicable for a STATUS.

Filename: Not applicable for a STATUS.

Values: Not applicable for a STATUS.

Value: Initial text to be displayed in the panel, if any.

Style(s): User defined styles only.

STYLE dialog element

The fields for a STYLE dialog element should be filled in as follows:

Type: STYLE (defines color and text attributes that can be used by the dialog or by individual controls).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code.

Font: This specifies the typeface to be associated with the style. You can choose this and the following three attributes from a common font dialog by pressing the button captioned .... The default font is MS Sans Serif.

Size: Specifies the size of the typeface. The default size is 8.

Bold: Specifies whether the bold attribute is required.

Italic: Specifies whether the italic attribute is required.

Position: Specifies whether text should be left justified, centered, or right justified in a TEXT control. This attribute does not apply to other controls.

Backgrnd: Can be one of BACKGRND, FOREGRND or a list of colors selected from a drop-down list. The default value, BACKGRND, is the Windows default button color.

Foregrnd: Can be one of BACKGRND, FOREGRND or a list of colors selected from a drop-down list. The default value, FOREGRND, is the Windows default text color.

TASKICON dialog element

The fields for a TASKICON dialog element should be filled in as follows:

Type: TASKICON (task bar icon).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code. When clicked, the icon will generate a <name>ICON event.

Position: Not applicable for a TASKICON: this always appears in the Task Bar Notification Area.

Caption: Not applicable for a TASKICON.

Filename: The name of the icon or executable file from which the icon image will be loaded. You can use the button captioned ... to select this. Note that if a full path is given, the program might fail when run on a different system and installed into a different directory. If the icon will be in the same directory as the program, just give the name.

Values: Not applicable for a TASKICON.

Value: Text to be displayed when the user places the cursor over the task bar icon.

Style(s): Not applicable for a TASKICON.

TEXT dialog element

The fields for a TEXT dialog element should be filled in as follows:

Type: TEXT (text control).

Name: As specified. This cannot be changed once the control exists, except by editing the DialogScript code.

Position: These are the top position, left hand edge position, width and height, respectively, of the control. You can also position and size the control using the mouse. Note that the width and height can be left blank to take default values. If the width is blank the control auto-sizes to fit the text.

Caption: Not applicable for a TEXT.

Filename: Not applicable for a TEXT.

Values: Not applicable for a TEXT.

Value: Initial value of text to be displayed in the control.

Style(s): TRANSPARENT - Use when positioning a TEXT element over a BITMAP.

User defined styles.

Multiple styles may be used, separated by semicolons.

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 (available in the Professional version only)

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

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 (Professional version only.)

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 You can add your own tools to the menu using the Add-In Tool Manager.

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*.

License Agreement

You are granted a free license to use this software for your own personal, non-commercial purposes. This software is not to be used for any business, governmental or military purpose. If you wish to use the software for such purposes then you must license the Standard or Professional edition.

No liability is accepted 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. It is entirely your responsibility to take all necessary precautions to safeguard against any such loss or damage.

Upgrading

We hope you enjoy using Personal VDS Freeware Edition.

If you find it useful then you may be interested in upgrading to one of the following versions:

Personal VDS '97 **$29 US**

This is the version of Personal VDS for users of Windows 95. It is a 32-bit program and supports long filenames, task bar tray icons, unlimited length strings and string lists and can read and write to the Windows Registry. Like the Freeware Edition it is not licensable for commercial use.

Visual DialogScript 2 **$69 US**

This version of Visual DialogScript is for power users, business users and professional developers who use - or are developing scripts for - Windows 95 or Windows NT 4. This version can create EXE files, and includes a royalty-free run-time license. Additional features include an icon editor and support for up to four add-in extensions. As a 32-bit program it supports long filenames, task bar tray icons, unlimited length strings and string lists and the Windows Registry.

For more information, and to download evaluation copies of these products, visit our Web site at http://www.jm-tech.com/.