

Desktop Control Language Help Contents



[What Is DCL?](#)



[System Requirements](#)



[A Quick DCL Course](#)



[Desktop Control Language Reference](#)



[Desktop Control Language Characteristics](#)



[Using the DCL Editor](#)



[Using the Dialog Editor](#)



[Using the Debugger](#)



[Sample Scripts](#)



[Translating MultiSet Scripts](#)

What Is DCL?

Welcome to **Desktop Control Language™ (DCL)**, a full-featured, yet easy-to-use scripting language that lets you control your Microsoft® Windows™ network environment. **DCL** is a product of McAfee, Inc.

DCL is built on the easy-to-learn Basic language. It provides all of the features of a major programming language, including subroutines, functions, strings, arrays, numeric support, and advanced flow control.

DCL provides an integrated development environment for editing, testing, and debugging scripts.

A **Macro Recorder** lets you automatically generate scripts to control other Windows applications. You can modify the code of the recorded macro as needed.

A syntax checker looks for entry errors. A **Debugger** lets you set breakpoints and watch variables and trace through scripts for logic errors.

DCL translates any **MultiSet** scripts you run into **DCL** scripts. The conversion is automatic.

A **Dialog Editor** lets you interactively design dialog boxes and define their controls, and then include the dialog box definition in your script. You can also capture dialog boxes from other applications and paste the captured controls in the **Dialog Editor**.

When you have finished writing and testing your script, you can build an executable (.EXE) file for distribution.

System Requirements

The following list contains the minimum system requirements to run this version of **DCL**.

- An 80386/SX or higher based computer.
- 4 megabytes of memory.
- One 1.44 MB (3-1/2") floppy disk drive.
- MS-DOS Version 5.0 or later.
- Microsoft Windows Version 3.1 or later, or Windows for Workgroups Version 3.1 or later.
- A monitor and VGA graphics card or other high-resolution graphics card compatible with Windows Version 3.1 or later.
- A Microsoft Windows-compatible mouse, recommended but not required.

This product supports, but does not require, local area networks running Novell NetWare 3.11, 3.12, or NetWare 4.0 in bindery emulation mode.

A Quick DCL Course

This section will introduce you to the elements that make up **DCL**'s implementation of Basic. If you have had Basic programming experience, we recommend that you peruse this section as well as [Desktop Control Language Characteristics](#), as different implementations of Basic may vary. If you are new to Basic programming, this section will introduce you to the fundamentals of the language. You may also want to purchase a Basic programming book from the many that are currently available on the market.



[A Simple DCL Script](#)



[Comments](#)



[Variables](#)



[Constants](#)



[Operators](#)



[Subroutines](#)



[Functions](#)



[Arrays](#)



[Conditional Statements](#)



[Loops](#)



[Where to Go from Here](#)

A Simple DCL Script

Here is a simple **DCL** script:

```
sub main()  
    msgbox "Hello World"  
end sub
```

Every **DCL** script contains a subroutine called `main`. A subroutine is a set of commands that perform a specific task. Simple Basic programs often have only one subroutine. More complex Basic programs often have more than one subroutine.

The commands `sub main` and `end sub` define the beginning and end of the subroutine. The subroutine contains only one statement `msgbox "Hello World"`.

Running the Script

To run this script:

- 1 Start the **DCL Editor** (DCLEDIT.EXE). If you allowed the installation to create a NetTools program group, choose the DCL Editor icon.
- 2 Type the second line of the above script. The first and third lines are already entered for you.
- 3 Choose the Start Script command from the Run menu.

When you run this script, the following message box is displayed:



Saving the Script

To save the script:

- 1 Choose the Save command from the **DCL Editor** File menu.
- 2 In the File Save As dialog box, enter **hello** in the File Name box.
- 3 Choose OK.

A text (ASCII) file named `hello.dcl` contains your script. The default extension for all scripts is `.dcl`.

Comments

A comment is a note you write in a script. It is not executed with your script. Its sole purpose is to provide information about the script. Comments are very useful when you have to modify a script.

A comment begins with a single quote (') and continues for the rest of the line.

The following script contains comments:

```
sub main()  
  ' Script Name: Hello  
  ' Written: 3/1/94  
  ' Author: J. Doe  
  
  msgbox "Hello World"      'display message box  
end sub
```

Notice that a comment can occupy the same line as a **DCL** command (if it follows the command).

Variables

In the previous script, we provided literal text (the actual letters, digits, and special characters) to be displayed in the message box. Literal text is enclosed in double quotation marks (""), for example, "Hello World". Although useful, literal text has its limitations. For example, what if we wanted to say "Hello Everybody" instead of "Hello World." We would have to write a new script.

Variables let you change the data displayed by and used in your scripts. Put simply, a variable is a name with which you associate data--a string of characters or a number. Variable names can be up to 40 characters long. They must start with a letter and can contain letters and digits. (Specific special characters are used to indicate the type of data contained in variables.)

Data Types

The most commonly used types of data in scripts are strings and integers, which are described in the following paragraphs. **DCL** supports many data types, allowing it accommodate almost any programming situation. These data types include integers, long integers, strings, dialogs, single precision floats, and double precision floats. The characteristics of these data types are described in detail under [DCL Data Types](#).

Strings

A string is a succession of letters, digits, spaces, and special characters. Here are some example strings:

```
"Hello"
"Hello "
"Hello World"
"Testing 1 2 3"
"" (Null string)
```

A string variable is simply a name representing a string. The actual string of characters represented by the variable may change. This script makes use of a string variable:

```
sub main()
    name$ = InputBox$("Enter your name:")
    msgbox "Hello " + name$
end sub
```

`InputBox$` is a function (a special type of command we'll discuss later) that displays a dialog box that prompts the user to enter his or her name. The name the user enters is stored in a string variable called `name$`. The contents of the string variable `name$` might be "Jack" or "Jane" or "Chris". The dollar (\$) symbol at the end of a variable name indicates that the variable is a string.

The `msgbox` statement displays a string that is made up of the literal "Hello " and the contents of the string variable `name$`. The two strings are combined (concatenated) by the '+' operator.

Here are the dialog boxes generated by the script.



Assume that the user types Chris and chooses OK. The following message is displayed:



Integers

Integers are whole numbers between -32,768 and 32,767. Long Integers are whole numbers between -2,147,483,648 and 2,147,483,647. The following script uses integers:

```
sub main()  
    SysRes%=SystemFreeResources  
    FreeMem&=SystemFreeMemory  
end sub
```

The `SystemFreeResources` function returns an integer between 0 and 100, indicating the percentage of free system resources. This value is stored in the integer variable `SysRes%`. An integer variable is indicated by the `%` symbol at the end of the name. (The percent symbol in an integer variable name has nothing to do with percentages.)

The `SystemFreeMemory` function returns a long integer representing the amount of free memory. The value is stored in the long integer variable `FreeMem&`. The `&` symbol at the end of a variable name indicates a long integer.

Variable names are not case-sensitive. **DCL** does not distinguish between upper- and lowercase letters in a variable name. However, you can use capitalization to make your variable names more readable. A common practice is to use a capital letter to indicate the beginning of a word. For example, `FreeMem&` is much more readable than `freemem&`.

In its current state, this script produces no visible output. We will expand it in a later section.

Constants

A constant is a value that does not change such as 0, 1, 2, 4, etc. To help make your scripts more readable, **DCL** provides symbolic constants, text strings that represent numeric values. For example the constant FALSE equals 0 and TRUE equals -1. You do not need to know the numeric value of a constant in order to use it.

The following script illustrates the use of symbolic constants:

```
sub main()
    SystemMouseTrails TRUE
    msgbox "Move the Mouse Around Now"
    SystemMouseTrails FALSE
end sub
```

The values TRUE and FALSE are used to turn the display of mouse trails (ghost images following the mouse as it moves) on and off.

Several **DCL** commands make use of symbolic constants. By convention, constant names are in uppercase. If a symbolic constant is provided for a command, it is indicated in the description of the command in the [Desktop Control Language Reference](#).

User-Defined Constants

The [Const](#) command lets you define your own constants.

The following script converts from Fahrenheit to Centigrade. The formula for converting from Fahrenheit to Centigrade is:

$$\text{Centigrade} = ((\text{Fahrenheit} - 32) * 5) / 9$$

The * is the symbol for multiplication. The / represents division.

The script performs the conversion, using the constant FREEZING instead of the literal 32.

```
Const FREEZING = 32
sub main()
    ftemp$ = InputBox$("Enter a Fahrenheit temperature.")
    fahrenheit% = val(ftemp$)
    centigrade% = ((fahrenheit% - FREEZING) * 5) / 9
    ctemp$ = str$(centigrade%)
    msgbox "Centigrade temperature:" + ctemp$
end sub
```

Constants must be defined before the `main` subroutine is executed.

In this script, the `val` function converts a string to a number, the `str$` function converts a number to a string.

When you define a constant, you should check the list of constants that have been predefined by **DCL** and avoid using these names. The **DCL** constants are listed under [DCL Constants](#).

Operators

The following operators are used by **DCL**:

Operator (Precedence: highest to lowest)	Description
()	parentheses
^	exponentiation
-	unary minus
/, *	multiplication and division
\	integer division
mod	modulo
+, -	addition and subtraction
=, <>, >, <, <=, >=	relational
not	logical negation
and	and
or	or
xor	exclusive or

For an explanation of a particular operator, see [Desktop Control Language Reference](#).

Subroutines

A subroutine is a separate block of **DCL** code (in the same script) that is called and executed as a unit. Short scripts usually do not require subroutines. When you write a long script, it is helpful to break tasks down into subroutines. The following code illustrates a subroutine:

```
sub HelloProcessing()  
    name$ = InputBox$("Enter your name:")  
    msgbox "Hello " + name$  
end sub  
sub main()  
    HelloProcessing  
  
    'Rest of Script  
  
end sub
```

In this example the subroutine HelloProcessing greets the user and gets his or her name. The block of code contained in the subroutine begins with the command `sub <NameOfSubroutine>` and ends with the command `end sub`. To call a subroutine, you enter the name of the subroutine as you would any other command. The "command" `HelloProcessing` causes the HelloProcessing subroutine to be executed. Note that the subroutine must be defined before the call to the subroutine.

For further information about subroutines, see [Sub...End Sub](#).

Functions

Functions are like subroutines. Unlike subroutines, however, functions return values (integers, strings, etc.). `InputBox$` is a **DCL** function that returns a string. When you call a **DCL** function, you use the returned value in an expression, and you must provide any data to be manipulated by the function. The term for this data is arguments or parameters. The general format for a function call is:

```
variableName = Function(Parameter1,Parameter2,etc.)
```

For example:

```
name$ = InputBox$("Enter your name:")
```

In this code sample, `name$` is a string variable that contains the string value returned by `InputBox$`. The argument "Enter your name:" is passed to the function and displayed in the dialog box generated by `InputBox$`.

The `len` function returns an integer representing the length of a string. For example:

```
StringLength%=len(name$)
```

In this example, the `len` function puts the length of the string `name$` in the integer variable `StringLength%`.

User-Defined Functions

You can write your own functions. Functions are useful for frequent tasks that return a value. The following script uses a function to convert from Fahrenheit to Centigrade.

```
Const FREEZING = 32
function FarToCent%(f%)
    FarToCent% = ((f% - FREEZING) * 5) / 9
end function
sub main()
    ftemp$ = InputBox$("Enter a Fahrenheit temperature.")
    fahrenheit% = val(ftemp$)
    centigrade% = FarToCent%(fahrenheit%)
    ctemp$ = str$(centigrade%)
    msgbox "Centigrade temperature:" + ctemp$
end sub
```

The function block is defined by the `function` and `end function` statements.

The `function` statement contains the function name, return type, and parameters passed to the function. The function name (`FarToCent%`) ends in a character that indicates the type of value the function returns. The function in our script returns an integer (denoted by the `%` symbol). One parameter (`f%`) is passed to the function.

The value to be returned is assigned to the function name (in our script, `FarToCent%`).

The processing of the function occurs as follows:

- 1 In the `main` subroutine, `FarToCent%(fahrenheit%)` calls the `FarToCent%` function and passes it the `fahrenheit%` parameter.
- 2 The `FarToCent%` function receives the integer parameter as `f%` and uses it to represent the Fahrenheit temperature in the conversion to Centigrade.
- 3 The `FarToCent%` function performs the conversion and returns the result.
- 4 The returned value is put in the `Centigrade%` variable in the `main` subroutine.

Passing Parameters by Reference and by Value

As a default, parameters are passed to functions by **reference**. If you are new to programming, this means that the function-side parameters refer to the same variable as the calling-statement-side parameters. In other words, `fahrenheit%` and `f%` refer to the same variable. If you were to change `f%` in the `FarToCent%` function `fahrenheit%` in the `main` subroutine would also be changed.

The alternative to passing parameters by reference is to pass them by **value**. This means that the **contents** of the variable are passed to the function. The function-side parameter and the calling-statement-side parameter refer to different variables. To specify that a parameter is to be passed by value, precede the parameter in the `function` statement with `byval`. By adding `byval` to the `FarToCent%` function, you can cause `f%` and `fahrenheit%` to be entirely separate variables. You can change `f%` in the function without affecting `fahrenheit%` in the `main` subroutine.

```
Const FREEZING = 32
function FarToCent%(byval f%)
    FarToCent% = ((f% - FREEZING) * 5) / 9
end function
sub main()
    ftemp$ = InputBox$("Enter a Fahrenheit temperature.")
    fahrenheit% = val(ftemp$)
    centigrade% = FarToCent%(fahrenheit%)
    ctemp$ = str$(centigrade%)
    msgbox "Centigrade temperature:" + ctemp$
end sub
```

For a more in-depth discussion of functions, see the [Function...End Function](#) command.

Arrays

An array is a series of variables of the same type all having the same name. You differentiate between the elements of the array by using an integer index. Assume that you have an array of strings called `Cities$`.

Cities\$	Index
"Atlanta"	0
"Chicago"	1
"Los Angeles"	2
"Montreal"	3
"New York"	4

`Cities$(0)` contains the string "Atlanta"; `Cities$(1)` contains "Chicago", etc. The form for referencing an element of an array is:

`ArrayVariableName(Index)`

Here is a sample script that uses an array of strings:

```
sub main()
  dim Performance$(5) as String
  Performance$(0)="Poor"
  Performance$(1)="Fair"
  Performance$(2)="Good"
  Performance$(3)="Excellent"
  Performance$(4)="Perfect"
  gradestr$=InputBox$("Enter a grade from 0 to 100.")
  grade#=val(gradestr$)
  plevel%=grade#\25
  msgbox "The grade is " + Performance$(plevel%)
end sub
```

The activities required to use an array are:



Declare the array using the `dim` statement. This statement names the array, indicates the number of elements (and dimensions) and specifies the type of data to be contained in the array. The sample script defines an array named `Performance` of type `string`, containing 5 elements and one dimension.



Load the array with values. This essentially amounts to assigning a value to each element in the array. The `Performance(n)="value"` statements load the array. The number in parentheses is the index to the array. (Some **DCL** functions that use arrays load the initial values for you.) By default the first element of an array has 0 as its index. You can use the `Option Base` to set this value to 1, if desired.



Reference the array as needed. In the subsequent code, an input box asks the user to enter a number from 0 to 100. This returns a string, which the `val` function converts into the long integer `grade#`. We perform integer division of `grade#` by 25 to get an integer between 0 and 4. (Integer division returns only the nonfractional part of the quotient.) This integer is `plevel%`, which serves as an index to select the correct element from the array. The selected string is then displayed in a message box. For additional information about arrays, see the [dim](#) statement.

Conditional Statements

Conditional statements are executed only if a particular condition is true. **DCL** supports the following conditional statements: If...Then...Else and Select Case. They allow your script to perform different tasks under different conditions.

If...Then...Else

The `if...then` statement performs a statement (or set of statements) if a condition is true. Otherwise the statement is skipped. Here is an example of the `if...then` statement.

```
sub main()
    SysRes%=SystemFreeResources
    if SysRes% < 30 then
        msgbox "Your available resources are low."
    end if
end sub
```

This script informs the user if available system resources fall below 30 percent. The `if` keyword specifies the condition that must be true; the `then` keyword signals the beginning of one or more statements to be executed if the condition is true. The terms `if` and `then` must be on the same line. The `if...then` statement ends with the `end if` keyword.

The `if...then...else` statement lets you specify activities for the script to perform whether the condition is true or false. The `else` keyword precedes one or more statements to be executed when the condition is false.

```
sub main()
    SysRes%=SystemFreeResources
    if SysRes% < 30 then
        msgbox "Your available resources are low."
    else
        msgbox "Your available resources are fine."
    end if
end sub
```

A more complex form of the `if...then...else` statement lets you accommodate multiple conditions. The following `if...then...elseif...else` statement varies the message depending on the percentage of available system resources.

```
sub main()
    SysRes%=SystemFreeResources
    if SysRes% < 30 then
        msgbox "Your available resources are low."
    elseif SysRes% < 20 then
        msgbox "Resources extremely low. Close some Applications now."
    elseif SysRes% < 10 then
        msgbox "Resources are DANGEROUSLY low. Close Applications now."
    else
        'everything's ok
        msgbox "Your available resources are fine."
    end if
end sub
```

For further information, see [If...Then...Else](#).

Select Case

The `select case` statement also handles multiple conditions.

Like the previous `if...then...elseif...else` statement, the following `select case` statement varies the message depending on the percentage of available system resources.

```
sub main()
    SysRes%=SystemFreeResources
    select case SysRes%
        case 21 to 30
            msgbox "Your available resources are low."
        case 11 to 20
            msgbox "Resources extremely low. Close some Applications now."
        case 0 to 10
            msgbox "Resources are DANGEROUSLY low. Close Applications now."
        case else
            'everything's ok
            msgbox "Your available resources are fine."
    end select
end sub
```

The `select case` statement begins with `select case <variableName>` and ends with `end select`. The logic of the `select case` statement is to test each case in the `select case...end select` block. Each case statement specifies a condition. If the condition is true, the statement(s) associated with are executed. The `case else` statement is executed if none of the conditions tested are true. The breadth of expressions supported by the case statement make it a powerful tool for conditional processing. For further information, see [Select Case](#).

Loops

Loops are blocks of statements that perform the same action multiple times while a condition is true or until a condition is met. Each pass through a loop is called an iteration. **DCL** supports the following iterative statements: For...Next, While...Wend, and Do...Loop.

For...Next

A For...Next loop executes a group of statements a specified number of times. Here is a For...Next loop:

```
sub main()  
    for x% = 1 to 5 step 1  
        msgbox str$(x%)  
    next x%  
    msgbox "You're past the loop."  
end sub
```

In the For statement you specify a:



Counter variable, which in our example is `x%`.



Beginning and ending value for the counter variable, in our example 1 and 5 respectively.



Increment. This is provided by the `step` portion of the statement. The `step` can be omitted if the increment is 1.

The Next statement marks the end of the loop and increments the counter variable by the `step` value.

The statements inside the loop are executed repeatedly until the counter exceeds the ending value. In our example the loop is executed five times. When the counter exceeds the ending value, the script continues with the next statement after the For...Next loop.

You can use variables to represent starting and ending values and the increment. For example:

```
sub main()  
    startval%=1  
    endval%=10  
    incr%=2  
    for x% = startval% to endval% step incr%  
        msgbox str$(x%)  
    next x%  
    msgbox "You're past the loop."  
end sub
```

For more information, see For...Next.

While...Wend

A While...Wend loop executes a group of statements repeatedly while a condition is true. Once the condition is false, the statement following the Wend statement is executed.

```
sub main()  
    testval%=1  
    while testval% < 6  
        msgbox str$(testval%)  
        testval% = testval% +1  
    wend  
    msgbox "You're past the loop."  
end sub
```

The `While` statement marks the beginning of the loop and specifies the test condition. In our example, the condition is that the variable `testval%` must be less than 6.

The `Wend` statement marks the end of the loop.

The loop is executed repeatedly until `testval%` is equal to 6.

For more information, see [While...Wend](#).

Do...Loop

The `Do...Loop` statement executes a group of statements while or until a condition is true. It has several forms. For further information see, [Do...Loop](#).

Where to Go from Here

The purpose of this section was to give you an initial exposure to **DCL**. To increase your familiarity with the language, we recommend that you examine the numerous sample scripts provided for the individual **DCL** commands in [Desktop Control Language Reference](#).

Translating MultiSet Scripts

When you open a **MultiSet** script (with the extension .SET) from the **DCL Editor**, the **MultiSet** script is translated into a **DCL** script. This section is a technical appendix that describes the translation process.

The Mechanisms Of Translation

Translation of MultiSet scripts into Desktop Control Language (DCL) is accomplished through the use of a couple of mechanisms. First there is the actual MultiSet language runtime system, and secondly a set of carefully devised DCL functions that emulate MultiSet commands and functions as closely as possible.

For compatibility's sake, the original MultiSet language parsing and execution routines have been employed to provide the first phase of the translation. Instead of executing MultiSet commands however, when a script is read by the parser the components of the script are given to a translation routine which decides how each part of the script should be used to generate DCL code. The translations routine is aware of all possible MultiSet commands and constructs, and also has preset instructions on how to generate the proper DCL text.

The second part of the translation is a set of DCL functions that have been designed to emulate the existing MultiSet command functions. Since most of the MultiSet language cannot be simply mapped to an existing DCL command, these routines provide the bridge between the two languages. The functions, as you will see in your translated scripts, have names that are derived from their MultiSet predecessors.

The Process Of Translation

When you choose to open a MultiSet script file (a file ending with the extension .SET) from the DCL editor the translator is automatically called for you. The MultiSet script is given to the translator at this time, and when translation has been finished the newly generated DCL script will be presented to you.

During translation, the parser routine reads each character of each line of MultiSet code to find the individual symbols and keywords in the script. These symbols and keywords are generally referred to as "tokens." As these tokens are found, they are given to the translator routine to be mapped into their appropriate DCL counterparts. All of the translated calls are placed into the "main()" subroutine of the DCL script. When such a token is encountered that requires one of the specially designed DCL functions then that function text is added to the top of the script being generated. Once such a function has been added to the script, it will not be added again. Each function need only be added once because from that point onward it only needs to have the text generated to call it when needed.

Once all of the MultiSet text has been read through and translated, the DCL script that has been generated is given back to the DCL editor for presentation to the user.

Translation Requirements

In order to translate a script from MultiSet to DCL, there are a few requirements that must be met. First of all, you must be using the DCL editor, and have available the MSTODCL.DLL file. This DLL contains the translator routines. Without it you will not be able to translate MultiSet scripts. In addition, you need a working MultiSet script. Since the translator uses the same parser that MultiSet incorporates, any syntax errors in your MultiSet script will be reproduced by the translator and you will not get a complete translation. In addition, if your MultiSet script does not run correctly, then the resulting DCL script will run just as incorrectly.

Desktop Control Language Characteristics



DCL General Characteristics



DCL Script Execution



DCL Structures



DCL Compiler



DCL Variables



DCL Expression Evaluation



DCL Subroutines & Functions



DCL Error Handling



DCL Arrays



DCL Data Types



DCL Operator Precedence



DCL Comments












DCL Constants



DCL Limitations

DCL General Characteristics

-  DCL is compiled, not interpreted.
-  A DCL script consists of any number of user-defined functions, user-defined subroutines, and external DLL declarations.
-  The compiler and runtime operate under Windows 3.1 and Windows for Workgroups 3.11 or later.
-  A DCL script is an ASCII stream of text with a NULL terminator.
-  DCL scripts are completely self contained. Variables have a scope local to the function or subroutine in which they are declared. Further, any referenced functions or subroutines must occur within the same script.
-  Line numbers are not supported. Labels are used instead.
-  Quotes can be embedded within constant string expressions using two consecutive quotes, such as "John said, ""hello"", to Jane".
-  Numeric constant folding is supported. For example, the statement `i=10+23` is reduced before compiling to `i=33`.
Constant expressions involving strings are reduced at compile time, including calls to the function `CHR$()` where the passed parameter is a constant. For example, the statement `s$ = "Hello" + chr$(13) + chr$(10) + "world"` is reduced to the expression `s$ = "Hello\r\nworld"`.
-  The runtime is reentrant. Thus, two DCL scripts can execute concurrently.

DCL Script Execution

Execution of a DCL script begins with a subroutine with the predefined name `Main`.

DCL Structures



User-defined structures are not supported. DCL does support dialog structures.

DCL Compiler



All built-in statements, functions, and constants are implicitly declared and compiled in a resource file. Thus, there are no include files required for compiling.





Compiler metacommands are not supported.




The compiler is reentrant. Thus, two DCL scripts can be compiled simultaneously.


DCL Variables

 DCL supports integers, longs, shorts, strings, dialogs, single precision floats, and double precision floats. See [Data Types](#) for more information.

 Variables declared (explicitly or implicitly) within a subroutine are local to that subroutine.

 Variables that are not explicitly given a type (with the `dim` statement or using a type specifier) are given the type `integer`.

 All declared variables are assigned an initial value of 0. For strings, 0 equates to "" (or NULL).


 Internal type conversions are performed automatically between any two numeric quantities. Thus, the script author can freely assign numeric quantities without regard to type conversions. However, it is possible for an overflow error to occur when converting from larger to smaller types. This occurs when the larger type contains a numeric quantity that cannot be represented by the smaller type. For example, the following code will produce a runtime error:

```
dim amount as long
dim quantity as integer

amount = 400123      'assign a value out of range for int
quantity = amount    'attempt to assign to integer
```

Like many runtime errors, the overflow error is trappable.

 Loss of precision is not a runtime error.

 The declaration modifiers `STATIC`, `GLOBAL`, and `SHARED` are not supported.

DCL Expression Evaluation

DCL allows expressions to involve data of different types. When this occurs, the two arguments are converted to be of the same type by promoting the "smaller" of the two data types. For example, DCL will promote the value of `i%` to a `double` in the following expression:

```
result# = i% * d#
```

When evaluating an expression, DCL also looks at the resultant type. During evaluation, the data type of each operand of each subexpression is compared with the data type of the result. Each operand will then be promoted to be the same as the "largest" of the two operands and the result. For example, the following expression results in the value `2.5`, even though the division appears to involve two integer values:

```
d# = 10 / 4          'd# becomes 2.5
```

The expression evaluator realizes that the result of the expression is being assigned to a `double`, and promotes the two integer operands of the division to be of the same type as the result. After the promotion, the division is performed. If this promotion did not occur, then the above expression would result in `2`, which would be incorrect.

DCL Subroutines & Functions



Subroutines and functions are available to any other subroutine or function within the same script.



Arguments to functions and subroutines are passed automatically by reference. Passing by value is accomplished with the `BYVAL` keyword:

```
sub foo(byval a as integer)
:
end sub

sub main
d% = 10
foo d           'pass d by value
end sub
```

If a subroutine declares parameters by reference, then the caller can force a parameter to be passed by value by enclosing the parameter with parentheses:

```
sub foo(a as integer)
:
end sub

sub main
d% = 10
foo(d)         'force 'd' to be passed by value
end sub
```



Forward references are not allowed. The body of a referenced subroutine or function must appear before its reference in the script.



Recursive statements and functions are supported.

DCL Error Handling



DCL supports error handling in a manner that conforms to the Visual Basic error handling model. Individual error numbers may be different.



`ON ERROR` traps are valid only during the execution of the current subroutine or function.



Error traps are saved and restored within a user-defined subroutine or function.

DCL errors fall into three categories:



Trappable error numbers are between 10 and 1000.



Non-trappable error numbers are between 0 and 10. Internal errors and out of memory errors are all non-trappable.



Application specific error numbers are greater than or equal to 1000.

Error numbers will change in the next release of DCL.

DCL uses the Windows floating point emulator WIN87EM.DLL. Floating point exception errors are handled in a standard way using the SDK `__fpmath()` function. When the host application calls the compiler or the runtime, the floating point exception handler of the host application is saved and restored on exit.

DCL Arrays



Dynamic allocation of arrays is supported using both the DIM and the REDIM statements.



Arrays can be declared to contain any fundamental data type.



Arrays can have up to 60 dimensions.



Array dimensions and size can be changed dynamically using the REDIM statement.



Arrays are always passed by reference.

DCL Data Types

integer	Type Declarator	%
	Significant Digits	4
	Size	2 bytes (16 bits)
	Range	-32768 <= X <= 32767
long	Type Declarator	&
	Significant Digits	9
	Size	4 bytes (32 bits)
	Range	-2147483648 <= X <= 2147483647
string	Type Declarator	\$
	Significant Digits	N/A
	Size	1 byte per character
	Range	0 <= LENGTH < 32768
	Note	All strings are variable length. Fixed length strings are not yet supported.
single	Type Declarator	!
	Significant Digits	7
	Size	4 bytes (32 bits)
	Range	Negative Values: -3.402823E38 to -1.401298E-45 Positive Values: 1.401298E-45 to 3.402823E38
double	Type Declarator	#
	Significant Digits	15-16
	Size	8 bytes (64 bits)
	Range	Negative Values: 1.797693134862315E308 to -4.94066E-324 Positive Values: 4.94066E-324 to 1.797693134862315E308

DCL Operator Precedence

Operator	Description	Precedence order
()	parentheses	Highest
^	exponentiation	
-	unary minus	
/, *	multiplication and division	
\	integer division	
mod	modulo	
+, -	addition and subtraction	

=, <>, >, <, <=, >=	relational	
not	logical negation	
and	and	
or	or	
xor	exclusive or	Lowest

DCL Comments

Comments can be added to DCL code in the following manner:



All text between a single quote and the end of the line is ignored:

```
MsgBox "hello" 'display a message box
```



The REM statement causes the compiler to ignore the entire line:

```
REM This is a comment...
```





DCL supports C-style multi-line comment blocks /*...*/, as shown in the following example:

```
MsgBox "Before Comment"
/* This stuff is all commented out.
This line too will be ignored
This is the last line of the comment */
MsgBox "After Comment"
```

The C-style comments cannot be nested.


















DCL Constants

 Numerous symbolic constants that you can use with specific **DCL** commands are defined in the files ADSCON11.EBL and DCL.EBL. The constants defined for a particular **DCL** command are listed in Desktop Control Language Reference. Two constants are defined by **DCL** itself--TRUE and FALSE.

 The following constants are predefined for use with **DCL**:

ATTR_ARCHIVE
ATTR_DIRECTORY
ATTR_HIDDEN
ATTR_NONE
ATTR_NORMAL
ATTR_READONLY
ATTR_SYSTEM
ATTR_VOLUME
ENV_BOTH
ENV_DOS
ENV_WINDOWS
FALSE
NS_ACTIVE
NS_LOGGEDON
PO_LANDSCAPE
PO_PORTRAIT
TRUE
TYPE_DOS
TYPE_WINDOWS
VK_LBUTTON
VK_RBUTTON
VS_VERSION_INFO
WS_MAXIMIZED
WS_MINIMIZED
WS_RESTORED

DCL Limitations

-  Each running script is limited to 64K of data. The data segment contains dynamic variables (such as strings and arrays), constants, an event jump table, and external DLL call information.
-  Strings are limited to 32K in size (32767 bytes).
-  Script source size is limited to 42K.
-  Compiled code size is limited to 64K. The code segment contains the executable pcode.
-  The maximum size of the symbol table (used by runtime errors and the debugger) is 64K.
-  Arrays can have up to 60 dimensions.
-  Variable names are limited to 40 characters.
-  Labels are limited to 40 characters.
-  A given subroutine or function can have up to 100 variables, including variables that are passed.
-  The stack size for the runtime is 2048 bytes.
-  The number of open DDE channels is unlimited (limited only by available memory and system resources).
-  The number of open files is limited to 255, or the operating system limit, whichever is less.
-  The number of characters within a string literal (a string enclosed within quotes) is 255 characters. (Strings can be concatenated using the plus (+) operator with the normal string limit of 32767 characters.)
-  Number of nesting levels is limited by compiler memory.
-  Text file input/output buffer size is 512 bytes.
-  Queue playback buffer size is limited to 64K. With 10 bytes per event, this allows for 6553 events. This memory is buffered in blocks of 100 events (1000 bytes).
-  Each `gosub` requires 2 bytes of the DCL runtime stack.

Desktop Control Language Reference

A
B-C
D
E
F
G
H-K
L
M
N
O
P
Q
R
S
T
U-V
W-Z
Symbols

Command Categories

[Arrays](#)
[Clipboard Manipulation](#)
[Conversions](#)
[Date and Time Functions](#)
[Desktop Modifications](#)
[Dialog Creation](#)
[Dialog Display](#)
[Dialog Manipulation](#)
[Dynamic Data Exchange](#)
[DCL Environment Information](#)
[Environment Statements and Functions](#)
[Error Trapping](#)
[File Input and Output](#)
[Flow Control](#)
[Icons](#)
[Keyboard Manipulation](#)
[Math Statements and Functions](#)
[Menus](#)

Miscellaneous Statements and Functions

Mouse Events

Network Functions

Operators

Printer Manipulation

Procedure Statements

Strings

Variables and Constants

Viewport Window Manipulation

Window Manipulation

Alphabetical Listing

Symbols

'

..

*

=

+

=

-

=

/

=

<

=

<=

<=

<>

=

=

>

=

>=

>=

\

^

=

A

Abs

ActivateControl

AddIni

And

AnswerBox

AppActivate

AppClose

AppFileName\$

AppFind

AppGetActive\$

AppGetPosition

AppGetState

AppHide

AppList

AppMaximize

AppMinimize

AppMove

AppRestore

AppSetState

AppShow

AppSize

AppType

ArrayDims

ArraySort

Asc

AskBox\$
AskPassword\$
Atn
ATTR_ARCHIVE
ATTR_DIRECTORY
ATTR_HIDDEN
ATTR_NONE
ATTR_NORMAL
ATTR_READONLY
ATTR_SYSTEM
ATTR_VOLUME

B -- C

Beep
Begin Dialog
ButtonEnabled
ButtonExists
Call
CancelButton
CDBl
ChDir
ChDrive
CheckBox
CheckboxEnabled
CheckboxExists
Chr\$
CInt
Clipboard\$
ClipboardClear
CLng
Close
Combobox
ComboboxEnabled
ComboboxExists
Command\$
Const
Cos
CSng
CStr
CurDir\$

D

Date\$
DateSerial
DateValue
Day
DDEExecute
DDEInitiate
DDEPoke
DDERequest
DDETerminateAll
DDETerminate
DDETimeOut
Declare
DEFtype

DesktopCascade
DesktopSetColors
DesktopSetWallpaper
DesktopTile
Dialog
Dim
Dir\$
DirExists
DiskDrives
DiskFree
Do...Loop
DoEvents
DoKeys
DCLHomeDir\$
DCLOS\$
DCLVersion\$

E

EditEnabled
EditExists
EnableStopScript
End
ENV_BOTH
ENV_DOS
ENV_WINDOWS
Environ\$
EOF
Erl
Err
Error\$
Error
Exclusive
Exit Do
Exit For
Exit Function
Exit Sub
Exp

F

FALSE
FileAttr
FileCopy
FileDateTime
FileDirs
FileExists
FileLen
FileList
FileParse
FileType
FindFile\$
Fix
For...Next
FreeFile
Function...End Function

G

GetAttr
GetCheckbox
GetComboboxItem\$
GetComboboxItemCount
GetEditText\$
GetEnv
GetListboxItem\$
GetListboxItemCount
GetOption
GoSub
Goto
GroupBox

H -- K

Hex\$
HLine
Hour
HPage
HScroll
If...Then...Else
Input #
Input\$
InputBox\$
InStr
Int
Item\$
ItemCount
Kill

L

LBound
LCase\$
Left\$
Len
Let
Line\$
LineCount
LineInput #
ListBox
ListboxEnabled
ListboxExists
LOF
Log
LTrim\$

M

Main
MCI
Menu
MenuItemChecked
MenuItemEnabled
MenuItemExists
Mid\$
Minute

MkDir
Mod
Month
MsgBox
MsgClose
MsgOpen
MsgSetText
MsgSetThermometer

N

Name
NetAttach
NetConnectDrive
NetDetach
NetDirectoryRights
NetDisconnectDrive
NetGetDirectoryRights
NetMemberOf
NetStationID
NetUserName
NetworkStatus
Not
Now
NS_ACTIVE
NS_LOGGEDON
Null

O

Oct\$
OKButton
On Error
Open
OpenFileName\$
Option Base
OptionButton
OptionEnabled
OptionExists
OptionGroup
Or

P

PI
PO_LANDSCAPE
PO_PORTRAIT
PopupMenu
Print #
Print
PrinterGetOrientation
PrinterSetOrientation
PrintFile
PushButton

Q

QueEmpty
QueFlush

QueKeyDn
QueKeys
QueKeyUp
QueMouseClicked
QueMouseDbIClk
QueMouseDbIDn
QueMouseDn
QueMouseMove
QueMouseUp
QueSetRelativeWindow

R

Random
Randomize
ReadINI\$
ReadINISection
ReDim
RefreshIni
REM
Reset
RestoreEnv
Resume
Return
Right\$
Rmdir
Rnd
RTrim\$

S

SaveEnv
SaveFileName\$
Second
Seek
Select...Case
SelectBox
SelectButton
SelectComboboxItem
SelectListboxItem
SendKeys
SetAttr
SetCheckbox
SetEditText
SetEnv
SetIcon
SetIconTitle
SetOption
Sgn
Shell
ShowIcon
Sin
Sleep
SleepUntil
Snapshot
Space\$
Sqr

Stop
Str\$
StrComp
String\$
StringSort
Sub...End Sub
SystemFreeMemory
SystemFreeResources
SystemMouseTrails
SystemRestart
SystemTotalMemory
SystemWindowsDirectory\$
SystemWindowsVersion\$

T

Tan
TextBox
Text
Time
Timer
TimeSerial
TimeValue
Trim\$
TRUE
TYPE_DOS
TYPE_WINDOWS

U -- V











UBound
UCase\$
Val
ViewportClear
ViewportClose
ViewportOpen
VK_LBUTTON
VK_RBUTTON
VLine
VPage
VScroll

W -- Z




WaitForTaskCompletion
Weekday
While...Wend
WinActivate
WinClose
WinFind
WinList
WinMaximize
WinMinimize
WinMove
WinRestore
WinSize
Word\$
WordCount

Write #
WriteINI
WS_MAXIMIZED
WS_MINIMIZED
WS_RESTORED
Xor
Year



Arrays












Description	Function/Statement
Change default lower limit	 <u>Option Base</u>
Declare and initialize	 <u>Dim</u>
	 <u>FileDir</u>
	 <u>FileList</u>
	 <u>ReDim</u>
	 <u>ReadINISection</u>
Find the limits	 <u>LBound</u>
	 <u>UBound</u>
Manipulate an array	 <u>ArrayDims</u>
	 <u>ArraySort</u>

Clipboard Manipulation










Description	Function/Statement
Capture a screen or window to the clipboard	 <u>Snapshot</u>
Clear the clipboard object	 <u>ClipboardClear</u>
Get contents of clipboard	 <u>Clipboard\$</u>

Conversions

Description	Function/Statement
ANSI value to string	 <u>Chr\$</u>
Date to serial number	 <u>DateSerial</u>

	 <u>DateValue</u>
Number to string	 <u>CStr</u>
	 <u>Hex\$</u>
	 <u>Oct\$</u>
	 <u>Str\$</u>
One numeric type to another	 <u>CDbl</u>
	 <u>CInt</u>
	 <u>CLng</u>
	 <u>CSng</u>
String to ASCII value	 <u>Asc</u>
String to number	 <u>Val</u>

Date and Time Functions

Description	Function/Statement
Date to serial number	 <u>DateSerial</u>
	 <u>DateValue</u>
Get the current date or time	 <u>Date\$</u>
	 <u>Now</u>
	 <u>Time\$</u>
Serial number to date	 <u>Day</u>
	 <u>Month</u>
	 <u>Weekday</u>
	 <u>Year</u>

Serial number to time



Hour



Minute



Second

Set the date or time



Date\$



Time\$

Time a process



Timer

Time to serial number



TimeSerial



TimeValue

Desktop Modifications



DesktopCascade



DesktopSetColors



DesktopSetWallpaper



DesktopTile

Dialog Creation



Begin Dialog...EndDialog



CancelButton



Checkbox



Combobox



Dialog



Dim



GroupBox



ListBox



OKButton



OptionButton



OptionGroup



PushButton



Text



TextBox

Dialog Display



AnswerBox



AskBox\$



AskPassword\$



InputBox\$



MsgBox



MsgClose



MsgOpen



MsgSetText



MsgSetThermometer



OpenFileName\$



PopupMenu



SaveFileName\$



SelectBox

Dialog Manipulation



ActivateControl



ButtonEnabled



ButtonExists



CheckboxEnabled



CheckboxExists



ComboboxEnabled



ComboboxExists



EditEnabled



EditExists



GetCheckbox



GetComboboxItem\$



GetComboboxItemCount



GetEditText\$



GetListboxItem\$



GetListboxItemCount



GetOption



ListboxEnabled



ListboxExists



OptionEnabled



OptionExists



SelectButton



SelectComboboxItem



SelectListboxItem



SetCheckbox



SetEditText



SetOption

Dynamic Data Exchange (DDE)



DDEExecute



DDEInitiate



DDEPoke



DDERequest



DDETerminate



DDETerminateAll



DDETimeOut

DCL Environment Information



DCLHomeDir\$



DCLOS\$









DCLVersion\$















Environment Statements and Functions























Description	Function/Statement
Control an MCI device	 <u>MCI</u>
End the working session	 <u>SystemRestart</u>
Get information about the system	 <u>ReadINI\$</u>
	 <u>ReadINISection</u>
	 <u>SystemFreeMemory</u>
	 <u>SystemFreeResources</u>
	 <u>SystemTotalMemory</u>
	 <u>SystemWindowsDirectory\$</u>
	 <u>SystemWindowsVersion\$</u>
Modify the Windows environment	 <u>AddIni</u>
	 <u>RefreshIni</u>
	 <u>SystemMouseTrails</u>
	 <u>WriteINI</u>
Save and restore the environment	 <u>RestoreEnv</u>
	 <u>SaveEnv</u>
Set and get environment variables	 <u>Environ\$</u>
	 <u>GetEnv</u>
	 <u>SetEnv</u>

Error Trapping

Description	Function/Statement
Get error messages	 <u>Error\$</u>
Get error-status data	 <u>Erl</u>
Get or set error-status data	 <u>Err</u>
Simulate run-time errors	 <u>Error</u>
Trap errors while a program is running	 <u>On Error</u>
	 <u>Resume</u>

File Input and Output

Description	Function/Statement
Access or create a file	 <u>Open</u>
Close files	 <u>Close</u>
	 <u>Reset</u>
Copy a file	 <u>FileCopy</u>
Get information about a file	 <u>EOF</u>
	 <u>FileAttr</u>
	 <u>FileDateTime</u>
	 <u>FileExists</u>
	 <u>FileLen</u>
	 <u>FileList</u>
	 <u>FileType</u>
	 <u>FindFile\$</u>
	 <u>FreeFile</u>
	



















Manage disk drives or directories	<u>Loc</u>
	
	<u>LOF</u>
	
	<u>Seek</u>
	
	<u>ChDir</u>
	
	<u>ChDrive</u>
	
	<u>CurDir\$</u>
	
Manage files	<u>DiskDrives</u>
	
	<u>DiskFree</u>
	
	<u>MkDir</u>
	
	<u>RmDir</u>
	
	<u>DirExists</u>
	
	<u>Dir\$</u>
	
Read from a file	<u>FileDirs</u>
	
	<u>FileParse</u>
	
	<u>Kill</u>
	
Set or get file attributes	<u>Name</u>
	
	<u>Input #</u>
	
	<u>Input\$</u>
Set read-write position in a file	
	<u>Line Input #</u>
	
	<u>GetAttr</u>
Write to a file	
	<u>SetAttr</u>
	
	<u>Seek</u>
	

Print #



Write #

Flow Control

Description	Function/Statement
Branch	 <u>GoSub...Return</u>
	 <u>Goto</u>
	 <u>On Error</u>
	 <u>EnableStopScript</u>
Exit or pause the program	 <u>End</u>
	 <u>SleepUntil</u>
	 <u>Stop</u>
	 <u>WaitForTaskCompletion</u>
	 <u>Do...Loop</u>
Loop	 <u>Exit Do</u>
	 <u>Exit For</u>
	 <u>For...Next</u>
	 <u>While...Wend</u>
	 <u>If...Then...Else</u>
Make decisions	 <u>Select Case</u>
	 <u>Sleep</u>
Pause script	 <u>Exclusive</u>
Prevent other applications	 <u>Shell</u>
Run another program	

Yield control to other applications



DoEvents

Icons



SetIcon










SetIconTitle













ShowIcon

Keyboard Manipulation

Description	Function/Statement
Play back a string of keystrokes	 <u>DoKeys</u>
Record and play keystrokes, using the event queue	 <u>QueEmpty</u>  <u>QueFlush</u>  <u>QueKeyDn</u>  <u>QueKeys</u>  <u>QueKeyUp</u>  <u>SendKeys</u>

Math Statements and Functions

Description	Function/Statement
General calculations	 <u>Exp</u>  <u>Log</u>  <u>Sqr</u>
Generate random numbers	 <u>Random</u>  <u>Randomize</u>  <u>Rnd</u>
Get absolute value	 <u>Abs</u>
Get the sign of an expression	 <u>Sgn</u>
Numeric conversion	 <u>Fix</u>  <u>Int</u>

Trigonometry



Atn



Cos



Sin



Tan

Menus

Description

Issue menu command



Menu

Manipulate menu items



MenuItemChecked



MenuItemEnabled



MenuItemExists

Miscellaneous Statements and Functions

Description

Comment



Rem



,

=

Get command-line arguments



Command\$

Sound a beep



Beep

Mouse Events



QueEmpty



QueFlush



QueMouseClicked



QueMouseDown



QueMouseDbIDn



QueMouseDn



QueMouseMove



QueMouseUp



QueSetRelativeWindow

Network Functions



NetAttach



NetConnectDrive



NetDetach



NetDirectoryRights



NetDisconnectDrive



NetGetDirectoryRights



NetMemberOf



NetStationID









NetUserName







NetworkStatus

Operators

Description Function/Statement





Arithmetic

*
=

+
=

-
=

/
=

\
=

^
=

Mod



Comparison

<
=

<=

>
=

>=

<>
=

=

Logical

And

Not

Or

Xor








Printer Manipulation


PrinterGetOrientation

PrinterSetOrientation















PrintFile























Procedure Statements

Description	Function/Statement
Call a subroutine	 <u>Call</u>
Declare a reference to an external routine	 <u>Declare</u>
Define a procedure	 <u>Function...End Function</u>  <u>Main</u>  <u>Sub...End Sub</u>
Exit a procedure	 <u>Exit Function</u>  <u>Exit Sub</u>

Strings

Description	Function/Statement
Convert to lowercase or uppercase letters	 <u>LCase\$</u>  <u>UCase\$</u>
Convert string to number	 <u>Val</u>
Create strings of repeating characters	 <u>Space\$</u>  <u>String\$</u>
Find the length of a string	 <u>Len</u>
Logical operators used in string comparisons	 <  =<  <=  <>  = 

Manipulate strings

>

>=

InStr

Left\$

LTrim\$

Mid\$

Null

Right\$

RTrim\$

StrComp

StringSort

Str\$

Trim\$

Item\$

ItemCount

Line\$

LineCount

Word\$

WordCount

Asc

Chr\$

<







Parsing

Work with ASCII and ANSI
values

Logical operators used in string
comparisons



Variables and Constants

Description	Function/Statement
Assign value	 <u>Let</u>
Declare variables or constants	 <u>Const</u>
	 <u>Dim</u>
	 <u>ReDim</u>
Set default data type	 <u>Deftype</u>

Viewport Window Manipulation



Print



ViewportClear



ViewportClose



ViewportOpen

Window Manipulation



AppActivate



AppClose



AppFileName\$



AppFind



AppGetActive\$



AppGetPosition



AppGetState



AppHide



AppList



AppMaximize



AppMinimize



AppMove



AppRestore



AppSetState



AppShow



AppSize



AppType



HLine



HPage



HScroll



VLine



VPage



VScroll



WinActivate



WinClose



WinFind



WinList



WinMaximize



WinMinimize



WinMove





WinRestore




WinSize


'

Description	Begins a comment line or comments the remainder of the current line.
Syntax	' text
Comments	Causes the compiler to skip all characters between this character and the end of the current line.
See Also	 Comments  REM Statement


*

Description	Multiplication operator.
Syntax	expression1 * expression2
Returns	Returns the product of expression1 and expression2.
See Also	 Operators


+

Description	Addition operator.
Syntax	expression1 + expression2
Returns	Returns the sum of expression1 and expression2.
See Also	 Operators

-



Description	Subtraction operator.
Syntax	expression1 - expression2
Returns	Returns the difference between expression1 and expression2.
See Also	 Operators

/



Description	Long Division operator.
Syntax	expression1 / expression2
Returns	Returns the quotient of expression1 and expression2.
See Also	

Operators



<

Description	Less Than operator.
Syntax	<code>expression1 < expression2</code>
Returns	TRUE if <code>expression1</code> is less than <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is less than <code>expression2</code> .
See Also	 Operators  Strings



<=

Description	Less Than or Equal To operator.
Syntax	<code>expression1 <= expression2</code>
Returns	TRUE if <code>expression1</code> is less than or equal to <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is less than or equal to <code>expression2</code> .
See Also	 Operators  Strings



<>

Description	Not Equal operator.
Syntax	<code>expression1 <> expression2</code>
Returns	TRUE if <code>expression1</code> is not equal to <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is not equal to <code>expression2</code> .
See Also	 Operators  Strings



=

Description	Equal To operator.
Syntax	<code>expression1 = expression2</code>
Returns	TRUE if <code>expression1</code> is equal to <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is equal to <code>expression2</code> .
See Also	 Operators  Strings

>

Description	Greater Than operator.
Syntax	<code>expression1 > expression2</code>
Returns	TRUE if <code>expression1</code> is greater than <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is greater than <code>expression2</code> .
See Also	 Operators  Strings

>=

Description	Greater Than or Equal To operator.
Syntax	<code>expression1 >= expression2</code>
Returns	TRUE if <code>expression1</code> is greater than or equal to <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is greater than or equal to <code>expression2</code> .
See Also	 Operators  Strings

\

Description	Integer Division operator.
Syntax	<code>expression1 \ expression2</code>
Returns	Returns the integer portion of the quotient of <code>expression1</code> divided by <code>expression2</code> . Any fractional part of the answer is

dropped. For example `3 \ 1` equals 1 rather than 1.5.

See Also



[Operators](#)

^

Description Exponentiation operator.

Syntax `expression1 ^ expression2`

Returns Returns `expression1` raised to the power specified by `expression2`.

See Also



[Operators](#)

Abs Function

Description Returns the absolute value of a given number.

Syntax `abs (number)`

Comment `number` is an integer, long integer, single-precision, or double-precision value.

Example



[Abs Example](#)

See Also



[Math Statements and Functions](#)

ActivateControl Statement



Description Sets the focus to the control with the specified name or ID.

Syntax 1 `ActivateControl ControlName$`

Syntax 2 `ActivateControl ControlID%`

Comments The control can be referenced using either the `ControlName$` or the `ControlID%`.

For push buttons, radio buttons, or check boxes, `ControlName$` is the name of the actual button. For listboxes, comboboxes, and edit boxes, `ControlName$` is the name that appears within the static text control that immediately precedes it in the window manager list.

A runtime error is generated if a control with `ControlName$` or `ControlID%` cannot be found.

This statement is used primarily to set the focus to a custom control within a dialog box. This is accomplished by setting the focus first to a known control that immediately precedes the custom control, then simulating a TAB key press:

```
ActivateControl "Portrait"  
DoKeys "{TAB}"
```


Example



[Dialog Examples](#)

See Also



[Dialog Manipulation](#)

AddIni Function



Description

Reads the .INI settings of the source .INI file and adds them to the destination .INI file.

Syntax

```
AddIni(srcfile$[, destfile$])
```

Comments

If the `destfile$` is not provided, WIN.INI is used. If the specified destination file does not exist, it is created. If an .INI setting is not present in the destination file, it is added. The function returns TRUE if it completes successfully or FALSE if it doesn't.

It is assumed that all entries in the `srcfile$` .INI file are in standard .INI format. Any entries that do not follow this Windows standard are ignored. If the specified source file does not exist, the function ends.

Example



[AddIni Example](#)

See Also



[Environment Statements and Functions](#)

And

Description

And operator.

Syntax

```
expression1 And expression2
```

Returns

TRUE if `expression1` is TRUE and `expression2` is TRUE, otherwise FALSE.

If the two operands are numeric, then the result is the bitwise AND of the two arguments.

Notes

If either of the two operands is a floating point number, then the two operands are first converted to longs, then a bitwise AND is performed.

Example



[And Example](#)

See Also



[Operators](#)

AnswerBox Function

Description

Displays a box that prompts the user for a response and indicates which button the user pressed.

Syntax

```
AnswerBox(prompt$ [,button1$ [,button2$
```

```
[,button3$]]))
```

Returns Returns an integer indicating which button was pushed (1 for the first button, 2 for the second, and so on). 0 is returned if the user cancels the dialog box (by pressing Escape).

Comments The dialog box is sized to hold the entire contents of `prompt$`. The `prompt$` string can contain CR/LF--`Chr$(13)+Chr$(10)`--to separate lines.

The maximum size of the dialog box is 5/8 the width of the screen and 3/4 the height of the screen. If a given line is too long, it will be word wrapped.

The `button$` parameters specify the labels for one or more buttons to appear below the displayed `prompt$`. If no buttons are specified, then "OK" and "Cancel" are used. Up to three buttons can be specified. The width of each button is determined by the width of the widest button.

You can use the '&' symbol to specify an accelerator key in the label of the button.

The dialog box uses the 8 point Helvetica font.

Example



[AnswerBox Example](#)

See Also



[Dialog Display](#)

AppActivate Statement



Description Activates the specified top-level window.

Syntax `AppActivate WindowName$`

Comments The `WindowName$` parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If the application is minimized, it will be restored by this command. If the window being activated is a full-screen DOS application, that application will be restored to full screen.

A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box) or if the window is not found.

Example



[AppActivate Example](#)

See Also





[Window Manipulation](#)

AppClose Statement





Description Closes the specified top-level application.



Syntax `AppClose [WindowName$]`

Comments	<p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the focus is used (i.e., the active window).</p> <p>A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).</p>
Example	 <p>AppClose Example</p>
See Also	 <p>Window Manipulation</p>

AppFileName\$ Function

Description	Returns the filename of the program that owns the top-level window of the given title.
Syntax	<code>AppFileName\$ (WindowName\$)</code>
Comments	<p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>For DOS applications, the filename returned is that of the actual executable program, not WINOLDAP.EXE.</p>
Example	 <p>AppFileName\$ Example</p>
See Also	 <p>Window Manipulation</p>

AppFind Function

Description	Returns the full name of the top-level window, given a partial window name.
Syntax	<code>AppFind\$ (partial_name\$)</code>
Comments	The name is specified using the same format as the WinActivate statement.
Example	 <p>AppFind Example</p>
See Also	 <p>Window Manipulation</p>

AppGetActive\$ Function

Description	Returns the title of the active window.
Syntax	<code>AppGetActive\$ ()</code>
Comments	This function is used to retrieve the title of the active top-level window. The returned value can be used in subsequent calls

to routines that require a title to a window.

If "" is returned, no window is active. This is a rare occurrence, it indicates that Windows may be in an unstable state.

Example 1



AppGetActive\$ Example

Example 2

```
n$ = AppGetActive$()  
AppMinimize n$
```

See Also



Window Manipulation

AppGetPosition Statement



Description

Gets the position of a specified top-level window.

Syntax

```
AppGetPosition x%,y%,width%,height%[,WindowName$]
```

Comments

The numeric arguments are filled with the pixel position of the window on the display. If an argument is not a variable reference, then the argument is ignored, as in the following example which only retrieves the position and ignores the width and height:

```
dim x as integer,y as integer  
AppGetPosition x,y,0,0,"Program Manager"
```

The WindowName\$ parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If WindowName\$ parameter is missing, then the window with the focus is used (i.e., the active window).

Example



AppGetPosition Example

See Also



Window Manipulation

AppGetState Function



Description

Returns an integer representing the state of the top-level window.

Syntax

```
AppGetState(WindowName$)
```

Returns

WS_MAXIMIZED	window is maximized
WS_MINIMIZED	window is minimized
WS_RESTORED	window is restored

Comments

The WindowNme\$ parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If WindowName\$ parameter is missing, then the window with the focus is used (i.e., the active window).

Example





AppGetState Example

See Also





Window Manipulation

AppHide Statement

Description	Hides the specified top-level window.
Syntax	<code>AppHide [WindowName\$]</code>
Comments	<p>Nothing happens if the window is already hidden.</p> <p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the focus is used (i.e., the active window).</p> <p>A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).</p>
Example	 AppHide Example
See Also	 Window Manipulation

AppList Statement

Description	Fills the specified string array with the names of all the active applications.
Syntax	<code>AppList ArrayOfAppNames\$</code>
Comments	<p>Before calling this function, you must declare the array to contain the names of the applications. Use the Dim command.</p> <p>After calling this function, use the lbound() and ubound() functions to determine the new size of the array.</p>
Example	 AppList Example
See Also	 Window Manipulation

AppMaximize Statement

Description	Maximizes the specified top-level window.
Syntax	<code>AppMaximize [WindowName\$]</code>
Comments	<p>Nothing happens if the window is already maximized or is hidden.</p> <p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the focus is used (i.e., the active window).</p> <p>A runtime error results if the window being activated is not</p>

enabled (which is the case if that application currently is displaying a modal dialog box).

Example



[AppMaximize Example](#)

See Also



[Window Manipulation](#)

AppMinimize Statement



Description

Minimizes the specified top-level window.

Syntax

```
AppMinimize [WindowName$]
```

Comments

Nothing happens if the window is already minimized or is hidden.

The `WindowName$` parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If `WindowName$` parameter is missing, the window with the focus is used (i.e., the active window).

A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).

Example



[AppMinimize Example](#)

See Also



[Window Manipulation](#)

AppMove Statement



Description

Sets the position of the specified top-level window to a given `x, y` pixel location.

Syntax

```
AppMove x%,y%[,WindowName$]
```

Comments

This statement has no effect if the top-level window is maximized.

It is valid to specify an `x, y` location that is not visible.

The `WindowName$` parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If `WindowName$` parameter is missing, the window with the focus is used (i.e., the active window).

A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).

Example





[AppMove Example](#)

See Also





Window Manipulation

AppRestore Statement



Description	Restores the specified top-level window.
Syntax	<code>AppRestore [WindowName\$]</code>
Comments	<p>This statement has an effect only if the specified window is either maximized or minimized.</p> <p><code>AppRestore</code> will do nothing if the specified window is hidden.</p> <p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the focus is used (i.e., the active window).</p> <p>A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).</p>
Example	 AppRestore Example
See Also	 Window Manipulation

AppSetState Statement



Description	Sets the state of the specified top-level window.						
Syntax	<code>AppSetState [WindowName\$]</code>						
Comments	<p>This statement sets the state of the specified top-level window to one of the following values:</p> <table><tr><td><code>WS_MAXIMIZED</code></td><td>maximize the window</td></tr><tr><td><code>WS_MINIMIZED</code></td><td>minimize the window</td></tr><tr><td><code>WS_RESTORED</code></td><td>restore the window</td></tr></table> <p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the focus is used (i.e., the active window).</p>	<code>WS_MAXIMIZED</code>	maximize the window	<code>WS_MINIMIZED</code>	minimize the window	<code>WS_RESTORED</code>	restore the window
<code>WS_MAXIMIZED</code>	maximize the window						
<code>WS_MINIMIZED</code>	minimize the window						
<code>WS_RESTORED</code>	restore the window						
Example	 AppSetState Example						
See Also	 Window Manipulation						

AppShow Statement

Description	Shows the specified top-level window.
Syntax	<code>AppShow [WindowName\$]</code>

Comments	<p>Nothing happens if the window is already displayed.</p> <p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the focus is used (i.e., the active window).</p> <p>A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).</p>
Example	 <p>AppShow Example</p>
See Also	 <p>Window Manipulation</p>

AppSize Statement

Description	Sets the width and height of the specified top-level window. It lets you size sizable and non-sizable windows.
Syntax	<code>AppSize width%,height%[,WindowName\$]</code>
Comments	<p>This statement works only if the specified application is restored (i.e., not minimized or maximized).</p> <p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the focus is used (i.e., the active window).</p> <p>A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).</p>
Example	 <p>AppSize Example</p>
See Also	 <p>Window Manipulation</p>

AppType Function

Description	Returns an integer representing the type of application owning the specified window:				
Syntax	<code>AppType [(WindowName\$)]</code>				
Returns	<table> <tr> <td><code>TYPE_DOS</code></td><td>DOS executable</td></tr> <tr> <td><code>TYPE_WINDOWS</code></td><td>Windows executable</td></tr> </table>	<code>TYPE_DOS</code>	DOS executable	<code>TYPE_WINDOWS</code>	Windows executable
<code>TYPE_DOS</code>	DOS executable				
<code>TYPE_WINDOWS</code>	Windows executable				
Comments	<p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the</p>				

focus is used (i.e., the active window).

Example



[AppType Example](#)

See Also



[Window Manipulation](#)

ArrayDims Function

Description

Returns an integer representing the number of dimensions in the specified array.

Syntax

`ArrayDims (arrayvariable)`

Comments

If the function indicates zero dimensions, the array was declared as an empty array (e.g. `dim x$()`).

Example 1



[ArrayDims Example](#)

Example 2

```
sub main()  
dim f$()           'allocate empty array  
files f$, "C:\*.BAT" 'fill the array  
if dims(f$) = 0 then exit sub 'exit if no  
                        'elements  
  
end sub
```

See Also



[Arrays](#)

ArraySort Statement

Description

Sorts a single-dimensioned array.

Syntax 1

`ArraySort s$()`

Syntax 2

`ArraySort a%()`

Syntax 3

`ArraySort a&()`

Syntax 4

`ArraySort a!()`

Syntax 5

`ArraySort a#()`

Comments

If a string array is specified, then the routine sorts alphabetically in ascending order (using case-sensitive string comparisons). If a numeric array is specified, the routine sorts lower numbers to the lowest array index locations.

A runtime error results if an array with more than one dimension is specified.

Example 1



[ArraySort Example](#)

Example 2

```
sub main()  
  dim a$(100)  
  :  
  :  
  ArraySort a$  
  result = SelectBox("Title", "Prompt:", a$)  
end sub
```

See Also



[Arrays](#)

Asc Function

Description Returns the numeric ASCII code for the first character of the specified string.

Syntax `asc(text$)`

Comments The return value is an integer between 0 and 255.

Example



[Asc Example](#)

See Also



[Conversions](#)



[Strings](#)

AskBox\$ Function

Description Displays a dialog box that asks the user to enter a string in an edit box, and returns the string the user enters.

Syntax `AskBox$(prompt$ [,default$])`

Returns Returns a string that the user typed, or returns an empty string indicating that the user canceled the dialog box.

Comments The dialog box is sized to the appropriate width depending on the width of `prompt`.

When the dialog box is displayed, the edit box has the focus.

If `default` is specified, then this is used as the initial contents of the edit field.

The dialog box has OK and Cancel buttons.

The dialog box uses the 8 point Helvetica font.

The maximum number of characters that can be typed into the edit box is 255.

Example



[AskBox\\$ Example](#)

See Also





[Dialog Display](#)

AskPassword\$ Function



Description Displays a dialog box that asks the user to enter his or her password in an edit box, and returns the string the user enters.

Syntax `AskPassword$(prompt$)`

Returns Returns the string that the user typed, or returns an empty string indicating that the user canceled the dialog box.

Comments	<p>Unlike the <code>AskBox</code> command, the user sees asterisks in place of the characters that are actually typed. This allows the input of passwords.</p> <p>When the dialog box is displayed, the edit box has the focus.</p> <p>The dialog box is sized to the appropriate width depending on the width of <code>prompt\$</code>.</p> <p>The dialog box has OK and Cancel buttons.</p> <p>The dialog box uses the 8 point Helvetica font.</p> <p>The maximum number of characters that can be typed into the edit box is 255.</p>
Example	 <p>AskPassword\$ Example</p>
See Also	 <p>Dialog Display</p>

Atn Function

Description	Returns a double-precision number representing the arctangent of a given number.
Syntax	<code>atn (number#)</code>
Comments	<p>To calculate the value of PI, use:</p> $4 * \text{ATN}(1)$
Example	 <p>Atn Example</p>
See Also	 <p>Math Statements and Functions</p>

ATTR_ARCHIVE

Description	Constant.
Value	32
Comments	Bit position of a file attribute indicating that a file hasn't been backed up. This value is used by GetAttr , SetAttr , and FileList .

ATTR_DIRECTORY

Description	Constant.
Value	16
Comments	Bit position of a file attribute indicating that a file is a directory entry. This value is used by GetAttr , SetAttr , and FileList .

ATTR_HIDDEN

Description	Constant.
--------------------	-----------

Value	2
Comments	Bit position of a file attribute indicating that a file is hidden. This value is used by GetAttr , SetAttr , and FileList .

ATTR_NONE

Description	Constant.
Value	64
Comments	Bit position of a file attribute indicating that a file has no other attributes set. This value is used by GetAttr , SetAttr , and FileList .

ATTR_NORMAL

Description	Constant.
Value	0
Comments	Bit position of a file attribute indicating that a file has no other attributes set. This value is used by GetAttr , SetAttr , and FileList .

ATTR_READONLY

Description	Constant.
Value	1
Comments	Bit position of a file attribute indicating that a file is read-only. This value is used by GetAttr , SetAttr , and FileList .

ATTR_SYSTEM

Description	Constant.
Value	4
Comments	Bit position of a file attribute indicating that a file is a system file. This value is used by GetAttr , SetAttr , and FileList .

ATTR_VOLUME

Description	Constant.
Value	8
Comments	Bit position of a file attribute indicating that a file is the volume label. This value is used by GetAttr , SetAttr , and FileList .

Beep Statement

Description	Makes a single system beep.
Syntax	beep

Example



[Beep Example](#)

Begin Dialog...End Dialog Statement

Description	The BEGIN DIALOG...END DIALOG block defines a dialog box template.
Syntax	<pre>begin dialog DialogName\$,x%,y%,width%,height% end dialog</pre>
Comments	<p>The <code>x</code>, <code>y</code> parameters are the dialog coordinates of the upper left hand corner of the dialog box relative to the parent window.</p> <p>The <code>width</code>, <code>height</code> parameters specify the width and height dimensions of the dialog box in dialog units.</p> <p>The <code>DialogName</code> parameter specifies the name used to dimension a variable of this type (i.e., a variable that refers to this dialog box template). Once a dialog template has been defined, a variable can be dimensioned using this name:</p> <pre>dim MyDlg as DialogName</pre> <p>An error is generated if the dialog box template is empty.</p> <p>A dialog template must have at least one PushButton, OKButton, or CancelButton. Otherwise, there will be no way to close the dialog box.</p> <p>Dialog units are defined as 1/4 the width of the font in the horizontal direction and 1/8 the height of the font in the vertical direction. All dialogs created by DCL use an 8 point Helvetica font.</p>

Example



[Dialog Examples](#)

See Also



[Dialog Creation](#)

ButtonEnabled Function



Description	Determines whether the specified button within the current window is enabled.
Syntax 1	<code>ButtonEnabled(ButtonName\$)</code>
Syntax 2	<code>ButtonEnabled(ButtonID%)</code>
Returns	Returns the integer TRUE if the specified button within the current window is enabled, otherwise this function returns FALSE.
Comments	The button can be specified either by its name (<code>ButtonName\$</code>) or using its ID (<code>ButtonID%</code>). <code>ButtonName\$</code> is the text on the button's label.

When a button is enabled, it can be pressed using the [SelectButton](#) statement.

Example



[ButtonEnabled Example](#)

See Also



[Dialog Manipulation](#)

ButtonExists Function



Description

Determines whether the specified button exists within the current window.

Syntax 1

`ButtonExists (ButtonName$)`

Syntax 2

`ButtonExists (ButtonID%)`

Returns

Returns the integer TRUE if the specified button exists within the current window, otherwise this function returns FALSE.

Comments

The button can be specified either by its name (`ButtonName$`) or using its id (`ButtonID%`). The `ButtonName$` is the text of button's label.

Example



[ButtonExists Example](#)

See Also



[Dialog Manipulation](#)

Call Statement

Description

Transfers control to the specified subroutine, optionally passing arguments.

Syntax

`call subroutine_name [(arguments)]`

Comments

Using this statement is equivalent to:

`subroutine_name [arguments]`

Use of the `call` statement is never required.

Example



[Call Example](#)

See Also



[Procedure Statements](#)

CancelButton Statement

Description

Defines a Cancel button that appears within a dialog box template.



Syntax

`CancelButton x%,y%,width%,height%`



Comments

This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).



The `x`, `y`, `width`, `height` parameters are specified in dialog coordinates. The `x`, `y` position is relative to the upper left

	corner of the dialog box.
Example	 Dialog Examples
See Also	 Dialog Creation



CDbl Function

Description	Returns the double-precision equivalent of the passed numeric expression.
Syntax	<code>cdbl (number#)</code>
Comments	This function has the same effect as assigning the numeric expression to a double precision variable.
Example	 CDbl Example
See Also	 Conversions



ChDir Statement

Description	Changes the current directory on the specified drive.
Syntax	<code>chdir newdir\$</code>
Comments	This statement will not change to the specified drive. If you do not include a drive in the <code>newdir\$</code> parameter, the current drive is assumed. This statement behaves the same as the DOS "cd" command.
Example	 ChDir Example
See Also	 File Input and Output



ChDrive Statement

Description	Changes the default drive.
Syntax	<code>chdrive DriveLetter\$</code>
Comments	Only the first character of <code>DriveLetter\$</code> is used. You can use the same string for the <code>ChDrive</code> and ChDir commands. <code>DriveLetter\$</code> is case insensitive. If <code>DriveLetter\$</code> is empty, then the current drive is not changed.
Example	 ChDrive Example
See Also	 File Input and Output



CheckBox Statement

Description	Defines a checkbox within a dialog box template.
Syntax	<code>CheckBox x%,y%,width%,height%,title\$, .Field</code>
Comments	<p>Checkboxes can be either on or off, depending on the value of <code>.Field</code>.</p> <p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>The <code>x</code>, <code>y</code>, <code>width</code>, <code>height</code> parameters are specified in dialog coordinates. The <code>x</code>, <code>y</code> position is relative to the upper left corner of the dialog box. The <code>width</code> and <code>height</code> parameters specify the dimensions of the check box and the label.</p> <p>On entry to the <code>Dialog</code> statement, the <code>.Field</code> variable is used to set the initial state of the checkbox. On exit from the <code>Dialog</code> statement, the <code>.Field</code> variable is used to determine the final state of the checkbox. If the value is 0, the checkbox is unchecked; 1 indicates that the checkbox is checked.</p> <p>The <code>title\$</code> parameter may contain an ampersand character to denote an underlined accelerator, such as "&Font" for <code>F</code>ont.</p>
Example	 Dialog Examples
See Also	 Dialog Creation




CheckboxEnabled Function

Description	Determines whether the specified checkbox in the current window is enabled.
Syntax 1	<code>CheckboxEnabled (CheckboxName\$)</code>
Syntax 2	<code>CheckboxEnabled (CheckboxID%)</code>
Returns	Returns the integer TRUE if the specified checkbox within the current window is enabled, otherwise this function returns FALSE.
Comments	<p>The checkbox can be specified either by its name (<code>CheckboxName\$</code>) or using its id (<code>CheckboxID%</code>). <code>CheckboxName\$</code> is the text of the check box label.</p> <p>When a checkbox is enabled, its state can be set using the <code>SetCheckbox</code> statement.</p>
Example	 Dialog Examples
See Also	 Dialog Manipulation


CheckboxExists Function

Description	Determines whether the specified checkbox exists within the current window.
Syntax 1	<code>CheckboxExists (CheckboxName\$)</code>
Syntax 2	<code>CheckboxExists (CheckboxID%)</code>
Returns	Returns the integer TRUE if the specified checkbox exists within the current window, otherwise this function returns FALSE.
Comments	The checkbox can be specified either by its name (<code>CheckboxName\$</code>) or using its ID (<code>CheckboxID%</code>). <code>CheckboxName\$</code> is the text of the check box label.
Example	 Dialog Examples
See Also	 Dialog Manipulation

Chr\$ Function

Description	Returns the character for the specified ASCII code.
Syntax	<code>chr\$ (AsciiCode%)</code>
Comments	<code>AsciiCode</code> must be an integer between 0 and 255. The <code>Chr\$()</code> function can be used within constant declarations, as in the following example: <code>const crlf\$ = chr\$(13) + chr\$(10)</code>
Example	 Chr\$ Example
See Also	 Conversions  Strings

CInt Function

Description	Returns the integer portion of the given expression. In cases of fractions, this function rounds to the nearest integer.
Syntax	<code>cint (number#)</code>
Comments	The passed numeric expression must be within the following range: <code>-32768 <= number <= 32767</code> A runtime error results if the passed expression is not within the above range. This function has the same effect as assigning a numeric expression to a variable of type integer.
Example	 CInt Example

See Also



[Conversions](#)





[Fix](#)





[Int](#)

Clipboard\$ Statement and Function

Description	The <code>Clipboard\$</code> statement puts a string in the clipboard. The <code>Clipboard\$</code> function returns the text contained in the clipboard.
Statement Syntax	<code>clipboard\$ NewContent\$</code>
Comments	This statement puts <code>NewContent\$</code> into the clipboard.
Function Syntax	<code>clipboard\$()</code>
Returns	Text contained in the clipboard.
Comments	If the clipboard doesn't contain text, or the clipboard is empty, then an empty string is returned.
Example	 Clipboard\$ Example
See Also	 Clipboard Manipulation

ClipboardClear Statement

Description	Clears (removes the contents of) the clipboard.
Syntax	<code>ClipboardClear</code>
Example	 ClipboardClear Example
See Also	 Clipboard Manipulation

CLng Function

Description	Returns a long integer representing the result of the given numeric expression.
Syntax	<code>clng(number#)</code>
Comments	The passed numeric expression must be within the following range: <code>-2147483648 <= number <= 2147483647</code> A runtime error results if the passed expression is not within the above range.

This function has the same effect as assigning a numeric expression to a long variable.

Example



[CLng Example](#)

See Also



[Conversions](#)

Close Statement

Description

Closes open file(s).

Syntax

```
close [[#] filename% [, [#] filename%]]
```

Comments

The file number is assigned by the [Open](#) command.

If no arguments are specified, this statement closes all files. Otherwise, this statement closes each specified file.

Example



[Input/Output Example](#)

See Also



[File Input and Output](#)

Combobox Statement

Description

Defines a combobox that appears within a dialog box template.

Syntax

```
ComboBox x%,y%,width%,height%,items$(),.Field
```

Comments

The `items$` array must be a single-dimension array of strings. The elements of this array are placed into the combobox when the dialog box is created. The `.Field` parameter defines the name used to extract which string occupies the combobox when the dialog box ends. On exit from the [Dialog](#) statement, the `.Field` contains an index to the item that is highlighted in the combobox.

This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).

The `x,y,width,height` parameters are specified in dialog coordinates. The `x,y` position is relative to the upper left corner of the dialog box. The `width` and `height` parameters define the dimensions of the drop-down list box.

Example



[Dialog Examples](#)

See Also





[Dialog Creation](#)

ComboboxEnabled Function





Description

Determines whether the specified combobox is enabled in the current window or dialog box.

Syntax 1	<code>ComboboxEnabled (name\$)</code>
Syntax 2	<code>ComboboxEnabled (id%)</code>
Comment	<p>The combobox can be specified either by <code>name\$</code> or <code>id%</code>. The <code>name\$</code> specifies the text that appears in the static control that immediately precedes the combobox control in the window list (or dialog template).</p> <p>This function returns the integer TRUE if the given combobox is enabled within the current window or dialog box, FALSE otherwise.</p> <p>A runtime error is generated if the specified combobox does not exist.</p>
Example	 Dialog Examples
See Also	 Dialog Manipulation

ComboboxExists Function


Description	Determines whether the specified combobox exists in the current window or dialog box.
Syntax 1	<code>ComboboxExists (name\$)</code>
Syntax 2	<code>ComboboxExists (id%)</code>
Comment	<p>The combobox can be specified either by <code>name\$</code> or <code>id%</code>. The <code>name\$</code> specifies the text that appears in the static control that immediately precedes the combobox control in the window list (or dialog template).</p> <p>This function returns the integer TRUE if the given combobox exists within the current window or dialog box, FALSE otherwise.</p>
Example	 Dialog Examples
See Also	 Dialog Manipulation

Command\$ Statement


Description	Returns a string representing the arguments from the command line used to start the application.
Syntax	<code>Command\$</code>
Comment	When running scripts from the DCL editor, you enter command-line arguments through the Arguments command.

Const Statement



Description	Declares a constant for use within the current script.
Syntax	<code>const name = expression [,name = expression]...</code>

Comments	The <code>name</code> is only valid within the current DCL script. The <code>expression</code> must be assembled from literals, or other constants. Calls to functions are not allowed.
See Also	 Variables and Constants



Cos Function

Description	Returns a double-precision number representing the cosine of a given angle.
Syntax	<code>cos (angle#)</code>
Comments	The <code>angle</code> parameter is given in radians.
See Also	 Math Statements and Functions

CSng Function

Description	Returns a single-precision number representing the result of the given numeric expression.
Syntax	<code>CSng (number#)</code>
Comments	This function has the same effect as assigning a numeric expression to a single-precision variable.
Example	 CSng Example
See Also	 Conversions

CStr Function

Description	Returns a string representing the result of the given expression.
Syntax	<code>CStr (number#)</code>
Example	 CStr Example
See Also	 Conversions

CurDir\$ Function

Description	Gets the current directory.
Syntax	<code>curdir\$[(drive\$)]</code>
Returns	Returns the current directory on the specified drive. If no drive is specified, the current directory on the current drive is returned.
Comments	A runtime error results if <code>drive\$</code> is invalid.

Example



[CurDir\\$ Example](#)

See Also



[File Input and Output](#)

Date\$ Statement and Function

Description The `date$` assignment statement sets the system date. The `date$` function returns the system date.

Assignment Syntax `date$ = newdate$`

Comments Sets the system date to the specified date. The format for `newdate$` is any of the following:

`MM-DD-YYYY`

`MM-DD-YY`

`MM/DD/YYYY`

`MM/DD/YY`

Example 1



[Date\\$ Statement Example](#)

Example 2

`date$ = "7-28-1992"`

Function Syntax

`date$[()]`

Returns The `date$` function returns the current system date as a 10 character string.

Comments The format for the returned date is `MM-DD-YYYY`.

Example



[Date\\$ Function Example](#)

See Also



[Date and Time Functions](#)

DateSerial Function

Description Returns a double-precision number representing the specified date. The number is returned in days where Dec 30, 1899 is 0.

Syntax `DateSerial(year%,month%,day%)`

Example



[DateSerial Example](#)

See Also






[Conversions](#)





[Date and Time Functions](#)



DateValue Function

Description	Returns a double-precision number representing the date contained in the specified string argument.
Syntax	<code>DateValue (date_string\$)</code>
Comments	<p>This function interprets the passed <code>date_string\$</code> parameter looking for a valid date specification. Date specifications vary depending on the international settings contained in the INTL section of the WIN.INI file.</p> <p>The <code>date_string\$</code> parameter can contain valid date items separated by date separators such as slash (/), minus (-), or comma (.). The date items must follow the ordering determined by the current date format settings in use by Windows.</p> <p>Date strings can contain an optional time specification, but this is not used in the formation of the returned value.</p> <p>Months can appear in their abbreviated formats, such as Jan, Feb, or Mar.</p>
Example	 DateValue Example
See Also	 Conversions  Date and Time Functions



Day Function

Description	Returns an integer representing the day of the month for the date encoded in the specified <code>serial</code> parameter. The value returned is between 1 and 31 inclusive.
Syntax	<code>day(serial#)</code>
Comment	You can obtain the value for the <code>serial#</code> parameter by using the DateSerial or DateValue command.
Example	 Day Example
See Also	 Date and Time Functions



DCLHomeDir\$ Function

Description	Returns the directory containing DCL.
Syntax	<code>DCLhomedir\$()</code>
Comments	This function is used to locate files that are part of the DCL system itself.
Example	 DCLHomeDir\$ Example
See Also	 DCL Environment Information



DCLOS\$ Function

Description	Returns a number indicating the host operating environment.
Syntax	DCLOS\$ ()
Returns	0 Windows 1 DOS
Example	 DCLOS\$ Example
See Also	 DCL Environment Information

DCLVersion\$ Function



Description	Returns the version of DCL.
Syntax	DCLVersion\$ ()
Comments	This function returns the major and minor version numbers in the format <code>major.minor</code> , as in "1.1".
Example	 DCLVersion\$ Example
See Also	 DCL Environment Information

DDEExecute Statement



Description	Sends an execute message to another application.
Syntax	DDEExecute <code>channel%</code> , <code>command\$</code>
Comments	<p>The <code>channel</code> must first be initiated using DDEInitiate. An error will result if <code>channel</code> is invalid.</p> <p>If the receiving application does not execute the instructions, a runtime error will be generated.</p> <p>The format of <code>command\$</code> depends on the receiving application.</p>
Example	 DDE Example
See Also	 Dynamic Data Exchange

DDEInitiate Function



Description	Initializes a DDE link to another application.
Syntax	DDEInitiate(<code>app\$</code> , <code>topic\$</code>)
Returns	Returns a unique integer used to subsequently refer to the open DDE channel.

Comments	<p>The function returns 0 if the link cannot be established. This will occur under the following circumstances:</p> <ul style="list-style-type: none"> • The specified application is not running • The topic was invalid for that application • Insufficient memory or system resources to establish DDE link
Example	 DDE Example
See Also	 Dynamic Data Exchange

DDEPoke Statement



Description	Sets the value of a data item in the receiving application associated with an open DDE link.
Syntax	<code>DDEPoke channel%,dataItem\$,value\$</code>
Comments	<p>The <code>channel</code> must first be initiated using <code>DDEInitiate</code>. An error will result if <code>channel</code> is invalid.</p> <p>The format for <code>dataItem\$</code> and <code>value\$</code> depends on the receiving application.</p>
Example	 DDE Example
See Also	 Dynamic Data Exchange

DDERequest Function



Description	Gets a data item from a receiving application.
Syntax	<code>DDERequest\$(channel%,dataItem\$)</code>
Returns	Returns a string representing the value of the given data item in the receiving application associated with the open DDE channel.
Comments	<p>The <code>channel</code> must first be initiated using DDEInitiate. An error will result if <code>channel</code> is invalid.</p> <p>The formats for <code>dataItem\$</code> and the returned value depend on the receiving application.</p>
Example	 DDE Example
See Also	 Dynamic Data Exchange

DDETerminate Statement



Description	Closes the specified DDE channel.
Syntax	<code>DDETerminate channel%</code>

Comments	<p>The <code>channel</code> must first be initiated using DDEInitiate. An error will result if <code>channel</code> is invalid.</p> <p>All open DDE channels are automatically terminated when the script ends.</p>
Example	 DDE Example
See Also	 Dynamic Data Exchange

DDETerminateAll Statement

Description	Closes all currently open DDE channels.
Syntax	<code>DDETerminateAll</code>
Comments	If you do not issue this statement, all open DDE channels are automatically terminated when the script ends.
Example	 DDE Example
See Also	 Dynamic Data Exchange

DDETimeout Statement

Description	Sets the number of milliseconds that must elapse before a DDE command times out.
Syntax	<code>DDETimeout milliseconds&</code>
Comments	<p>The default is 10000 (10 seconds).</p> <p>The timeout should be set before issuing other DDE commands.</p>
Example	 DDE Example
See Also	 Dynamic Data Exchange

Declare Statement

Description	Declares a subroutine or function within another DLL.
Syntax 1	<code>Declare sub <i>name</i> [lib <i>libname</i>\$ [alias <i>aliasname</i>\$]] [([<i>argumentlist</i>])]</code>
Syntax 2	<code>Declare function <i>name</i> [lib <i>libname</i> [alias <i>aliasname</i>\$]] [[<i>argumentlist</i>]] [as <i>type</i>]</code>
Comments	<p>This statement must precede any call to an external DLL routine.</p> <p>The <code>name</code> parameter is any DCL valid global name. When declaring functions, a type-declaration character can be included to indicate the return type.</p>

The `libname$` needs to be specified along with an optional `aliasname$`. The `libname$` parameter specifies the DLL that contains the external routine. All of the Windows API routines are contained in DLLs, such as "user", "kernel", "gdi", and so on. The file extension ".EXE" is implied if another extension is not given. If the `libname$` parameter is not the real name of the external routine as it appears within the external DLL, then an alias name must be given providing this name. For example, the following two statements declare the same routine.

```
declare function GetCurrentTime lib "user" ()
    as integer

declare function GetTime lib "user" alias
    "GetCurrentTime" as integer
```

Use an alias when the name of an external routine conflicts with the name of an internal routine or if the external routine name contains invalid characters.

The optional `argumentlist` specifies the arguments received by the routine. The argument list must match exactly with that of the referenced routine. Otherwise unpredictable results may occur. By default, DCL passes arguments by reference. Many DLL routines require a value rather than a reference. The `byval` keyword does this. For example, the following C routine:

```
int MessageBeep(int);
```

would be declared as follows:

```
declare sub MessageBeep lib "user"
    (byval n as integer)
```

The following shows how a C routine that requires a pointer to an integer would be declared (notice the missing `byval` keyword on the third parameter):

```
int SystemParametersInfo(int,int,
    int far *,int);

declare function SystemParametersInfo
    lib "user"
    (byval action as integer,
     byval uParam as integer,
     pi as integer,
     byval updateINI as integer)
```

Strings are always passed to DLL routines by reference - the `byval` keyword in this case is unnecessary. If a DLL routine modifies a passed string variable, there must be sufficient space within the string to hold the returned characters. This can be accomplished using the `space$()` function:

```
declare sub GetWindowsDirectory lib "user"
    (dir$,len%)
    :
    :
    dim s as string
    s = space$(128)
    GetWindowsDirectory s,128
```

For function declarations, the return type can be specified using a type declaration character (i.e., \$, %, or &), or by specifying the `[as type]` clause. The valid types are integer, long, and string.

The libraries containing the routines are loaded when the routine is called for the first time (i.e., not when the script is loaded). This allows a script to reference external DLLs that potentially do not exist.

The `declare` statement must appear before the function is used.

The `declare` statements are only valid during the life of that script.

Example



[Declare Example](#)

See Also



[Procedure Statements](#)

DEFtype Statement

Description

This statement controls automatic type declaration of variables.

Syntax

`DEFInt letterrange`

`DEFLng letterrange`

`DEFStr letterrange`

`DEFsng letterrange`

`DEFdbl letterrange`

Comments

Normally, if a variable is encountered that hasn't yet been declared with the `dim` statement, or does not appear with an explicit type declaration character, the variable is declared implicitly as an integer (`DEFInt A-Z`). This can be changed using the `DEFtype` statement to specify starting letter ranges for *type* other than integer. The *letterrange* parameter is used to specify starting letters. Thus, any variable that begins with a specified character will be declared using the specified *type*.

The syntax for *letterrange* is:

`letter [-letter] [,letter [-letter]]...`

`DEFtype` variable types are superseded by an explicit type declaration -- using a type declaration character or using the `dim` statement.

This statement only affects compiling of scripts.

Example



[DEFtype Example](#)

See Also



[Variables and Constants](#)

DesktopCascade Statement

Description Cascades all non-minimized top-level windows.

Syntax DesktopCascade

Example 
[DesktopCascade Example](#)

See Also 
[Desktop Modifications](#)

DesktopSetColors Statement

Description Changes the system colors to one of the predefined color sets in the CONTROL.INI file.

Syntax DesktopSetColors ControlPanelItemName\$

Example 
[DesktopSetColors Example](#)

See Also 
[Desktop Modifications](#)

DesktopSetWallpaper Statement

Description Changes the Windows wallpaper.

Syntax DesktopSetWallPaper filename\$,tile%

Comments This statement changes the Windows wallpaper to the bitmap specified by filename\$. The wallpaper will be tiled is tile is TRUE, otherwise the bitmap will be centered on the desktop.

To remove the wallpaper, set the filename\$ parameter to "":

DesktopSetWallPaper "",true

This statement makes permanent changes to the wallpaper by writing the new wallpaper information to the WIN.INI file.

Example 
[DesktopSetWallpaper Example](#)

See Also 
[Desktop Modifications](#)

DesktopTile Statement




Description Tiles all non-minimized top-level windows.

Syntax DesktopTile

Example 
[DesktopTile Example](#)

See Also 
[Desktop Modifications](#)

Dialog Statement and Function

Description	The <code>Dialog</code> statement displays a dialog box. The <code>Dialog</code> function displays a dialog box and returns an integer representing the button that was pushed.
Statement Syntax	<code>Dialog UserDlg as dialog</code>
Comments	<p>Displays the dialog box specified by the dialog template <code>UserDlg</code>.</p> <p>If the user selects Cancel, a runtime error is generated. This error can be trapped using <code>ON ERROR</code>.</p> <p>The <code>Dialog</code> statement returns only after the user presses OK, Cancel, or another push button.</p>
Function Syntax	<code>Dialog(UserDlg as dialog)</code>
Returns	<p>-1 OK button was pushed.</p> <p>0 Cancel button was pushed.</p> <p>>0 A push button was selected. The returned number represents which button was pushed based on its order in the dialog template. 1 represents the first button, 2 represents the second, etc.</p>
Comments	<p>This function displays the dialog box associated with the template <code>UserDlg</code>, and returns which button was pushed. Unlike the statement form, this function will not generate a runtime error when Cancel is selected.</p>
Examples	 Dialog Statement and Function Example  A Comprehensive Dialog Example
See Also	 Dialog Creation

Dim Statement

Description	<p>Declares a list of variables and their corresponding types and sizes.</p> <p>A special form of the DIM statement is used to store a dialog definition in memory.</p>
Syntax	<code>dim name [(_{<subscripts>})] [as type] [,name [(_{<subscripts>})] [as type]]...</code>
Comments	<p>If a type declaration character is used (such as %, &, or \$), the optional <code>[as type]</code> expression is not allowed.</p> <p>Up to 60 array dimensions are allowed.</p> <p>The total size of an array (not counting space for strings) is limited to 42K.</p> <p>Dynamic arrays are declared by not specifying any bounds:</p> <pre>dim a()</pre>

Any DIM statements declared within a subroutine or function are local to that subroutine or function.

If a variable is seen that has not been explicitly declared with DIM, it is implicitly declared using the type specifier character (% , \$, or &). If this character is missing, then integer is assumed.

Example



[Dim Example](#)

Use with Dialogs

A special form of the DIM statement stores the preceding dialog definition ([BeginDialog...EndDialog](#)) in memory.

The syntax of this statement is:

```
dim dialog_name as UserDialog
```

Where `dialog_name` is a variable you define and `UserDialog` is a DCL reserved word.

For an example of this command, see [Dialog Examples](#).

See Also



[Arrays](#)



[Dialog Creation](#)



[ReDim](#)



[Variables and Constants](#)

DirExists Function



Description

Determines whether the specified `path$` exists and is a directory.

Syntax

```
DirExists(path$)
```

Comments

The function returns the integer TRUE or FALSE.

The path can be a partial path, such as windows.

Example



[DirExists Example](#)

See Also



[File Input and Output](#)

Dir\$ Function

Description

Searches a disk directory.

Syntax

```
dir$([filespec$])
```

Returns

If `filespec$` is specified, then this function returns the first file matching that `filespec`. If `filespec$` is not specified, then this function returns the next file matching the initial `filespec`.

Comments The `filespec$` argument can include wildcards, such as `*` and `?`.

An error is generated if `dir$` is called without first calling it with a valid `filespec`.

If there is no matching `filespec`, then an empty string is returned.

If no path is specified on `filespec$`, then the current directory is used.

This function does not find hidden files and directories.

Example 
[Dir\\$ Example](#)

See Also 
[File Input and Output](#)

DiskDrives Statement

Description This statement grabs all of the valid drive letters and packs them into the specified array.

Syntax `DiskDrives list$()`

Comments The array is resized to hold the exact number of valid drives.

The `list$` parameter must be a single dimension array of strings.

Use the functions [lbound\(\)](#) and [ubound\(\)](#) to determine the size of the resultant array.

Example 
[DiskDrives Example](#)

See Also 
[File Input and Output](#)

DiskFree Function

Description Returns a long integer representing the free space (in bytes) available on the specified drive.

Syntax `DiskFree([drive$])`



Comments If `drive$` is empty or not specified, then the current drive is assumed.

Only the first character of the `drive$` string is used.



Example 
[DiskFree Example](#)

See Also 
[File Input and Output](#)



Do...Loop Statement

Description	This statement repeats a block of DCL statements while a condition is TRUE or until a condition is TRUE.
Syntax 1	<pre>do {while until} <i>condition</i> <i>statements</i> loop</pre>
Comment	<p>Using <code>do while</code> causes the statements in the loop to be executed while the specified condition is true.</p> <p>Using <code>do until</code> causes the statements in the loop to be executed until the specified condition is true.</p> <p>The <i>condition</i> parameter specifies any Boolean expression.</p>
Syntax 2	<pre>do <i>statements</i> loop {while until} <i>condition</i></pre>
Comment	Syntax 2 has the same effect as Syntax 1.
Syntax 3	<pre>do <i>statements</i> loop</pre>
Comment	If the {while until} conditional clause is not specified, then the loop repeats the statements forever (or until an <u>exit do</u> statement is encountered).
Example	 Do...Loop Example
See Also	 Flow Control



DoEvents Statement

Description	The <code>DoEvents</code> statement and function yield control to other applications.
Statement Syntax	<code>DoEvents</code>
Function Syntax	<code>DoEvents [()]</code>
Returns	The function returns the value 0.
Comments	When running in <u>exclusive</u> mode, the only way other applications can multitask is for the script to call this statement/function.
Example	 DoEvents Example
See Also	 Flow Control



DoKeys Statement

Description	Uses the Windows journaling mechanism to play keystrokes into the Windows environment.
Syntax	<code>DoKeys KeyString\$</code>
Comments	The format for <code>KeyString\$</code> is the same as that used for QueKeys . This statement will not affect the current event queue.
Example	 DoKeys Example
See Also	 Keyboard Manipulation

EditEnabled Function



Description	Determines if an edit box is enabled within the current window or dialog box.
Syntax 1	<code>EditEnabled(name\$)</code>
Syntax 2	<code>EditEnabled(id%)</code>
Returns	Returns the integer TRUE if the given edit box is enabled within the active window or dialog box, FALSE otherwise.
Comments	If enabled, the edit box can be given the focus using the ActivateControl statement. The <code>name\$</code> parameter specifies the text that appears within the static control that immediately precedes the edit control in the window list (or dialog template). Alternatively, the <code>id%</code> of the edit box can be specified.
Example	 Dialog Examples
See Also	 Dialog Manipulation

EditExists Function



Description	Determines if an edit box exists within the current window or dialog box.
Syntax 1	<code>EditExists(name\$)</code>
Syntax 2	<code>EditExists(id%)</code>
Returns	Returns the integer TRUE if the given edit box exists within the active window or dialog box, FALSE otherwise.
Comments	If there is no active window, FALSE will be returned. The <code>name\$</code> parameter specifies the text that appears within the static control that immediately precedes the edit control in the window list (or dialog template). Alternatively, the <code>id%</code> of the edit box can be specified.
Example	 Dialog Examples
See Also	

EnableStopScript Statement



Description	Controls whether a script may be halted by the user.
Syntax	<code>EnableStopScript condition%</code>
Comments	<p>The <code>condition%</code> parameter can be one of the following constants:</p> <ul style="list-style-type: none">• <code>ESS_ENABLE</code> (enable script breaking). This is the default.• <code>ESS_DISABLE</code> (disable script breaking).• <code>ESS_ENABLE_INTERACTIVE</code> (allows the script to be broken only when executing in interactive [debug] mode).
Example	 EnableStopScript Example
See Also	 Flow Control

End Statement

Description	Terminates execution of the current script.
Syntax	<code>end</code>
Comments	<p>This statement can appear multiple times in a script, so that you conditionally end the script. It can also appear in functions and subroutines.</p> <p>All open files are closed. All open DDE channels are closed.</p>
Example	 End Example
See Also	 Flow Control

ENV_BOTH

Description	Constant meaning both DOS and Windows; used by the SetEnv command.
--------------------	--

ENV_DOS



Description	Constant meaning DOS environment; used by the GetEnv , SetEnv , RestoreEnv , and SaveEnv commands.
--------------------	--

ENV_WINDOWS



Description	Constant meaning Windows environment; used by the GetEnv , SetEnv , RestoreEnv , and SaveEnv commands.
--------------------	--

Environ\$ Function




Description	Returns the value of the specified environment variable. Note: For greater convenience, we recommend using the GetEnv function instead.
Syntax 1	<code>environ\$(variable\$)</code>
Syntax 2	<code>environ\$(VariableNumber%)</code>
Comments	<p>If <code>variable\$</code> is specified, then this function looks for that variable in the environment. If the variable name cannot be found, then an empty string is returned.</p> <p>If <code>VariableNumber</code> is specified, then this function looks for the Nth variable within the environment (the first variable being number 1). If there is no such environment variables, an empty string is returned. Otherwise, the entire entry from the environment is returned in the format:</p> <pre>variable=value</pre>
Example	 Environ\$ Example
See Also	 Environment Statements and Functions

EOF Function

Description	Determines whether end of file has been reached.
Syntax	<code>eof(filenumber%)</code>
Returns	Returns the integer TRUE if the end of file has been reached for the given file, otherwise FALSE.
Comments	The <code>filenumber</code> parameter is a number that is used by DCL to refer to the open file -- the number passed to the open statement.
Example	 Input/Output Example
See Also	 File Input and Output

Erl Function

Description	Not used.
Syntax	<code>erl[()]</code>
Returns	Returns the integer 0 (DCL does not support line numbers).
See Also	 Error Trapping


Err Statement and Function

Description The `err` function returns an integer representing the runtime error that caused the current error trap. The `err` statement sets the value returned by the `err` function to a specific value.

Statement Syntax `err = value%`

Function Syntax `err[()]`

Comments The `err` function can only be used while within an error trap. When a function or statement ends, the value returned by `err` is reset to 0.

Example 
[Err Example](#)

See Also 
[Error Trapping](#)

Error Statement

Description Simulates the occurrence of the specified runtime error.

Syntax `error errornumber%`

Comments The `errornumber` parameter can be a built-in error number, or a user defined error number. The `err` function can be used within the error trap handler to determine the value of the error.

Example 
[Error Statement and Function Example](#)

See Also 
[Error Trapping](#)

Error\$ Function

Description Returns the text corresponding to the given error number or the most recent error.

Syntax `error$ [(errornumber%)]`

Comments If `errornumber` is omitted, then the function returns the text corresponding to the most recent runtime error. If no runtime error has occurred, then an empty string is returned ("").



If the `error` statement was used to generate a user-defined runtime error, this function will return an empty string ("").

Example 
[Error Statement and Function Example](#)


See Also 
[Error Trapping](#)

Exclusive Statement





Description	Sets or unsets exclusive mode.
Syntax	<code>exclusive NewState%</code>
Comments	<p>When set (<code>NewState</code> is <code>TRUE</code>), then the script will not yield control to other applications - no other applications will execute concurrently. When a script is not running in exclusive mode (<code>NewState</code> is <code>FALSE</code>), then other applications can multitask while the script is running.</p> <p>By default, scripts do not run in exclusive mode.</p> <p>If a script is running in exclusive mode and there is an infinite loop, you will be unable to abort the script (the computer will hang). Thus, caution must be taken to make sure that a script has no errors or infinite loops.</p> <p>Scripts running exclusively run faster than scripts not in exclusive mode.</p> <p>If a dialog box is encountered (<code>MsgBox</code>, <code>AskBox\$</code>, <code>AnswerBox</code>, ...) while a script is running in <code>exclusive</code> mode, then that dialog box is system modal, meaning that a user can only interact with that dialog and not any other applications that may be running concurrently. All other dialog boxes, applications, and buttons will be "locked out".</p>
Example	 Exclusive Example
See Also	 Flow Control

Exit Do Statement



Description	Exits a <code>do ...loop</code> .
Syntax	<code>exit do</code>
Comments	This statement can only appear within a <u><code>do ...loop</code></u> statement. It causes execution to continue with the next statement after the <code>loop</code> clause.
See Also	 Flow Control

Exit For Statement



Description	Exits a <code>for ...next</code> loop.
Syntax	<code>exit for</code>
Comments	<p>This statement ends the <u><code>for ...next</code></u> block in which it appears. Execution will continue on the line immediately after the <code>NEXT</code> statement.</p> <p>This statement can only appear within a <code>for ...next</code> block.</p>
Example	 Exit Statement Examples
See Also	

Flow Control



Exit Function Statement

Description	Exits the current function.
Syntax	<code>exit function</code>
Comments	<p>This statement ends execution of the function in which it appears. Execution will continue on the statement or function following the call to this function.</p> <p>This statement can only appear within a function.</p>
Example	 Exit Statement Examples
See Also	 Procedure Statements

Exit Sub Statement

Description	Exits the current subroutine.
Syntax	<code>exit sub</code>
Comments	<p>This statement ends the current subroutine. Execution is transferred to the statement following the call to the current subroutine.</p> <p>This statement can appear anywhere within a subroutine. It cannot appear within a function.</p>
Example	 Exit Statement Examples
See Also	 Procedure Statements

Exp Function

Description	Returns a double-precision number representing the value of e raised to the power of x .
Syntax	<code>exp (x#)</code>
Comments	<p>Range of x: $0 \leq x \leq 709.782712893$</p> <p>A runtime error is generated if x is out of the above specified range.</p>
Example	 Exp Example
See Also	 Math Statements and Functions

FALSE

Description	Constant.
--------------------	-----------

Returns	0
Comments	Used in conditionals and Boolean expressions

FileAttr Function

Description	Returns an integer representing the file mode or file handle.						
Syntax	<code>FileAttr(filenumber%,attribute%)</code>						
Returns	Returns the file mode (if <code>attribute</code> is 1), or the operating system file handle (if <code>attribute</code> is 2).						
Comments	<p>If <code>attribute</code> is 1, then one of the following values is returned:</p> <table> <tr> <td>1</td><td>input</td></tr> <tr> <td>2</td><td>output</td></tr> <tr> <td>8</td><td>append</td></tr> </table> <p>The <code>filenumber</code> parameter is a number that is used by DCL to refer to the open file -- the number passed to the <u><code>open</code></u> statement.</p>	1	input	2	output	8	append
1	input						
2	output						
8	append						

Example  [Input/Output Example](#)

See Also  [File Input and Output](#)

FileCopy Function



Description	Copies the specified file(s) to the given destination.
Syntax	<code>FileCopy(src\$, dest\$)</code>
Comments	<p>The function returns the integer TRUE if successful; otherwise it returns FALSE.</p> <p>Wildcards are permitted in the <code>src\$</code> string. They are the same as the DOS wildcards.</p> <p>If the <code>src\$</code> parameter specifies wildcards, all files matching the <code>src\$</code> parameter are copied to <code>dest\$</code> with the proper modification made to the destination file name so that it matches the source file according to the wildcards.</p>

Example  [FileCopy Example](#)




See Also  [File Input and Output](#)

FileDateTime Function



Description	Returns a double-precision number representing the date and time of the given file. The number is returned in days where Dec 20, 1899 is 0.
--------------------	---

Syntax	<code>FileDateTime(filename\$)</code>
Comments	This function retrieves the date and time of the file specified by <code>filename\$</code> . A runtime error results if the file does not exist. The value returned can be used with the date/time functions (i.e., <code>year()</code> , <code>month()</code> , <code>day()</code> , <code>weekday()</code> , <code>minute()</code> , <code>second()</code> , <code>hour()</code>) to extract the individual elements.
Example	 FileDateTime Example
See Also	 File Input and Output




FileDirs Statement

Description	Fills an array with directory names from disk.
Syntax	<code>FileDirs array\$() [,dirspec\$]</code>
Comments	<p>The <code>array\$()</code> is any previously declared string array. The <code>FileDirs</code> function reallocates this array to exactly hold all of the directory names matching a given specification.</p> <p>The <code>dirspec\$</code> parameter specifies the file search mask, such as:</p> <pre>T*.C:*.*</pre> <p>If the <code>dirspec\$</code> parameter is not specified, then <code>*.*</code> is used, which fills the array with all the sub-directory names within the current directory.</p>
Example	 FileDirs Example
See Also	 Arrays  File Input and Output

FileExists Function

Description	Determines if a given filename is valid.
Syntax	<code>FileExists(filename\$)</code>
Returns	Returns the integer TRUE if the <code>filename\$</code> is a valid file, FALSE otherwise.
Example	 FileExists Example
See Also	 File Input and Output

FileLen Function

Description	Returns a long integer representing the length of the specified file in bytes.
Syntax	<code>FileLen(filename\$)</code>
Comments	This function is used to retrieve the length of a file without first opening the file. A runtime error results if the file does not exist.
Example	 FileLen Example
See Also	 File Input and Output  LOF

FileList Statement

Description

Fills an array with filenames from disk.

Syntax

FileList array\$() [,filespec\$ [,fileattr%]]

Comments

The array\$() is any previously declared string array. The files function reallocates this array to exactly hold all of the files matching a given filespec.

The filespec\$ parameter specifies the file search mask, such as:

```

*.EXE          *.DOC          t*.DO?

```


If the filespec\$ parameter is not specified, *.* is used.

The fileattr parameter is a number indicating what types of files you want included in the list. It can be any combination of the following:

Constant	Value	Description
ATTR_NORMAL	0	Read-only, archive, subdirectory, and files with no attributes
ATTR_READONLY	1	Read-only files
ATTR_HIDDEN	2	Hidden files
ATTR_SYSTEM	4	System files
ATTR_VOLUME	8	Volume label
ATTR_DIRECTORY	16	MS-DOS directories
ATTR_ARCHIVE	32	Files changed since last backup
ATTR_NONE	64	Files with no attributes


If the fileattr parameter is not specified, then the value 97 is used (ATTR_READONLY or ATTR_ARCHIVE or ATTR_NONE or ATTR_DIRECTORY). This value retrieves the same set of files normally returned by the DOS dir command.

Example




[FileList Example](#)

See Also



[Arrays](#)



File Input and Output

FileParse Statement

Description	Takes a given filename and extracts a given portion of the filename from it.																		
Syntax	FileParse\$(filename\$[,operation])																		
Comments	<p>The filename\$ parameter can specify an valid DOS filename (does not have to exist). For example:</p> <pre>..\TEST.DAT C:\SHEETS\TEST.DAT TEST.DAT</pre> <p>The optional operation parameter specifies which portion of the filename\$ to extract. It can be any of the following values.</p> <table><tr><td>0</td><td>full name</td><td>C:\SHEETS\TEST.DAT</td></tr><tr><td>1</td><td>drive</td><td>C</td></tr><tr><td>2</td><td>path</td><td>C:\SHEETS</td></tr><tr><td>3</td><td>name</td><td>TEST.DAT</td></tr><tr><td>4</td><td>root</td><td>TEST</td></tr><tr><td>5</td><td>extension</td><td>DAT</td></tr></table> <p>If operation is not specified, then the full name is returned. A runtime error will result if operation is not one of the above values.</p>	0	full name	C:\SHEETS\TEST.DAT	1	drive	C	2	path	C:\SHEETS	3	name	TEST.DAT	4	root	TEST	5	extension	DAT
0	full name	C:\SHEETS\TEST.DAT																	
1	drive	C																	
2	path	C:\SHEETS																	
3	name	TEST.DAT																	
4	root	TEST																	
5	extension	DAT																	

Example



[FileParse Example](#)

See Also



[File Input and Output](#)

FileType Function

Description	Returns an integer representing the file type.				
Syntax	<code>FileType(filename\$)</code>				
Returns	Returns one of the following file type constants: <table><tr><td><code>TYPE_DOS</code></td><td>a DOS executable file</td></tr><tr><td><code>TYPE_WINDOWS</code></td><td>a Windows executable file</td></tr></table>	<code>TYPE_DOS</code>	a DOS executable file	<code>TYPE_WINDOWS</code>	a Windows executable file
<code>TYPE_DOS</code>	a DOS executable file				
<code>TYPE_WINDOWS</code>	a Windows executable file				
Comments	This function is used to determine whether a file is a Windows executable or DOS executable. If one of the above values is not returned, then the file type is unknown.				

Example



[FileType Example](#)



See Also



[File Input and Output](#)

FindFile\$ Function



Description	Searches for <code>file\$</code> and, if found, returns a full path to it. If the file is not found, the return value is a null-string.
Syntax	<code>fullpath\$=FindFile\$(file\$)</code>
Comments	The search follows normal Windows search order: current directory, Windows directory, system directory, and then the PATH environment variable.
Example	 <u>FindFile\$ Example</u>
See Also	 <u>File Input and Output</u>

Fix Function

Description	Returns the integer part of <code>number</code> .
Syntax	<code>fix(number#)</code>
Comments	This function returns the integer part of the given value by removing the fractional part. The sign is preserved. No rounding occurs. For example:

```
fix(4.5)  'returns 4
fix(-4.5) 'returns -4
```

Example



[Fix Example](#)

See Also



[CInt](#)



[Int](#)



[Math Statements and Functions](#)

For...Next Statement

Description	Repeats a block of statement a specified number of times, incrementing a loop counter by a given increment each time through the loop.
Syntax	<pre>for counter = start to end [step increment] next [counter]</pre>
Comments	<p>If <code>increment</code> is not specified, then 1 is assumed.</p> <p>The first time through the loop, <code>counter</code> is equal to <code>start</code>. Each time through the loop, <code>increment</code> is added to <code>counter</code> by the amount specified in <code>increment</code>.</p> <p>The <code>for...next</code> statement continues executing until:</p> <ul style="list-style-type: none">• An <code>exit for</code> statement is encountered <p style="text-align: center;">OR</p> <ul style="list-style-type: none">• When <code>counter</code> is greater than <code>end</code>. <p>If <code>end > start</code> then <code>increment</code> must be positive. If <code>end < start</code>, then <code>increment</code> must be negative.</p> <p>The <code>for...next</code> statements can be nested. In such a case, the <code>next [counter]</code> statement applies to the innermost <code>for...next</code>.</p> <p>The <code>next [counter]</code> can be optimized for next loops by separating each counter with a comma. The ordering of the counters must be consistent with the nesting order (innermost</p>

counter appearing before outermost counter):

```
next i,j
```

Example



[For...Next Example](#)

See Also



[Flow Control](#)

FreeFile Function

Description Returns the next available file number.

Syntax `FreeFile[()]`

Comment The returned integer is suitable for use in the [open](#) statement.

Example



[FreeFile Example](#)

See Also



[File Input and Output](#)

Function...End Function Statement

Description Creates a user-defined function.

Syntax

```
function name[(parameter [as <type>]...)] [as  
<type>]  
  
...  
  
...  
  
name = <expression>  
end function
```

Comments The return value is determined by the statement:

```
name = <expression>
```

The name of the function following DCL naming conventions. It can include type declaration characters: %, &, and \$.

If no assignment is encountered before the function exits, then 0 will be returned for numeric functions, and an empty string will be returned for string functions.

The type of the return value is determined by the `as <type>` clause on the function statement itself. As an alternative, a type declaration character can be added to the function name:

```
function Test() as string  
    Test = "Hello World"  
end function  
  
function Test$()  
    Test = "Hello World"  
end function
```

Parameters are passed to a function by reference, meaning that any modifications to a passed parameter changes that variable in the caller. To avoid this, simply enclose variable names in parenthesis, as in the following example function

calls:

```
i = UserFunction(10,12,(j))
```

If a function is not to receive a parameter by reference, then the optional `byval` keyword can be used:

```
function Test(byval FileName as string)
    as string
end function
```

A function returns to the caller when either of the following statements is encountered:

```
end function
exit function
```

The function definition must precede the statements that call the function.

The function cannot be inside a subroutine (including `sub(main)`) or inside another function.

Functions can be recursive.

See Also



[Procedure Statements](#)

GetAttr Function

Description	Returns an integer representing the attributes of the specified file.
Syntax	<code>GetAttr(filename\$)</code>
Returns	The attribute value returned contains the sum of the following attributes.

Constant	Value	Description
ATTR_NORMAL	0	Read-only, archive, subdirectory, and files with no attributes
ATTR_READONLY	1	Read-only files
ATTR_HIDDEN	2	Hidden files
ATTR_SYSTEM	4	System files
ATTR_VOLUME	8	Volume label
ATTR_DIRECTORY	16	MS-DOS directories
ATTR_ARCHIVE	32	Files changed since last backup
ATTR_NONE	64	Files with no attributes

These attributes are the same as those used by DOS.

Example



[GetAttr Example](#)

See Also





[File Input and Output](#)



GetCheckbox Function



Description	Returns an integer representing the state of the specified check box.
--------------------	---

Syntax 1	<code>GetCheckbox (name\$)</code>
Syntax 2	<code>GetCheckbox (id%)</code>
Comments	<p>This function is used to determine the state of a check box, given its <code>name\$</code> (the text of its label) or <code>id%</code>. The return value will be one of the following:</p> <ul style="list-style-type: none"> 0 check box has no check 1 check box contains a check 2 check box is grayed
Example	 Dialog Examples
See Also	 Dialog Manipulation

GetComboboxItem\$ Function

Description	Returns the text corresponding to an item number in a combobox.
Syntax 1	<code>GetComboboxItem\$ (name\$, ItemNumber%)</code>
Syntax 2	<code>GetComboboxItem\$ (id%, ItemNumber\$)</code>
Comments	<p>The combobox must exist within the current window or dialog box, otherwise a runtime error is generated.</p> <p>You can use the <code>name\$</code> or <code>id%</code> parameter to specify the combobox. The <code>name\$</code> parameter specifies the text that appears in the static control that immediately precedes the combobox control in the window list (or dialog template).</p> <p>The <code>ItemNumber%</code> parameter is the line number of the desired combobox item.</p> <p>An empty string will be returned if the combobox does not contain textual items.</p>
Example	 Dialog Examples
See Also	 Dialog Manipulation

GetComboboxItemCount Function

Description	Returns an integer representing the number of items in the specified combobox.
Syntax 1	<code>GetComboBoxItemCount (name\$)</code>
Syntax 2	<code>GetComboBoxItemCount (id%)</code>
Comments	<p>You can use the <code>name\$</code> or <code>id%</code> parameter to specify the combobox. <code>Name\$</code> specifies the text that appears in the static control that immediately precedes the combobox control in the window list (or dialog template).</p> <p>A runtime error is generated if the specified combobox does</p>

not exist within the current window or dialog box.

Example



[Dialog Examples](#)

See Also



[Dialog Manipulation](#)

GetEditText\$ Function



Description

Returns the textual content of the specified edit box.

Syntax 1

`GetEditText$(name$)`

Syntax 2

`GetEditText$(id%)`

Comments

The name of an edit control is determined by scanning the window list (or dialog template) for a static control labeled `name$` that is immediately followed by an edit control. A runtime error is generated if an edit control with that name cannot be found within the active window.

For edit controls that do not have a preceding static control, the `id%` can be used to absolutely reference the control.

Example



[Dialog Examples](#)

See Also



[Dialog Manipulation](#)

GetEnv Function



Description

Returns the value of the given environment variable for DOS or WINDOWS.

Syntax

`GetEnv$(var$[, mode$])`

Comments

The `mode$` parameter can be `ENV_DOS` or `ENV_WINDOWS`.

If `mode$` is `ENV_DOS`, the variable is returned from the DOS environment, otherwise it is returned from the Windows environment.

If `mode` is unspecified, the default is `ENV_WINDOWS`.

Example



[GetEnv Example](#)



See Also



[Environment Statements and Functions](#)



GetListboxItem\$ Function



Description	Returns the string corresponding to a list box item.
Syntax 1	<code>GetListboxItem\$(name\$, item%)</code>
Syntax 2	<code>GetListboxItem\$(id%, item%)</code>
Comments	<p>This function retrieves the text of a given item in a listbox. The <code>item%</code> parameter is the item's position in the list, where 1 is the first item. The <code>item%</code> parameter must be between 1 and the number of items in the listbox.</p> <p>The listbox can be specified using either its <code>id%</code> or the <code>name\$</code> (label) of the static control that immediately precedes the listbox control in the window list (or dialog template).</p> <p>A runtime error is generated if the specified listbox cannot be found within the active window.</p>
Example	 Dialog Examples
See Also	 Dialog Manipulation

GetListboxItemCount Function



Description	Returns an integer representing the number of items in the specified listbox.
Syntax 1	<code>GetListboxItemCount(name\$)</code>
Syntax 2	<code>GetListboxItemCount(id%)</code>
Comments	<p>The listbox can be specified using either its <code>id%</code> or the <code>name\$</code> (label) of the static control that immediately precedes the listbox control in the window list (or dialog template).</p> <p>A runtime error is generated if the specified listbox cannot be found within the active window.</p>
Example	 Dialog Examples
See Also	 Dialog Manipulation

GetOption Function



Description	Determines whether a given option button is checked.
Syntax 1	<code>GetOption(name\$)</code>
Syntax 2	<code>GetOption(id%)</code>
Returns	Returns the integer TRUE if the option is set, FALSE otherwise.
Comments	The option button must exist within the current window or dialog box.

The option button can be referenced given its `name$` (label) or its `id%`. A runtime error will be generated if the given option button does not exist.

Example



[Dialog Examples](#)

See Also



[Dialog Manipulation](#)

GetUserName Function

Description Use the [NetUserName](#) function instead.

GoSub Statement

Description Executes the specified subroutine.

Syntax `gosub label`

Comments This statement causes execution to continue at the specified label. Execution can later be returned to the statement following the `gosub` by using the [return](#) statement.

The `label` parameter must be a label within the current function or subroutine. The `gosub` parameter outside the context of the current function or subroutine is not allowed.

Note: It is a sounder programming practice to write a named subroutine using a [Sub...End Sub](#) block.

Example



[GoSub Example](#)

See Also



[Flow Control](#)

Goto Statement

Description Transfers execution to the line containing the specified label.

Syntax `goto <label>`

Comments The compiler will produce an error if `label` does not exist.

The `label` must appear within the same subroutine or function as the `goto`.

Labels must begin with a letter and end with a colon. Keywords cannot be used as labels. Labels are not case sensitive.

Example





[Goto Example](#)

See Also





[Flow Control](#)



GroupBox Statement

Description	Defines a groupbox within a dialog box template.
Syntax	<code>GroupBox x%,y%,width%,height%,title\$</code>
Comments	<p>A groupbox is simply a visual element used to enclose other controls within a dialog box.</p> <p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>The <code>x,y,width,height</code> parameters are specified in dialog coordinates. The <code>x,y</code> position is relative to the upper left corner of the dialog box.</p>
Example	 <p>Dialog Examples</p>
See Also	 <p>Dialog Creation</p>



Hex\$ Function

Description	Returns a string containing the hexadecimal equivalent of the specified number.
Syntax	<code>hex\$(number&)</code>
Comments	<p>The returned string contains only the number of hexadecimal digits necessary to represent the number, up to a maximum of 8.</p> <p>The <code>number</code> parameter can be any type, but is rounded to the nearest whole number before converting to hexadecimal. If the passed number is an integer, then a maximum of 4 digits are returned; otherwise, up to 8 digits can be returned.</p>
Example	 <p>Hex\$ Example</p>
See Also	 <p>Conversions</p>



HLine Statement

Description	Scrolls the window with the focus left or right by the specified number of lines. This feature is useful when the contents are wider than the window.
Syntax	<code>HLine [lines%]</code>
Comments	If the <code>lines%</code> parameter is omitted, then the window is scrolled right by 1 line.
Example	 <p>HLine Example</p>
See Also	 <p>Window Manipulation</p>



Hour Function

Description	Returns an integer representing the hour of the day encoded in the specified <code>serial#</code> parameter. The value returned is between 0 and 23 inclusive.
Syntax	<code>hour(serial#)</code>
Comment	You can obtain the value for the <code>serial</code> parameter by using the TimeSerial or TimeValue command.
Example	 Hour Example
See Also	 Date and Time Functions

HPage Statement

Description	Scrolls the window with the focus left or right by the specified number of pages. This feature is useful when the contents are wider than the window.
Syntax	<code>HPage [pages%]</code>
Comments	If the <code>pages%</code> parameter is omitted, then the window is scrolled right by 1 page.
Example	 HPage Example
See Also	 Window Manipulation

HScroll Statement

Description	Sets the thumb mark on the horizontal scroll bar attached to the current window.
Syntax	<code>HScroll percentage%</code>
Comments	The position is given as a percentage of the total range associated with that scroll bar. For example, if the <code>percentage%</code> parameter is 50, then the thumb is positioned in the middle of the scroll bar.
Example	 HScroll Example
See Also	 Window Manipulation

If...Then...Else Statement

Description	Conditionally executes a statement or group of statements.
Syntax 1	<code>if <condition> then <statement></code> <code>[else <statement>]</code>
Syntax 2	<code>if <condition> then</code>

```

        [<statement>]
elseif <condition> then
        [<statement>]]
else
        [<statement>]]
end if

```

Comments In the single line version, the <statement> must be a single statement. Optionally, many statements can be separated using the colon (:).

Example  [If...Then...Else Example](#)

See Also  [Flow Control](#)

Input # Statement

Description Reads comma-delimited data from a file into variables.

Syntax `input [#]filename%,variable[,variable]...`

Comments This statement reads data from the file referenced by `filename%` into the given variables.

Each `variable` must be type matched to the data in the file. For example, a string variable must be matched to a string in the file.

All data items are separated by commas in the file. Leading spaces are ignored. Strings must be enclosed in quotes:

```
10,"Hello World",192,6
```

The `filename%` parameter is a number that is used by DCL to refer to the open file -- the number passed to the [open](#) statement.

The `filename%` must reference a file opened in `input` mode.

See Also  [File Input and Output](#)

Input\$ Function

Description Returns a character string containing the first `numbytes` characters read from the given file.

Syntax `input$(numbytes%, [#]filename%)`

Comments The `input$` function reads all characters, including spaces and carriage returns.

The `filename%` must reference a file opened in `input` mode.

The `filenumber%` parameter is a number that is used by DCL to refer to the open file -- the number passed to the [open](#) statement.

See Also



[File Input and Output](#)

InputBox\$ Function

Description	Presents a dialog box displaying a prompt and returns the user's response.
Syntax	<code>InputBox\$(prompt\$ [,title\$ [,default\$ [,x%,y%]])</code>
Returns	Returns the text contained in the edit box when the user presses OK. If the user Cancels the dialog box, an empty string is returned.
Comments	<p>A default response can be specified in the <code>default\$</code> parameter.</p> <p>The <code>prompt\$</code> parameter can contain multiple lines, each separated with a Carriage Return/Line Feed (<code>chr\$(13) + chr\$(10)</code>).</p> <p>The <code>title\$</code> parameter specifies the text that appears in the dialog box's caption. If the <code>title\$</code> parameter is not specified, no title appears in the dialog's caption.</p> <p>The <code>x</code> and <code>y</code> parameters are specified in twips (1/20th of a point or 1/1440 of an inch). This allows the dialog box to be positioned in a device independent manner. If the position is not specified, then the dialog box is positioned on or near the object containing the executing script.</p>

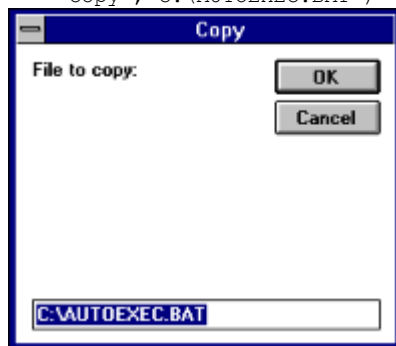
Example 1



[InputBox\\$ Example](#)

Example 2

```
s$ = InputBox$("File to copy:",  
             "Copy", "C:\AUTOEXEC.BAT")
```





See Also







[Dialog Display](#)

InStr Function

Description	Searches a string for a substring.
Syntax	<code>instr([start%,] search\$,find\$)</code>

Returns	Returns an integer representing the character position of <code>find\$</code> within <code>search\$</code> .
Comments	<p>If the string is found, its character position within <code>search\$</code> is returned, with 1 being the character position of the first character.</p> <p>If <code>start%</code> is specified, then the search starts at that character position within <code>search\$</code>. The <code>start%</code> parameter must be between 1 and 65535. If not specified, the search starts at the beginning (<code>start% = 1</code>).</p> <p>If the string is not found, or <code>start%</code> is greater than the length of <code>search\$</code>, or if <code>search\$</code> is empty, then 0 is returned.</p>
Example	 InStr Example
See Also	 Strings

Int Function

Description	Returns the integer part of a given number.
Syntax	<code>int (number#)</code>
Comments	This function returns the first integer less than (rounds down) the given value. The sign is preserved.
Example	 Int Example
See Also	 Fix  CInt  Math Statements and Functions

Item\$ Function

Description	Gets a set of contiguous, delimited items from a text string..
Syntax	<code>item\$(text\$,first%,last% [,delimiters\$])</code>
Returns	Returns all of the items between <code>first</code> and <code>last</code> within the specified text.
Comments	<p>The <code>first</code> parameter specifies the first item in the sequence to return. The lowest value for <code>first</code> is 1. All items between <code>first</code> and <code>last</code> are returned.</p> <p>By default, items are separated by commas and end-of-lines. This can be changed by specifying different delimiters in the <code>delimiters\$</code> parameter.</p> <p>If <code>first</code> is greater than the number of items in <code>text\$</code>, then</p>

an empty string is returned.

If `last` is greater than the number of items in `text$`, then all items from `first` to the end of `text` are returned.

Example



[Item\\$ and ItemCount Example](#)

See Also



[Strings](#)

ItemCount Function

Description

Returns an integer representing the number of items in the specified text.

Syntax

```
ItemCount(text$ [,delimiters$])
```

Comments

By default, items are separated by commas and end-of-lines. This can be changed by specifying different delimiters in the `delimiters$` parameter. For example, to parse items using a backslash:

```
n = itemCount(text$,"\\")
```

The first `text$` item is 1.

Example



[Item\\$ and ItemCount Example](#)

See Also



[Strings](#)

Kill Statement

Description

Deletes one or more files.

Syntax

```
kill filespec$
```

Comments

This command deletes all files matching `filespec$`.

The `filespec$` parameter can contain wildcards, such as `*` and `?`.

This function behaves the same as the "del" command in DOS.

Example



[Kill Example](#)

See Also



[File Input and Output](#)

LBound Function

Description



Determines the smallest subscript for a dimension of an array.

Syntax



```
lbound(ArrayVariable() [,dimension%])
```

Returns



Returns an integer representing the lower bound of the specified dimension of the specified array variable. If the array

	has no dimension, a runtime error is returned.
Comments	The first dimension is assumed if <code>dimension</code> is not specified (i.e., <code>dimension = 1</code>).
Example	 LBound Example
See Also	 Arrays



LCase\$ Function

Description	Returns the lower case equivalent of the specified string.
Syntax	<code>lcase\$(str\$)</code>
Example	 LCase\$ Example
See Also	 Strings



Left\$ Function

Description	Returns the leftmost <code>NumChars%</code> characters from a given string.
Syntax	<code>left\$(str\$,NumChars%)</code>
Comments	If <code>NumChars%</code> is 0, then an empty string is returned. If <code>NumChars%</code> is greater than or equal to the number of characters in the specified string, then the entire string is returned.
Example	 Left\$ Example
See Also	 Strings



Len Function

Description	Returns an integer representing the number of characters in a given string.
Syntax	<code>len(str\$)</code>
Comments	If <code>str</code> is empty, 0 is returned.
Example	 Len Example
See Also	 Strings



Let Statement

Description	Assigns the result of an expression to a variable.
Syntax	<code>[let] variable = expression</code>
Comments	<code>let</code> is supported for compatibility with other implementations of DCL.
Example	 Let Example
See Also	 Variables and Constants

Line\$ Function

Description	Gets a set of lines (delimited by CR/LFs) from the specified text.
Syntax	<code>line\$(text\$,first%[,last%])</code>
Returns	Returns a single line or group of lines between <code>first</code> and <code>last</code> .
Comments	Lines are delimited by CR/LF pairs. If <code>last</code> is not specified, then only one line is returned. If <code>first</code> is greater than the number of lines in <code>text\$</code> , an empty string is returned. If <code>last</code> is greater than the number of lines in <code>text\$</code> , all lines from <code>first</code> to the end of text are returned.
Example	 Line\$ Example
See Also	 Strings

LineCount Function

Description	Returns an integer representing the number of lines in the specified text.
Syntax	<code>LineCount(text\$)</code>
Comments	Lines are delimited by CR/LF pairs.
Example	 LineCount Example
See Also	 Strings

LineInput # Statement

Description	Reads a line into a string variable.
Syntax	<code>lineinput [#]filenumber%,text\$</code>
Comments	This statement reads an entire line into the given string variable <code>text\$</code> . The file is read up to the next carriage return.

The file pointer is positioned after the terminating CR/LF.

The `filenumber%` parameter is a number that is used by DCL to refer to the open file -- the number passed to the `open` statement.

The `filenumber%` must reference a file opened in `input` mode.

The `text$` parameter is any string variable reference.

Example



[Input/Output Example](#)

See Also



[File Input and Output](#)

ListBox Statement

Description Defines a listbox that appears within a dialog box template.

Syntax `Listbox x%,y%,width%,height%,items$(),.Field`

Comments The `items$` array must be a single-dimension array of strings. The elements of this array are placed into the listbox when the dialog box is created. The `.Field` parameter defines the name used to extract which string occupies the listbox when the dialog box ends. On exit from the [Dialog](#) statement, the `.Field` will contains an index to the item that is highlighted in the listbox.

This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).

The `x,y,width,height` parameters are specified in dialog coordinates. The `x,y` position is relative to the upper left corner of the dialog box.

Example



[Dialog Examples](#)

See Also



[Dialog Creation](#)

ListboxEnabled Function



Description Determines whether a listbox is enabled within the current window or dialog box.

Syntax `ListboxEnabled(name$ | id%)`

Returns Returns the integer TRUE if the given listbox is enabled within the active window or dialog box, FALSE otherwise.

Comments If there is no active window, FALSE is returned.

The `name$` parameter specifies the text that appears within the static control that immediately precedes the listbox control in the window list (or dialog template). Alternatively, the `id%` of the listbox can be specified.

Example

[Dialog Examples](#)

See Also

[Dialog Manipulation](#)

ListboxExists Function

**Description**

Determines whether a listbox exists within the current window or dialog box.

Syntax

```
ListboxExists(name$ | id%)
```

Returns

Returns the integer TRUE if the given listbox exists within the active window or dialog box, FALSE otherwise.

Comments

If there is no active window, FALSE is returned.

The `name$` parameter specifies the text that appears within the static control that immediately precedes the listbox control in the window list (or dialog template). Alternatively, the `id%` of the listbox can be specified.

Example

[Dialog Examples](#)

See Also

[Dialog Manipulation](#)

Loc Function

Description

Returns an integer representing the position of the file pointer in the given file.

Note: We recommend using the [Seek](#) function instead.

Syntax

```
loc(filenum%)
```

Comments

The `filenum%` parameter is a number that is used by DCL to refer to the open file -- the number passed to the [open](#) statement.

See Also

[File Input and Output](#)

LOF Function

Description

Returns an integer representing the number of bytes in the given file.

Syntax

```
lof(filenum%)
```

Comments



The `filenum` parameter is a number that is used by DCL to refer to the open file -- the number passed to the [open](#) statement.

Example



[Input/Output Example](#)

See Also


Log Function

Description	Returns a double-precision number representing the natural logarithm of a given number.
Syntax	<code>log (number#)</code>
Comments	The value of <code>number</code> must be greater than 0.
Example	 Log Example
See Also	 Math Statements and Functions

LTrim\$ Function

Description	Returns the specified string with the leading spaces removed.
Syntax	<code>ltrim\$ (str\$)</code>
Example	 LTrim\$ Example
See Also	 Strings

Main Statement

Description	Defines the Main subroutine for the script.
Syntax	<code>sub main() end sub</code>
Comments	This defines the subroutine that receives execution control from the host application.
See Also	 Procedure Statements

MCI Function



Description	Executes MCI (multimedia) command.
Syntax	<code>mci(command\$,result\$ [,error\$])</code>
Returns	Returns the integer 0 if the function was successful; otherwise it returns an error number.
Comments	<p>If an error occurs, then the optional <code>error\$</code> parameter is set to the text corresponding to the error.</p> <p>If the <code>command\$</code> returns a value, then that value is contained in <code>result\$</code>.</p> <p>The <code>mci</code> function accepts any MCI command as defined in the <i>Multimedia Programmers Reference</i> in the Windows 3.1 SDK.</p>

Example

[MCI Example](#)

See Also

[Environment Statements and Functions](#)

Menu Statement

**Description**

Issues the specified menu command from the active window.

Syntax

`Menu MenuItem$`

Comments

The `MenuItem` parameter specifies the complete menu item name, each menu level separated with a period. For example, the "Open" command on the "File" menu is represented by: "File.Open". Cascading menu items may have multiple periods, one for each popup menu, such as "File.Layout.Vertical". Menu items can also be specified using numeric index values. For example, to select the third menu item from the File menu, use "File.#3". To select the 4th item from the third menu, use "#3,#4". Separators count as items.

Items from an application's system menu can be selected by beginning the menu item specification with a period, such as ".Restore" or ".Minimize".

A runtime error will result if the menu item specification does not specify a menu item. For example, "File" specifies a menu popup, rather than a menu item. The menu item "File.Blank Blank" is not a valid menu item.

When comparing menu item names, this statement removes periods (.), spaces, and &. Further, all characters after a backspace or tab are removed. Thus, the menu item "&Open... \aCtrl+F12" translates simply to "Open".

A runtime error is generated if the menu item cannot be found or is not enabled at the time that this statement is encountered.

Example

[Menu Example](#)

See Also

[Menus](#)

MenuItemChecked Function

**Description**

Determines whether a menu item in the active window is checked.

Syntax

`MenuItemChecked(MenuItemName$)`

Returns

Returns the integer TRUE if the given menu item exists and is checked, FALSE otherwise.

Comments

The `MenuItemName$` parameter specifies a complete menu item or menu item popup following the same format as that

used by the [Menu](#) statement.


See Also



[Menus](#)


MenuItemEnabled Function



Description	Determines whether a menu item in the active window is enabled.
Syntax	<code>MenuItemEnabled(MenuItemName\$)</code>
Returns	Returns the integer TRUE if the given menu item exists and is enabled, FALSE otherwise.
Comments	The <code>MenuItemName\$</code> parameter specifies a complete menu item or menu item popup following the same format as that used by the Menu statement.
See Also	 Menus

MenuItemExists Function



Description	Determines whether a menu item in the active window exists.
Syntax	<code>MenuItemExists(MenuItemName\$)</code>
Returns	Returns the integer TRUE if the given menu item exists, FALSE otherwise.
Comments	The <code>MenuItemName</code> parameter specifies a complete menu item or menu item popup following the same format as that used by the Menu statement.
Examples	<pre>sub main() if MenuItemExists("File.Open") then beep if MenuItemExists("File") then MsgBox "There is a File menu." end sub</pre>
See Also	 Menus

Mid\$ Function

Description	The <code>mid\$</code> function returns a substring. The <code>mid\$</code> function can also replace a substring with new text.
Syntax 1	<code>mid\$(str\$,start% [,length%])</code>
Returns	Returns a substring from the specified string. The substring starts at character position <code>start%</code> for <code>length%</code> number of characters.
Comments	If <code>length%</code> is not specified, then the entire string starting at <code>start%</code> is returned. If <code>start%</code> is greater than the length of <code>str\$</code> , then an empty

string is returned.

Syntax 2

```
mid$(str$,start%[,length%]) = newvalue$
```

Comments

This statement replaces one part of a string with another. The `str$` parameter specifies the string variable containing the substring to replace. The substring within `str$` which is replaced starts at the character position specified by `start%` for `length%` number of characters. If `length%` is not specified, then the rest of the string is assumed.

The `newvalue$` parameter is any string or string expression. The resultant string is never longer than the original length of `str$`. Any extra characters at the end of `newvalue$` are ignored.

Example



[Mid\\$ Example](#)

See Also



[Strings](#)

Minute Function

Description

Returns an integer representing the minute of the day encoded in the specified `serial#` parameter. The value returned is between 0 and 59 inclusive.

Syntax

```
minute(serial#)
```

Comment

You can obtain the value for the `serial#` parameter by using the [TimeSerial](#) or [TimeValue](#) command.

Example



[Minute Example](#)

See Also



[Date and Time Functions](#)

MkDir Statement

Description

Creates a new directory.

Syntax

```
mkdir dir$
```

Comments

This command behaves just like the DOS "md" command.

Example



[MkDir Example](#)

See Also



[File Input and Output](#)

Mod

Description

Modulo operator.

Syntax

```
expression1 mod expression2
```

Returns

Returns the remainder of `expression1 / expression2`.

Notes The two operands are converted to whole numbers before performing the modulo operation.

Example 
[Mod Example](#)


See Also 
[Operators](#)

Month Function

Description Returns an integer representing the month of the date encoded in the specified `serial#` parameter. The value returned is between 1 and 12 inclusive.

Syntax `month(serial#)`

Comment You can obtain the value for the `serial#` parameter by using the [DateSerial](#) or [DateValue](#) command.

Example 
[Month Example](#)

See Also 
[Date and Time Functions](#)

MsgBox Statement and Function

Description The `MsgBox` statement displays a message box; the `MsgBox` function displays a message box and returns an integer representing the button that was pressed.

Statement Syntax `MsgBox msg$ [,type% [,title$]]`

Function Syntax `MsgBox(msg$ [,type% [,title$]])`





Returns The `MsgBox` function returns one of the following numbers:

- 1 OK was pressed
- 2 Cancel was pressed
- 3 Abort was pressed
- 4 Retry was pressed
- 5 Ignore was pressed
- 6 Yes was pressed
- 7 No was pressed

Comments The dialog box is sized to hold the entire contents of `msg$`. The `msg$` string can contain CR/LF to separate lines. If a given line is too long, it will be word wrapped.

The `type` parameter is the sub of the any of the following values:

- 0 display OK button only
- 1 display OK, Cancel buttons

- 2 display Abort, Retry, Ignore buttons
- 3 display Yes, No, Cancel buttons
- 4 display Yes, No buttons
- 5 display Retry, Cancel buttons
- 16 display "stop" icon 
- 32 display "question mark" icon 
- 48 display "exclamation point" icon 
- 64 display "information" icon 
- 0 first button is the default button
- 256 second button is the default button
- 512 third button is the default button
- 0 Application modal - the current application is suspended until dialog box is closed
- 4096 System modal - all applications are suspended until the dialog box is closed

The default value for `type` is 0 (display only the OK button, making it the default).

The default value for `title$` is "DCL".

Example



[MsgBox Example](#)

See Also



[Dialog Display](#)

MsgClose Statement

Description Closes a message window that was opened with the [MsgOpen](#) statement.

Syntax `MsgClose`

Comments Nothing will happen if there is no open message window.

Example



[Message Example](#)

See Also



[Dialog Display](#)

MsgOpen Statement

Description Displays a window with a message.

Syntax `MsgOpen msg$, timeout%, isCancel%, isThermometer%[, x%, y%]`

Comments The message can be displayed permanently, or for a specified number of seconds, or until an optional Cancel button is pressed.

The displayed message can be changed by calling the [MsgSetText](#) statement

The `timeout%` parameter causes the window to be removed after that number of seconds. The `timeout%` parameter has no effect if its value is zero.

The `isCancel` parameter controls whether or not a Cancel button appears within the window beneath the displayed message. If TRUE, then a Cancel button appears. If not specified, or FALSE, then no Cancel button is created. If a user presses the Cancel button at runtime, a trappable runtime error is generated. In this manner, a message box can be displayed and processing can continue as normal, aborting only when the process is canceled by pressing the Cancel button.

The `isThermometer` parameter controls whether there is a thermometer. If TRUE, then a thermometer is created between the text and the optional Cancel button. The thermometer initially indicated 0% complete, and can be changed using the [MsgSetThermometer](#) statement.

The optional `x`, `y` parameters specify the location of the upper left corner of the message box, in twips (1/20th of a point or 1/1440 of an inch). If this point is not specified, then the window is centered on top of the parent.

Only one message window can be opened at any one time. The message window is removed automatically when a script terminates.

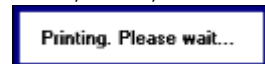
Example 1



[Message Example](#)

Example 2

```
MsgOpen "Printing. Please wait...",  
0, FALSE, FALSE
```



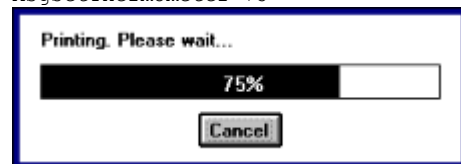
Example 3

```
MsgOpen "Printing. Please wait... ",  
0, TRUE, FALSE
```



Example 4

```
MsgOpen "Printing. Please wait... ",  
0, TRUE, TRUE  
MsgSetThermometer 75
```





See Also





[Dialog Display](#)

MsgSetText Statement





Description	Changes the text within an open message box (one that was previously opened with MsgOpen).
Syntax	<code>MsgSetText newtext\$</code>
Comments	The message box is resized to accommodate the new text. A runtime error will result if a message box is not currently open (using MsgOpen).
Example	 Message Example
See Also	 Dialog Display


MsgSetThermometer Statement

Description	Changes the percentage filled in the thermometer of an open message box (one that was previously opened with MsgOpen).
Syntax	<code>MsgSetThermometer percentage%</code>
Comments	A runtime error will result if a message box is not currently open (using MsgOpen), or if the value of <code>percentage%</code> is not between 0 and 100 inclusive.
Example	 Message Example
See Also	 Dialog Display

Name Statement

Description	Renames a file.
Syntax	<code>name oldfile\$ as newfile\$</code>
Example	 Name Example
See Also	 File Input and Output

NetAttach Function

Description	Attaches the current user to the specified server using the given user ID and password.
Syntax	<code>NetAttach(server\$, user\$, password\$)</code>
Comments	This function returns the integer TRUE if it is successful in attaching to the specified server or FALSE if the user is already attached, the password verification fails, or the server is not available.
Example	 NetAttach Example

See Also



[Network Functions](#)

NetConnectDrive Function



Description	Connects a local drive letter with a network path.
Syntax	<code>NetConnectDrive(localpath\$, networkpath\$ [, [root%], passwd\$])</code>
Comments	<p>Returns the integer TRUE if the drive is successfully mapped, otherwise returns FALSE.</p> <p>If <code>root%</code> is specified, then the drive is created with a <i>false root</i>. If <code>passwd\$</code> is specified it is used in attempting to connect the network drive. Note that <code>root%</code> and <code>passwd\$</code> may not be supported in all networks.</p> <p><code>root%</code> defaults to FALSE, and <code>passwd\$</code> defaults to none.</p> <p><code>localpath\$</code> should be specified in the form "E:".</p> <p><code>networkpath\$</code> is a valid network path, meaning <i>server/volume:...</i> or a UNC pathname.</p>

Example



[NetConnectDrive Example](#)

See Also



[Network Functions](#)

NetDetach Function



Description	Detaches the current user from the specified file server.
Syntax	<code>NetDetach(server\$)</code>
Comment	Returns the integer TRUE is returned if the drive is successfully detached, otherwise returns FALSE.
Caution	No safety checks are performed during the detach operation. Detaching from a server is a dangerous thing to do, and this function in no way guarantees that it will be done without causing some sort of system failure.

Example



[NetDetach Example](#)

See Also



[Network Functions](#)

NetDirectoryRights Function



Description	Returns the integer TRUE if the current user has the given rights for the specified network directory path.
Syntax	<code>NetDirectoryRights(path\$, rights\$)</code>
Comments	The <code>path\$</code> parameter may be any legal reference to a directory. This includes full paths with drive letter, server/volume, or UNC;

or partial paths that are relative to the current directory.

The `rights$` parameter is a series of characters that have network specific meanings. For example, "ROS" on NetWare means Read, Open & Search permissions. If the user has **all** of these permissions for the specified path, the return value is TRUE.

Example



[NetDirectoryRights Example](#)

See Also



[Network Functions](#)

NetDisconnectDrive Function

Description

Disconnects a local drive letter from a network path.

Syntax

```
NetDisconnectDrive (networkdrive$)
```

Comments

The function takes a single string parameter which represents the drive letter that is to be disconnected. It returns an integer value 0 or -1. A 0 (or FALSE) indicates function failure, a -1 (or TRUE) indicates success.

Example



[NetDisconnectDrive Example](#)

See Also



[Network Functions](#)

NetGetDirectoryRights Function



Description

Returns a string representing the effective rights for the current user on the specified path.

Syntax

```
NetGetDirectoryRights$ (path$)
```

Comments

The rights are identified as a series of characters compatible with those used by the network operating system that is in use.

Example



[NetGetDirectoryRights Example](#)

See Also



[Network Functions](#)

NetMemberOf Function



Description

Determines whether the current user is a member of the specified group.

Syntax

```
NetMemberOf (group$, server$)
```

Comments

This function returns the integer TRUE or FALSE.

If the `server$` parameter is specified then the groups on that server are searched, otherwise the primary server is used.

If the user USERA is a member of the group SMALLGRP, and SMALLGRP is a member of BIGGROUP, USERA is a member of BIGGROUP.

Example



[NetMemberOf Example](#)

See Also



[Network Functions](#)

NetStationID Function



Description

Returns a network-dependent station id as a string.

Syntax

`NetStationID$()`

Comments

On Novell networks, the station's Ethernet address is returned.

Example



[NetStationID Example](#)

See Also



[Network Functions](#)

NetUserName Function



Description

Returns the user name associated with the given server.

Syntax

`NetUserName$([server$])`

Comments

If server isn't specified then the primary server is used.

Example



[NetUserName Example](#)

See Also



[Network Functions](#)

NetworkStatus Function



Description

Returns an integer representing the network status as a 16 bit (WORD) set of flags.

Syntax

`NetworkStatus()`

Comments

Each bit of the returned status has different significance. Currently, there are two bit flags.

NS_ACTIVE (0x0001) The network redirector is loaded

NS_LOGGEDON (0x0002) A user is actively logged onto a server/the network.

Normally, this function is compared to the value 3 to determine if additional network calls can or should be made.

Example





[NetworkStatus Example](#)

See Also





[Network Functions](#)

Not

Description	Not operator.
Syntax	<code>NOT expression1</code>
Returns	TRUE if <code>expression1</code> is FALSE, otherwise returns TRUE. If the operand is numeric, then the result is the bitwise NOT of the argument.
Notes	If the operand is a floating point value (either single or double), then it is first converted to a long, then a bitwise NOT is performed.
Example	 Not Example
See Also	 Operators

Now Function

Description	Returns a double-precision number representing the current date and time. The number is returned in days since Dec 20, 1899.
Syntax	<code>now()</code>
Example	 Now Example
See Also	 Date and Time Functions


NS_ACTIVE

Description	Constant used by the NetworkStatus command.
--------------------	---

NS_LOGGEDON

Description	Constant used by the NetworkStatus command.
--------------------	---

Null Function



Description	Returns a null string (a string that contains no characters and requires no storage).
Syntax	<code>null[()]</code>
Comments	An empty string ("") can also be used to remove all characters from a string. However, an empty string still requires some memory for storage. Null strings require no memory.
Example	 Null Example

See Also





[Strings](#)

Oct\$ Function

Description	Returns a string containing the octal equivalent of the specified number.
Syntax	<code>oct\$(number%)</code>
Comments	<p>The returned string contains only the number of octal digits necessary to represent the number.</p> <p>The <code>number</code> parameter can be any type, but is rounded to the nearest whole number before converting to octal.</p>
Example	 Oct\$ Example
See Also	 Conversions

OKButton Statement

Description	Defines an OK button that appears within a dialog box template.
Syntax	<code>OKButton x%,y%,width%,height%</code>
Comments	<p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>The <code>x</code>, <code>y</code>, <code>width</code>, <code>height</code> parameters are specified in dialog coordinates. The <code>x</code>, <code>y</code> position is relative to the upper left corner of the dialog box.</p>
Example	 Dialog Examples
See Also	 Dialog Creation

On Error Statement

Description	Defines the action taken when a trappable runtime error occurs.
Syntax	<code>on error {goto <label> resume next goto 0}</code>
Comments	<p>The form <code>on error goto <label></code> causes execution to transfer to the specified label when a runtime error occurs.</p> <p>The form <code>on error resume next</code> causes execution to continue to the next line after the line that caused the error.</p> <p>The form <code>on error goto 0</code> causes any existing error trap to be removed.</p> <p>If an error trap is in effect when the script ends, then an error will be generated.</p>

An error trap is only active within the subroutine or function in which it appears.

Once an error trap has gained control, appropriate action should be taken, and then control should be resumed using the resume statement.

If an error occurs within the error handler, the current routines error trap is disabled and a runtime error results.

Example



On Error Example

See Also



Error Trapping



Flow Control

Open Statement

Description

Opens a file.

Syntax

```
open filename$ [for {input | output | append}] as  
[#] filenumber%
```

Comments

This statement opens a file for a given mode, assigning the open file to the supplied `filenumber`.

The `filename` parameter is a string expression that contains a valid DOS filename.

The `filenumber` parameter is a number between 1 and 255. The `FreeFile()` function can be used to determine an available file number.

The different modes are defined as follows:

Input Opens an existing file for input.

Output Opens an existing file, truncating its length to zero, or creates a new file.

Append Opens an existing file, positioning the file pointer at the end of the file, or creates a new file.

If the `[for mode]` is missing, then `append` is used.

Example



Input/Output Example

See Also



File Input and Output

OpenFileName\$ Function

Description



Displays the common file open dialog box (from COMMDLG.DLL), allowing the user to select a file.

Syntax



```
OpenFileName$(title$,extensions$)
```

Returns



Returns the full DOS pathname of the file the user selected, or an empty string if the user canceled the dialog box.

Comments	The <code>title\$</code> parameter specifies the title that appears on the dialog box's caption.
	The <code>extensions\$</code> parameter specifies the available file types. This string should be in the following format:
	<code>"type:ext[,ext][;type:ext[,ext]]..."</code>
	where <i>ext</i> is a valid file extension, like *.BAT or *.?F?, and <i>type</i> is a string that identifies this type to the user.
Example	 OpenFileName\$ Example
See Also	 Dialog Display

Option Base Statement



Description	Sets the lower bound for array declarations. By default, the lower bound used for all array declarations is 0.
Syntax	<code>option base {0 1}</code>
Comments	This statement must appear outside of any functions or subroutines.
Example	 Option Base Example
See Also	 Arrays

OptionButton Statement



Description	Defines a push button with the specified text that appears within a dialog box template.
Syntax	<code>OptionButton x%,y%,width%,height%,title\$</code>
Comments	<p>The <code>title\$</code> parameter may contain an ampersand character to denote an underlined accelerator, such as "&Font" for <u>F</u>ont.</p> <p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>The <code>x,y,width,height</code> parameters are specified in dialog coordinates. The <code>x,y</code> position is relative to the upper left corner of the dialog box.</p>
Example	 Dialog Examples
See Also	 Dialog Creation

OptionEnabled Function



Description	Determines whether an option button is enabled within the current window or dialog box.
--------------------	---

Syntax 1	<code>OptionEnabled(name\$)</code>
Syntax 2	<code>OptionEnabled(id%)</code>
Returns	Returns the integer TRUE if the specified option button is enabled within the current window or dialog box, otherwise this function returns FALSE.
Comments	<p>If an option button is enabled, its value can be set using the <u>SetOption</u> statement.</p> <p>The option button can be referenced given either its <code>name\$</code> (caption) or its <u><code>id%</code></u>.</p>
Example	 Dialog Examples
See Also	 Dialog Manipulation

OptionExists Function



Description	Determines whether an option button exists within the current window or dialog box.
Syntax 1	<code>OptionExists(name\$)</code>
Syntax 2	<code>OptionExists(id%)</code>
Returns	Returns the integer TRUE if the specified option button exists within the current window or dialog box, otherwise this function returns FALSE.
Comments	The option button can be referenced given either its <code>name\$</code> (caption) or its <u><code>id%</code></u> .
Example	 Dialog Examples
See Also	 Dialog Manipulation

OptionGroup Statement


Description	Starts a group of option buttons within a dialog box template and defines the name used to determine which option button (from the group of option buttons) is selected when the dialog box ends.
Syntax	<code>OptionGroup .Field</code>
Comments	<p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>On exit from the <u>Dialog</u> statement, the <code>.Field</code> will contain the index of the selected option button, with 0 being the first option button.</p>
Example	 Dialog Examples
See Also	

Dialog Creation



Or

Description	Or operator.
Syntax	<code>expression1 OR expression2</code>
Returns	TRUE if either <code>expression1</code> is TRUE or <code>expression2</code> is TRUE, otherwise FALSE. If the two operands are numeric, the result is the bitwise OR of the two arguments.
Notes	If either of the two operands is a floating point number, the two operands are first converted to longs, then a bitwise OR is performed.
Example	 Or Example
See Also	 Operators


PI

Description	Pi constant.
Syntax	<code>PI</code>
Returns	3.141592653589793238462643383279
Notes	PI can also be determined using the following formula: $4 * \text{atn}(1)$
Example	 PI Example

PO_LANDSCAPE

Description	Constant used with the <code>PrinterSetOrientation</code> statement to align the paper horizontally.
Returns	2
See Also	 PrinterGetOrientation  PrinterSetOrientation

PO_PORTRAIT

Description	Constant used with the <code>PrinterSetOrientation</code> statement to align the paper vertically.
Returns	1
See Also	

[PrinterGetOrientation](#)



[PrinterSetOrientation](#)

PopupMenu Function



Description Creates a popup menu using the string elements in the given array and returns an integer representing the user's response.

Syntax `PopupMenu (MenuItems$ ())`

Returns Returns the index of the selected item. If no item is selected (the popup menu is canceled), then a value of 1 less than the lower bound is returned.

Comments Each array element is used as a menu item. An empty string results in a separator bar in the menu.

The popup menu is created with the upper left corner at the current mouse position.

A runtime error results if `MenuItems$` is not a single-dimension array.

Only one popup menu can be displayed at a time. An error will result if another script executes this function while a popup menu is visible.

Example



[PopupMenu Example](#)

See Also



[Dialog Display](#)

Print Statement

Description Writes data to a viewport window.

Syntax `print expression [{, | ;} expression]...`

Comments Strings are written in their literal form, with no enclosing quotes.

Integers and longs are written with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.

Each `expression` is separated either with a comma (,) or a semicolon (;). A comma means that the next expression is output in the next print zone. A semicolon means that the next expression is output immediately after the current expression. Print zones are defined every 14 spaces.

If the last expression is not followed by a comma or semicolon, then a carriage return is printed to the file. If the last expression in the list ends with a semicolon, no carriage return is printed - the next print statement will output information immediately following the expression. If the last expression in the list ends with a comma, the file pointer is positioned at the start of the next print zone on the current line.

If no viewport window is open, then the statement is ignored. Printing information to a viewport window is a convenient way to output debugging information.

Example



[Print Example](#)

See Also



[Viewport Window Manipulation](#)

Print # Statement

Description

Writes data to a disk file.

Syntax

```
print #filename%, expression [{, | ;} expression]...
```

Comments

The `filename` parameter is a number that is used by DCL to refer to the open file -- the number passed to the [open](#) statement.

Strings are written in their literal form, with no enclosing quotes.

Integers and longs are written with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.

Each `expression` is separated either with a comma (,) or a semicolon (;). A comma means that the next expression is output in the next print zone. A semi-colon means that the next expression is output immediately after the current expression. Print zones are defined every 14 spaces.

If the last expression is not followed by a comma or semi-colon, then a carriage return is printed to the file. If the last expression in the list ends with a semi-colon, no carriage return is printed - the next print statement will output information immediately following the expression. If the last expression in the list ends with a comma, the file pointer is positioned at the start of the next print zone on the current line.

The [write](#) statement always outputs information ending with a carriage-return. Thus, if a `print` statement is followed by a `write` statement, the file pointer is positioned on a new line.

The `print` statement can only be used with files that are opened in output or append modes.

Example



[Print # Example](#)

See Also



[File Input and Output](#)

PrinterGetOrientation Function



Description

Retrieves the orientation of the default printer--the printer specified in the `device=` line in the `[windows]` section of the WIN.INI file.

Syntax

```
PrinterGetOrientation()
```

Returns

Returns the integer `PO_PORTRAIT` if the printer orientation is set

to portrait, otherwise if returns PO LANDSCAPE.

Comments This function loads the printer driver and therefore may be slow.

Example



[PrinterGetOrientation Example](#)

See Also



[Printer Manipulation](#)

PrinterSetOrientation Statement



Description Sets the orientation of the default printer--the printer specified in the device= line in the [windows] section of the WIN.INI file.

Syntax `PrinterSetOrientation NewSetting%`

Comments NewSetting% is PO PORTRAIT or PO LANDSCAPE.

This command loads the printer driver for the default printer, and therefore may be slow.

Example



[PrinterSetOrientation Example](#)

See Also



[Printer Manipulation](#)

PrintFile Function



Description Invokes the Windows 3.1 shell functions that cause an application to execute and print a file.

Syntax `PrintFile(filename$)`

Returns Returns an integer representing the ID of the executing task.

Comments This function is only available under Windows 3.1.

The application to be executed must be associated with the file extension of the file specified by this command. This association is established in the [Extensions] section of the WIN.INI file. For example, if the Notepad application is associated with the .TXT extension, the Notepad application is started when DCL executes a PrintFile command for a .TXT file.

This command does not support .EXE, .COM, .BAT, or .PIF files.

Example



[PrintFile Example](#)

See Also





[Printer Manipulation](#)




PushButton Statement

Description Defines a push button within a dialog box template.



Syntax `PushButton x%,y%,width%,height%,title$`

Comments	<p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>When a push button is selected, the Dialog statement ends.</p> <p>The <code>x,y,width,height</code> parameters are specified in dialog coordinates. The <code>x,y</code> position is relative to the upper left corner of the dialog box.</p> <p>The <code>title\$</code> parameter may contain an ampersand character to denote an underlined accelerator, such as "&Font" for Font.</p>
Example	 <p>Dialog Examples</p>
See Also	 <p>Dialog Creation</p>

QueEmpty Statement

Description	Empties the current event queue.
Syntax	<code>QueEmpty</code>
Comments	After this statement, QueFlush will do nothing.
Example	 <p>Queue Example</p>
See Also	 <p>Keyboard Manipulation</p>  <p>Mouse Events</p>

QueFlush Statement




Description	Plays back events that are stored in the current event queue.
Syntax	<code>QueFlush isSaveState%</code>
Comments	<p>After QueFlush is finished, the queue is empty.</p> <p>The QueFlush statement uses the Windows journaling mechanism to replay the mouse and keyboard events stored in the queue. As a result, the mouse position may be changed. Furthermore, events can be played into any Windows application, including DOS applications running in a window.</p> <p>If <code>isSaveState</code> is TRUE, then QueFlush saves the state of the CAPSLOCK, NUMLOCK, SCROLL LOCK, and INSERT, and restores the state after the QueFlush is complete. If FALSE, these states are not restored.</p> <p>The function does not return until the entire queue has been played.</p>
Example	 <p>Queue Example</p>
See Also	 <p>Keyboard Manipulation</p>



Mouse Events

QueKeyDn Statement



Description	Appends key down events for the specified keys to the end of the current event queue.
Syntax	<code>QueKeyDn Keys\$</code>
Comments	<p>The format for <code>Keys\$</code> is the same as for the <u>QueKeys</u> <code>KeyString\$</code> parameter, with the exception that parentheses are illegal.</p> <p>The <u>QueFlush</u> command is used to play back the events stored in the current event queue.</p>
Example	 <p><u>Queue Example</u></p>
See Also	 <p><u>Keyboard Manipulation</u></p>  <p><u>QueKeys</u> for list of special keys.</p>

QueKeys Statement



Description	Appends keystroke information to the current event queue.																																																								
Syntax	QueKeys KeyString\$																																																								
Comments	<p>To specify any key on the keyboard, simply use that key, such as "a" for lower case a, or "A" for upper case a.</p> <p>Sequences of keys are specified by appending them together: "abc" or "dir /w".</p> <p>The keys +, ^, ~, and % are special and must be specified within brackets. For example, to specify the percent, use "{%}".</p> <p>The keys {} also have special meaning. To specify one of these keys, enclose it within brackets, such as "{ }".</p> <p>Keys that are not displayed when you press that key are described within brackets, such as {ENTER} or {UP}. A list of these keys follows:</p> <table><tr><td>{BACKSPACE}</td><td>{BS}</td><td>{BREAK}</td><td>{CAPSLOCK}</td></tr><tr><td>{CLEAR}</td><td>{DELETE}</td><td>{DEL}</td><td>{DOWN}</td></tr><tr><td>{END}</td><td>{ENTER}</td><td>{ESCAPE}</td><td>{ESC}</td></tr><tr><td>{HELP}</td><td>{HOME}</td><td>{INSERT}</td><td>{LEFT}</td></tr><tr><td>{NUMLOCK}</td><td>{NUMPAD0}</td><td>{NUMPAD1}</td><td>{NUMPAD2}</td></tr><tr><td>{NUMPAD3}</td><td>{NUMPAD4}</td><td>{NUMPAD5}</td><td>{NUMPAD6}</td></tr><tr><td>{NUMPAD7}</td><td>{NUMPAD8}</td><td>{NUMPAD9}</td><td>{NUMPAD/}</td></tr><tr><td>{NUMPAD*}</td><td>{NUMPAD-}</td><td>{NUMPAD+}</td><td>{NUMPAD.}</td></tr><tr><td>{PGDN}</td><td>{PGUP}</td><td>{PRTSC}</td><td>{RIGHT}</td></tr><tr><td>{TAB}</td><td>{UP}</td><td>{F1}</td><td>{SCROLLLOCK}</td></tr><tr><td>{F2}</td><td>{F3}</td><td>{F4}</td><td>{F5}</td></tr><tr><td>{F6}</td><td>{F7}</td><td>{F8}</td><td>{F9}</td></tr><tr><td>{F10}</td><td>{F11}</td><td>{F12}</td><td>{F13}</td></tr><tr><td>{F14}</td><td>{F15}</td><td>{F16}</td><td></td></tr></table>	{BACKSPACE}	{BS}	{BREAK}	{CAPSLOCK}	{CLEAR}	{DELETE}	{DEL}	{DOWN}	{END}	{ENTER}	{ESCAPE}	{ESC}	{HELP}	{HOME}	{INSERT}	{LEFT}	{NUMLOCK}	{NUMPAD0}	{NUMPAD1}	{NUMPAD2}	{NUMPAD3}	{NUMPAD4}	{NUMPAD5}	{NUMPAD6}	{NUMPAD7}	{NUMPAD8}	{NUMPAD9}	{NUMPAD/}	{NUMPAD*}	{NUMPAD-}	{NUMPAD+}	{NUMPAD.}	{PGDN}	{PGUP}	{PRTSC}	{RIGHT}	{TAB}	{UP}	{F1}	{SCROLLLOCK}	{F2}	{F3}	{F4}	{F5}	{F6}	{F7}	{F8}	{F9}	{F10}	{F11}	{F12}	{F13}	{F14}	{F15}	{F16}	
{BACKSPACE}	{BS}	{BREAK}	{CAPSLOCK}																																																						
{CLEAR}	{DELETE}	{DEL}	{DOWN}																																																						
{END}	{ENTER}	{ESCAPE}	{ESC}																																																						
{HELP}	{HOME}	{INSERT}	{LEFT}																																																						
{NUMLOCK}	{NUMPAD0}	{NUMPAD1}	{NUMPAD2}																																																						
{NUMPAD3}	{NUMPAD4}	{NUMPAD5}	{NUMPAD6}																																																						
{NUMPAD7}	{NUMPAD8}	{NUMPAD9}	{NUMPAD/}																																																						
{NUMPAD*}	{NUMPAD-}	{NUMPAD+}	{NUMPAD.}																																																						
{PGDN}	{PGUP}	{PRTSC}	{RIGHT}																																																						
{TAB}	{UP}	{F1}	{SCROLLLOCK}																																																						
{F2}	{F3}	{F4}	{F5}																																																						
{F6}	{F7}	{F8}	{F9}																																																						
{F10}	{F11}	{F12}	{F13}																																																						
{F14}	{F15}	{F16}																																																							

Keys can be combined with SHIFT, CTRL, and ALT using the reserved keys "+", "^", and "%" respectively:

Shift+Enter	"+{ENTER}"
Ctrl+C	"^C"
Alt+F2	"%{F2}"

To specify a modifier key combined with a sequence of consecutive keys, group the key sequence within parentheses, as in the following example:

Shift+A, Shift+B, Shift+C	"+(abc)"
Ctrl+F1, Ctrl+F2	"^({F1}{F2})"

Use "~" as a shortcut for embedding ENTER within a key sequence:

"ab~de"

To embed quotes, use two quotes in a row:

"This is a ""test"" of the system"

Key sequences can be repeated using a repeat count within brackets:

"{a 10}"	produces 10 "a" keys
"{ENTER 2}"	produces 2 Enter keys

Example



[Queue Example](#)

See Also



[Keyboard Manipulation](#)

QueKeyUp Statement

Description Appends key up events for the specified keys to the end of the current event queue.

Syntax `QueKeyUp Keys$`

Comments The format for `Keys$` is the same as for the [QueKeys](#) `KeyString$` parameter, with the exception that parentheses are illegal.

The [QueFlush](#) command is used to play back the events stored in the current event queue.

Example



[Queue Example](#)

See Also



[Keyboard Manipulation](#)



[QueKeys](#) for list of special keys.

QueMouseClicked Statement

Description Adds a mouse click to the current event queue.

Syntax `QueMouseClicked button%,x%,y%`

Comments A mouse click consists of a mouse button down at position

x , y , immediately followed by a mouse button up.

The `button` parameter specifies which button to queue: either `VK_LBUTTON` or `VK_RBUTTON`.

The `QueFlush` command is used to play back the events stored in the current event queue.

Example



[Queue Example](#)

See Also



[Mouse Events](#)

QueMouseDbIClk Statement



Description

Adds a mouse double click to the current event queue.

Syntax

```
QueMouseDbIClk button%, x%, y%
```

Comments

A mouse double click consists of a mouse DN/UP/DN/UP at position x , y . The events are queued in such a way that a double-click is registered during queue playback.

The `button` parameter specifies which button to queue: either `VK_LBUTTON` or `VK_RBUTTON`.

The `QueFlush` command is used to play back the events stored in the current event queue.

Example



[Queue Example](#)

See Also



[Mouse Events](#)

QueMouseDbIDn Statement



Description

Adds a mouse double down to the current event queue.

Syntax

```
QueMouseDbIDn button%, x%, y%
```

Comments

A double down consists of a mouse DN/UP/DN at position x , y .

The `button%` parameter specifies which button to queue: either `VK_LBUTTON` or `VK_RBUTTON`.

The `QueFlush` command is used to play back the events stored in the current event queue.

Example



[Queue Example](#)

See Also





[Mouse Events](#)

QueMouseDn Statement





Description



Adds a mouse down to the current event queue.

Syntax	<code>QueMouseDn button%,x%,y%</code>
Comments	<p>The <code>button</code> parameter specifies which button to queue: either <code>VK_LBUTTON</code> or <code>VK_RBUTTON</code>.</p> <p>The <code>QueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example	 <p>Queue Example</p>
See Also	 <p>Mouse Events</p>

QueMouseMove Statement

Description	Adds a mouse move to the current event queue.
Syntax	<code>QueMouseMove button%,x%,y%</code>
Comments	<p>The <code>button</code> parameter specifies which button to queue: either <code>VK_LBUTTON</code> or <code>VK_RBUTTON</code>.</p> <p>The <code>QueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example	 <p>Queue Example</p>
See Also	 <p>Mouse Events</p>

QueMouseUp Statement

Description	Adds a mouse up to the current event queue.
Syntax	<code>QueMouseUp button%,x%,y%</code>
Comments	<p>The <code>button</code> parameter specifies which button to queue: either <code>VK_LBUTTON</code> or <code>VK_RBUTTON</code>.</p> <p>The <code>QueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example	 <p>Queue Example</p>
See Also	 <p>Mouse Events</p>

QueSetRelativeWindow Statement

Description	Adjusts all mouse positions relative to the specified window.
Syntax	<code>QueSetRelativeWindow hWnd%</code>
Comments	<p>This statement affects all subsequent <code>Que . . .</code> commands.</p> <p>The <code>hWnd%</code> parameter is a handle to a window in the Windows environment. If <code>hWnd%</code> is 0, then the window with the focus is used (i.e., the active window).</p>

The [QueFlush](#) command is used to play back the events stored in the current event queue.

Example 1



[Queue Example](#)

Example 2

```
sub main()  
'Adjust mouse coordinates relative to Notepad  
hWnd = WinFind("Notepad")  
QueSetRelativeWindow hWnd  
end sub
```

See Also



[Mouse Events](#)

Random Function

Description Generates a random number.

Syntax `random(min&,max&)`

Returns Returns a long integer greater than or equal to `min` and less than or equal to `max`.

Example



[Random Example](#)

See Also



[Math Statements and Functions](#)

Randomize Function

Description Initializes the random number generator with a new seed.

Syntax `randomize [seed&]`

Comments If `seed` is not specified, then the current value of the system clock is used.

Example



[Randomize Example](#)

See Also



[Math Statements and Functions](#)

ReadINI\$ Function

Description Returns the value of the specified item from the specified INI file.

Syntax `ReadINI$(section$,item$[,filename$])`

Comments The `filename$` parameter, if specified, contains the name of the INI file to read. Otherwise, the WIN.INI file is used.

The `section$` parameter specifies the section that contains the desired variable, such as "windows". Section names are specified without the enclosing brackets.

The `item$` parameter specifies which item to retrieve the value of.

Example

[ReadINI\\$ Example](#)

See Also

[Environment Statements and Functions](#)

ReadINISection Statement

Description

Reads all of the item names from a given section of the specified INI file.

Syntax

```
ReadINISection section$,items$()[,filename$]
```

Comments

The `filename$` parameter, if specified, contains the name of an INI file. Otherwise, the WIN.INI file is used.

The `section$` parameter specifies the section that contains the desired variables, such as "windows". Section names are specified without the enclosing brackets.

The `items$` parameter refers to the item name on the left side of the = sign. This parameter must be a single dimension array of strings (see [Dim](#) statement). On return, this will contain one array element for each variable in the specified INI section. Use the [lbound](#) and [ubound](#) functions to determine its size on return.

Example

[ReadINISection Example](#)

See Also

[Arrays](#)



[Environment Statements and Functions](#)

ReDim Statement

Description

Redimensions an array, specifying a new upper and lower bound for a given arrays dimensions.

Syntax

```
redim variablename (subscriptRange) [as type],...
```

Comments

The `variablename` parameter specifies the name of an existing array (previously declared using the [dim](#) statement).

Caution: The ReDim statement deletes any data already in the array.

The `subscriptRange` parameter specifies the new upper and lower bounds for each dimension of the array using the following syntax:

```
[lower% to] upper% [, [lower% to] upper%]...
```

If `lower%` is not specified, then 0 is used (or the value set using the `option base` statement).

Arrays can be dynamically dimensioned by first declaring them with no initial size using the `dim` statement:


```
dim a$( )
```

Then, using the `redim` statement, the actual size can be specified:

```
redim a$(0 to 8,6 to 10)
```

The number of dimensions of an array cannot be changed once the array has been given dimensions--either by declaring it with initial dimensions using `dim` or by a previous use of `redim`.

The *type* parameter can be used to specify the array element type. The following can be used: `integer`, `long`, `string`.

Example



[ReDim Example](#)

See Also



[Arrays](#)



[Variables and Constants](#)

REM Statement

Description

Causes the compiler to skip all characters on that line.

Syntax

```
REM text
```

Example



[REM Example](#)

See Also



[Comments](#)

RefreshIni Statement



Description

Reloads the WIN.INI file from disk, thus refreshing all WIN.INI settings.

Syntax

```
RefreshIni
```

Comments

This forces all hand-edited changes to the WIN.INI file to be activated.

Example



[RefreshIni Example](#)

See Also



[Environment Statements and Functions](#)

Reset Statement

Description

Closes all open files, writing out all I/O buffers.

Syntax

```
reset
```

Example



[Reset Example](#)

See Also



RestoreEnv Function



Description	Restores the set of environment variables from the virtual stack for DOS or for Windows saved by the SaveEnv function.
Syntax	<code>RestoreEnv([\$mode])</code>
Comments	<p>Returns the integer TRUE if the function is successful, otherwise FALSE.</p> <p>Separate stacks are kept for the Windows and DOS environments.</p> <p>The <code>mode\$</code> parameter can be ENV_DOS or ENV_WINDOWS.</p> <p>If mode is unspecified, the default is ENV_WINDOWS.</p>

Example



[RestoreEnv Example](#)

See Also



[Environment Statements and Functions](#)

Resume Statement

Description	Ends an error handler and continues execution.
Syntax	<code>resume {[0] next label}</code>
Comments	<p>The form <code>resume [0]</code> causes execution to continue with the statement that caused the error.</p> <p>The form <code>resume next</code> causes execution to continue with the statement following the statement that caused the error.</p> <p>The form <code>resume label</code> causes execution to continue at the specified label.</p>

Example



[Resume Example](#)

See Also



[Error Trapping](#)

Return Statement

Description	Transfers execution control to the statement following the most recent gosub .
Syntax	<code>return</code>
Comments	A runtime error results if a <code>return</code> statement is encountered without a corresponding <code>gosub</code> statement.

Example





[Return Example](#)

See Also





[Flow Control](#)



Right\$ Function

Description	Returns the rightmost <code>NumChars</code> characters from a specified string.
Syntax	<code>right\$(str\$,NumChars\$)</code>
Comments	If <code>NumChars</code> is greater than or equal to the length of the string, then the entire string is returned. If <code>NumChars</code> is 0, then an empty string is returned.
Example	 Right\$ Example
See Also	 Strings


Rmdir Statement

Description	Removes the specified directory.
Syntax	<code>rmdir dir\$</code>
Comments	This command behaves just like the DOS "rd" command.
Example	 Rmdir Example
See Also	 File Input and Output

Rnd Function

Description	Returns a single-precision random number between 0 and 1.						
Syntax	<code>rnd[(number#)]</code>						
Comments	If <code>number#</code> is omitted, the next random number is returned. Otherwise, the <code>number#</code> parameter has the following meaning: <table><tr><td><code>number# < 0</code></td><td>Always returns the same number</td></tr><tr><td><code>number# = 0</code></td><td>Returns the last number generated</td></tr><tr><td><code>number# > 0</code></td><td>Returns the next random number</td></tr></table>	<code>number# < 0</code>	Always returns the same number	<code>number# = 0</code>	Returns the last number generated	<code>number# > 0</code>	Returns the next random number
<code>number# < 0</code>	Always returns the same number						
<code>number# = 0</code>	Returns the last number generated						
<code>number# > 0</code>	Returns the next random number						
Example	 Rnd Example						
See Also	 Math Statements and Functions						

RTrim\$ Function

Description	Returns the string with the trailing spaces removed.
Syntax	<code>rtrim\$(str\$)</code>
Example	 RTrim\$ Example

See Also



[Strings](#)

SaveEnv Function



Description	Pushes the current environment variable set onto a virtual stack so that they can later be restored by the RestoreEnv function.
Syntax	<code>SaveEnv ([mode\$])</code>
Comments	<p>Returns the integer TRUE if the function is successful, otherwise FALSE.</p> <p>Separate stacks are kept for the Windows and DOS environments.</p> <p>The <code>mode\$</code> parameter can be ENV_DOS or ENV_WINDOWS.</p> <p>If mode is unspecified, the default is ENV_WINDOWS.</p>

Example



[SaveEnv Example](#)

See Also



[Environment Statements and Functions](#)

SaveFileName\$ Function

Description	Displays the common file save dialog box (from COMMDDL.G.DLL), allowing the user to select a file. If the file already exists, the user is prompted to overwrite it.
Syntax	<code>SaveFileName\$(title\$ [,extensions\$])</code>
Returns	Returns a full DOS pathname of the file that the user selected.
Comments	<p>The <code>title\$</code> parameter specifies the title that appears on the dialog box's caption.</p> <p>The <code>extensions\$</code> parameter specifies the available file types. This string should be in the following format:</p> <p style="text-align: center;"><code>"type:ext[,ext][;type:ext[,ext]]..."</code></p> <p>where <i>ext</i> is a valid file extension, like *.BAT or *.?F?, and <i>type</i> is a string that identifies this type to the user.</p> <p>By default, the first extension in appearing within <code>extensions</code> is used.</p>

Example



[SaveFileName\\$ Example](#)



See Also




[Dialog Display](#)

Second Function

Description	Returns an integer representing the second of the day encoded in the specified <code>serial#</code> parameter. The value returned is between 0 and 59 inclusive.
--------------------	--

Syntax	<code>second(serial#)</code>
Comment	You can obtain the value for the <code>serial#</code> parameter by using the TimeSerial or TimeValue command.
Example	 Second Example
See Also	 Date and Time Functions

Seek Statement and Function

Description	The <code>seek</code> function returns the file pointer for a given file. The <code>seek</code> statement sets the file pointer.
Function Syntax	<code>seek(filenumbers%)</code>
Statement Syntax	<code>seek [#] filenumbers%,position&</code>
Comments	The <code>filenumbers</code> parameter is a number that is used by DCL to refer to the open file -- the number passed to the open statement.
See Also	 File Input and Output

Select...Case Statement

Description	Executes a block of DCL statements depending on the value of a given expression.												
Syntax	<pre> select case testexpression [case expressionlist [statement_block]] [case expressionlist [statement_block]] : [case else [statement_block]] end select </pre>												
Comments	<p>The <code>select case</code> statement uses the following arguments:</p> <table> <tr> <td><code>testexpression</code></td><td>Any numeric or string expression</td></tr> <tr> <td><code>statement_block</code></td><td>Any group of DCL statements</td></tr> <tr> <td><code>expressionlist</code></td><td>Any of the following:</td></tr> <tr> <td></td><td><code>expression [,expression]...</code></td></tr> <tr> <td></td><td><code>expression to expression</code></td></tr> <tr> <td></td><td><code>is relational_operator expression</code></td></tr> </table> <p>If the <code>testexpression</code> matches any of the expressions contained in <code>expressionlist</code>, the accompanying block of</p>	<code>testexpression</code>	Any numeric or string expression	<code>statement_block</code>	Any group of DCL statements	<code>expressionlist</code>	Any of the following:		<code>expression [,expression]...</code>		<code>expression to expression</code>		<code>is relational_operator expression</code>
<code>testexpression</code>	Any numeric or string expression												
<code>statement_block</code>	Any group of DCL statements												
<code>expressionlist</code>	Any of the following:												
	<code>expression [,expression]...</code>												
	<code>expression to expression</code>												
	<code>is relational_operator expression</code>												

DCL statements is executed.

The resultant type of *expression* must be the same as that of *testexpression*.

Multiple expression ranges can be used within a single *case* clause. For example:

```
case 1 to 10,12,15, is > 40
```

Only the *statement_block* associated with the first matching expression will be executed.

A *select...end select* expression can also be represented with the *if...then* expression. The use of the *select* statement, however, may be more readable.

Example



[Select...Case Example](#)

See Also



[Flow Control](#)

SelectBox Function



Description	Displays a dialog box containing a listbox of strings. If the user selects an item, the index of that item is returned.
Syntax	<code>SelectBox(title\$,prompt\$,items\$)</code>
Returns	Returns an integer representing the index of the item that the user selected. The first item is 0. The value -1 is returned if the user selects Cancel.
Comments	<p>The <i>items\$</i> parameter must be a single-dimension array of strings, otherwise a runtime error is generated.</p> <p>The <i>title\$</i> parameter specifies the name that appears in the dialog box's caption. The <i>prompt\$</i> parameter specifies the name that appears immediately above the listbox containing the array items.</p> <p>The dialog box uses the 8 point Helvetica font.</p>

Example



[SelectBox Example](#)

See Also



[Dialog Display](#)

SelectButton Statement



Description	Simulates a mouse click on a button, given the button's name or ID.
Syntax 1	<code>SelectButton ButtonName\$</code>
Syntax 2	<code>SelectButton ButtonID%</code>
Comments	<p>You can reference the button by <i>name\$</i> (caption) or <i>id%</i>.</p> <p>A runtime error is generated if a button with the given <i>name\$</i></p>

or `id%` cannot be found in the active window.

Example



[Dialog Examples](#)

See Also



[Dialog Manipulation](#)

SelectComboboxItem Statement



Description

Selects an item from a combobox, given the name or ID of the combobox and the name or line number of the item.

Syntax

```
SelectComboboxItem name$ | id%,ItemName$ |  
ItemNumber%[,isDoubleClick%]
```

Comments

The combobox can be referenced either by `name$` or by `id%`. `Name$` specifies the text that appears in the static control that immediately precedes the combobox control in the window list (or dialog template).

A runtime error is generated if a combobox with the given `name$` or `id%` cannot be found within the active window, or if an item with the given name or line number cannot be found within that combobox.

If the second parameter is either 0 or an empty string (""), all selections will be removed from the combobox.

The optional `isDoubleClick%` parameter specifies if this item is to be selected via a double click or single click. If not specified, then the item is selected using a single click.

Example



[Dialog Examples](#)

See Also



[Dialog Manipulation](#)

SelectListboxItem Statement



Description

This statement selects an item from a list box, given the name or ID of the listbox and the name or line number of the item.

Syntax

```
SelectListboxItem name$ | id%,ItemName$ |  
ItemNumber%[,isDoubleClick%]
```

Comments

The listbox must exist within the current window or dialog, otherwise a runtime error will be generated.

The listbox can be referenced either by `name$` or by `id%`. `Name$` specifies the text that appears in the static control that immediately precedes the listbox control in the window list (or dialog template). A runtime error is generated if a listbox with that name cannot be found within the active window.

If the second parameter is 0 or an empty string (""), then all selections are removed from the listbox. For multi-select listboxes, `SelectListBoxItem` will select additional items (i.e., it will not remove the selection from the currently selected

items).

Example



[Dialog Examples](#)

See Also



[Dialog Manipulation](#)

SendKeys Statement



Description

Sends the specified keys to the active application, optionally waiting for the keys to be processed before continuing.

Syntax

```
SendKeys KeyString$ [,wait%]
```

Comments

The format for `KeyString$` is the same as that used for [DoKeys](#).

If `wait` is TRUE (or not specified), the statement waits for the keys to be completely processed before continuing. Otherwise, execution continues immediately.

Example



[SendKeys Example](#)

See Also



[Keyboard Manipulation](#)

SetAttr Statement

Description

Changes the attributes of the specified file to the given value.

Syntax

```
SetAttr filename$,attribute%
```

Comments

A runtime error results if the file cannot be found.

The `attribute%` parameter contains the sum of the following attributes.

Constant	Value	Description
ATTR_NORMAL	0	Read-only, archive, subdirectory, and files with no attributes
ATTR_READONLY	1	Read-only files
ATTR_HIDDEN	2	Hidden files
ATTR_SYSTEM	4	System files
ATTR_VOLUME	8	Volume label
ATTR_DIRECTORY	16	MS-DOS directories
ATTR_ARCHIVE	32	Files changed since last backup
ATTR_NONE	64	Files with no attributes

These attributes are the same as those used by DOS.

Example



[SetAttr Example](#)

See Also



[File Input and Output](#)

SetCheckbox Statement

Description	Sets the state of the checkbox with the given name or ID.
Syntax 1	<code>SetCheckbox name\$,state%</code>
Syntax 2	<code>SetCheckbox id%,state%</code>
Comments	<p>The checkbox can be specified either by its <code>name\$</code> or using its <code>id%</code>. <code>Name\$</code> is the text of the checkbox label.</p> <p>If <code>state</code> is 1, the box is checked. If <code>state</code> is 0, the check is removed. If <code>state</code> is 2, then the box is grayed (only applicable for 3-state check boxes).</p> <p>A runtime error is generated if a check box with the specified name cannot be found in the active window.</p> <p>This statement has the side-effect of setting the focus to the given checkbox.</p>

Example



[Dialog Examples](#)

See Also



[Dialog Manipulation](#)

SetEditText Statement



Description	Sets the content of an edit control, given its name or ID.
Syntax 1	<code>SetEditText name\$,content\$</code>
Syntax 2	<code>SetEditText id%,content\$</code>
Comments	<p>The <code>name\$</code> parameter specifies the text that appears within the static control that immediately precedes the edit control in the window list (or dialog template). Alternatively, the <code>id%</code> of the edit box can be specified.</p> <p>This statement has the side-effect of setting the focus to the given edit control.</p>

Example



[Dialog Examples](#)

See Also



[Dialog Manipulation](#)

SetEnv Function



Description	Sets the specified environment variable to the given value for DOS, Windows, or Both.
Syntax	<code>SetEnv (var\$, value\$[, mode\$])</code>
Comments	<p>When you are running Windows, there are two environments. One environment is used for Windows applications and one for DOS applications.</p> <p>To control which environment you set variables for, use the <code>mode\$</code> parameter. The <code>mode\$</code> can be:</p> <p>-- ENV_DOS, which sets environment variables for DOS</p>

applications.

- ENV_WINDOWS, which sets environment variables for Windows.
- ENV_BOTH, which sets environment variables for both DOS and Windows.

If it is an invalid value, or not set, the default ENV_WINDOWS is used.

Changes made to Windows environment variables are available to all Windows applications--including those that have already been launched.

The function returns TRUE if successful, or FALSE otherwise. If the functions fails, the environment may be full.

Example



[SetEnv Example](#)

See Also



[Environment Statements and Functions](#)

SetIcon Statement



Description

Specifies which icon to show when [ShowIcon](#) is called.

Syntax

```
SetIcon iconfile$ [,icon%]
```

Comments

The `iconfile$` specifies an .EXE, .DLL, or .ICO file. `Icon%` is the zero-based index into the list of icons the file contains. If unspecified, `icon%` is assumed to be 0.

If the icon cannot be found or is not a valid format, or if the value of `icon%` exceeds the number of icons in the file, a runtime error is generated.

This function has no affect on scripts run from the editor or debugger.

See Also



[Icons](#)

SetIconTitle Statement



Description

Sets the title to be displayed under the icon (if shown) for a compiled script.

Syntax

```
SetIconTitle title$
```

Comments

This function has no affect on scripts run from the editor or debugger.



See Also





[Icons](#)

SetOption Statement



Description	Clicks on the specified option button.
Syntax 1	<code>SetOption name\$</code>
Syntax 2	<code>SetOption id%</code>
Comments	<p>The option button can be referenced either by its <code>name\$</code> (caption) or its <code>id%</code>.</p> <p>A runtime error is generated if the option button cannot be found within the active window.</p>
Example	 Dialog Examples
See Also	 Dialog Manipulation

Sgn Function

Description	Returns an integer indicating if a number is less than, greater than, or equal to zero.
Syntax	<code>sgn (number)</code>
Comments	<p>Returns 1 if <code>number</code> is greater than 0.</p> <p>Returns 0 if <code>number</code> is equal to 0.</p> <p>Returns -1 if <code>number</code> is less than 0.</p> <p>The <code>number</code> parameter is a numeric expression of any type.</p>
Example	 Sgn Example
See Also	 Math Statements and Functions

Shell Function

Description	Runs the program (with any specified parameters) contained in <code>command\$</code> .
Syntax	<code>shell (command\$ [,WindowStyle%])</code>
Returns	If successful, the function returns an integer representing the task ID. For other return codes, see "Errors" below.
Comments	<p>The optional <code>WindowStyle%</code> parameter specifies the state of the application window after execution. It can be any of the following values:</p> <ol style="list-style-type: none">1 Normal window with focus2 Minimized with focus3 Maximized with focus4 Normal window without focus7 Minimized without focus

An error is generated if unsuccessful. A return value less than or equal to 32 specifies an error.

Errors

This function returns the value 31 if there is no association for the specified file type or if there is no association for the specified action within the file type. The other possible error values are as follows:

- 0 System was out of memory, executable file was corrupt, or relocations were invalid.
- 2 File was not found.
- 3 Path was not found.
- 5 Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.
- 6 Library required separate data segments for each task.
- 8 There was insufficient memory to start the application.
- 10 Windows version was incorrect.
- 11 Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
- 12 Application was designed for a different operating system.
- 13 Application was designed for MS-DOS 4.0.
- 14 Type of executable file was unknown.
- 15 Attempt was made to load a real-mode application (developed for an earlier version of Windows).
- 16 Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.
- 19 Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
- 20 Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
- 21 Application requires Microsoft Windows 32-bit extensions.

Example



[Shell Example](#)

See Also



[Flow Control](#)

ShowIcon Statement



Description

Displays an icon on the desktop while the script is running.

Syntax

```
ShowIcon [show%]
```

Comments

Normally, an icon does not show up on the desktop while a compiled script is running.

If `show%` is set to TRUE (the default value), an icon will appear on the desktop when the script is run. The purpose of the icon

is to allow the user to stop the script if the user has been permitted this authority (through the [EnableStopScript](#) command).



This function has no affect on scripts run from the editor or debugger.

See Also





[Icons](#)

Sin Function

Description	Returns a double-precision number representing the sine of a given angle.
Syntax	<code>sin(angle#)</code>
Comments	The <code>angle#</code> parameter is given in radians.
Example	 Sin Example
See Also	 Math Statements and Functions

Sleep Statement

Description	Causes the script to pause for a specified number of milliseconds.
Syntax	<code>sleep milliseconds&</code>
Comments	While paused, other applications can execute.
Example	 Sleep Example
See Also	 Flow Control

SleepUntil Function



Description	Pauses the script until the specified time is reached.
Syntax	<code>SleepUntil(time\$[,showdialog%[,text\$[, caption\$[,cancel%[,minimizebox]]]]], usenetttime%)</code>
Comments	<p>The integer TRUE is returned if the pause is completed normally. FALSE is returned if an error is encountered or if the user cancels the operation.</p> <p>The <code>time\$</code> value must be in one of the following formats:</p> <p>12 Hour - "HH:MM xm"</p> <p>24 Hour - "HH:MM"</p> <p>HH = hours (may be 1 or 2 digits for 12 hour format, but must be 2 digits for 24 hour)</p> <p>MM = minutes</p> <p>xm = am/pm indicator (the 'x' will be replaced by 'a' or 'p')</p> <p>Colons (":") are expected between the hour and minute portions of the time. In the 12-hour format, a space is expected before the "xm" portion.</p> <p>Any deviation from this time format will generate a runtime error.</p> <p>You can set <code>showdialog%</code> to TRUE or FALSE. If TRUE, a</p>

dialog is displayed with the specified features. `text$` specifies the text to display in the dialog box. The dialog resizes to display the text. If text not specified, no text is used. The `caption$` parameter specifies the caption to be used. If not given, then "Sleeping Until"+`time$` is used.

If `cancel%` is specified, a Cancel button appears in the dialog box. Clicking on the Cancel button stops the sleep and returns FALSE.

If `minimizebox%` is specified, then the dialog can be minimized. It will show a clock icon.

Set `usenetttime%` to TRUE to use network time instead of DOS time.

Example



[SleepUntil Example](#)

See Also



[Flow Control](#)

Snapshot Statement



Description

Takes a snapshot of a particular section of the screen and saves it to the clipboard.

Syntax

`snapshot [spec%]`

Comments

The `spec` parameter specifies the screen area as follows:

- 0 Entire screen
- 1 Client area of the active application
- 2 Entire window of the active application
- 3 Client area of the active window
- 4 Entire window of the active window

Before the snapshot is taken, each application is updated. This ensures that any application in the middle of drawing will have a chance to finish before the snapshot is taken.

There is a slight delay if the specified window is large.

Example



[Snapshot Example](#)

See Also



[Clipboard Manipulation](#)

Space\$ Function

Description

Returns a string containing the specified number of spaces.

Syntax

`space$ (NumSpaces%)`

Comments

NumSpaces must be between 0 and 32767.

Example



[Space\\$ Example](#)

See Also



[Strings](#)

Sqr Function

Description Returns a double-precision number representing the square root of a given value.

Syntax `sqr (number#)`

Comments The `number` parameter must be greater than or equal to 0.

Example



[Sqr Example](#)

See Also



[Math Statements and Functions](#)

Stop Statement

Description Stops execution of a script.

Syntax `stop`

Comments This command terminates execution of the current script and displays the message: "Stopped at line X", where X is the line number containing the stop statement. All open files are closed. All open DDE channels are closed.

Example



[Stop Example](#)

See Also



[Flow Control](#)

StrComp Function

Description Compares two strings.

Syntax `StrComp (string1$, string2$ [, compare%])`

Returns Returns an integer value indicating the result of comparing the two string arguments:

```
0    string1$ = string2$
1    string1$ > string2$
-1   string1$ < string2$
```

Comments The `StrComp` function compares two strings and returns an integer indicating the result of the comparison. The comparison can be either case-sensitive or case-insensitive, depending on the value of the optional `compare%` parameter:

```
0    Case-sensitive comparison
1    Case-insensitive comparison
```

If `compare%` is not specified, then the comparison is case-sensitive (0).

Example



StrComp Example


See Also






Strings

StringSort Function




Description	Sorts a one-dimensional array of strings in ascending order.
Syntax	<code>StringSort list\$()</code>
Comments	If an array of more than one dimension is specified, a runtime error is generated.
See Also	 <u>Strings</u>

Str\$ Function

Description	Returns a string representation of the given number.
Syntax 1	<code>str\$(number%)</code>
Syntax 2	<code>str\$(number&)</code>
Syntax 3	<code>str\$(number!)</code>
Syntax 4	<code>str\$(number#)</code>
Comments	<p>If <code>number</code> is negative, then the returned string will contain a leading minus sign.</p> <p>If <code>number</code> is positive, then the returned string will contain a leading space.</p> <p>Singles are printed using only 7 significant digits. Doubles are printed using 15-16 significant digits.</p>
Example	 <u>Str\$ Example</u>
See Also	 <u>Conversions</u>  <u>Strings</u>

String\$ Function

Description	Returns a string of <code>number%</code> length consisting of a repetition of the specified filler character.
Syntax 1	<code>string\$(number%,CharCode%)</code>
Syntax 2	<code>string\$(number%,str\$)</code>
Comments	<p>If <code>CharCode</code> is specified, the character with this ASCII value is used as the filler character.</p> <p>If <code>str</code> is specified, then the first character of this string is used as the filler character.</p>
Example	

String\$ Example

See Also



Strings

Sub...End Sub Statement

Description Declares a subroutine.

Syntax

```
sub name[(parameter [as type]...)]
end sub
```

Comments Parameters are passed to a subroutine by reference, meaning that any modification to a passed parameter changes that variable in the caller. To avoid this, simply enclose variable names in parentheses, as in the following example function calls:

```
UserSub 10,12,(j)
```

If a subroutine is not to receive a parameter by reference, the optional `byval` keyword can be used:

```
sub Test byval FileName as string
end sub
```

A subroutine terminates when one of the following statements is encountered:

```
end sub
exit sub
```

The name of the subroutine must follow DCL naming conventions. It cannot include type declaration characters.

Subroutines can be recursive.

See Also



Procedure Statements

SystemFreeMemory Function

Description Returns a long integer representing the number of bytes of free memory.

Syntax

```
SystemFreeMemory()
```

Example



SystemFreeMemory Example

See Also



Environment Statements and Functions

SystemFreeResources Function





Description Returns an integer representing the percentage of free system resources.

Syntax

```
SystemFreeResources()
```

Comments The returned value is between 0 and 100.

Example  [SystemFreeResources Example](#)


See Also  [Environment Statements and Functions](#)


SystemMouseTrails Statement

Description Turns on or off mouse trails.

Syntax `SystemMouseTrails state%`

Comments The setting is saved in the INI file permanently.
This option is only available under Windows 3.1.


Example  [SystemMouseTrails Example](#)


See Also  [Environment Statements and Functions](#)

SystemRestart Statement

Description Restarts Windows, much like the Window Setup program.

Syntax `SystemRestart`


Example  [SystemRestart Example](#)


See Also  [Environment Statements and Functions](#)

SystemTotalMemory Function

Description Returns a long integer representing the total available free memory in Windows in bytes.

Syntax `SystemTotalMemory()`


Example  [SystemTotalMemory Example](#)


See Also  [Environment Statements and Functions](#)

SystemWindowsDirectory\$ Function

Description Returns the directory where Windows is stored, such as "C:\WINDOWS".

Syntax `SystemWindowsDirectory$()`

Example  [SystemWindowsDirectory\\$ Example](#)

See Also  [Environment Statements and Functions](#)

SystemWindowsVersion\$ Function

Description Returns the Windows version, such as "3.0" or "3.1".

Syntax `SystemWindowsVersion$()`

Example 
[SystemWindowsVersion\\$ Example](#)

See Also 
[Environment Statements and Functions](#)

Tan Function

Description Returns a double-precision number representing the tangent of the specified angle.

Syntax `tan(angle#)`

Comments The `angle` parameter is given in radians.

Example 
[Tan Example](#)

See Also 
[Math Statements and Functions](#)

Text Statement

Description Defines a text control in a dialog template.

Syntax `Text x%,y%,width%,height%,title$`

Comments The purpose of `Text` controls is simply to display text.

The `title$` parameter will be truncated if the width of the control is insufficient to hold the entire content. The `title$` parameter may contain an ampersand character to denote an underlined accelerator, such as "&Font" for Font.

This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).

The `x,y,width,height` parameters are specified in dialog coordinates. The `x,y` position is relative to the upper left corner of the dialog box.

Example 
[Dialog Examples](#)

See Also 
[Dialog Creation](#)

TextBox Statement

Description Defines a text-entry field that appears within a dialog box template.

Syntax `TextBox x as integer,y as integer,width as`

`integer,height as integer,.Field`

Comments

This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).

The `x,y,width,height` parameters are specified in dialog coordinates. The `x,y` position is relative to the upper left corner of the dialog box.

When the dialog box is created, the content of the `.Field` is used to set the initial content of the textbox. When the `Dialog` statement returns, the `.Field` is used to determine the final content of the textbox. The `.Field` always contains a string.

Example



[Dialog Examples](#)

See Also



[Dialog Creation](#)

Time\$ Statement and Function

Description

The `time$` assignment statement sets the system time; the `time$` function returns the system time.

Assignment Syntax

`time$ = newtime$`

Comments

This statement sets the system time to the time contained in the specified string.

The format for `newtime$` must be one of the following:

`HH`

`HH:MM`

`HH:MM:SS`

A 24 hour clock is used.

Example



[Time Statement Example](#)

Function Syntax

`time$()`

Returns

Returns the system time as an 8 character string.

Comments

The format of the returned string is `HH:MM:SS`.

Example



[Time Function](#)

See Also



[Date and Time Functions](#)

Timer Function

Description

Returns a long integer representing the number of seconds that have occurred since midnight.

Syntax

`timer`

Example



[Timer Example](#)

See Also



[Date and Time Functions](#)

TimeSerial Function

Description

Returns a double-precision number representing the given time with today's date. The number is returned in days where Dec 30, 1899 is 0.

Syntax

`TimeSerial(hour%,minute%,second%)`

Example



[TimeSerial Example](#)

See Also



[Date and Time Functions](#)

TimeValue Function

Description

Returns a double-precision number representing the time contained in the specified string argument.

Syntax

`TimeValue(time_string$)`

Comments

This function interprets the passed `time_string$` parameter looking for a valid time specification. Time specifications vary depending on the international settings contained in the INTL section of the WIN.INI file.

The `time_string$` parameter can contain valid time items separated by time separators such as colon (:) or period (.). The time items must follow the ordering determined by the current time format settings in use by Windows.

Time strings can contain an optional date specification, but this is not used in the formation of the returned value.

If a particular time item is missing, then the missing time items are set to zero. For example, the string "10 pm" would be interpreted as "22:00:00".

Example



[TimeValue Example](#)

See Also



[Date and Time Functions](#)

Trim\$ Function

Description

Removes leading and trailing spaces.

Syntax


`trim$(str$)`

Returns

Returns a copy of the passed string expression (`str$`) with leading and trailing spaces removed.

Example



See Also [Trim\\$ Example](#)
 [Strings](#)

TRUE

Description Constant.
Returns -1
Comments Used in conditionals and boolean expressions.



TYPE_DOS

Description Constant.
Returns 1
Comment Used with the [AppType](#) function to indicate a DOS application.



TYPE_WINDOWS

Description Constant.
Syntax 2
Comment Used with the [AppType](#) function to indicate a Windows application.



UBound Function

Description Returns an integer representing the upper bound of the specified dimension of the specified array variable.
Syntax `ubound(ArrayVariable() [,dimension%])`
Comments The first dimension (1) is assumed if `dimension` is not specified.
Example  [UBound Example](#)
See Also  [Arrays](#)



UCase\$ Function

Description Returns the uppercase equivalent of the specified string.
Syntax `ucase$(str$)`
Example  [UCase\\$ Example](#)
See Also  [Strings](#)

Val Function

Description	Converts a given string expression to a double-precision number.										
Syntax	<code>val (number\$)</code>										
Comments	<p>The <code>number\$</code> parameter can contain any of the following:</p> <ul style="list-style-type: none">-- Leading minus sign (for non hex or octal numbers only)-- Hexadecimal number in the format: <code>&H<hex digits></code>-- Octal number in the format: <code>&O<octal digits></code>-- Floating point number, which can contain a decimal point and optional exponent <p>Spaces, tabs, and linefeeds are ignored.</p> <p>If <code>number\$</code> does not contain a number, 0 is returned.</p> <p>The <code>val ()</code> function continues to read characters from the string up to the first non-numeric character.</p> <p>The <code>val ()</code> function always returns a double-precision floating point value. This value is forced to the data type of the assigned variable.</p>										
Example 1	 Val Example										
Example 2	<p>The following table shows valid strings and their numeric equivalent:</p> <table><tr><td>"1 2 3"</td><td>123</td></tr><tr><td>"12.3"</td><td>12.3</td></tr><tr><td>"&HFFFF"</td><td>-1</td></tr><tr><td>"&O77"</td><td>64</td></tr><tr><td>" 12.345E-02"</td><td>.12345</td></tr></table>	"1 2 3"	123	"12.3"	12.3	"&HFFFF"	-1	"&O77"	64	" 12.345E-02"	.12345
"1 2 3"	123										
"12.3"	12.3										
"&HFFFF"	-1										
"&O77"	64										
" 12.345E-02"	.12345										
See Also	Conversions  Strings										

ViewportClear Statement

Description	Clears the open viewport window.
Syntax	<code>ViewportClear</code>
Comments	The statement has no effect if no viewport is opened.
Example	 Viewport Example
See Also	 Viewport Window Manipulation

ViewportClose Statement

Description Closes an open viewport window.

Syntax ViewportClose

Example 
[Viewport Example](#)

See Also 
[Viewport Window Manipulation](#)

ViewportOpen Statement

Description Opens a viewport window.

Syntax ViewportOpen [title\$ [,x%,y% [,width%,height%]]]

Comments The optional `title$` parameter specifies the text to appear in the viewport's caption. The `x` and `y` parameters specify an optional initial position in twips, and the optional `width` and `height` parameters specify an optional initial width and height for the viewport window.

This statement has no effect if a viewport window is already open.

Combined with the [print](#) statement, a viewport window is a convenient place to output debugging information.

The viewport window is closed when the DCL host application is terminated.

The buffer size for the viewport is 32K. Information from the start of the buffer is removed to make room for additional information being appended to the end of the buffer.

The following keys work within a viewport window:


Up	Scroll up by one line
Down	Scroll down by one line
Home	Scroll to the first line in the viewport window
End	Scroll to the last line in the viewport window
PgUp	Scroll the viewport window down by one page
PgUp	Scroll the viewport window up by one page
Ctrl+PgUp	Scroll the viewport window left by one page
Ctrl+PgDn	Scroll the viewport window right by one page

Only 1 viewport window can be open at any one time. Any scripts with `print` statements will output information into the same viewport window.


Example 
[Viewport Example](#)

See Also 
[Viewport Window Manipulation](#)



VK_LBUTTON

Description	Constant used with the <code>QueMouse . . .</code> commands to represent the left button.
Value	1
See Also	 Mouse Events



VK_RBUTTON

Description	Constant used with the <code>QueMouse . . .</code> commands to represent the right button.
Value	2
See Also	 Mouse Events

VLine Statement

Description	Scrolls the window with the focus up or down by the specified number of lines.
Syntax	<code>VLine [lines%]</code>
Comments	If the <code>lines</code> parameter is omitted, then the window is scrolled down by 1 line.
Example	 VLine Example
See Also	 Window Manipulation

VPage Statement

Description	Scrolls the window with the focus up or down by the specified number of pages.
Syntax	<code>VPage [pages%]</code>
Comments	If the <code>pages</code> parameter is omitted, then the window is scrolled down by 1 page.
Example	 VPage Example
See Also	 Window Manipulation

VScroll Statement

Description	Sets the thumb mark on the vertical scroll bar attached to the current window.
Syntax	<code>VScroll percentage%</code>
Comments	The position is given as a percentage of the total range

associated with that scroll bar. For example, if the `percentage%` parameter is 50, then the thumb is positioned in the middle of the scroll bar.

Example



[VScroll Example](#)

See Also



[Window Manipulation](#)

WaitForTaskCompletion Function



Description

Waits until the task specified by `taskid%` is exited.

Syntax

```
WaitForTaskCompletion taskid%
```

Comments

The `taskid%` is the return value from the [Shell](#) statement.

If `taskid%` is not currently running, control returns immediately.

Example



[WaitForTaskCompletion Example](#)

See Also



[Flow Control](#)

Weekday Function

Description

Returns an integer representing the day of the week of the date encoded in the specified `serial` parameter. The value returned is between 1 and 7 inclusive where 1 is Sunday.

Syntax

```
weekday(serial#)
```

Comment

You can obtain the value for the `serial#` parameter by using the [DateSerial](#) or [DateValue](#) command.

Example



[Weekday Example](#)

See Also



[Date and Time Functions](#)

While...Wend Statement

Description

Repeats a statement or group of statements while a condition is TRUE.

Syntax

```
while <condition>  
    [<statement>]  
wend
```

Example



[While...Wend Example](#)



See Also



[Flow Control](#)

WinActivate Statement



Description	Activates the window with the given name or window handle.
Syntax 1	<code>WinActivate window_name\$</code>
Syntax 2	<code>WinActivate hWnd%</code>
Comments	<p>The <code>window_name\$</code> parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. Optionally, a partial name can be used, such as "Word" for "Microsoft Word".</p> <p>A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:</p> <pre>WinActivate "Notepad Find"</pre> <p>In the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.</p> <p>If the <code>hWnd%</code> parameter is specified rather than the <code>window_name\$</code> parameter, then focus is set immediately to the window with that handle.</p> <p>Windows without captions cannot be activated using this command.</p>
Example	 WinActivate Example
See Also	 Window Manipulation

WinClose Statement



Description	Closes the given window.
Syntax	<code>WinClose [window_name\$]</code>
Comments	<p>If no <code>window_name\$</code> parameter is specified, the window with the focus is closed.</p> <p>The <code>window_name\$</code> parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. Optionally, a partial name can be used, such as "Word" for "Microsoft Word".</p> <p>A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:</p> <pre>WinActivate "Notepad Find"</pre> <p>In the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.</p>

This command differs from the [AppClose](#) command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window.

Example



[WinClose Example](#)

See Also



[Window Manipulation](#)

WinFind Function



Description

Returns an integer representing a handle to the specified window.

Syntax

`WinFind (name$)`

Comments

The `name$` is specified using the same format as that used by the [WinActivate](#) statement.

Example



[WinFind Example](#)

See Also



[Window Manipulation](#)

WinList Function



Description

Fills the passed array with the handles to all the top-level windows.

Syntax

`WinList hWnd$()`

Comments

After calling this function, use the [lbound\(\)](#) and [ubound\(\)](#) functions to determine the new size of the array.

Example



[WinList Example](#)

See Also



[Window Manipulation](#)

WinMaximize Statement



Description

This command maximizes the specified window.

Syntax

`WinMaximize [window_name$]`

Comments

If no `window_name$` parameter is specified, the window with the focus is maximized.

The `window_name$` parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. Optionally, a partial name can be used, such as "Word" for "Microsoft Word".

A hierarchy of windows can be specified by separating each window name with a vertical bar (|), as in the following example:

```
WinActivate "Notepad|Find"
```

In the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

This command differs from the [AppMaximize](#) command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window.

Example



[WinMaximize Example](#)

See Also



[Window Manipulation](#)

WinMinimize Statement



Description

Minimizes the specified window.

Syntax

```
WinMinimize [window_name$]
```

Comments

If no `window_name$` parameter is specified, the window with the focus is minimized.

The `window_name$` parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. Optionally, a partial name can be used, such as "Word" for "Microsoft Word".

A hierarchy of windows can be specified by separating each window name with a vertical bar (|), as in the following example:

```
WinActivate "Notepad|Find"
```

In the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

This command differs from the [AppMinimize](#) command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window.

Example



[WinMinimize Example](#)

See Also



[Window Manipulation](#)

WinMove Statement



Description

Moves the specified window.

Syntax

```
WinMove x%,y% [window_name$]
```

Comments

This command moves the given window to the given x,y position. If no `window_name$` parameter is specified, then the

window with the focus is moved.

The `window_name$` parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. Optionally, a partial name can be used, such as "Word" for "Microsoft Word".

A hierarchy of windows can be specified by separating each window name with a vertical bar (`|`), as in the following example:

```
WinActivate "Notepad|Find"
```

In the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

This command differs from the `AppMove` command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window. When moving child windows, remember that the `x%` and `y%` parameters are relative to the client area of the parent window.

Example



[WinMove Example](#)

See Also



[Window Manipulation](#)

WinRestore Statement



Description

Restores the specified window.

Syntax

```
WinRestore [window_name$]
```

Comments

If no `window_name$` parameter is specified, the window with the focus is restored.

The `window_name$` parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. Optionally, a partial name can be used, such as "Word" for "Microsoft Word".

A hierarchy of windows can be specified by separating each window name with a vertical bar (`|`), as in the following example:

```
WinActivate "Notepad|Find"
```

In the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

This command differs from the `AppRestore` command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window.

Example



[WinRestore Example](#)

See Also



[Window Manipulation](#)

WinSize Statement



Description Resizes the specified window.

Syntax `WinSize width%,height% [window_name$]`

Comments This command resizes the given window to the specified width and height. If no `window_name$` parameter is specified, the window with the focus is resized.

The `window_name$` parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. Optionally, a partial name can be used, such as "Word" for "Microsoft Word".

A hierarchy of windows can be specified by separating each window name with a vertical bar (|), as in the following example:

```
WinActivate "Notepad|Find"
```

In the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

This command differs from the [AppSize](#) command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window.

Example



[WinSize Example](#)

See Also



[Window Manipulation](#)

Word\$ Function

Description Extracts words from a text source.

Syntax `word$(text$,first%[,last%])`

Returns Returns a single word or sequence of words between `first` and `last`.

Comments The `first` parameter specifies the first word in the sequence to return. If `last` is not specified, then only that word is returned. If `last` is specified, then all words between `first` and `last` will be returned, including all spaces, tabs, and end-of-lines that occur between those words.

Word are separated by spaces, tabs, and end-of-lines.

If `first` is greater than the number of words in `text$`, then an empty string is returned.

If `last` is greater than the number of words in `text$`, then all words from `first` to the end of `text` are returned.

Example



[Word\\$ Example](#)

See Also



[Strings](#)

WordCount Function

Description

Returns an integer representing the number of words in the specified `text`.

Syntax

`WordCount (text$)`

Comments

Word are separated by spaces, tabs, and end-of-lines.

Example



[WordCount Example](#)

See Also



[Strings](#)

Write # Statement

Description

Writes a list of expressions to the specified file.

Syntax

`write [#]filename% [,expressionlist]`

Comments

The file referenced by `filename` must be opened in either `output` or `append mode`.

The `filename` parameter is a number that is used by DCL to refer to the open file -- the number passed to the [open](#) statement.

See Also



[File Input and Output](#)

WriteINI Statement

Description

Writes a new value into an INI file.

Syntax

`WriteINI section$,ItemName$,value$[,filename$]`

Comments

The `filename$` parameter, if specified, contains the name of an INI file. If `filename$` is not specified, the `WIN.INI` file is used.

The `section$` parameter specifies the section which contains the desired variables, such as "windows". Section names are specified without the enclosing brackets.

The `ItemName$` parameter specifies which item from within the given section you want to change. If `ItemName$` is an empty string (""), then the entire section specified by `section$` is deleted.

The `value$` parameter specifies the new value for the given item. If `value$` is an empty string (""), then the item specified by

ItemName\$ is deleted from the INI file.

Example



[WriteINI Example](#)

See Also



[Environment Statements and Functions](#)

WS_MAXIMIZED

Description Constant used with the [AppSetState](#) and [AppGetState](#) statements to indicate a maximized window state.

Value 1

WS_MINIMIZED

Description Constant used with the [AppSetState](#) and [AppGetState](#) statements to indicate a minimized window state.

Value 2

WS_RESTORED

Description Constant used with the [AppSetState](#) and [AppGetState](#) statements to indicate a normal window state.

Value 3

Xor

Description Xor operator.

Syntax expression1 XOR expression2

Returns If both operands are relational, then XOR returns the logical exclusive OR of expression1 and expression2. In other words, XOR returns TRUE only if both operands are not equal.

If both operands are numeric, the result is the bitwise XOR of the arguments.

Notes If either of the two operands is a floating point number, the two operands are first converted to longs, then a bitwise XOR is performed.

Example



[Xor Example](#)

See Also




[Operators](#)


Year Function

Description Returns an integer representing the year of the date encoded in the specified serial parameter. The value returned is between 100 and 9999 inclusive.

Syntax `year(serial#)`

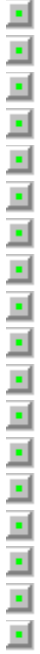
Comment You can obtain the value for the `serial#` parameter by using the [DateSerial](#) or [DateValue](#) command.

Example 
[Year Example](#)

See Also 
[Date and Time Functions](#)

This is an ID assigned to the control by the program. It can be obtained through Windows diagnostic utilities, such as Spy.

Sample Scripts



Note: In almost all of the sample scripts, the `sub main()` and `end sub` statements are commented out because the **DCL Editor** includes these commands in each new script. These commands are not commented out in sample scripts where functions are defined or multiple subroutines used.

A

[Abs](#)

[AddIni](#)

[And](#)

[AnswerBox](#)

[AppActivate](#)

[AppClose](#)

[AppFileName\\$](#)

[AppFind](#)

[AppGetActive\\$](#)

[AppGetPosition](#)

[AppGetState](#)

[AppHide](#)

[AppList](#)

[AppMaximize](#)

[AppMinimize](#)

[AppMove](#)

[AppRestore](#)

[AppSetState](#)

[AppShow](#)

[AppSize](#)

[AppType](#)

[ArrayDims](#)

[ArraySort](#)

[Asc](#)

AskBox\$

AskPassword\$

Atn

B -- C

Beep

ButtonEnabled

ButtonExists

Call

CDBl

ChDir

ChDrive

Chr\$

CInt

Clipboard\$

ClipboardClear

CLng

CSng

CStr

CurDir\$

D

Date\$ Statement

Date\$ Function

DateSerial

DateValue

Day

DCLHomeDir\$

DCLOS\$

DCLVersion\$

DDE Commands

Declare

DEFtype

DesktopCascade

DesktopSetColors

DesktopSetWallpaper

DesktopTile

Dialog Commands

OK, Cancel, and Push Buttons

A Comprehensive Dialog Example

Dialog Statement and Function

Dim

DirExists

Dir\$

DiskDrives

DiskFree

Do...Loop

DoEvents

DoKeys

E

EnableStopScript

End

Environ\$

Err

Error Statement and Function

Exclusive

Exit Commands

Exp

F

FileCopy

FileDateTime

FileDirs

FileExists

FileLen

FileList

FileParse

FileType

FindFile\$

Fix

For...Next

FreeFile

G

GetAttr

GetEnv

GoSub

Goto

H -- K

Hex\$

HLine

Hour

HPage

HScroll

If...Then...Else

InputBox\$

Input/Output Commands

InStr

Int

Item\$ and ItemCount

Kill

L

LBound

LCase\$

Left\$

Len

Let

Line\$

LineCount

Log

LTrim\$

M

MCI

Menu

Message Commands

Mid\$

Minute

MkDir

Mod

Month

MsgBox

N

Name

NetAttach

NetConnectDrive

NetDetach

NetDirectoryRights

NetDisconnectDrive

NetGetDirectoryRights

NetMemberOf

NetStationID

NetUserName

NetworkStatus

Not

Now

Null

O

Oct\$

On Error

OpenFileName\$

Option Base

Or

P

PI

PopupMenu

Print

Print #

PrinterGetOrientation

PrinterSetOrientation

PrintFile

Q

Queue Commands

R

Random

Randomize
ReadINI\$
ReadINISection
ReDim
REM
RefreshIni
Reset
RestoreEnv
Resume
Return
Right\$
RmDir
Rnd
RTrim\$

S

SaveEnv
SaveFileName\$
Second
Select...Case
SelectBox
SendKeys
SetAttr
SetEnv
Sgn
Shell
Sin
Sleep
SleepUntil
Snapshot
Space\$
Sqr
Stop
StrComp
Str\$
String\$
SystemFreeMemory
SystemFreeResources
SystemMouseTrails
SystemRestart
SystemTotalMemory
SystemWindowsDirectory\$
SystemWindowsVersion\$

T

Tan
Time Statement
Time Function
Timer
TimeSerial

TimeValue

Trim\$

U -- V

UBound

UCase\$

Val

Viewport Commands

VLine

VPage

VScroll

W -- Z

WaitForTaskCompletion

Weekday

While...Wend

WinActivate

WinClose

WinFind

WinList

WinMaximize

WinMinimize

WinMove

WinRestore

WinSize

Word\$

WordCount

WriteINI

Xor

Year

'DDE Example



Copy

Copy Script

Print Script

Close Window

```
'sub main()
'   This script illustrates the use of the DDE commands.
'   The current shell is used to demonstrate this.
'   NetTools Applications Manager is assumed to be the
'   shell.
'
'   Demonstrated in this script are:
'   DDEInitiate, DDETimeout, DDERequest, DDETerminate,
'   DDEExecute, and DDETerminateAll

dim channel as integer
dim winshell as string
dim topic as string
dim reqstr as string
dim grouplist() as string
dim i as integer
dim nl as integer
dim selitem as integer
dim groupname as string

winshell = "AppMan"
topic = "AppMan"

'   initiate a DDE conversation with the shell
channel = DDEInitiate(winshell,topic)
DDETimeout 5000           'set time out to 5 seconds (default is 10000 milliseconds)
if channel = 0 then
    msgbox "Unable to initiate DDE link with "+winshell+ ". Script will end now."
end
end if

'   Get a list of the groups in the shell
reqstr = DDERequest$(channel,"Groups")
'   Now the group names have to be placed into an array
'   for use in a Selection Box
nl = LineCount(reqstr) - 1
redim grouplist(1 to nl)
for i = 1 to nl
    grouplist(i) = Line$(reqstr,i)
next i

'   Now that we have the groups in a usable list,
'   allow the user to make selections to
'   view the items in the group
viewgroups:
selitem = SelectBox("Groups",winshell+" Groups:",grouplist)
if selitem = 0 then goto shutdown

'   Get the group name, and enumerate the item names
groupname = grouplist(selitem)
reqstr = DDERequest$(channel,groupname)
'   Display the items in a message box
msgbox reqstr
```

```
        goto viewgroups

shutdown:
    DDETerminate channel
    msgbox "Now we'll create a group, and then delete it."
    channel = DDEInitiate(winshell,topic)
    DDEExecute channel,"[CreateGroup(TestGrp)]"
    DDEExecute channel,"[ShowGroup(TestGrp,1)]"
    msgbox "Look at your shell now and see the new group before we delete it."
    DDEExecute channel,"[DeleteGroup(TestGrp)]"
    DDETerminateAll
    end
'end sub
```

Dialog Examples



OK, Cancel, and Push Buttons



A Comprehensive Dialog Example



Dialog Statement and Function Example

'OK, Cancel, and Push Buttons



Copy Script



Close Window

```
'sub main()
  'Example of a simple Dialog box
  Begin Dialog UserDialog 16,32,233,105, "Title"
    OKButton 176,7,43,14
    CancelButton 176,30,44,14
    Text 6,34,158,8, "This is a sample text field."
    PushButton 176,52,44,14, "Other #1"
    PushButton 177,76,43,14, "Other #2"
  End Dialog

  dim adialog as UserDialog

  returncode = Dialog(adialog)
  '  returncode contains the value of the button selected
  '  OK = -1
  '  Cancel = 0
  '  Other buttons are numbered 1 to N where N is the number
  '  of other buttons on the dialog
  select case returncode
    case -1
      msgbox "OK pressed."
    case 0
      msgbox "Cancel pressed."
    case else
      msgbox "Other button #"+str$(returncode)+" pressed."
  end select
'end sub
```

'A Comprehensive Dialog Example



```
'sub main()
'Example showing the entire process of defining, displaying,
'and interpreting the input from a dialog box.

'Initialization
'-----
'ListBox$ and ComboBox are single-dimensional arrays to hold the
'contents of the list and combo box in the dialog box.
Dim ListBox1$() as string
Dim ComboBox1$() as string

Dim stf$ as string          ' static text variable
stf$ = "123456789012345678901234567890"

'In this example, the Text field uses the str$
'variable to get its field name. This can
'be done for any of the dialog control types except:
'OKButton, CancelButton, TextBox
'
'Note that the str$ variable must have a
'value assigned BEFORE you can define the dialog box.

'Define dialog box
'-----
Begin Dialog UserDialog 16,32,304,168, "Sample Dialog Box"
    OKButton 251,9,44,14
    CancelButton 252,30,44,14
    PushButton 252,51,44,14, "Pushbutton1"
    PushButton 252,73,44,14, "PushButton2"
    GroupBox 13,9,84,59, "Sample Group Box"
    OptionGroup .OptionGroup1
        OptionButton 21,24,65,14, "Option 1"
        OptionButton 21,44,66,14, "Option 2"
    CheckBox 15,78,79,14, "Sample Checkbox", .CheckBox1
    Text 14,105,79,8, stf$
    TextBox 16,120,81,12, .TextBox1
    ListBox 114,14,120,48, ListBox1$, .ListBox1
    ComboBox 113,68,120,84, ComboBox1$, .ComboBox1
End Dialog

'Prepare to display dialog box.
'-----
'Declare the dialog type using Dim ... as UserDialog command
Dim aSampleDialog as UserDialog

'Load the list box and combo box arrays with app window names
AppList ComboBox1$
AppList ListBox1$

'Load the text box with an initial value
aSampleDialog.TextBox1 = "123456789012345678901234567890"
'Note that you reference a dialog box field as follows:
'    <DialogBoxName>.<FieldName>
```

```

'Display the Dialog
'-----
a% = Dialog(aSampleDialog)      'Returns integer indicating button chosen

'Interpret user input
'-----
'This example builds a string describing the contents of the
'dialog box when the user finished making changes.
crlf$ = chr$(13)+chr$(10)
dlgstr$ = "Button pushed = "
'Determine button pushed.
select case a%
    case -1
        dlgstr$ = dlgstr$ + "OK"
    case 0
        dlgstr$ = dlgstr$ + "Cancel"
    case 1
        dlgstr$ = dlgstr$ + "PushButton1"
    case 2
        dlgstr$ = dlgstr$ + "PushButton2"
end select
dlgstr$ = dlgstr$ + crlf$
'Determine new value of each dialog box component
dlgstr$ = dlgstr$ + "Option = " + str$(aSampleDialog.OptionGroup1) + crlf$
dlgstr$ = dlgstr$ + "Checkbox = " + str$(aSampleDialog.CheckBox1) + crlf$
dlgstr$ = dlgstr$ + "TextBox = " + aSampleDialog.TextBox1 + crlf$
dlgstr$ = dlgstr$ + "ListBox = " + ListBox1$(aSampleDialog.ListBox1) + crlf$
dlgstr$ = dlgstr$ + "ComboBox = " + aSampleDialog.ComboBox1
'Display the dlgstr$ string in a message box.
msgbox dlgstr$
'end sub

```

'Exit Statement Examples



```
function testfunc (a as integer) as integer
    testfunc = 0
    if a = 2 then
        Exit Function
    end if
    msgbox "no premature exit from testfunc"
    testfunc = 100
end function

sub testsub (a as integer)
    if a = 4 then
        Exit Sub
    end if
    msgbox "no premature exit from testsub"
end sub

sub main()
    'examples of EXIT statements
    for i% = 1 to 10
        if i% >= 1 and i% <= 2 then
            b% = testfunc(i%)
            msgbox str$(b%)
        end if
        if i% >= 3 and i% <= 4 then
            testsub(i%)
        end if
        if i% = 7 then
            Exit For
        else
            msgbox "no exit yet "+str$(i%)
        end if
    next i%
    msgbox "we just exited from the for loop"
end sub
```


'Input/Output Example



```
'sub main()
'Example of file input/output functions
'Open, Close, Eof, FileAttr, Line Input#, Lof
'Because of the use of ViewPort... commands,
'this script should not be run under DOS.

dim fileno as integer
dim flen as integer
dim fileatr as integer
dim osfile as integer
dim inline as string
dim linedesc as string
dim linemult as integer

ViewPortOpen "AUTOEXEC.BAT"
ViewPortClear
fileno = FreeFile()           'get next available file number
Open "C:\AUTOEXEC.BAT" for input as fileno
flen = Lof(fileno)           'get length of file
Print "File Autoexec.Bat - Length"+str$(flen)
fileatr = FileAttr(fileno,1)
osfile = FileAttr(fileno,2)
Print "File is opened for ";
Select Case fileatr
    Case 1
        Print "Input";
    Case 2
        Print "Output";
    Case 8
        Print "Append";
End Select
Print " and has an Operating System Filehandle of"+str$(osfile)
Print string$(50,"-")
while not eof(fileno)
    Line Input #fileno, inline
    Print inline
wend
Close fileno
msgbox "Click OK when you're finished viewing the file."

ViewPortClear
Print "Creating a simple output file."
fileno = FreeFile()
Open "C:\JUNK.TXT" for output as fileno
for i% = 1 to 10
    Write #fileno,"Line"+str$(i%),i% * 10
next i%
Close fileno
Print "Reading created file."
Print string$(50,"-")
fileno = FreeFile()
Open "C:\JUNK.TXT" for input as fileno
while not eof(fileno)
```

```

        print "Seek Position"+str$(Seek(fileno))
        print "File Position"+str$(loc(fileno)) 'print the file position
        Input #fileno,linedesc,linemult
        Print linedesc+", Value ="&str$(linemult)
wend
Close fileno
msgbox "Click OK when you're finished viewing the file."

ViewPortClear
Print "Display Autoexec.Bat again...this time in a different way."
fileno = FreeFile()
Open "C:\AUTOEXEC.BAT" for input as fileno
flen = lof(fileno)
aexec$ = Input$(flen,fileno) 'read the entire file at once
print aexec$
Close fileno
msgbox "Click OK when you're finished viewing the file."

ViewPortClear
Print "Now create Junk.Txt using Print# instead of Write#"
fileno = FreeFile()
Open "C:\JUNK.TXT" for output as fileno
for i% = 1 to 10
    Print #fileno,i%,"test","more"
next i%
Close fileno
fileno = FreeFile()
Open "C:\JUNK.TXT" for input as fileno
stuff$ = Input$(lof(fileno),fileno)
Close fileno
print stuff$
msgbox "Click OK when you're finished viewing the file."
ViewPortClose
'end sub

```

'Message Example



```
'sub main()
  'Example of MsgOpen, MsgSetText, MsgSetThermometer, and MsgClose

  MsgOpen "Message Box - 0% complete.",0,TRUE,TRUE
  on error goto cancelpressed
  for i% = 1 to 100
    sleep 100
    msg$ = "Message Box -"+str$(i%)+"% complete."
    MsgSetText msg$
    MsgSetThermometer i%
  next i%
  MsgClose
  msgbox "Finished Normally"
end

cancelpressed:
  MsgClose
  on error goto 0
  msgbox "Cancel was selected."
end

'end sub
```

'Queue Example



```
'sub main()  
    'Example of QUE commands  
    dim npWnd as integer  
    dim npname as string  
  
    npname = AppFind$("Notepad")  
    AppActivate npname  
    npWnd = WinFind(npname)  
    QueEmpty      'empty the queue  
    QueKeyDn "D"  
    QueKeyUp "a"  
    QueKeys "vid"  
    QueMouseClicked VK_RBUTTON,1,1  
    QueMouseDblClk VK_RBUTTON,1,1  
    QueMouseDblDn VK_RBUTTON,1,1  
    QueMouseDown VK_RBUTTON,1,1  
  
    QueMouseMove 100,100  
    QueMouseUp VK_RBUTTON,100,100  
    QueSetRelativeWindow npWnd  
    QueFlush TRUE  
    QueEmpty  
'end sub
```


'Abs Function Example



```
'sub main()  
    'ABS() - absolute value  
    dim a as single  
    dim b as integer  
  
    a = 2.34  
    b = abs(a)  
    msgbox str$(b)  
'end sub
```

'AddIni Function Example



```
'sub main()  
  'Example of AddIni  
  
  'add the contents of myini.ini to win.ini  
  a% = AddIni("myini.ini","win.ini")  
  if a% then  
    msgbox "INI settings added to Win.INI"  
  else  
    msgbox "AddIni failed."  
  end if  
end sub
```

'And Operator Exanmple



```
'sub main()  
    'AND statement  
    dim a as integer  
    dim b as integer  
  
    a = 5  
    b = 9  
    if (a < 6) AND (b > 8) then  
        msgbox "Both conditions were true."  
    else  
        msgbox "Both conditions were not true."  
    end if  
  
    if (a < 6) AND (b > 9) then  
        msgbox "Both conditions were true."  
    else  
        msgbox "Both conditions were not true."  
    end if  
'end sub
```


'AnswerBox Function Example



```
'sub main()  
  'AnswerBox example  
  
  'This is a default answer box without  
  'specifying the buttons  
  r% = AnswerBox("Example prompt?")  
  msgbox str$(r%)    'display the result  
  r% = AnswerBox("Example prompt 2?","&Maybe","&Ok","Maybe &Not")  
  msgbox str$(r%)    'display the result  
'end sub
```

'AppActivate Statement Example



```
'sub main()  
    'AppActivate example  
  
    'activate Applications Manager  
    AppActivate "Applications Manager"  
'end sub
```

'AppClose Statement Example



```
'sub main()  
  'AppClose example  
  
  'search for a copy of Notepad that is running  
  appname$ = AppFind$("Notepad")  
  
  'close the copy of Notepad that was found  
  AppClose appname$  
'end sub
```

'AppFileName\$ Function Example



```
'sub main()  
    'Example of AppFileName$  
  
    'find a copy of Notepad that is running  
    appname$ = AppFind$("Notepad")  
  
    'get the name of the program it was executed from  
    appfile$ = AppFileName$(appname$)  
  
    'display the file name  
    MsgBox appfile$  
'end sub
```

'AppFind Function Example



```
'sub main()  
    'Example of AppFind$  
  
    appname$ = AppFind$("Notepad")  
    MsgBox appname$  
'end sub
```

'AppGetActive\$ Function Example



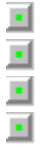
```
'sub main()  
    'Example of AppGetActive$  
  
    'activate a copy of Notepad  
    appname$ = AppFind$("Notepad")  
    AppActivate appname$  
  
    'now see if it is active  
    appname$ = AppGetActive$()  
    MsgBox appname$  
'end sub
```

'AppGetPosition Statement Example



```
'sub main()  
    'Example of AppGetPosition  
    dim x, y, w, h as integer  
  
    'Get the position, width and height of Notepad  
    appname$ = AppFind$("Notepad")  
    AppGetPosition x, y, w, h, appname$  
  
    'display them  
    MsgBox appname$+ " is at "+str$(x)+ ", "+str$(y)+ " with width of "+str$(w)+ " and height of  
"+str$(h)  
'end sub
```

'AppGetState Function Example



```
'sub main()  
    'Example of AppGetState  
  
    appname$ = AppFind$("Notepad")  
    appstate% = AppGetState(appname$)  
    if appstate% = WS_MAXIMIZED then  
        MsgBox appname$ + " is Maximized - " +str$(appstate%)  
    else  
        if appstate% = WS_MINIMIZED then  
            MsgBox appname$ + " is Minimized - " + str$(appstate%)  
        else  
            if appstate% = WS_RESTORED then  
                MsgBox appname$ + " is Restored - " +str$(appstate%)  
            else  
                MsgBox appname$ + " is in an unknown state - " + str$(appstate%)  
            end if  
        end if  
    end if  
end if  
end sub
```


'AppHide Statement Example



```
'sub main()  
    'Example of AppHide and AppShow  
  
    appname$ = AppFind$("Notepad")  
    AppHide appname$  
    MsgBox appname$+ " is now hidden."  
    AppShow appname$  
    MsgBox appname$+ " is no longer hidden."  
'end sub
```

'AppList Statement Example



```
'sub main()  
    'example of AppList  
    dim appnames(1) as string  
  
    AppList appnames  
    for i% = lbound(appnames) to ubound(appnames)  
        MsgBox appnames(i%)  
    next i%  
'end sub
```

'AppMaximize Statement Example



```
'sub main()  
    'example of AppMaximize, AppRestore, and AppMinimize  
  
    appname$ = AppFind$("Notepad")  
    AppMaximize appname$  
    msgbox appname$+ " is maximized."  
    AppRestore appname$  
    msgbox appname$+ " is restored."  
    AppMinimize appname$  
    msgbox appname$+ " is minimized."  
'end sub
```

'AppMinimize Statement Example



```
'sub main()  
    'example of AppMaximize, AppRestore, and AppMinimize  
  
    appname$ = AppFind$("Notepad")  
    AppMaximize appname$  
    msgbox appname$+ " is maximized."  
    AppRestore appname$  
    msgbox appname$+ " is restored."  
    AppMinimize appname$  
    msgbox appname$+ " is minimized."  
'end sub
```

'AppMove Statement Example



```
'sub main()  
    'example of AppMove  
    dim ax, ay as integer  
  
    '    get current position of Notepad to save  
    appname$ = AppFind$("Notepad")  
    AppGetPosition ax, ay, 0, 0, appname$  
  
    '    move Notepad around a little  
    for i% = 0 to 200  
        AppMove i%, i%, appname$  
    next i%  
  
    AppMove ax, ay, appname$  
'end sub
```

'AppRestore Statement Example



```
'sub main()  
    'example of AppMaximize, AppRestore, and AppMinimize  
  
    appname$ = AppFind$("Notepad")  
    AppMaximize appname$  
    msgbox appname$+ " is maximized."  
    AppRestore appname$  
    msgbox appname$+ " is restored."  
    AppMinimize appname$  
    msgbox appname$+ " is minimized."  
'end sub
```

'AppSetState Statement Example



```
'sub main()  
    'example of AppSetState  
  
    appname$ = AppFind$("Notepad")  
    AppSetState WS_MAXIMIZED, appname$  
    msgbox "Notepad is maximized"  
    AppSetState WS_RESTORED, appname$  
    msgbox "Notepad is restored"  
    AppSetState WS_MINIMIZED, appname$  
    msgbox "Notepad is minimized"  
'end sub
```

'AppShow Statement Example



```
'sub main()  
    'example of AppHide and AppShow  
  
    appname$ = AppFind$("Notepad")  
    AppHide appname$  
    MsgBox appname$+ " is now hidden."  
    AppShow appname$  
    MsgBox appname$+ " is no longer hidden."  
'end sub
```


'AppSize Statement Example



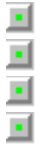
```
'sub main()  
    'example of AppSize  
    dim ax, ay, aw, ah as integer  
  
    appname$ = AppFind$("Notepad")  
    AppSetState WS_RESTORED, appname$  
    AppGetPosition ax, ay, aw, ah, appname$  
    for i% = 10 to 200  
        AppSize i%, i%, appname$  
    next i%  
    AppSize aw, ah, appname$  
    AppMove ax, ay, appname$  
    AppSetState WS_MINIMIZED, appname$  
'end sub
```

'AppType Function Example



```
'sub main()  
    'example of AppType  
  
    msgbox "After you press OK, you have 10 seconds      to activate an app to check."  
    sleep 10000      'wait for 10 seconds  
    appname$ = AppGetActive$()  
    at% = AppType(appname$)  
    select case at%  
        case TYPE_DOS  
            msgbox "DOS Application"  
        case TYPE_WINDOWS  
            msgbox "WINDOWS Application"  
    end select  
'end sub
```

'ArrayDims Function Example



```
'sub main()  
    'example of ArrayDims  
    dim apps(1) as string  
    dim oapps(1,1) as string  
  
    'ArrayDims shows how many dimensions an array  
    '    was declared with  
    msgbox "apps(1) has "+str$(arraydims(apps))+ " dimension(s)."  
    msgbox "oapps(1,1) has "+str$(arraydims(oapps))+ "dimension(s)."  
'end sub
```

'ArraySort Statement Example



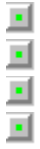
```
'sub main()  
    'example of ArraySort  
    dim apps(1) as string  
  
    'get list of running applications  
    AppList apps  
    'sort it  
    ArraySort apps  
    'display them  
    for i% = lbound(apps) to ubound(apps)  
        msgbox apps(i%)  
    next i%  
'end sub
```

'Asc Function Example



```
'sub main()  
    'example of ASC()  
    dim acode as integer  
    dim astr as string  
  
    astr = "This is a string."  
    'the ASC function returns the ASCII code for  
    'the first character of the given string  
    acode = asc(astr)  
    msgbox "The first character of the string (" + astr + ") has an ASCII code of " + str$(acode)  
'end sub
```

'AskBox\$ Function Example



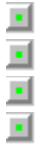
```
'sub main()  
    'example of AskBox$  
  
    userinput$ = AskBox$("Key in something below:", "This is the default value.")  
    msgbox "You entered -> "+userinput$  
'end sub
```

'AskPassword\$ Function Example



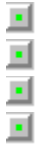
```
'sub main()  
    'example of AskPassword$  
  
    mypassword$ = AskPassword$("Type in a fake password:")  
    msgbox "The password you typed was -> "+mypassword$  
'end sub
```

'Atn Function Example



```
'sub main()  
    'example of ATN (arctangent)  
    dim mypi as double  
  
    mypi = 4 * atn(1)  
    msgbox str$(mypi)  
'end sub
```


'Beep Statement Example



```
'sub main()  
    'example of BEEP  
  
    beep  
'end sub
```

'ButtonEnabled Function Example



```
'sub main()  
    'Example of ButtonEnabled  
  
    appn$ = AppFind$("Notepad")  
    if appn$ <> "" then  
        AppActivate appn$  
        Menu "File.Page Setup"  
        if ButtonEnabled("OK") then  
            msgbox "OK button is enabled."  
        else  
            msgbox "OK button is not enabled."  
        end if  
    end if  
end sub
```

'ButtonExists Function Example



```
'sub main()  
  'Example of ButtonExists  
  
  appn$ = AppFind$("Notepad")  
  if appn$ <> "" then  
    AppActivate appn$  
    Menu "File.Page Setup"  
    if ButtonExists("Cancel") then  
      msgbox "Cancel button exists."  
    else  
      msgbox "Cancel button does not exist."  
    end if  
  end if  
end sub
```

'Call Statement Example



```
sub doubleit (a as integer)
    msgbox str$(a * 2)
end sub
```

```
sub main()
    'Example of the Call statement

    'the following statements do the same thing
    Call doubleit (3)
    doubleit(3)
end sub
```

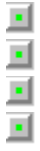
'CDBl Function Example



```
'sub main()
    'example of CDBL
    dim adouble as double
    dim asingle as single
    dim ainteger as integer
    dim along as long

    asingle = pi
    adouble = pi
    ainteger = 30000
    along = 88000
    adouble = cdbl(ainteger)
    msgbox str$(adouble)
    adouble = cdbl(along)
    msgbox str$(adouble)
    msgbox str$(asingle)
    adouble = pi
    msgbox str$(adouble)
    'Now convert the single to double. You'll see a
    'change in the value due to conversion of a
    'smaller precision to double precision.
    adouble = cdbl(asingle)
    msgbox str$(adouble)
'end sub
```

'ChDir Statement Example



```
'sub main()  
    'example of ChDir  
  
    msgbox CurDir$("C")  
    ChDir "C:\  
    msgbox CurDir$("C")  
'end sub
```

'ChDrive Statement Example



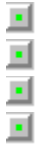
```
'sub main()  
    'example of ChDrive  
  
    msgbox CurDir$()  
    chdrive "D"  
    msgbox CurDir$()  
'end sub
```

'Chr\$ Function Example



```
'sub main()  
    'example of Chr$()  
  
    'List the letters of the alphabet  
    startcode = asc("A")  
    alphstr$ = ""  
    msgbox "The alphabet starts at code "+str$(startcode)  
    for i% = startcode to startcode +25  
        alphstr$ = alphstr$ +chr$(i%)  
    next i%  
    msgbox alphstr$  
'end sub
```


'CInt Function Example



```
'sub main()  
    'Example of CINT  
    dim a as integer  
  
    a = cint(pi)    'convert PI to an integer  
    msgbox str$(a)  
'end sub
```

'Clipboard\$ Statement and Function Example



```
'sub main()  
    'Example of Clipboard$(), Clipboard$, and  
    'ClipboardClear  
    'WARNING: after you run this, your clipboard  
    'contents will be gone  
  
    msgbox Clipboard$()  
    ClipboardClear  
    msgbox Clipboard$()  
    Clipboard$ "I was here!"  
    msgbox Clipboard$()  
    ClipboardClear  
end sub
```

'ClipboardClear Statement Example



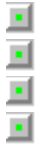
```
'sub main()  
    'Example of Clipboard$(), Clipboard$, and  
    'ClipboardClear  
    'WARNING: after you run this, your clipboard  
    'contents will be gone  
  
    msgbox Clipboard$()  
    ClipboardClear  
    msgbox Clipboard$()  
    Clipboard$ "I was here!"  
    msgbox Clipboard$()  
    ClipboardClear  
'end sub
```

'CLng Function Example



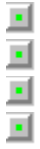
```
'sub main()  
    'Example of CLNG  
    dim a as long  
  
    a = clng(pi)    'convert PI to a long integer  
    msgbox str$(a)  
'end sub
```

'CSng Function Example



```
'sub main()  
    'Example of CSNG()  
  
    dim adouble as double  
    dim asingle as single  
  
    adouble =pi  
    msgbox str$(adouble)  
    asingle = csng(adouble)  
    msgbox str$(asingle)  
'end sub
```

'CStr Function Example



```
'sub main()  
    'Example of CSTR  
  
    dim adouble as double  
    dim astring as string  
  
    adouble = pi  
    astring = cstr(adouble)  
    msgbox astring  
'end sub
```

'CurDir\$ Function Example



```
'sub main()  
    'Example of CurDir$()  
    dim dstr as string  
  
    dstr = CurDir$()  
    msgbox dstr  
    dstr = CurDir$("F")    'for a particular drive  
    msgbox dstr  
'end sub
```

'Date\$ Statement Example



```
'sub main()  
    'Example of the Date$ command (not function)  
    dim currentdate as string  
    dim newdate as string  
  
    currentdate = Date$  
    newdate = "1-1-1980"      'This is the earliest  
                              'date that can be  
                              'assigned  
  
    Date$ = newdate  
    msgbox Date$  
    Date$ = currentdate  
    msgbox Date$  
'end sub
```


'Date\$ Function Example



```
'sub main()  
    'Example of Date$()  
    dim dstr as string  
  
    dstr = Date$  
    msgbox dstr  
    dstr = Date$()  
    msgbox dstr  
'end sub
```

'DateSerial Function Example



```
'sub main()  
    'example of DateSerial  
    dim dserial as double  
  
    dserial = DateSerial(1899,12,30) 'Earliest date should be 0  
    msgbox str$(dserial)  
    dserial = DateSerial(1999,12,30) '100 years later (note leap years included)  
    msgbox str$(dserial)  
'end sub
```

'DateValue Function Example



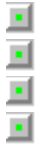
```
'sub main()  
    'Example of DateValue  
    dim dval as double  
  
    dval = DateValue(Date$) 'value of current date  
    msgbox str$(dval)  
'end sub
```

'Day Function Example



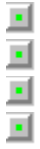
```
'sub main()  
    'Example of Day() function (day of month)  
    'will show the current date's day of month.  
    msgbox str$(Day(DateValue(Date$)))  
'end sub
```

'DCLHomeDir\$ Function Example



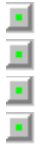
```
'sub main()  
    'DCLHomeDir$ example  
  
    a$ = DCLHomeDir$  
    msgbox a$  
'end sub
```

'DCLOS\$ Function Example



```
'sub main()  
  'Example of DCLOS() function  
  
  a% = DCLOS()  
  select case a%  
    case 0  
      msgbox "Windows"  
    case 1  
      msgbox "DOS"  
  end select  
'end sub
```

'DCLVersion\$ Function Example



```
'sub main()  
  'Example of DCLVersion$()  
  
  a$ = DCLVersion$()  
  msgbox a$  
'end sub
```

'Declare Statement Example



```
declare sub MessageBeep lib "user" (byval n as integer)
declare function GetDebugState lib "user" alias "GetSystemDebugState" () as long

'sub main()
'Example of Declare statement
'All declared external functions and procedures
'exist outside of any DCL function or
'subroutines.
dim debugstate as long

MessageBeep(-1)      'standard system beep
debugstate = GetDebugState()
msgbox str$(debugstate)
'end sub
```


'DEFtype Statement Example



```
DEFInt i - k, x - z
DEFStr s - v
DEFDbl a - c
```

```
'sub main()
  'example of DEFtype statement
  'This statement is used to define certain
  'letters of the alphabetic character set to
  'be used as a certain type of variable.

  i = 1
  msgbox str$(i)
  j = 2.5
  msgbox str$(j)
  s = "test"
  msgbox s
  a = 3
  msgbox str$(a)
  b = 5.6
  msgbox str$(b)
'end sub
```

'DesktopCascade Statement Example



```
'sub main()  
    'Example of DesktopCascade  
  
    DesktopCascade  
'end sub
```

'DesktopSetColors Statement Example



```
'sub main()  
    'example of DesktopSetColors  
  
    msgbox "Click to change to WingTips"  
    DesktopSetColors "WingTips"  
    msgbox "Click to change to Arizona"  
    DesktopSetColors "Arizona"  
'end sub
```

'DesktopSetWallpaper Statement Example



```
'sub main()  
    'Example of DesktopSetWallpaper  
  
    msgbox "Click to set wallpaper to CASTLE tiled."  
    DesktopSetWallpaper "castle.bmp",true  
    msgbox "Click to set wallpaper to CASTLE centered."  
    DesktopSetWallpaper "castle.bmp",false  
    msgbox "Click to clear wallpaper."  
    DesktopSetWallpaper "",true  
'end sub
```

'DesktopTile Statement Example



```
'sub main()  
    'Example of DesktopTile  
  
    DesktopTile  
'end sub
```

'Dialog Statement and Function Example



```
'sub main()
  'Example of using the DIALOG function and command

  'Here is our dialog box specification.
  Begin Dialog UserDialog 16,32,148,72, "Sample"
    OKButton 54,11,41,14
    CancelButton 55,40,41,14
  End Dialog

  'Create a dialog box type variable
  dim mydialog as UserDialog

  'First the function method
  retval% = Dialog(mydialog)
  select case retval%
    case -1
      msgbox "OK pressed."
    case 0
      msgbox "Cancel pressed."
  end select

  'Now the command method
  'First create an error handler in case "Cancel"
  'is pressed.
  on error goto cancelpressed
  Dialog mydialog
  msgbox "OK pressed."
  goto endprog
cancelpressed:
  msgbox "Cancel pressed."
endprog:
  on error goto 0
'end sub
```

'Dim Statement Example



```
'sub main()
    'Examples of the DIM statement

    'These two statements declare integers
    dim a as integer
    dim b%

    a = 1
    b% = 1

    'These two statements declare strings
    dim s as string
    dim t$

    s = "test"
    t$ = "test"

    'Array declarations
    'When an array is declared, its elements are
    'numbered 0 through N-1 where N is the size of
    'the array. Using the OPTION BASE statement,
    'you can begin numbering at 1.

    dim j(5) as integer      'array of 5 integer values
    for x% = 0 to 4
        j(x%) = x%
        msgbox str$(j(x%))
    next x%

    dim k(5) as string       'array of 5 strings
    for x% = 0 to 4
        k(x%) = chr$(65+x%)
        msgbox k(x%)
    next x%

    dim l(3,3) as string     'two-dimensional array
                                'containing a total of
                                '9 strings, 3 by 3

    for x% = 0 to 2
        for y% = 0 to 2
            l(x%,y%) = str$(x%)+str$(y%)
            msgbox l(x%,y%)
        next y%
    next x%

    'Using explicit array subscripts
    dim p(6 to 10) as integer
    for x% = 6 to 10
        p(x%) = x%
        msgbox str$(p(x%))
    next x%
'end sub
```

'DirExists Function Example



```
'sub main()  
  'Example of DirExists  
  
  a$ = "C:\DOS"  
  if DirExists(a$) then  
    msgbox a$+ " exists"  
  else  
    msgbox a$+ " does not exist"  
  end if  
end sub
```


'Dir\$ Function Example



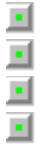
```
'sub main()
  'Example of Dir$() function

  a$ = dir$("c:\*.bat")  'initialize file search
                        '(filespec parameter required)

  do
    msgbox a$           'display file name
    a$ = dir$()         'next file name
                        '(no parameter specified)

  loop while a$ <> ""
'end sub
```

'DiskDrives Statement Example



```
'sub main()  
    'Example of DiskDrives command  
    dim ddrives() as string  
  
    DiskDrives ddrives  
    for i% = lbound(ddrives) to ubound(ddrives)  
        alldrives$ = alldrives$ +ddrives(i%)  
    next i%  
    msgbox alldrives$  
'end sub
```

'DiskFree Function Example



```
'sub main()  
    'Example of DiskFree function  
    dim ddrives() as string  
    dim freespace as long  
  
    DiskDrives ddrives  
    for i% = lbound(ddrives) to ubound(ddrives)  
        if ddrives(i%) <> "A" and ddrives(i%) <> "B" then  
            freespace = DiskFree(ddrives(i%))  
            msgbox "Free space on drive "+ddrives(i%)+ ": is "+str$(freespace)  
        end if  
    next i%  
'end sub
```

'Do...Loop Statement Example



```
'sub main()
    'Examples of DO-LOOP statements
    'All of the following loops perform the same task.
    'The difference is primarily in WHEN the loop exit
    'condition is checked.

    a% = 1
    do while a% = 1
        a% = msgbox("DO-WHILE-LOOP Click cancel to go to next loop.",1)
    loop

    a% = 1
    do until a% <> 1
        a% = msgbox("DO-UNTIL-LOOP Click cancel to go to next loop.",1)
    loop

    a%=1
    do
        a% = msgbox("DO-LOOP-WHILE Click cancel to go to next loop.",1)
    loop while a% = 1

    a% = 1
    do
        a% = msgbox("DO-LOOP-UNTIL Click cancel to go to next loop.",1)
    loop until a% <> 1

    a% = 1
    do
        a% = msgbox("DO-LOOP Click cancel to go to next loop.",1)
        if a% <> 1 then
            exit do
        end if
    loop
'end sub
```

'DoEvents Statement Example



```
'sub main()  
    'Examples of DoEvents and DoEvents()  
  
    DoEvents  
    x% = DoEvents()  
'end sub
```

'DoKeys Statement Example



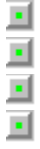
```
'sub main()  
    'Example of DoKeys  
    dim alttab as string  
  
    alttab = "%{TAB}"  
    msgbox "Press OK to do first Alt-Tab"  
    DoKeys alttab  
    msgbox "Press OK to Alt-Tab back to original application"  
    DoKeys alttab  
'end sub
```

'EnableStopScript Statement Example



```
'sub main()  
  'Example of EnableStopScript  
  
  EnableStopScript ESS_ENABLE  
    'control-break enabled  
  EnableStopScript ESS_DISABLE  
    'control-break disabled  
  EnableStopScript ESS_INTERACTIVE  
    'control-break only in script editor  
'end sub
```

'End Statement Example



```
'sub main()  
    'Example of END  
  
    msgbox "You'll see this one."  
    end  
    msgbox "You won't see this one."  
'end sub
```


'Environ\$ Function Example



```
'sub main()  
  'Example of environ$() function  
  
  a$ = environ$("path")  
  msgbox a$  
'end sub
```

'Err Statement and Function Example



```
'sub main()  
    'Example of Err(), Error, and Error$()  
    dim i as integer  
  
    i=1  
nexterror:  
    on error goto errortrap  
    select case i  
        case 1  
            error 100  
        case 2  
            error 101  
        case 3  
            error 102  
    end select  
end  
  
errortrap:  
    msgbox "Error #"+str$(err())+ "; "+Error$(err())  
    i = i +1  
    goto nexterror  
'end sub
```

'Error Statement and Function Example



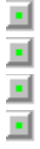
```
'sub main()  
    'Example of Err(), Error, and Error$()  
    dim i as integer  
  
    i=1  
nexterror:  
    on error goto errortrap  
    select case i  
        case 1  
            error 100  
        case 2  
            error 101  
        case 3  
            error 102  
    end select  
end  
  
errortrap:  
    msgbox "Error #"+str$(err())+ "; "+Error$(err())  
    i = i +1  
    goto nexterror  
'end sub
```

'Exclusive Statement Example



```
'sub main()  
  'Examples of the Exclusive statement  
  
  Exclusive TRUE      'no multitasking for now  
  Exclusive FALSE     'allow other programs to run  
                      'again  
  
'end sub
```

'Exp Function Example



```
'sub main()  
    'Example of EXP function  
  
    dim result as double  
  
    result = exp(1)  
    msgbox str$(result)  
'end sub
```

'FileCopy Function Example



```
'sub main()  
  'Example of FileCopy  
  
  a% = FileCopy("c:\*.bat","c:\batch")  
  if a% then  
    msgbox "File copy successful."  
  else  
    msgbox "File copy failed."  
  end if  
end sub
```

'FileDateTime Function Example



```
'sub main()  
    'Example of FileDateTime  
    dim fdt as double  
    dim sfdt as string  
  
    fdt = FileDateTime("C:\AUTOEXEC.BAT")  
    sfdt = str$(month(fdt)) + "/" + str$(day(fdt)) + "/" + str$(year(fdt)) + " " + str$(hour(fdt)) +  
    ":" + str$(minute(fdt)) + " " + str$(second(fdt))  
    msgbox sfdt  
'end sub
```

'FileDirs Statement Example



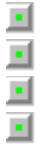
```
'sub main()  
    'Example of FileDirs statement  
    dim fdirs() as string  
  
    'list all directories on drive C:  
    FileDirs fdirs,"C:\*.*"   
  
    for i% = lbound(fdirs) to ubound(fdirs)  
        fdirlist$ = fdirlist +fdirs(i%) + "; "  
    next i%  
    msgbox fdirlist$  
'end sub
```


'FileExists Function Example



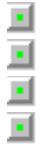
```
'sub main()  
    'Example of FileExists() function  
  
    if FileExists("C:\AUTOEXEC.BAT") then  
        msgbox "Autoexec Exists."  
    else  
        msgbox "Autoexec Does Not Exists."  
    end if  
    if FileExists("D:\NADA.FIL") then  
        msgbox "NADA.FIL Exists."  
    else  
        msgbox "NADA.FIL Does Not Exist."  
    end if  
'end sub
```

'FileLen Function Example



```
'sub main()  
    'Example of the FileLen function  
    dim flen as long  
  
    flen = FileLen("C:\AUTOEXEC.BAT")  
    msgbox "Autoexec.Bat is"+str$(flen)+ " bytes long."  
'end sub
```

'FileList Statement Example



```
'sub main()  
    'Example of FileList statement  
    dim ffiles() as string  
    dim atrib as integer  
  
    atrib = ATTR_NORMAL  
    filelist ffiles,"C:\*.*",atrib  
    for i% = lbound(ffiles) to ubound(ffiles)  
        flist$ = flist$ + ffiles(i%) + "; "  
    next i%  
    msgbox flist$  
'end sub
```

'FileParse Statement Example



```
'sub main()  
    'Examples of FileParse$()  
    dim filespec as string  
  
    filespec = "C:\DOS\COMMAND.COM"  
  
    'full file specification  
    a$ = FileParse$(filespec,0)  
    msgbox "Full Filespec = " + a$  
  
    'drive  
    a$ = FileParse$(filespec,1)  
    msgbox "Drive = " + a$  
  
    'path  
    a$ = FileParse$(filespec,2)  
    msgbox "Path = " + a$  
  
    'name  
    a$ = FileParse$(filespec,3)  
    msgbox "Filename = " + a$  
  
    'root  
    a$ = FileParse$(filespec,4)  
    msgbox "File Rootname = " + a$  
  
    'extension  
    a$ = FileParse$(filespec,5)  
    msgbox "File Extension = " + a$  
'end sub
```

'FileType Function Example



```
sub DisplayFileType (ftype as integer)
    select case ftype
        case TYPE_DOS
            msgbox "DOS"
        case TYPE_WINDOWS
            msgbox "WINDOWS"
        case else
            msgbox "Unknown Filetype"
    end select
end sub

sub main()
    'example of FileType function

    ftype% = FileType("C:\DOS\COMMAND.COM")
    DisplayFileType(ftype)
    ftype% = FileType("C:\DOS\DOSSHELL.EXE")
    DisplayFileType(ftype)
    ftype% = FileType("D:\WINDOWS\notepad.exe")
    DisplayFileType(ftype)
end sub
```

'FindFile\$ Function Example



```
'sub main()  
  'Example of FindFile$  
  
  fp$ = FindFile$("notepad.exe")  
  if fp$ <> "" then  
    msgbox "File found as "+fp$  
  else  
    msgbox "File not found."  
  end if  
end sub
```

'Fix Function Example



```
'sub main()  
    'Example of the Fix() function  
    dim adouble as double  
    dim aint as integer  
  
    adouble = pi  
    msgbox str$(adouble)  
    aint = fix(adouble)  
    msgbox str$(aint)  
'end sub
```

'For...Next Statement Example



```
'sub main()
    'Examples of FOR-NEXT loops
    dim x as integer      'used as iteration variable
                          'for FOR-NEXT

    dim xstart as integer
    dim xend as integer

    'simple form
    for x = 1 to 5
        msgbox str$(x)
    next x

    'step form
    for x = 1 to 10 step 2
        msgbox str$(x)
    next x

    'backward form
    for x = 10 to 1 step -2
        msgbox str$(x)
    next x

    'change of indexes
    for x = 69 to 74
        msgbox chr$(x)
    next x

    'variable range
    xstart = 69
    xend = 74
    for x = xstart to xend
        msgbox "VAR - "+chr$(x)
    next x

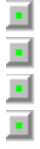
    'premature exit
    for x = 1 to 10
        if x = 6 then
            exit for
        end if
        msgbox "Exit - "+str$(x)
    next x
'end sub
```


'FreeFile Function Example



```
'sub main()  
    'Example of FreeFile%()  
    dim nextfile as integer  
  
    nextfile = FreeFile()  
    msgbox str$(nextfile)  
'end sub
```

'GetAttr Function Example



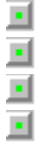
```
'sub main()  
    'Example of GetAttr  
  
    i% = GetAttr("C:\IO.SYS")  
    msgbox str$(i%)  
'end sub
```

'GetEnv Function Example



```
'sub main()  
  'Example of SetEnv and GetEnv$()  
  tmp$ = GetEnv$("TMP",ENV_WINDOWS)  
  tv$ = AskBox$("New Value For TMP Environment Variable:")  
  a% = SetEnv("TMP",tv$,ENV_WINDOWS)  
  msgbox "New value for TMP is "+ GetEnv$("TMP",ENV_WINDOWS)  
  'restore old value  
  a% = SetEnv("TMP",tmp$,ENV_WINDOWS)  
'end sub
```

'GoSub Statement Example



```
'sub main()  
    'Examples of GOSUB and RETURN  
  
    for x% = 1 to 5  
        gosub mylabel  
    next x%  
end  
  
mylabel:  
    msgbox "Here we are!"  
    return  
'end sub
```

'Goto Statement Example



```
'sub main()  
    'Example of GOTO  
  
    for x% = 1 to 5  
        if x% = 3 then goto enditall  
    next x%  
    msgbox "Error"  
    end  
  
enditall:  
    msgbox "Ended properly"  
    end  
'end sub
```

'Hex\$ Function Example



```
'sub main()  
    'Example of Hex$() function  
  
    i% = 31  
    h$ = hex$(i%)  
    msgbox h$  
'end sub
```

'HLine Statement Example



```
'sub main()  
  'Examples of HLINE  
  
  ViewPortOpen  
  ViewPortClear  
  Print "Here is some test data."  
  HLine 50  
  sleep 2000  
  HLine -50  
  ViewPortClose  
'end sub
```

'Hour Function Example



```
'sub main()  
    'Example of hour() function  
    dim dt as double  
  
    dt = Now  
    msgbox str$(hour(dt)) 'current hour  
'end sub
```


'HPage Statement Example



```
'sub main()  
  'Examples of HPage  
  
  ViewPortOpen  
  ViewPortClear  
  Print "This is some test data"  
  HPage 1  
  sleep 2000  
  HPage -1  
  ViewPortClose  
'end sub
```

'HScroll Statement Example



```
'sub main()  
  'Example of HSCROLL  
  
  ViewPortOpen  
  ViewPortClear  
  Print "This is some test data for the viewport scroll test."  
  sleep 2000  
  HScroll 50 '50 percent scroll  
  sleep 2000  
  HScroll 0      'no scroll  
  sleep 2000  
  ViewPortClose  
'end sub
```

'If...Then...Else Statement Example



```
'sub main()  
  'Example IF-THEN-ELSE-END IF statement  
  dim a%  
  
  if a% = 1 then  
    'do this stuff if a is 1  
  elseif a% = 2 then  
    'otherwise if a is 2 then do this stuff  
  else  
    'if a is neither 1 nor 2 then do this stuff  
  end if  
end sub
```

'InputBox\$ Function Example



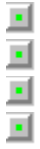
```
'sub main()  
    'Example of InputBox$() function  
  
    a$ = InputBox$("Enter your description:", "Description","",100,200)  
    msgbox a$  
'end sub
```

'InStr Function Example



```
'sub main()  
    'Example of InStr() function  
    dim teststring as string  
  
    teststring = "The quick brown fox jumps over the lazy dog."  
    'search starting at position 1  
    i% = InStr(1,teststring,"quick")  
    if i% = 0 then  
        msgbox "'quick' was not found"  
    else  
        msgbox "'quick' was found at position"+str$(i%)  
    end if  
    'search starting at position 10  
    i% = InStr(10,teststring,"quick")  
    if i% = 0 then  
        msgbox "'quick' was not found"  
    else  
        msgbox "'quick' was found at position"+str$(i%)  
    end if  
'end sub
```

'Int Function Example



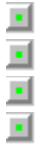
```
'sub main()  
    'example of INT function  
    dim adouble as double  
  
    adouble = pi  
    msgbox str$(adouble)  
    msgbox str$(int(adouble))  
'end sub
```

'Item\$ and ItemCount Example



```
'sub main()  
    'Example of Item$() and ItemCount  
    dim pathstr as string  
  
    pathstr = environ$("PATH") 'get the path  
    msgbox "There are"+str$(itemcount(pathstr,";"))+" items in the path."  
    msgbox pathstr  
    msgbox item$(pathstr,3,4,";")  
'end sub
```

'Kill Statement Example



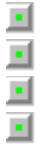
```
'sub main()  
    'Example of KILL command  
  
    kill "c:\junk.txt"  
'end sub
```


'LBound Function Example



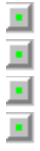
```
'sub main()  
    'Examples of LBOUND  
    dim ia1(8) as integer  
    dim ia2(65 to 70) as integer  
  
    msgbox str$(lbound(ia1))  
    msgbox str$(lbound(ia2))  
'end sub
```

'LCase\$ Function Example



```
'sub main()  
    'Example of LCase$  
    dim teststr as string  
  
    teststr = "THIS IS A TEST"  
    msgbox teststr  
    teststr = lcase$(teststr)  
    msgbox teststr  
'end sub
```

'Left\$ Function Example



```
'sub main()  
    'Example of left$()  
    dim teststr as string  
  
    teststr = "This is a test."  
    msgbox teststr  
    teststr = left$(teststr,7)  
    msgbox teststr  
'end sub
```

'Len Function Example



```
'sub main()  
    'Example of Len()  
    dim teststr as string  
  
    teststr = "This is a test."  
    msgbox ""+teststr+ " ' is"+str$(len(teststr))+ " characters long."  
'end sub
```

'Let Statement Example



```
'sub main()  
    'Examples of Let statement  
  
    let a% = 1  
    let s$ = "test"  
'end sub
```

'Line\$ Function Example



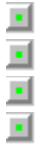
```
'sub main()  
    'Example of Line$()  
    dim crlf as string  
    dim testlines as string  
  
    crlf = chr$(13) +chr$(10)  
    'carriage return followed by a linefeed  
    testlines = "line 1" +crlf  
    testlines = testlines + "line 2" +crlf  
    testlines = testlines + "line 3" +crlf  
    testlines = testlines + "line 4"  
    msgbox testlines  
    msgbox line$(testlines,2,3)  
'end sub
```

'LineCount Function Example



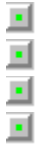
```
'sub main()  
    'Example of LineCount  
    dim crlf as string  
    dim testlines as string  
  
    crlf = chr$(13) + chr$(10)  
    'carriage return followed by a linefeed  
    testlines = "line 1" + crlf  
    testlines = testlines + "line 2" +crlf  
    testlines = testlines + "line 3" +crlf  
    testlines = testlines + "line 4"  
    msgbox testlines  
    msgbox str$(linecount(testlines))+ " lines total."  
'end sub
```

'Log Function Example



```
'sub main()  
    'Example of log() function  
  
    msgbox "Log of 1 is "+str$(log(1))  
    msgbox "Log of 2 is "+str$(log(2))  
'end sub
```


'LTrim\$ Function Example



```
'sub main()  
    'Example of LTrim$()  
    dim teststring as string  
  
    teststring = "      testing      "  
    msgbox "*" +teststring + "*"   
    msgbox "*" +ltrim$(teststring) + "*"   
'end sub
```

'MCI Function Example



```
'sub main()  
    'Example MCI command  
    'NOTE:  This program may not run on your machine.  
  
    dim resultstr as string  
    dim errorstr as string  
  
    MCI("play cdrom from 1 to 20",resultstr,errorstr)  
'end sub
```

'Menu Statement Example



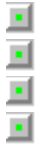
```
'sub main()  
    'Example of Menu command  
  
    aname$ = AppFind$("Notepad")  
    AppActivate aname$  
    Menu "File.Page Setup"  
'end sub
```

'Mid\$ Function Example



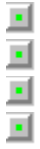
```
'sub main()  
    'Example of mid$() function and command  
    dim teststr as string  
  
    teststr = "This is a test."  
    msgbox teststr  
    msgbox mid$(teststr,3,5)      'starting at 3 and  
                                'retrieving 5 characters  
    mid$(teststr,3,5) = "      "  
    msgbox teststr  
    msgbox mid$(teststr,3,5)      'starting at 3 and  
                                'retrieving 5 characters  
  
'end sub
```

'Minute Function Example



```
'sub main()  
    'Example of minute() function  
    dim dt as double  
  
    dt = Now  
    msgbox str$(minute(dt))    'current minute  
'end sub
```

'MkDir Statement Example



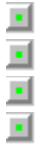
```
'sub main()  
    'Example of mkdir  
  
    Mkdir "C:\testdir"  
'end sub
```

'Mod Operator Example



```
'sub main()  
    'Example of MOD  
  
    msgbox str$(5 mod 3)    'should display 2--the  
                           'remainder after division  
  
'end sub
```

'Month Function Example



```
'sub main()  
    'Example of Month  
  
    msgbox str$(month(Now))    'display current month  
'end sub
```


'MsgBox Statement and Function Example



```
'sub main()
    'Examples of MsgBox function and statement

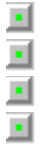
    for i% = 0 to 5
        MsgBox "message", i%, "title"
    next i%
    for i% = 0 to 5
        result = MsgBox("message", i%, "title")
        select case result
            case 1
                msgbox "OK was pressed"
            case 2
                msgbox "Cancel was pressed"
            case 3
                msgbox "Abort was pressed"
            case 4
                msgbox "Retry was pressed"
            case 5
                msgbox "Ignore was pressed"
            case 6
                msgbox "Yes was pressed"
            case 7
                msgbox "No was pressed"
        end select
    next i%

    'In this dialog box the second button is the default.
    result=msgbox("Hello World",3+256,"title")

    'In this dialog box the Stop symbol is displayed.
    result=msgbox("Hello World",3+256+16,"title")

'end sub
```

'Name Statement Example



```
'sub main()  
    'Example of Name statement  
  
    Name "C:\JUNK.TXT" as "C:\NEWJUNK.TXT"  
        'renames JUNK.TXT to NEWJUNK.TXT  
'end sub
```

'NetAttach Function Example



```
'sub main()  
  'Example of NetAttach  
  
  s$ = AskBox$("Server to attach to:")  
  u$ = AskBox$("Username:")  
  p$ = AskPassword$("Password:")  
  if NetAttach(s$,u$,p$) then  
    msgbox("Attach Successful.")  
  else  
    msgbox("Attach Failed.")  
  end if  
end sub
```

'NetConnectDrive Function Example



```
'sub main()  
    'Example of NetConnectDrive  
  
    lp$ = AskBox$("Local drive to connect to:")  
    np$ = AskBox$("Network path to connect:")  
    if NetConnectDrive(lp$,np$) then  
        msgbox "Drive connected."  
    else  
        msgbox "Drive connection failed."  
    end if  
end sub
```

'NetDetach Function Example



```
'sub main()  
    'Example of NetDetach  
  
    s$ = AskBox$("Server To Detach:")  
    if NetDetach(s$) then  
        msgbox "Server Detached."  
    else  
        msgbox "Error Detaching Server."  
    end if  
end sub
```

'NetDirectoryRights Function Example



```
'sub main()  
  'Example of NetDirectoryRights  
  
  p$ = AskBox$("Path to get rights for:")  
  r$ = AskBox$("Rights to search for:")  
  if NetDirectoryRights(p$,r$) then  
    msgbox "You have "+r$+ " rights for "+p$  
  else  
    msgbox "You do not have "+r$+ " rights for "+p$  
  end if  
'end sub
```

'NetDisconnectDrive Function Example



```
'sub main()  
    'Example of NetDisconnectDrive() function  
    dim drive as string  
    dim netpath as string  
  
    drive = "k"  
    netpath = "sweng_1/sys2:"  
  
    if NetConnectDrive(drive,netpath) then  
        msgbox "Drive "+drive+ " has been connected."  
        if NetDisconnectDrive(drive) then  
            msgbox "Drive "+drive+ " has been disconnected."  
        else  
            msgbox "Disconnect Failed."  
        end if  
    else  
        msgbox "Connection Failed."  
    end if  
end sub
```

'NetGetDirectoryRights Function Example



```
'sub main()  
    'Example of NetGetDirectoryRights$()  
  
    p$ = AskBox$("Path to get rights list for:")  
    r$ = NetGetDirectoryRights$(p$)  
    msgbox "Your rights for "+p$+ " are "+r$  
'end sub
```


'NetMemberOf Function Example



```
'sub main()  
  'Example of NetMemberOf  
  
  g$ = AskBox$("Group To Search For:")  
  s$ = AskBox$("Server To Search:")  
  if NetMemberOf(g$,s$) then  
    msgbox "You are a member of that group."  
  else  
    msgbox "You are not a member of that group."  
  end if  
'end sub
```

'NetStationID Function Example



```
'sub main()  
    'Example of NetStationID$()  
  
    nsi$ = NetStationID$()  
    msgbox "Your ethernet address is: "+nsi$  
'end sub
```

'NetUserName Function Example



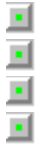
```
'sub main()  
    'Example of NetUserName  
  
    s$ = AskBox$("Server:")  
    nu1$ = NetUserName$()  
    nu2$ = NetUserName$(s$)  
    msgbox "You are "+nu1$+ " on your primary server, and "+nu2$+ " on "+s$+ "."  
'end sub
```

'NetworkStatus Function Example



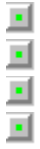
```
'sub main()  
    'Example of NetworkStatus function  
  
    ns% = NetworkStatus()  
    msgbox str$(ns%)  
    if ns% AND NS_ACTIVE then  
        msgbox "Network is active."  
    else  
        msgbox "Network is not active."  
    end if  
    if ns% AND NS_LOGGEDON then  
        msgbox "Logged On."  
    else  
        msgbox "Not Logged On."  
    end if  
'end sub
```

'Not Operator Example



```
'sub main()  
    'Example of NOT operator  
    dim testvar as integer  
  
    testvar = TRUE  
    if testvar then  
        msgbox "TestVar is TRUE"  
    end if  
  
    testvar = FALSE  
    if NOT testvar then  
        msgbox "TestVar is NOT TRUE"  
    end if  
'end sub
```

'Now Function Example



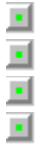
```
'sub main()  
    'Example of the Now function  
    dim cdt as double  
  
    cdt = Now  
    msgbox "Current date is "+str$(month(cdt))+ "/" +str$(day(cdt))+ "/" +str$(year(cdt))  
'end sub
```

'Null Function Example



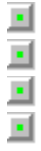
```
'sub main()  
    'Example of the Null function  
    dim teststr as string  
  
    teststr = "testing"  
    msgbox "*" + teststr + "*"  
    teststr = null  
    msgbox "*" + teststr + "*"  
'end sub
```

'Oct\$ Function Example



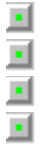
```
'sub main()  
    'Example of Oct$() function  
  
    msgbox Oct$(9)  
'end sub
```


'On Error Statement Example



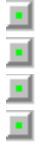
```
'sub main()  
    'Example of ON-ERROR statement  
  
    on error goto elabel      'enable error trap  
    error 101                 'simulate an error  
    end  
  
elabel:                      'error trap label  
    on error goto 0          'disable error trapping  
    end  
'end sub
```

'OpenFileName\$ Function Example



```
'sub main()  
    'Example of OpenFileName$  
  
    selffile$ = OpenFileName$("Open File", "All Files:*.bmp, *.wmf;Bitmaps:*.bmp;Metafiles:*.wmf")  
    msgbox "Selected File = "+selffile$  
'end sub
```

'Option Base Statement Example



Option Base 1

```
'sub main()  
    'Example of Option Base statement  
    dim a(5) as integer  
  
    msgbox str$(lbound(a))  
'end sub
```

'Or Operator Example



```
'sub main()  
    'OR statement  
    dim a as integer  
    dim b as integer  
  
    a = 5  
    b = 9  
    if (a < 6) OR (b > 8) then  
        msgbox "One of the conditions was true."  
    else  
        msgbox "Neither condition was true."  
    end if  
  
    if (a < 6) OR (b > 9) then  
        msgbox "One of the conditions was true."  
    else  
        msgbox "Neither condition was true."  
    end if  
'end sub
```

'PI Example



```
'sub main()  
    'Example of the PI function  
    dim adouble as double  
  
    adouble = pi  
    msgbox str$(adouble)  
'end sub
```

'PopupMenu Function Example



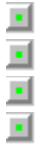
```
'sub main()  
  'Example of PopupMenu  
  dim apps$() as string  
  dim result as integer  
  
  AppList apps$  
  result = PopupMenu(apps$)  
  if result >= lbound(apps$) then  
    msgbox "You chose "+apps$(result)  
  else  
    msgbox "You chose nothing -"+str$(result)  
  end if  
end sub
```

'Print Statement Example



```
'sub main()  
    'Example of Print  
  
    ViewPortOpen  
    ViewPortClear  
    print "this is some data"  
    sleep 5000  
    ViewPortClose  
'end sub
```

'Print # Statement Example



```
'sub main()  
  open "test.dat" for output as #1  
  print #1,10,34,"Hello World";  
  a = 10  
  s$ = "this is a test"  
  print #1,a;s$,  
  print #1,67  
  close #1  
'end sub
```


'PrinterGetOrientation Function Example



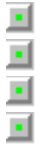
```
'sub main()  
  'Example of PrinterGetOrientation and  
  'PrinterSetOrientation  
  dim oldorient as integer  
  
  oldorient = PrinterGetOrientation()  
  select case oldorient  
    case PO_PORTRAIT  
      msgbox "Printer is set up for portrait print."  
    case PO_LANDSCAPE  
      msgbox "Printer is set up for landscape print."  
  end select  
  PrinterSetOrientation oldorient  
'end sub
```

'PrinterSetOrientation Statement Example



```
'sub main()  
  'Example of PrinterGetOrientation and  
  'PrinterSetOrientation  
  dim oldorient as integer  
  
  oldorient = PrinterGetOrientation()  
  select case oldorient  
    case PO_PORTRAIT  
      msgbox "Printer is set up for portrait print."  
    case PO_LANDSCAPE  
      msgbox "Printer is set up for landscape print."  
  end select  
  PrinterSetOrientation oldorient  
'end sub
```

'PrintFile Function Example



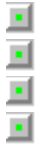
```
'sub main()  
    'Example of PrintFile  
  
    taskid = printfile("c:\testfile.txt")  
    msgbox "Your file is printing as task #" + str$(taskid)  
'end sub
```

'Random Function Example



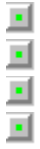
```
'sub main()  
    'Example of Random function  
  
    ViewPortOpen  
    ViewPortClear  
    for j% = 1 to 10  
        i% = Random(1,100)  
        print i%  
    next j%  
    sleep 5000  
    ViewPortClose  
'end sub
```

'Randomize Function Example



```
'sub main()  
    'Example of Randomize statement  
  
    Randomize 65  
    Randomize  
'end sub
```

'ReadIni\$ Function Example



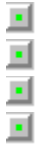
```
'sub main()  
  'Example of ReadIni$  
  
  winshell$ = ReadIni$("boot","shell","system.ini")  
  msgbox winshell$  
'end sub
```

'ReadIniSection Statement Example



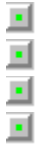
```
'sub main()  
    'Example of ReadIniSection  
    dim iniitems() as string  
  
    ReadIniSection "boot",iniitems,"system.ini"  
    ViewPortOpen  
    print "[Boot] section items from SYSTEM.INI"  
    print " "  
    for i% = lbound(iniitems) to ubound(iniitems)  
        print iniitems(i%)  
    next i%  
    sleep 5000  
    ViewPortClose  
'end sub
```

'ReDim Statement Example



```
'sub main()  
    'Example of ReDim  
    dim stuff(5) as integer  
  
    msgbox str$(ubound(stuff))  
    redim stuff(10)  
    msgbox str$(ubound(stuff))  
'end sub
```


'REM Statement Example



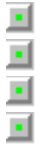
```
'sub main()  
    'Example of REM  
  
REM  this is also a comment  
'end sub
```

'RefreshIni Statement Example



```
'sub main()  
    'Example of RefreshIni  
  
    RefreshIni  
'end sub
```

'Reset Statement Example



```
'sub main()  
    'Example of RESET statement  
  
    Reset  
    'close all open files (writes out i/o buffers)  
'end sub
```

'RestoreEnv Function Example



```
'sub main()  
  'Example of SaveEnv and RestoreEnv  
  
  a% = SaveEnv(ENV_WINDOWS)  
  a% = RestoreEnv(ENV_WINDOWS)  
'end sub
```

'Resume Statement Example



```
'sub main()  
    'Example of Resume statement  
  
    on error goto testerror  
    error 101  
    msgbox "resumed as anticipated"  
    on error goto 0  
    end  
  
testerror:  
    resume next  
'end sub
```

'Return Statement Example



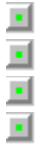
```
'sub main()  
    'Examples of GOSUB and RETURN  
  
    for x% = 1 to 5  
        gosub mylabel  
    next x%  
end  
  
mylabel:  
    msgbox "Here we are!"  
    return  
'end sub
```

'Right\$ Function Example



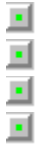
```
'sub main()  
    'Example of right$()  
    dim teststr as string  
  
    teststr = "This is a test."  
    msgbox teststr  
    teststr = right$(teststr,7)  
    msgbox teststr  
'end sub
```

'Rmdir Statement Example



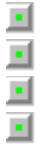
```
'sub main()  
    'Example of Rmdir  
  
    Rmdir "C:\testdir"  
'end sub
```


'Rnd Function Example



```
'sub main()  
    'Example of Rnd function  
  
    ViewPortOpen  
    Randomize  
    for i% = 1 to 10  
        print Rnd(1)  
    next i%  
    sleep 5000  
    ViewPortClose  
'end sub
```

'RTrim\$ Function Example



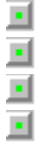
```
'sub main()  
    'Example of RTrim$()  
    dim teststring as string  
  
    teststring = "      testing      "  
    msgbox "*" +teststring + "*"   
    msgbox "*" +rtrim$(teststring) + "*"   
'end sub
```

'SaveEnv Function Example



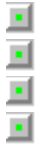
```
'sub main()  
    'Example of SaveEnv and RestoreEnv  
  
    a% = SaveEnv(ENV_WINDOWS)  
    a% = RestoreEnv(ENV_WINDOWS)  
'end sub
```

'SaveFileName\$ Function Example



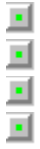
```
'sub main()  
    'Example of SaveFileName$  
  
    selffile$ = SaveFileName$("Open File", "All Files:*.bmp, *.wmf;Bitmaps:*.bmp;Metafiles:*.wmf")  
    msgbox "Selected File = "+selffile$  
'end sub
```

'Second Function Example



```
'sub main()  
    'Example of second() function  
    dim dt as double  
  
    dt = Now  
    msgbox str$(second(dt))....'current second  
'end sub
```

'Select...Case Statement Example



```
'sub main()  
    'Example of SELECT-CASE statement  
  
    i% = 1  
    select case i%  
        case 1  
            msgbox "i% is 1"  
        case 2  
            msgbox "i% is 2"  
    end select  
'end sub
```

'SelectBox Function Example



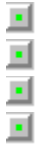
```
'sub main()  
  'example of SelectBox function  
  dim alist() as string  
  
  AppList alist  
  result% = SelectBox("Application List","Pick One",alist)  
  msgbox "You selected "+alist(result%)  
'end sub
```

'SendKeys Statement Example



```
'sub main()  
    'Example of SendKeys  
    dim alttab as string  
  
    alttab = "%{TAB}"  
    msgbox "Press OK to do first Alt-Tab"  
    SendKeys alttab,TRUE  
    msgbox "Press OK to Alt-Tab back to original application"  
    SendKeys alttab,FALSE  
'end sub
```


'SetAttr Statement Example



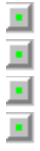
```
'sub main()  
    'Example of SetAttr  
  
    SetAttr "C:\AUTOEXEC.BAT",0  
    'set file attribute to NORMAL  
'end sub
```

'SetEnv Function Example



```
'sub main()  
    'Example of SetEnv and GetEnv$()  
    tmp$ = GetEnv$("TMP",ENV_WINDOWS)  
    tv$ = AskBox$("New Value For TMP Environment Variable:")  
    a% = SetEnv("TMP",tv$,ENV_WINDOWS)  
    msgbox "New value for TMP is "+ GetEnv$("TMP",ENV_WINDOWS)  
    'restore old value  
    a% = SetEnv("TMP",tmp$,ENV_WINDOWS)  
'end sub
```

'Sgn Function Example



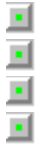
```
'sub main()  
    'Example of SGN() function  
  
    i% = -2  
    msgbox str$(sgn(i%))  
    i% = 0  
    msgbox str$(sgn(i%))  
    i% = 2  
    msgbox str$(sgn(i%))  
'end sub
```

'Shell Function Example



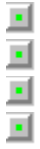
```
'sub main()  
    'Example of Shell function  
  
    taskid% = Shell("notepad.exe",1)  
'end sub
```

'Sin Function Example



```
'sub main()  
    'Example of Sin() function  
  
    result# = sin(0)  
    msgbox str$(result#)  
    result# = sin(1)  
    msgbox str$(result#)  
'end sub
```

'Sleep Statement Example



```
'sub main()  
    'Example of Sleep command  
  
    msgbox "Press OK to sleep 5 seconds"  
    sleep 5000  
'end sub
```

'SleepUntil Function Example



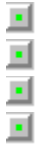
```
'sub main()
  'Example of SleepUntil

  t$ = time$()      'get current time
  t$ = left$(t$,5)   'get hours and
                    'minutes only
  m$ = right$(t$,2)  'get minutes value
  h$ = left$(t$,2)   'get hour value
  x = val(m$)        'get the value of the minutes
  y = val(h$)        'get value of hour string
  x = x + 2          'increment minutes
                    'to wait 2 minutes

  if x > 59 then
    y = y + 1
    x = x - 59
  end if
  h$ = str$(y)       'convert hours
                    'back to string
  h$ = right$(h$,len(h$)-1)
                    'get rid of leading blank from conversion
  if len(h$) = 1 then h$ = "0" + h$
  m$ = str$(x)       'convert minutes
                    'back to string likewise
  m$ = right$(m$,len(m$)-1)
  if len(m$) = 1 then m$ = "0" + m$
  t$ = h$+ " "+m$    'reconstruct time
                    'value
  retval% = SleepUntil(t$,TRUE, "Press CANCEL to quit.", "Sleeping Until "+t$,TRUE,TRUE)
'end sub
```

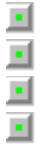


'Space\$ Function Example



```
'sub main()  
  'Example of Space$() function  
  
  stuff$ = null  
  msgbox "*" + stuff$ + "  
  stuff$ = space$(10)  
  msgbox "*" + stuff$ + "  
'end sub
```

'Sqr Function Example



```
'sub main()  
    'Example of SQR function  
  
    msgbox str$(sqr(100))  
'end sub
```

'Stop Statement Example



```
'sub main()  
    'Example of Stop statement  
  
    Stop  
'end sub
```

'StrComp Function Example



```
'sub main()
    'Example of StrComp function

    'Case-sensitive comparison
    teststr$ = "This is a test string"
    result% = StrComp(teststr$, "THIS IS A TEST STRING", 0)

    select case result%
        case -1
            msgbox "teststr$ is less than the compare string"
        case 0
            msgbox "teststr$ is equal to the compare string"
        case 1
            msgbox "teststr$ is greater than compare string"
    end select

    'Case-insensitive comparison
    result% = StrComp(teststr$, "THIS IS A TEST STRING", 1)
    select case result%
        case -1
            msgbox "teststr$ is less than the compare string"
        case 0
            msgbox "teststr$ is equal to the compare string"
        case 1
            msgbox "teststr$ is greater than compare string"
    end select
end sub
```

'Str\$ Function Example



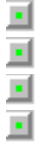
```
'sub main()  
    'Example of str$() function  
  
    i% = 3  
    s$ = str$(i%)  
    msgbox s$  
'end sub
```

'String\$ Function Example



```
'sub main()  
    'Example of String$() function  
  
    s$ = String$(10,65)  
    msgbox s$  
    s$ = String$(10,"B")  
    msgbox s$  
'end sub
```

'SystemFreeMemory Function Example



```
'sub main()  
    'Example of SystemFreeMemory  
  
    msgbox "Free Memory ="&str$(SystemFreeMemory)  
'end sub
```

'SystemFreeResources Function Example



```
'sub main()  
    'Example of SystemFreeResources  
  
    msgbox "Free Resources =" +str$(SystemFreeResources)+ "%"  
'end sub
```


'SystemMouseTrails Statement Example



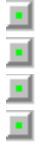
```
'sub main()  
  'Example of SystemMouseTrails  
  
  SystemMouseTrails TRUE  
  msgbox "Move the Mouse Around Now"  
  SystemMouseTrails FALSE  
'end sub
```

'SystemRestart Statement Example



```
'sub main()  
  'Example of SystemRestart  
  'WARNING:  this will restart Windows if you run it  
  
  SystemRestart  
'end sub
```

'SystemTotalMemory Function Example



```
'sub main()  
    'example of SystemTotalMemory  
  
    msgbox "Total System Memory ="&str$(SystemTotalMemory)  
'end sub
```

'SystemWindowsDirectory\$ Function Example



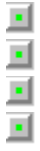
```
'sub main()  
    'Example of SystemWindowsDirectory$  
  
    msgbox SystemWindowsDirectory$  
'end sub
```

'SystemWindowsVersion\$ Function Example



```
'sub main()  
    'Example of SystemWindowsVersion$  
  
    msgbox SystemWindowsVersion$  
'end sub
```

'Tan Function Example



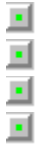
```
'sub main()  
    'Example of Tan() function  
  
    i# = Tan(1)  
    msgbox str$(i#)  
'end sub
```

'Time\$ Statement Example



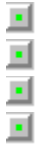
```
'sub main()  
    'Example of Time command  
  
    time$ = "21:30:40"  
'end sub
```

'Time\$ Function Example



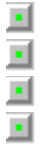
```
'sub main()  
    'Example of Time$() function  
  
    msgbox Time$()  
'end sub
```


'Timer Function Example



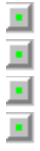
```
'sub main()  
    'Example of Timer function  
    dim tval as long  
  
    tval = Timer  
    msgbox str$(tval)  
'end sub
```

'TimeSerial Function Example



```
'sub main()  
    'Example of TimeSerial function  
    dim tser as double  
  
    tser = TimeSerial(21,40,44)  
'end sub
```

'TimeValue Function Example



```
'sub main()  
    'Example of TimeValue function  
    dim tval as double  
  
    tval = TimeValue("21:40:44")  
'end sub
```

'Trim\$ Function Example



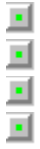
```
'sub main()  
    'Example of Trim$()  
    dim teststring as string  
  
    teststring = "      testing      "  
    msgbox "*" +teststring + "*"   
    msgbox "*" +trim$(teststring) + "*"   
'end sub
```

'UBound Function Example



```
'sub main()  
    'Examples of UBOUND  
    dim ia1(8) as integer  
    dim ia2(65 to 70) as integer  
  
    msgbox str$(ubound(ia1))  
    msgbox str$(ubound(ia2))  
'end sub
```

'UCase\$ Function Example



```
'sub main()  
    'Example of UCase$  
    dim teststr as string  
  
    teststr = "this is a test"  
    msgbox teststr  
    teststr = ucase$(teststr)  
    msgbox teststr  
'end sub
```

'Val Function Example



```
'sub main()  
    'Example of Val function  
  
    teststr$ = "123.3"  
    tval# = val(teststr$)  
    msgbox str$(tval)  
'end sub
```

'Viewport Example



```
'sub main()  
  'Example of ViewPort commands  
  
  'Create a veiwport window  
  ViewPortOpen "My ViewPort"  
  For i% = 1 to 20  
    Print i%  
  Next i%  
  msgbox "Press OK to clear the viewport."  
  ViewPortClear  
  msgbox "Press OK to close the viewport."  
  ViewPortClose  
'end sub
```


'VLine Statement Example



```
'sub main()  
  'Examples of VLINE  
  
  ViewPortOpen  
  ViewPortClear  
  for i% = 1 to 50  
    Print "Here is some test data."  
  next i%  
  VLine 50  
  sleep 2000  
  VLine -50  
  ViewPortClose  
'end sub
```

'VPage Statement Example



```
'sub main()  
  'Examples of VPage  
  
  ViewPortOpen  
  ViewPortClear  
  for i% = 1 to 50  
    Print i%  
  next i%  
  VPage 1  
  sleep 2000  
  VPage -1  
  ViewPortClose  
'end sub
```

'VScroll Statement Example



```
'sub main()  
  'Example of VSCROLL  
  
  ViewPortOpen  
  ViewPortClear  
  for i% = 1 to 50  
    Print "Test data for viewport scroll test."  
  next i%  
  sleep 2000  
  VScroll 50 '50 percent scroll  
  sleep 2000  
  VScroll 1   'no scroll  
  sleep 2000  
  ViewPortClose  
'end sub
```

'WaitForTaskCompletion Function Example



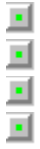
```
'sub main()  
    'Example of WaitForTaskCompletion  
  
    taskid% = Shell("notepad",1)  
    'The next statement pauses until Notepad  
    'is shut down  
    WaitForTaskCompletion taskid%  
    msgbox "All done."  
'end sub
```

'Weekday Function Example



```
'sub main()  
  'Example of Weekday()  
  
  wday% = weekday(Now)  
  select case wday%  
    case 1  
      msgbox "Today Is Sunday"  
    case 2  
      msgbox "Today Is Monday"  
    case 3  
      msgbox "Today Is Tuesday"  
    case 4  
      msgbox "Today Is Wednesday"  
    case 5  
      msgbox "Today Is Thursday"  
    case 6  
      msgbox "Today Is Friday"  
    case 7  
      msgbox "Today Is Saturday"  
  end select  
'end sub
```

'While...Wend Statement Example



```
'sub main()  
    'Example of WHILE-WEND  
  
    i% = 4  
    while i% > 1  
        msgbox "not yet"  
        i% = i% -1  
    wend  
    msgbox str$(i%)  
'end sub
```

'WinActivate Statement Example



```
'sub main()  
    'Example of WinActivate  
  
    appn$ = AppFind$("Notepad")  
    WinActivate appn$  
'end sub
```

'WinClose Statement Example



```
'sub main()  
    'Example of WinClose  
  
    appn$ = AppFind$("Notepad")  
    WinClose appn$  
'end sub
```


'WinFind Function Example



```
'sub main()  
    'Example of WinFind  
  
    appn$ = AppFind$("Notepad")  
    hWnd% = WinFind(appn$)  
    msgbox str$(hWnd%)  
'end sub
```

'WinList Function Example



```
'sub main()  
    'Example of WinList  
    dim hWindows() as integer  
  
    WinList hWindows  
    for i% = lbound(hWindows) to ubound(hWindows)  
        msgbox str$(hWindows(i%))  
    next i%  
'end sub
```

'WinMaximize Statement Example



```
'sub main()  
    'Example of WinMaximize  
  
    appn$ = AppFind$("Notepad")  
    WinMaximize appn$  
'end sub
```

'WinMinimize Statement Example



```
'sub main()  
    'Example of WinMinimize  
  
    appn$ = AppFind$("Notepad")  
    WinMinimize appn$  
'end sub
```

'WinMove Statement Example



```
'sub main()  
    'Example of WinMove  
  
    appn$ = AppFind$("Notepad")  
    for i% = 0 to 100  
        WinMove i%, i%, appn$  
    next i%  
'end sub
```

'WinRestore Statement Example



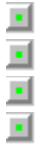
```
'sub main()  
    'Example of WinRestore  
  
    appn$ = AppFind$("Notepad")  
    WinRestore appn$  
'end sub
```

'WinSize Statement Example



```
'sub main()  
    'Example of WinSize  
  
    appn$ = AppFind$("Notepad")  
    for i% = 1 to 200  
        WinSize i%, i%, appn$  
    next i%  
'end sub
```

'Word\$ Function Example



```
'sub main()  
    'Example of WORD$() and WORDCOUNT functions  
    dim teststr as string  
  
    teststr = "The quick brown fox jumps over the lazy dog."  
    for i% = 1 to wordcount(teststr)  
        msgbox word$(teststr,i%,i%)  
    next i%  
'end sub
```


'WordCount Function Example



```
'sub main()  
    'Example of WORD$() and WORDCOUNT functions  
    dim teststr as string  
  
    teststr = "The quick brown fox jumps over the lazy dog."  
    for i% = 1 to wordcount(teststr)  
        msgbox word$(teststr,i%,i%)  
    next i%  
'end sub
```

'WriteIni Statement Example



```
'sub main()  
    'Example of WriteIni statement  
  
    WriteIni "anewsection","anewitem","value","win.ini"  
'end sub
```

'Xor Operator Example



```
'sub main()
    'Example of XOR
    dim a as integer
    dim b as integer

    a = 5
    b = 9
    if (a < 6) XOR (b > 8) then
        msgbox "Both conditions were not the same."
    else
        msgbox "Both conditions were the same--either TRUE or FALSE."
    end if

    if (a < 6) XOR (b > 9) then
        msgbox "Both conditions were not the same."
    else
        msgbox "Both conditions were the same--either TRUE or FALSE."
    end if
end sub
```

'Year Function Example



```
'sub main()  
    'Example of Year  
  
    msgbox str$(year(Now))    'display current year  
'end sub
```

Using the DCL Editor

Creating, Opening, Saving, and Printing Scripts



[Creating a New Script](#)



[Opening an Existing File](#)



[Opening a Recent File](#)



[Saving a New Script](#)



[Saving an Existing File](#)



[Saving an Existing File Under a New Name](#)



[Making an Executable from the Current Script](#)



[Distributing an Executable](#)



[Printing a Script](#)



[Setting Up for Printing](#)



[Exiting DCL Editor](#)

Working with Text



[Editing Text](#)



[Selecting Text](#)



[Text Editing Keys](#)



[Finding Text](#)



[Replacing Text](#)

Using the Toolbar and Status Bar



[Using the Toolbar](#)



[Using the Status Bar](#)

Running Scripts



[Running a Script](#)

Using Tools



Starting the Debugger



Starting the Dialog Editor



Recording a Macro



Events Recorded by DCL



Checking the Syntax

Using Windows



Working with Document Windows

Getting Help



Getting Help

Commands



DCL Editor Commands

Creating a New Script

To create a new script:

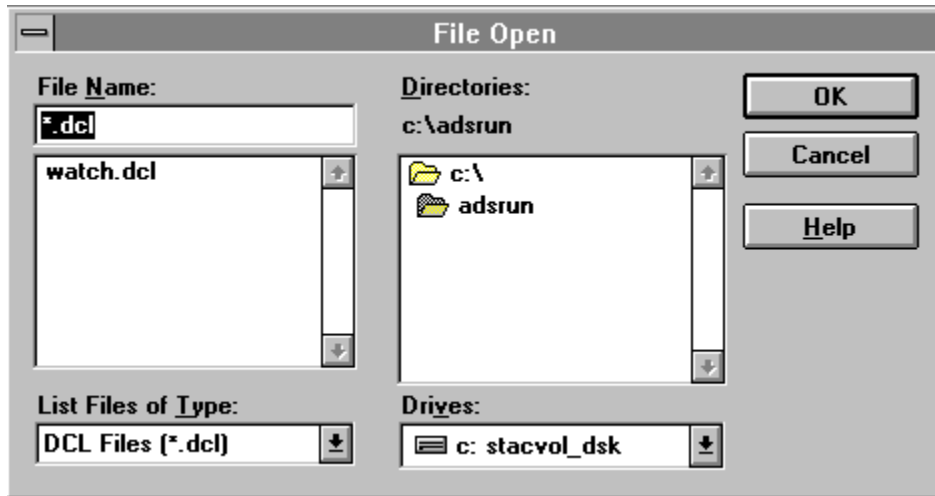
1. Choose the New command from the File menu.
2. Enter the commands that make up the script.
If you need help for a particular command, position the cursor on the line containing the command and press F1.
3. Save your script as described under [Saving a New Script](#).
4. Run your script as described under [Running a Script](#).

Opening an Existing File

To open an existing file:

1. Choose the Open command from the File menu.

The File Open dialog box is displayed.



2. Select the Drive and Directory containing the file.
3. Enter the File Name. Or select the file from the File Name list box.

You can change the types of files displayed in the File Name list box by selecting another file type from the List Files of Type box.

4. Choose OK.

Opening a Recent File

The last four files you saved are displayed toward the bottom of the File menu.

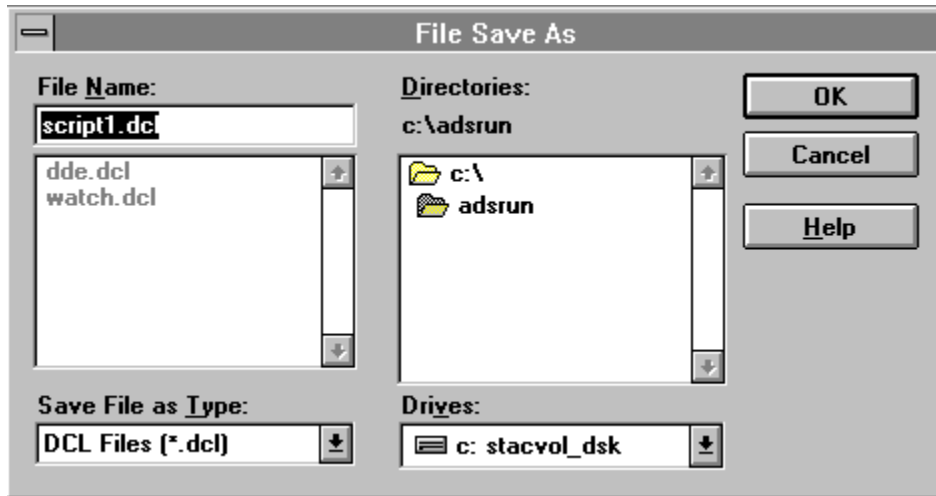
To open one of these files, choose the file name from the File menu.

Saving a New Script

To save a new script:

1. Choose the Save command from the File menu.

The File Save As dialog box is displayed.



2. If you need to change the current path, select the Drive and Directory to contain the file.
3. Enter the File Name.
4. Choose OK.

Saving an Existing File

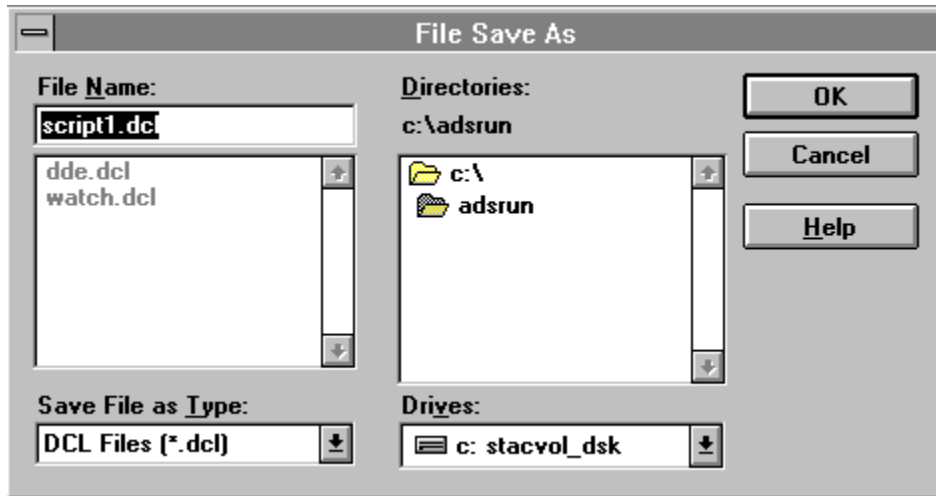
To save an existing file, choose the Save command from the File menu.

Saving an Existing File Under a New Name

To save an existing file under a new name:

1. Choose the Save As command from the File menu.

The File Save As dialog box is displayed.



2. If you want to change the current path, select the Drive and Directory to contain the file.
3. Enter the File Name.
4. From the Save File As Type box, select the desired file type.
5. Choose OK.

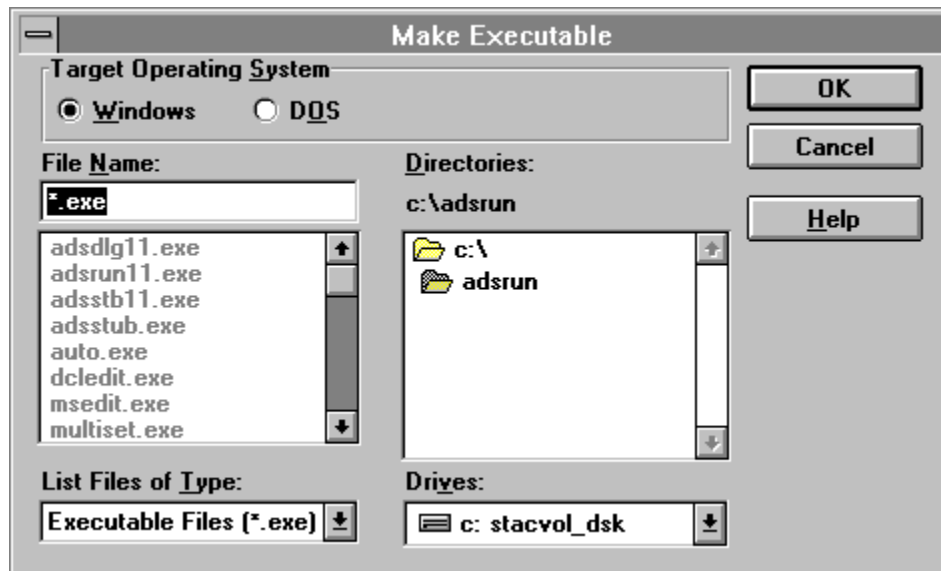
If a file having the same name already exists, you will be asked to confirm that you want to overwrite the file.

Making an Executable from the Current Script

To make an executable from the current script:

1. Choose the Make Exe command from the File menu.

The Make Executable dialog box is displayed.



2. Select the Target Operating System Windows or DOS.
3. If you want to change the current path, select the Drive and Directory to contain the file.
4. Enter a new File Name or accept the default executable name.
5. Choose OK.

If a file having the same name already exists, you will be asked to confirm that you want to overwrite the file.

Note: If you create an executable to run under DOS, make sure the **DCL** commands in your script are supported under DOS. The [Desktop Control Language Reference](#) indicates if a command is **not** supported under DOS.













Related Topic:



[Distributing an Executable](#)

Distributing an Executable

When you distribute an executable to users in your organization to run under Windows, the following files must be in the same directory as the executable or on the path:

	ADSCON11.EBL	
	ADSENV.DLL	
	ADSPUB11.DLL	
	ADSRUN11.DLL	
	ADSUTILS.DLL	
	CTL3D.DLL	
	DCL.EBL	
	WWNETWAR.DLL	(For network functionality)
	WWNET.INI	"
	NWCALLS.DLL	"
	NWLOCALE.DLL	"
	NWNET.DLL	"

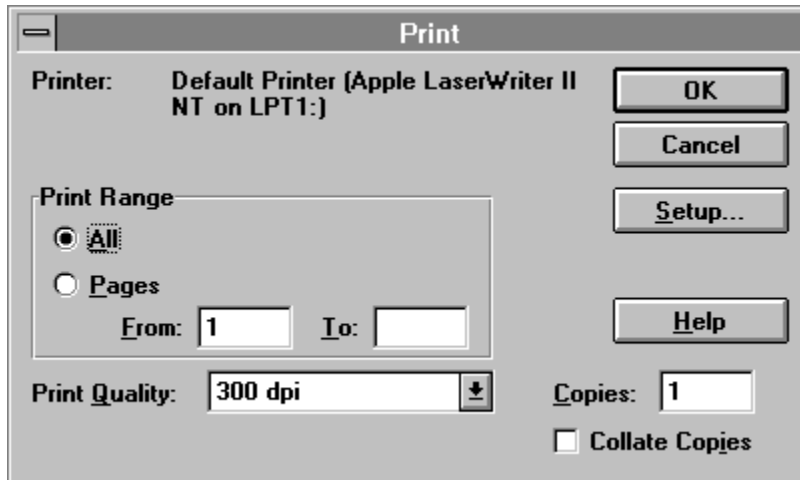
No support files are required for executables running under DOS.

Printing a Script

To print a script:

1. Choose the Print command from the File menu.

The Print dialog box is displayed.



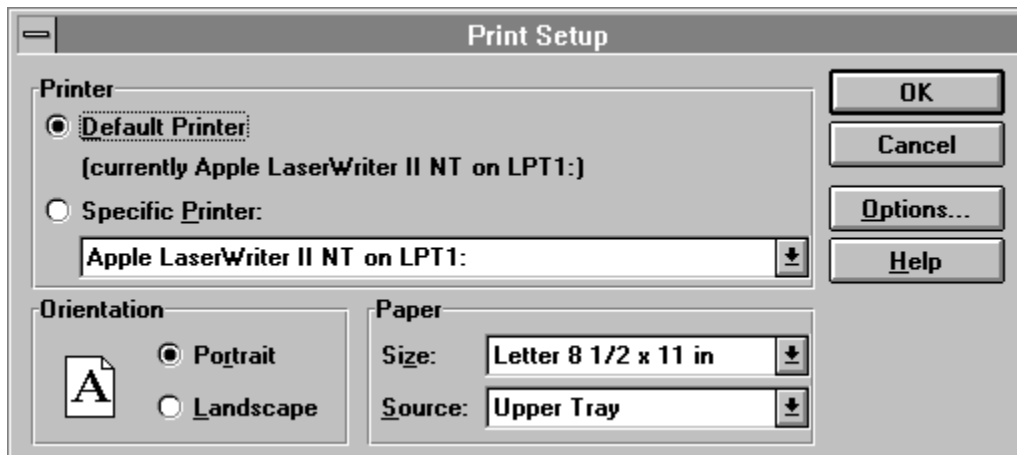
2. If you want to change the printer or printer settings, choose the Setup button and see [Setting Up for Printing](#).
3. Select the Print Range. Your choices are all of the file or a range of pages.
4. Change any of these print options as desired: Print Quality (depending on your printer), Copies, Collate Copies.
5. Choose OK.

Setting Up for Printing

To set up your printer:

1. Choose the Print Setup command from the File menu.

The Print Setup dialog box is displayed.



2. Select the Default Printer or another Specific Printer.
3. Select the Orientation of the paper: portrait or landscape.
4. Select the desired Paper Size and Source, depending on your printer.
5. If you want to change the setting for your printer, choose the Options button. One or more dialog boxes specific to your printer will be displayed.
6. Choose OK.

Exiting DCL Editor

To exit **DCL Editor**, choose the Exit command from the File menu.

Editing Text

The **DCL Editor** functions like other Windows text editors.

Use the Cut command on the Edit menu to remove a block of text you have selected from your script and store it in the Windows clipboard.

Use the Copy command to copy the selected text to the clipboard.

Use the Paste command to insert the text currently in the clipboard at the insertion point in the current document window.

Use the Select All command to select all of the text in the current document window.

Use the Move Text Left and Move Text Right commands to shift the selected text one tab stop left or right.

Use the Undo command to undo the last editing operation.

Related Topic:



[Text Editing Keys](#)



[Selecting Text](#)

Selecting Text

To Select a block of text for an editing operation:

- 1 Position the insertion point at the beginning of the text to be selected and click the left mouse button.
- 2 Move the insertion point to the end of the block to be selected.
- 3 Hold down the Shift key and press the left mouse button.

OR

- 1 Move the insertion point to the beginning of the text to be selected.
- 2 Hold the Shift key.
- 3 Move the cursor to the end of the text to be selected.

You can choose the Select All command from the Edit menu to select all of the text in the current document window.

Related Topic:



[Editing Text](#)

Finding Text

To find a string of text in a script:

1. Choose the Find command from the Edit menu.

The Find dialog box is displayed.



2. Enter the text string to search for in the Find What text box.

Shortcut: If you select the text you want to search for before choosing the Find command, the selected text is put in the Find What text box for you.

3. Select the Match Case check box if you want the search to differentiate between upper- and lowercase letters.
4. Specify the Direction of the search up or down.
5. Choose the Find Next button to begin the search.

The first occurrence of the string after the insertion point is highlighted.

6. To continue searching for subsequent occurrences of the string, choose Find Next.

The next occurrence of the string is highlighted.

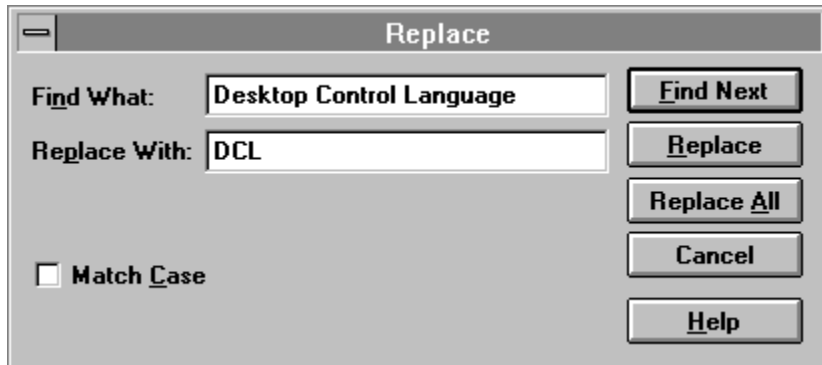
Hint: You can edit the script with the Find dialog box displayed. Use the mouse or press Alt+F6 to move between the dialog box and the script.

Replacing Text

To replace one text string with another:

1. Choose the Replace command from the Edit menu.

The Replace dialog box is displayed.



2. Enter the text string to search for in the Find What text box.

Shortcut: If you select the text you want to search for before choosing the Replace command, the selected text is put in the Find What text box for you.

3. Enter the replacement text in the Replace With box.
4. Select the Match Case check box if you want the search-and-replace operation to differentiate between upper- and lowercase letters.
5. Choose the Find Next button to begin the search.

The first occurrence of the string after the insertion point is highlighted.

6. If you want to substitute the replacement text for the string, choose the Replace button. (The search for the next matching string continues after the replacement is made.)

OR

If you want to go on to the next occurrence of the string without making the text change, choose the Find Next button.

OR

If you want to change all subsequent occurrences of the string to the replacement text, choose the Replace All button. (To avoid accidental changes, we recommend that you use this feature with caution.)

Using the Toolbar

The Toolbar is a row of buttons representing frequently used **DCL Editor** commands. Instead of using the menus, you can choose a Toolbar button to execute a command.

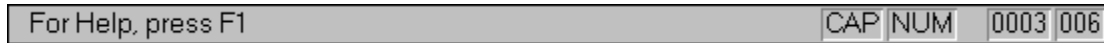
The Toolbar looks like this. Click any button you want information about.



To toggle the Toolbar display on or off, choose Toolbar from the View menu.

Using the Status Bar

The Status Bar appears at the bottom of the **DCL Editor** window. The Status Bar looks like this.



When a menu is displayed, the Status Bar provides a description of the currently selected item.

The Status Bar also indicates when the Caps Lock or Num Lock key is on.

The current line and column of the cursor are displayed in far right corner of the Status Bar.

To toggle the display of the Status Bar on or off, choose Status Bar from the View menu.

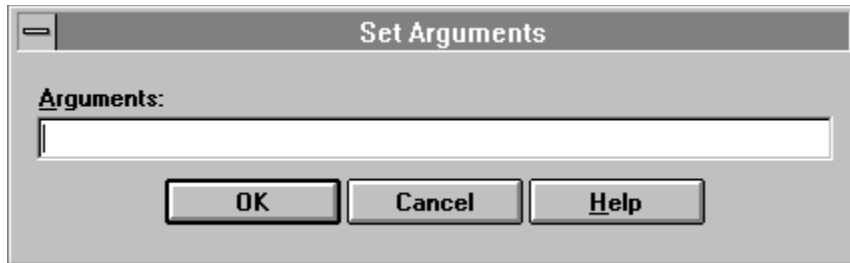
Running a Script

To run a script that does not use command-line arguments, choose Start Script from the Run menu.

To run a script that uses command-line arguments, follow these steps:

1. Choose the Arguments command from the Run menu.

The Set Arguments dialog box is displayed.



2. In the Set Arguments dialog box, enter the command-line arguments to be passed to the script's `main` subroutine and choose OK. (In your script, you can get these arguments through the `Command$` statement.)
3. Choose Start Script from the Run menu.

To stop the script that is currently executing, choose End Script from the Run menu.

Running Executables: If the script has been saved as an executable file (using the File/Make Exe command), you can use the **Applications Manager** or Program Manager File/Run function or File Manager to run the script. For more information, see [Making an Executable from the Current Script](#).

Starting the Debugger

To start the **DCL Debugger**, choose Debugger from the Tools menu.

Related Topic:



[Using the Debugger](#)

Starting the Dialog Editor

To start the **DCL Dialog Editor**, choose Dialog Editor from the Tools menu.

Related Topic:



[Using the Dialog Editor](#)

Recording a Macro

DCL's Recorder captures Windows events and translates them into **DCL** statements that can be inserted in a script. For a list of the of the events you can record, see [Recorded Events](#).


To record Windows events:


1. Choose Recorder from the Tools menu.

The Script Recorder Options dialog box is displayed.





2. Make any desired changes to the dialog box:

 **Include Comments:** Indicate whether to include descriptive comments in the code generated by the Recorder.

 **High-Level Statements:** Indicate whether to consolidate multiple events into one High-Level Statement where possible. (For example, selecting a menu is recorded as a menu command rather than a series of mouse commands.)

 **Keyboard:** Indicate whether to capture events resulting from keyboard activity.

 **Mouse:** Indicate whether to capture events resulting from mouse activity then indicate whether the coordinate system is relative to the window or screen.

 **Stop Recording On:** Specify the key combination to stop the Recorder.

3. Choose OK.

The **DCL Editor** window is minimized. The Recorder window is displayed.



4. Perform the Windows events you want to record.

You can interrupt recording by choosing the Pause button in the Recorder window. Then choose the Record button when you want to resume recording.

5. When you have finished recording events, choose the End button in the Recorder window.
6. The **DCL Editor** window is restored; the Insert Recording dialog box is displayed.



7. Press F9 (or use mouse) to switch to your script.
8. Position the insertion point where you want to insert the recorded code.
9. Press F9 (or use mouse) to switch back to the Insert Recording dialog box.
10. Choose OK.

Related topic:



[Recorded Events](#)

Recorded Events

The Recorder recognizes the following Windows actions:

Application Focus Switch

The Recorder recognizes when an application receives the focus and records this as a `WinActivate` statement if you perform an activity within the application. The focus can be shifted using any of the normal Windows application switching methods (Alt+Tab, Ctrl+Esc, clicking the mouse on another application, selecting the "Switch To..." command from an application's control menu, etc.).

Window focus shifts are also recorded to ensure that subsequent actions occur (and are synchronized) within the appropriate window.

Statements:



WinActivate



AppActivate

Window Scrolling

Interactions with windows that respond to scrolling messages (windows that have built-in Windows scroll bars, like Notepad and Write) are recorded.

Statements



VLine



VPage



HLine



HPage



HScroll



VScroll

Mouse Activity

Mouse down and up actions are compressed to high-level forms when possible. Further, mouse activity that leads to a recognizable result, such as resizing a window, is compressed out completely.

Statements



QueFlush




QueMouseClicked



QueMouseDown



 QueMouseDblDn



QueMouseDn



QueMouseMove



QueMouseUp



QueSetRelativeWindow

Keyboard Activity

Keyboard press/release actions are converted. Complex key combinations, such as Ctrl+Escape or Alt+Shift+C are converted into a readable form. Keyboard activity that results in a higher level command are compressed out completely, such as the following keystroke sequence, which results in a window reposition: Alt+Space, M, Right, Right, Right, Enter.

Keyboard activity that modifies mouse events is also recorded, such as holding down the Shift key and clicking the left mouse button.

Statements



DoKeys



QueFlush



QueKeys



QueKeyDn



QueKeyUp

Window Management

The Recorder recognizes high-level interactions with windows, such as movement, sizing, activation, minimizing, maximizing, and restoring. The Recorder differentiates between interactions with applications, popup windows, and child MDI windows.

Statements



AppMaximize



AppMinimize



AppRestore



AppMove



AppSize



WinMove



WinSize



WinMaximize



WinMinimize



WinRestore

Menu Commands

The Recorder watches for interactions with an application's built-in menu and records the results of these interactions. All intermediate events used to select the menu choice (with the keyboard and mouse) are not recorded.

Statement



Menu

Dialog Box Interaction

The Recorder recognizes interactions with standard Windows controls, such as edit boxes, check boxes, options buttons, list boxes, and combo boxes. Also, interactions with known controls of other applications are supported, such as Borland's custom controls and the controls in Visual Basic. Note: The results of dialog box interactions are recorded rather than the mouse/keyboard actions which resulted in those changes.

The Recorder also recognizes mixed dialogs - dialog boxes with a mixture of standard controls and custom controls. Many applications such as Control Panel, Corel Draw, and Lotus Ami Pro, use this technique. In this case, standard control interactions are recorded normally, while interactions with custom controls are recorded using high-level mouse and keyboard statements.

Statements



ActivateControl



SelectComboBoxItem



SelectButton



SelectListboxItem



SetCheckbox



SetEditText



SetOption

Checking the Syntax

To check the syntax of your script:

1. Choose Syntax Check from the Tools menu.

Upon encountering an error, the syntax checker stops at the line containing the error and displays an error message in a message box and in the Status Bar.

2. Correct the error.
3. Repeat Steps 1 and 2 until your script is free of syntax errors.

If your script is free of errors, the message "Syntax OK" is displayed in the Status Bar.

Working with Document Windows

DCL Editor allows you to have several scripts open for editing at the same time. Each script is contained in a document window. Document windows can be moved, resized, maximized to fill the entire screen, minimized to an icon, or restored.

The names of the currently open scripts are displayed at the bottom of the Window menu. To switch to another open script, choose the script name from the Window menu or click inside the window containing the script.

To display the document windows in a stair-step arrangement, choose the Cascade command from the Window menu.

To display the document windows in a tiled arrangement, choose Tile from the Window menu.

To neatly arrange the icons at the bottom of the **DCL Editor** window, choose Arrange Icons from the Window menu.

Getting Help

DCL provides extensive Help that describes how to use the graphical script building environment and documents each **DCL** statement, function, system variable, etc.

To display the Table of Contents for the **DCL** Help, choose Contents from the Help menu.

To search an index of **DCL** Help keywords, choose Search for Help On from the Help menu.


For information on Microsoft Windows Help, choose How to Use Help from the Help menu.

Context Sensitive Help

When you request context-sensitive Help, **DCL** displays a particular Help topic, based on your current activity. Context-sensitive Help is provided for each **DCL** language element, menu item, Toolbar button, and dialog box.

For Help on a particular **DCL** language element (statement, function, or system variable) enter the language element in the document and press F1.


For Help on a menu item (command):

- 1 Press Shift + F1 or choose the  button from the Toolbar.

The cursor changes to indicate that you are requesting context-sensitive Help.

- 2 Choose the menu and command you want Help on.

For Help on a Toolbar button:

- 1 Press Shift + F1 or choose the  button from the Toolbar.

The cursor changes to indicate that you are requesting context-sensitive Help.

- 2 Choose the Toolbar button you want Help on.

For Help on the current dialog box, choose the Help button.

Text Editing Keys

You can use the following keys when exiting scripts:

Key	Description
Backspace	Delete the selection or delete the character preceding the cursor
Tab	Insert a tab character
Enter	Insert a new line, breaking the current line
Ctrl+Insert, Ctrl+C	Copy
Shift+Insert, Ctrl+V	Paste
Shift+Delete, Ctrl+X	Cut
Insert	Toggle between insert and overwrite typing modes
Delete	Delete the selection or delete the character following the cursor
Up	Move the cursor up one line
Down	Move the cursor down one line
Left	Move the cursor left one character position
Right	Move the cursor right one character position
PgUp	Move the cursor up by one windowful
PgDn	Move the cursor down by one windowful
Ctrl+PgUp	Scroll the window left by one windowful
Ctrl+PgDn	Scroll the window right by one windowful
Ctrl+Left	Move the cursor left by one word
Ctrl+Right	Move the cursor right by one word
Home	Move the cursor to the start of the line
End	Move the cursor after the last character on the line
Ctrl+Home	Move the cursor to the first character in the macro
Ctrl+End	Move the cursor after the last character in the macro
Shift+Cursor Move	Drag the selection as the cursor moves
F1	Context-sensitive help for current command in editing window
Shift+F1	Context-sensitive help for current mouse cursor position

Using the Debugger



[Starting the Debugger](#)



[Running a Script](#)



[Tracing Through a Script](#)



[Setting Breakpoints](#)



[Using Watch Variables](#)



[Exiting the Debugger](#)



[Editing Text](#)



[Selecting Text](#)



[Finding Text](#)



[Replacing Text](#)



[Using the Toolbar](#)



[Editing Dialogs](#)



[Getting Help](#)

Keys



[Shortcut Keys](#)

Commands



[Debugger Commands](#)

Starting the Debugger

Before starting the **Debugger**, use the **DCL Editor Tools/Syntax Check** command to ensure that your script is free of syntax errors. The **Debugger** will not run or trace through a script with syntax errors.

To start the **Debugger**, choose the **DCL Editor Tools/Debugger** command.

Running a Script

You can run your script at full speed to reach the point where you want to begin debugging.

Before running the script, set a breakpoint (as described under [Setting Breakpoints](#)) where you want to begin debugging. You can set breakpoints throughout your script.

To run a script at full speed, choose the Start command from the Run menu. The script runs until it encounters the first breakpoint. (You must use the Run/Arguments command in the **DCL Editor** window to set command-line arguments, which are passed to sub main.)

At this point you can trace through the script as described under [Tracing Through a Script](#) or continue at full speed to the next breakpoint by choosing the Continue command from the Run menu.

To pause the execution of the script, choose the Break command from the Run menu.

To stop the script, choose the End command from the Run menu.

Tracing Through a Script

Tracing means executing your script one line at a time. You can trace through your script from the beginning or run the script at full speed to the point where you want to begin debugging (as described under [Running a Script](#)). While you are tracing through a script, the current line (called the instruction pointer) is highlighted.

To execute the current line and move the instruction pointer to the next line, choose the Single Step command from the Debug menu.

If you want to skip over a user-defined function or subroutine, choose the Procedure Step command. This command is the same as the Single Step command, except that it does not trace into user-defined subroutines or functions.

To reposition the instruction pointer, move the cursor to the line you want to make current and choose the Set Next Statement command from the Debug menu.

Setting Breakpoints

To set a breakpoint, position the insertion point on the line where you want to set the break point and choose Toggle Breakpoint from the Debug menu. Do not set a breakpoint on a line that contains no code.

To toggle a breakpoint off, position the insertion point on the line where the breakpoint has been set and choose Toggle Breakpoint from the Debug menu.

To remove all breakpoints, choose Clear All Breakpoints from the Debug menu.

You can set up to 255 breakpoints in a script.

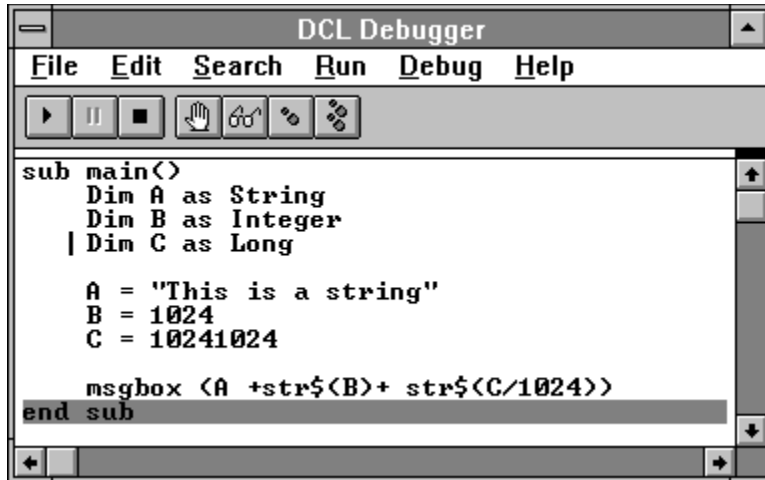
All breakpoints are removed when you exit the **Debugger**.

Using Watch Variables

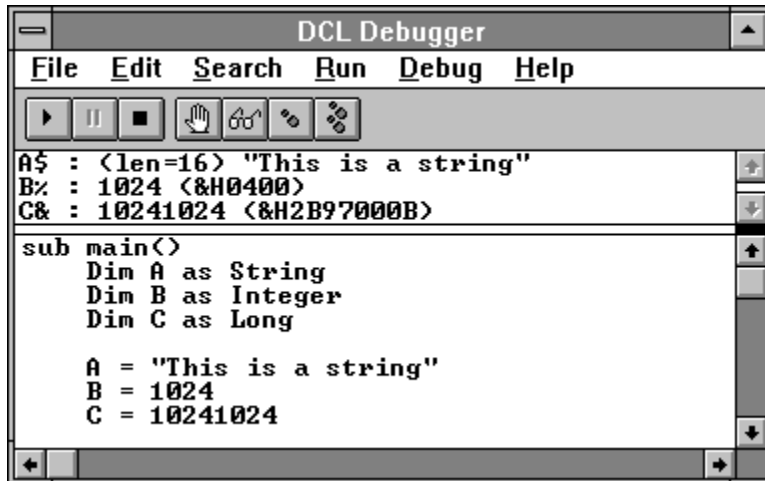
Watch variables appear in the watch pane of the **Debugger** window. As you step through your script, the current value of the variable and other information are displayed in the watch pane. If the variable is not currently in scope, the message <Not in Context> appears.

Opening the Watch Pane

When you start the **DCL Debugger** the watch pane is not open. The splitter (pictured in the following window) for the closed pane appears just below the Toolbar. This is a window with the watch pane closed.



To open the watch pane (pictured in the following window), click inside the splitter, hold the mouse button down and drag the pane to the desired size. This is a window with the watch pane open.

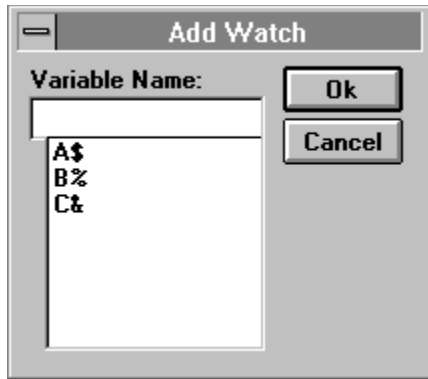


Adding a Watch Variable

For a variable to appear in the watch pane, you must explicitly add it. To add a watch variable:

- 1 Choose the Add Watch command from the Debug menu.

The Add Watch dialog box is displayed.



- 2 Enter the name of the variable you want to watch or select it from the list box. (The variable will not show up in the dialog box until you have begun to step through the script.)
- 3 Choose OK.

Once added the variable appears in the watch pane.

Deleting a Watch Variable

To delete a variable from the watch pane:

- 1 Select the variable in the watch pane.
- 2 Choose the Delete Watch command from the Debug menu or press the Del key.

Exiting the Debugger

When you have finished debugging your script, choose Exit & Update from the File menu. Your script will be updated with any changes you made during the debugging process.

If you do **not** want to update your script, choose Exit from the File menu. You will be asked whether to update your script before the **Debugger** is closed. Answer No.

Editing Text

The **Debugger** provides the basic text editing functions.

Use the Cut command on the Edit menu to remove a block of text you have selected and store it in the Windows clipboard.

Use the Copy command to copy the selected text to the clipboard.

Use the Paste command to insert the text currently in the clipboard at the insertion point.

Use the Clear command to remove the block of text you have selected. It is not written to the clipboard. You cannot undo the Clear command.

Related Topic:



[Shortcut Keys](#)



[Selecting Text](#)

Selecting Text

To Select a block of text for an editing operation:

- 1 Position the insertion point at the beginning of the text to be selected and click the left mouse button.
- 2 Move the insertion point to the end of the block to be selected.
- 3 Hold down the Shift key and press the left mouse button.

OR

- 1 Move the insertion point to the beginning of the text to be selected.
- 2 Hold the Shift key.
- 3 Move the cursor to the end of the text to be selected.

Related Topic:



[Editing Text](#)

Finding Text

To find a string of text in a script:

1. Choose the Find command from the Search menu.

The Find dialog box is displayed.



2. Enter the text string to search for in the Find What text box.
3. Select the Match Case check box if you want the search to differentiate between upper- and lowercase letters.
4. Specify the Direction of the search--up or down.
5. Choose the Find Next button to begin the search.

The first occurrence of the string after the insertion point is highlighted.

6. To continue searching for subsequent occurrences of the string, choose Find Next.

The next occurrence of the string is highlighted.

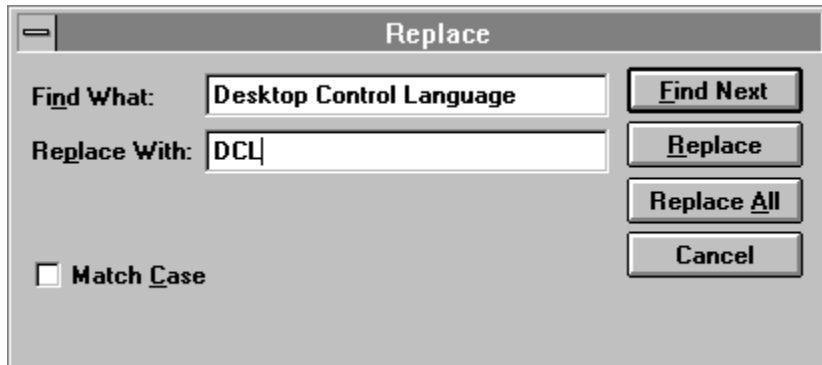
Note: You can also continue the search by closing the dialog box (by choosing Cancel) and choosing Find Next from the Search menu.

Replacing Text

To replace one text string with another:

1. Choose the Replace command from the Search menu.

The Replace dialog box is displayed.



2. Enter the text string to search for in the Find What text box.
3. Enter the replacement text in the Replace With box.
4. Select the Match Case check box if you want the search-and-replace operation to differentiate between upper- and lowercase letters.
5. Choose the Find Next button to begin the search.

The first occurrence of the string after the insertion point is highlighted.

6. If you want to substitute the replacement text for the string, choose the Replace button. (The search for the next matching string continues after the replacement is made.)

OR

If you want to go on to the next occurrence of the string without making the text change, choose the Find Next button.

OR

If you want to change all occurrences of the string to the replacement text, choose the Replace All button. (To avoid accidental changes, we recommend that you use this feature with caution.)

Using the Toolbar

The Toolbar is a row of buttons representing frequently used **Debugger** commands. Instead of using the menus, you can choose a Toolbar button to execute a command.

The Toolbar looks like this. Click any button you want information about.



Editing Dialogs

You can use the **Dialog Editor** to insert a new dialog box template in the script you are debugging or update an existing dialog box template from your script.

To create a new dialog box template to insert in your script:

1. Choose the Insert New Dialog command from the Edit menu.
2. Use the **Dialog Editor** to create the dialog box template.
3. Use the **Dialog Editor** Exit & Update command to exit the **Dialog Editor** and insert the template in the script.

To update a dialog box template from your script:

1. Select the text containing the dialog box template.
2. Choose the Edit Dialog command from the Edit menu.
3. Use the **Dialog Editor** to modify the dialog box template.
4. Use the **Dialog Editor** Exit & Update command to exit the **Dialog Editor** and replace the old template in the script.

Related Topic:



[Using the Dialog Editor](#)

Getting Help

DCL provides extensive Help that describes how to use the **Debugger**.

To display the Table of Contents for the **Debugger** Help, choose Contents from the Help menu.

To search an index of **DCL** Help keywords, choose Search for Help On from the Help menu.

For information on Microsoft Windows Help, choose How to Use Help from the Help menu.

Context Sensitive Help

When you request context-sensitive Help, **DCL** displays a particular Help topic, based on your current activity.

For Help on a particular **DCL** language element (statement, function, or system variable) enter the language element in the document and press F1.

For help on a menu item or Toolbar button:

1. Press Shift + F1.

The cursor changes to indicate that you are requesting context-sensitive Help.

2. Choose the menu and command, or Toolbar button you want Help on.

Shortcut Keys

You can use the following keys when debugging scripts:

Key	Description
Esc	Closes the Debugger
Ctrl+Insert or Ctrl+C	Copy
Shift+Insert or Ctrl+V	Paste
Shift+Delete or Ctrl+X	Cut
Del	Clear
	OR
	Deletes the current Watch variable if the Watch window is active
F3	Find Next
Shift+F8	Procedure Step
F8	Single Step
F9	Toggle Breakpoint on or off
F6	Moves between the Watch window and the editing workspace if the Watch window is open
F1	Context-sensitive help for current command in editing window
Shift+F1	Context-sensitive help for menu items and Toolbar buttons

Select the operating system in which the executable is to run--Windows or DOS.

This is the currently selected printer.

Use the Print Range buttons to indicate whether you want to print the entire script or a range of pages.

Use the From and To boxes to enter the beginning and ending pages if you selected Pages as the Print Range.

Select the desired print quality. The available choices depend on the type of printer currently selected.

Enter the number of copies you want to print.

If you are printing more than one copy, check this box on to collate the copies.

Choose the Setup button to change the current printer or modify printing options.

Select the default printer or another installed printer. The list box displays all installed printers.

Select the paper orientation--portrait or landscape.

Select the paper size. The choices depend upon the type of printer.

Select the source of the paper being fed into the printer. The choices depend upon the type of printer.

Choose the Options button to change printer-specific settings.

Enter the string of text to search for.

Check the Match Case box on to differentiate between upper- and lowercase letters in the search.

Indicate whether to search up (toward the beginning of the file) or down (toward the end of the file).

Choose the Find Next button to search for the next occurrence of the string.

Enter the replacement text.

Choose the Replace button to substitute the replacement text for the highlighted text string.

Choose the Replace All button to replace all occurrences of the string, beginning at the insertion point.

Enter the command-line arguments to be passed to the script's `main` subroutine.

Check this box on to include descriptive comments in the code generated by the Recorder.

Check this box on to consolidate multiple events into one High-Level Statement where possible. (For example, selecting a menu is recorded as a menu command rather than a series of mouse commands.)

Check this box on to capture events resulting from keyboard activity.

Check this box on to capture events resulting from mouse activity.

If you are going to capture mouse activity, click one of the Relative... buttons to indicate whether the coordinate system is relative to the window or screen.

Specify the key combination to stop the Recorder. Click on Ctrl or Alt then select the key from the Key box.

Opens a window to create a new script.

Related Topic:



[Creating a New Script](#)

Opens an existing file.

Related Topic:



[Opening an Existing File](#)

Saves the current script.

Related Topics:



[Saving a New Script](#)



[Saving an Existing File](#)



[Saving an Existing File Under a New Name](#)

Saves the current script as an executable file.

Related Topic:



[Making an Executable from the Current Script](#)

Removes the selected text from the current script and stores it in the clipboard.

Related Topic:



[Editing Text](#)

Copies the selected text from the current script to the clipboard.

Related Topic:



[Editing Text](#)

Pastes the contents of the clipboard at the insertion point.

Related Topic:



[Editing Text](#)

Starts the current script.

Related Topic:



[Running a Script](#)

Stops the script that is currently running.

Related Topic:



[Running a Script](#)

Starts the **DCL Debugger**.

Related Topic:



[Using the Debugger](#)

Starts the **DCL Dialog Editor**.

Related Topic:



[Using the Dialog Editor](#)

Begins capturing Windows events to be translated into **DCL** statements and inserted in a script.

Related Topic:



[Recording a Macro](#)

Checks the current script for syntax errors.

Related Topic:



[Checking the Syntax](#)

Prints all or part of the current script.

Related Topic:



[Printing a Script](#)

Displays the Table of Contents for **DCL** Help.

Related Topic:



[Getting Help](#)

Displays the context-sensitive help cursor. Then choose a menu command or Toolbar button for help specific to the command or button.

Related Topic:



[Getting Help](#)

Pause recording.

End recording.

Record.

Starts the script.

Related Topic:



[Running a Script](#)

Pauses the running script.

Related Topic:



[Running a Script](#)

Stops the running script.

Related Topic:



[Running a Script](#)

Sets or removes a breakpoint at the current line.

Related Topic:



[Setting Breakpoints](#)

Adds a watch variable.

Related Topic:



[Using Watch Variables](#)

Executes the current line of the script and moves the instruction pointer to the next line to be executed.

Related Topic:



[Tracing Through a Script](#)

Executes the current line of the script and moves the instruction pointer to the next line to be executed.

Related Topic:



[Tracing Through a Script](#)

Enter the name of a variable to watch or select the variable from the list.

DCL Editor Commands

File Menu Commands



New



Open



Close



Save



Save As



Print



Print Setup



Make Exe



Recent File



Exit

Edit Menu Commands



Undo



Cut



Copy



Paste



Select All



Find



Replace



Move Text Left



Move Text Right

View Menu Commands



Toolbar



Status Bar

Run Menu Commands



Start Script



End Script



Arguments

Tools Menu Commands



Debugger



Dialog Editor



Recorder



Syntax Check

Window Menu Commands



Cascade



Tile



Arrange Icons



Current Windows

Help Menu Commands



Help Contents



Search for Help On



How to Use Help



About DCL

Shift + F1 Command



Context Sensitive Help

Keys



Text Editing Keys

New

Use the New command to open a window to create a new script.

Related Topic:



[Creating a New Script](#)

Open

Use the Open command to open an existing file.

Related Topic:



[Opening an Existing File](#)

Close

Use the Close command to close the current script.

Save

Use the Save command to save the current new or existing script.

Related Topics:



[Saving a New Script](#)



[Saving an Existing File](#)

Save As

Use the Save As command to save an existing script under a new file name.

Related Topic:



[Saving an Existing File Under a New Name](#)

Print

Use the Print command to print all or part of the current script.

Related Topic:



[Printing a Script](#)

Print Setup

Use the Print Setup command to select the printer and set printing options.

Related Topic:



[Setting Up for Printing](#)

Make Exe

Use the Make Exe command to save the current script as an executable file.

Related Topic:



[Making an Executable from the Current Script](#)

Recent File

To open one of the last four files you saved, click on the file listed in the File menu.

Related Topic:



[Opening a Recent File](#)

Exit

Use the Exit command to exit the **DCL Editor**.

Related Topic:



[Exiting DCL Editor](#)

Undo

Use the Undo command to reverse the latest editing command.

Related Topic:



Editing Text

Cut

Use the Cut command to remove the selected text from the current script and store it in the clipboard.

Related Topic:



[Editing Text](#)

Copy

Use the Copy command to copy the selected text from the current script to the Windows clipboard.

Related Topic:



[Editing Text](#)

Paste

Use the Paste command to paste the contents of the clipboard at the insertion point.

Related Topic:



Editing Text

Select All

Use the Select All command to select all of the text in the script for an editing operation.

Related Topics:



[Editing Text](#)



[Selecting Text](#)

Find

Use the Find command to search for a text string in the current script.

Related Topic:



Finding Text

Replace

Use the Replace command to replace one text string with another in the current script.

Related Topic:



[Replacing Text](#)

Move Text Left

Use the Move Text Left command to shift the selected text one tab stop left.

Related Topics:



[Editing Text](#)



[Selecting Text](#)

Move Text Right

Use the Move Text Right command to shift the selected text one tab stop right.

Related Topics:



[Editing Text](#)



[Selecting Text](#)

Toolbar

Use the Toolbar command to display the Toolbar in the **DCL Editor** window.

Related Topic:



[Using the Toolbar](#)

Status Bar

Use the Status Bar command to display the Status Bar at the bottom of the **DCL Editor** window.

Related Topic:



[Using the Status Bar](#)

Start Script

Use the Start Script command to run the current script.

Related Topic:



[Running a Script](#)

End Script

Use the End Script command to stop the script that is currently running.

Related Topic:



[Running a Script](#)

Arguments

Use the Arguments command to enter command-line arguments to be passed to the `main` subroutine of the script you are going to run.

Related Topic:



[Running a Script](#)

Debugger

Use the Debugger command to start the **DCL Debugger**.

Related Topic:



[Using the Debugger](#)

Dialog Editor

Use the Dialog Editor command to start the **DCL Dialog Editor**.

Related Topic:



[Using the Dialog Editor](#)

Recorder

Use the Recorder command to begin capturing Windows events and that can be translated them into **DCL** statements and inserted in a script.

Related Topic:



[Recording a Macro](#)

Syntax Check

Use the Syntax Check command to check the current script for syntax errors.

Related Topic:



[Checking the Syntax](#)

Cascade

Use the Cascade command to display the document windows in a stair-step arrangement.

Related Topic:



[Working with Document Windows](#)

Tile

Use the Tile command to display the document windows in a tiled arrangement.

Related Topic:



[Working with Document Windows](#)

Arrange Icons

Use the Arrange Icons command to neatly arrange the icons at the bottom of the **DCL Editor** window.

Related Topic:



[Working with Document Windows](#)

Current Windows

The names of the currently open scripts are displayed at the bottom of the Window menu. To switch to another open script, choose the script name from the Window menu.

Related Topic:



[Working with Document Windows](#)

Help Contents

Use the Help Contents command to display the Table of Contents for **DCL** Help.

Related Topic:



[Getting Help](#)

Search for Help On

Use the Search for Help On command to search an index of **DCL** Help keywords.

Related Topic:



[Getting Help](#)

How to Use Help

Use the How to Use Help command for information on how the use Microsoft Windows Help.

Related Topic:



[Getting Help](#)

Context Sensitive Help

Choose the {blc help_but.bmp} button from the Toolbar or press Shift + F1 to display the context-sensitive help cursor then choose a menu command or Toolbar button for help specific to the command or button.

Related Topic:



[Getting Help](#)

About DCL Editor

Use the About DCL command to display the version number and memory information.

Related Topic:



[Getting Help](#)

Document Window

This is the current document window. Use this window to create a new script or edit an existing one.

Related Topic:



[Creating a New Script](#)

File Open dialog box

Use the File Open dialog box to open an existing file.



Click on any item you want information about.

Related Topic:



[Opening an Existing File](#)

File Save As dialog box

Use the Save As dialog box to save a new script or to save an existing file under a new name.



Click on any item you want information about.

Related Topics:



[Saving a New Script](#)



[Saving an Existing File Under a New Name](#)

Print dialog box

Use the Print dialog box to all or part of the current script.



Click on any item you want information about.

Related Topic:



[Printing a Script](#)

Print Setup dialog box

Use the Print Setup dialog box to select the printer and set printing options.



Click on any item you want information about.

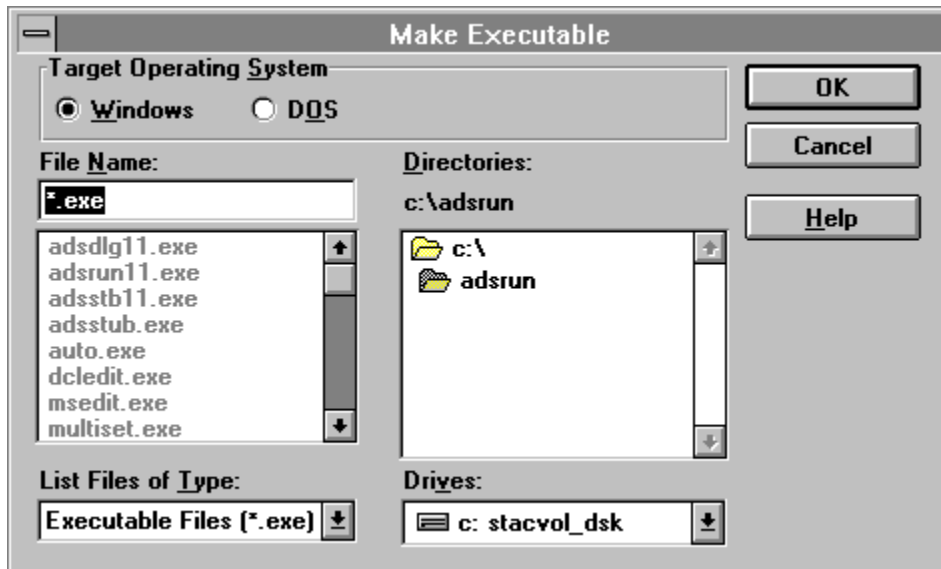
Related Topic:



[Setting Up for Printing](#)

Make Executable dialog box

Use the Make Executable dialog box to save the current script as an executable file.



Click on any item you want information about.

Related Topic:



[Making an Executable from the Current Script](#)

Find dialog box

Use the Find dialog box to search for a text string in the current script.



Click on any item you want information about.

Related Topic:



Finding Text

Replace dialog box



Find dialog box

Use the Replace command to replace one text string with another in the current script.



Click on any item you want information about.

Related Topic:



Replacing Text

Set Arguments dialog box

Use the Arguments dialog box to enter command-line arguments to be passed to the `main` subroutine of the script you are going to run.



Click on any item you want information about.

Related Topic:



[Running a Script](#)

Script Recorder Options dialog box

Use the Script Recorder Options dialog box to specify which script recording options you want to use and begin recording.



Click on any item you want information about.

Related Topic:



[Recording a Macro](#)

Insert Recording dialog box

Use the Insert Recording dialog box to insert the recorded events into the current script.



Click on any item you want information about.

Related Topic:



[Recording a Macro](#)

Title Bar

The title bar is located along the top of a window. It contains the name of the application and document.

To move the window, drag the title bar. Note: You can also move dialog boxes by dragging their title bars.

A title bar may contain the following elements:



Application Control-menu button



Document Control-menu button



Maximize button



Minimize button



Name of the application



Name of the document



Restore button

Scroll Bars

Displayed at the right and bottom edges of the document window. The scroll boxes inside the scroll bars indicate your vertical and horizontal location in the document. You can use the mouse to scroll to other parts of the document.

Size command

Use this command to display a four-headed arrow so you can size the active window with the arrow keys.

After the pointer changes to the four-headed arrow:

1. Press one of the DIRECTION keys (left, right, up, or down arrow key) to move the pointer to the border you want to move.
2. Press a DIRECTION key to move the border.
3. Press ENTER when the window is the size you want.

Note: This command is unavailable if you maximize the window.

Mouse Shortcut: Drag the size bars at the corners or edges of the window.

Move command


Use this command to display a four-headed arrow so you can move the active window or dialog box with the arrow keys.

Note: This command is unavailable if you maximize the window.

Minimize command

Use this command to reduce the active window to an icon.

Shortcuts

Mouse: Click the minimize icon  on the title bar.

Maximize command

Use this command to enlarge the active window to fill the available space.

Shortcut

Mouse: Click the maximize icon  on the title bar; or double-click the title bar.

Next Window

Use this command to switch to the next open document window. **DCL Editor** determines which window is next according to the order in which you opened the windows.

Shortcut Keys: CTRL+F6

Previous Window

Use this command to switch to the previous open document window. **DCL Editor** determines which window is previous according to the order in which you opened the windows.

Shortcut Keys: SHIFT+CTRL+F6

Close command

Use this command to close the active window or dialog box.

Double-clicking a Control-menu box is the same as choosing the Close command.

Shortcuts Keys:

CTRL+F4 closes a document window

ALT+F4 closes the application window or dialog box

Restore command

Use this command to return the active window to its size and position before you chose the Maximize or Minimize command.

Switch to command

Use this command to display a list of all open applications. Use this "Task List" to switch to or close an application on the list.

Shortcut: Keys: CTRL+ESC

Dialog Box Options

When you choose the Switch To command, you will be presented with a dialog box with the following options:

Task List: Select the application you want to switch to or close.

Switch To: Makes the selected application active.

End Task: Closes the selected application.

Cancel: Closes the Task List box.

Cascade: Arranges open applications so they overlap and you can see each title bar. This option does not affect applications reduced to icons.

Tile: Arranges open applications into windows that do not overlap. This option does not affect applications reduced to icons.

Arrange Icons: Arranges the icons of all minimized applications across the bottom of the screen.

No Help Available

No help is available for this area of the window.

Debugger Commands

File Menu Commands



Exit & Update



Exit

Edit Menu Commands



Cut



Copy



Paste



Clear



Insert New Dialog



Edit Dialog

Search Menu Commands



Find



Find Next



Replace

Run Menu Commands



Start



End



Break

Debug Menu Commands



Add Watch



Delete Watch



Single Step



Procedure Step



Toggle Breakpoint



Clear All Breakpoints



Set Next Statement

Help Menu Commands



Help Contents



Search for Help On



How to Use Help



About Debugger

Shift + F1 Command



Context Sensitive Help

Keys



Shortcut Keys

Exit & Update

When you have finished debugging your script, choose Exit & Update from the File menu. Your script will be updated with any changes you made during the debugging process.

Related Topic:



[Exiting the Debugger](#)

Exit

If you want to exit the **Debugger** without updating your script, choose Exit from the File menu. You will be asked whether to update your script before the **Debugger** is closed. Answer No.

Related Topic:



[Exiting the Debugger](#)

Cut

Use the Cut command to remove the selected text from the script and store it in the clipboard.

Related Topic:



Editing Text

Copy

Use the Copy command to copy the selected text from the script to the Windows clipboard.

Related Topic:



[Editing Text](#)

Paste

Use the Paste command to paste the contents of the clipboard at the insertion point.

Related Topic:



[Editing Text](#)

Clear

Use the Clear command to remove the selected text from the editing window.

Related Topic:



Editing Text

Insert New Dialog

Use the Insert New Dialog command to start the **Dialog Editor** and create a new dialog box template.

Related Topic:



[Editing Dialogs](#)

Edit Dialog

Use the Edit Dialog command to start the **Dialog Editor** and graphically edit the selected dialog box template.

Related Topic:



[Editing Dialogs](#)

Find

Use the Find command to search for a text string.

Related Topic:



Finding Text

Find Next

Use the Find Next command to search for the next occurrence of a selected text string.

Related Topic:



[Finding Text](#)

Replace

Use the Replace command to replace one text string with another.

Related Topic:



Replacing Text

Start/Continue

Use the Start command to run the script. Use the Continue command to resume running the script.

Related Topic:



[Running a Script](#)

End

Use the Stop Script command to stop the running script.

Related Topic:



[Running a Script](#)

Break

Use the Break command to pause the running script.

Related Topic:



[Running a Script](#)

Add Watch

Use the Add Watch command to set up a watch variable.

Related Topic:



[Using Watch Variables](#)

Delete Watch

Use the Delete Watch command to remove a watch variable.

Related Topic:



[Using Watch Variables](#)

Single Step

Use the Single Step command to execute the current line of the script and move the instruction pointer to the next line to be executed.

Related Topic:



[Tracing Through a Script](#)

Procedure Step

Use the Procedure Step command to execute the current line of the script and move the instruction pointer to the next line to be executed. The Procedure Step command differs from the Single Step command in that it does not trace through user-defined procedures line by line. It does, however, execute the procedure.

Related Topic:



[Tracing Through a Script](#)

Toggle Breakpoint

Use the Toggle Breakpoint command to set a breakpoint (or remove the breakpoint) at the current line.

Related Topic:



[Setting Breakpoints](#)

Clear All Breakpoints

Use the Clear All Breakpoints command to remove all breakpoints from the script.

Related Topic:



[Setting Breakpoints](#)

Set Next Statement

Use the Set Next Statement command to reposition the instruction pointer where desired in the current function or procedure.

Related Topic:



[Tracing Through a Script](#)

Help Contents

Use the Help Contents command to display the Table of Contents for **Debugger** Help.

Related Topic:



[Getting Help](#)

Search for Help On

Use the Search for Help On command to search an index of **DCL** Help keywords.

Related Topic:



[Getting Help](#)

How to Use Help

Use the How to Use Help command for information on how the use Microsoft Windows Help.

Related Topic:



Getting Help

About Debugger

Use the About Debugger command to display the version number and memory information.

Related Topic:



[Getting Help](#)

Context Sensitive Help

Press Shift + F1 to display the context-sensitive help cursor then choose a menu command or Toolbar button for help specific to the command or button.

Related Topic:



[Getting Help](#)

Choose the Cancel button to discard changes and exit the current dialog box.

The Directories box lists the directory tree for the selected Drive. Scroll through the list and double-click on the desired directory.

The Drives box lists the drives that are available for you to use.

Enter the file name in the File Name box or select the file name from the list of files in the current directory.

Choose the Help button for information about the current dialog box and the procedures it is used in.

Choose the OK button to save your changes and exit the current dialog box.

The Save File As Type box lists the possible file types this file can be saved as.

Use the List Files of Type box to change the type of files displayed in the File Name list box.

