

#1 S2 **Desktop Control Language Characteristics**

{bml j_bullet.bmp}DCL General Characteristics

{bml j_bullet.bmp}DCL Script Execution

{bml j_bullet.bmp}DCL Structures

{bml j_bullet.bmp}DCL Compiler

{bml j_bullet.bmp}DCL Variables

{bml j_bullet.bmp}DCL Expression Evaluation

{bml j_bullet.bmp}DCL Subroutines & Functions

{bml j_bullet.bmp}DCL Error Handling

{bml j_bullet.bmp}DCL Arrays

{bml j_bullet.bmp}DCL Data Types

{bml j_bullet.bmp}DCL Operator Precedence

{bml j_bullet.bmp}DCL Comments

{bml j_bullet.bmp}DCL Constants

{bml j_bullet.bmp}DCL Limitations

¹ LanguageCharacteristics

² DCL Characteristics

#3 S4 K5 +6 DCL General Characteristics

- {bmc bullet.bmp} DCL is compiled, not interpreted.
- {bmc bullet.bmp} A DCL script consists of any number of user-defined functions, user-defined subroutines, and external DLL declarations.
- {bmc bullet.bmp} The compiler and runtime operate under Windows 3.1 and Windows for Workgroups 3.11 or later.
- {bmc bullet.bmp} A DCL script is an ASCII stream of text with a NULL terminator.
- {bmc bullet.bmp} 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.
- {bmc bullet.bmp} Line numbers are not supported. Labels are used instead.
- {bmc bullet.bmp} Quotes can be embedded within constant string expressions using two consecutive quotes, such as "John said, ""hello"", to Jane".
- {bmc bullet.bmp} 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"`.
- {bmc bullet.bmp} The runtime is reentrant. Thus, two DCL scripts can execute concurrently.

³ LANGCHAR_DCL_General_Characteristics

⁴ General Characteristics

⁵ DCL, general characteristics

⁶ LangChar: 1005

#7 S8 K9 +10 **DCL Script Execution**

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

⁷ LANGCHAR_DCL_Script_Execution

⁸ Script Execution

⁹ Script execution

¹⁰ LangChar: 1010

#11 S12 K13 +14 **DCL Structures**

{bmc bullet.bmp} User-defined structures are not supported. DCL does support dialog structures.

¹¹ LANGCHAR_DCL_Structures

¹² Structures

¹³ Structures

¹⁴ LangChar: 1020

#15 S16 K17 +18 **DCL Compiler**

{bmc bullet.bmp} 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.

{bmc bullet.bmp} Compiler metacommands are not supported.

{bmc bullet.bmp} The compiler is reentrant. Thus, two DCL scripts can be compiled simultaneously.

¹⁵ LANGCHAR_DCL_Compiler

¹⁶ Compiler

¹⁷ Compiler

¹⁸ LangChar: 1025

DCL Variables

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

{bmc bullet.bmp} Variables declared (explicitly or implicitly) within a subroutine are local to that subroutine.

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

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

{bmc bullet.bmp} 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.

{bmc bullet.bmp} Loss of precision is not a runtime error.

{bmc bullet.bmp} The declaration modifiers `STATIC`, `GLOBAL`, and `SHARED` are not supported.

¹⁹ LANGCHAR_DCL_Variables

²⁰ Variables

²¹ Variables

²² LangChar: 1030

#23 S24 K25 +26

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.

²³ LANGCHAR_DCL_Expression_Evaluation

²⁴ Expression Evaluation

²⁵ Expression evaluation

²⁶ LangChar: 1035

DCL Subroutines & Functions

{bmc bullet.bmp} Subroutines and functions are available to any other subroutine or function within the same script.

{bmc bullet.bmp} 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
```

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

{bmc bullet.bmp} Recursive statements and functions are supported.

²⁷ LANGCHAR_DCL_Subroutines_Functions

²⁸ Subroutines & Functions

²⁹ Subroutines; Functions

³⁰ LangChar: 1040

DCL Error Handling

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

{bmc bullet.bmp} `ON ERROR` traps are valid only during the execution of the current subroutine or function.

{bmc bullet.bmp} Error traps are saved and restored within a user-defined subroutine or function.

DCL errors fall into three categories:

{bmc bullet.bmp} Trappable error numbers are between 10 and 1000.

{bmc bullet.bmp} Non-trappable error numbers are between 0 and 10. Internal errors and out of memory errors are all non-trappable.

{bmc bullet.bmp} 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.

³¹ LANGCHAR_DCL_Error_Handling

³² Error Handling

³³ Error handling

³⁴ LangChar: 1045

#35 S36 K37 +38

DCL Arrays

{bmc bullet.bmp}
statements.

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

{bmc bullet.bmp}

Arrays can be declared to contain any fundamental data type.

{bmc bullet.bmp}

Arrays can have up to 60 dimensions.

{bmc bullet.bmp}
statement.

Array dimensions and size can be changed dynamically using the REDIM

{bmc bullet.bmp} Arrays are always
passed by reference.

³⁵ LANGCHAR_DCL_Arrays

³⁶ Arrays

³⁷ Arrays

³⁸ LangChar: 1050

#39 #40 #41 #42

DCL Data Types

integer

Type Declarator

%

Significant Digits

4

Size

2 bytes (16 bits)

Range

 $-32768 \leq X \leq 32767$

long

Type Declarator

&

Significant Digits

9

Size

4 bytes (32 bits)

Range

 $-2147483648 \leq X \leq 2147483647$

string

Type Declarator

\$

Significant Digits

N/A

³⁹ LANGCHAR_DCL_Data_Types⁴⁰ Data Types⁴¹ Data Types⁴² LangChar: 1055

X

Size

1 byte per character

Range

$0 \leq \text{LENGTH} < 32768$

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: $-3.402823\text{E}38$ to $-1.401298\text{E}-45$

Positive Values: $1.401298\text{E}-45$ to $3.402823\text{E}38$

double

Type Declarator

#

Significant Digits

15-16

Size

8 bytes (64 bits)

Range

Negative Values: $1.797693134862315\text{E}308$ to $-4.94066\text{E}-324$

Positive Values: $4.94066\text{E}-324$ to $1.797693134862315\text{E}308$

#43 S44 K45 +46 DCL Operator Precedence

Operator

Description

Precedence order

()

parentheses

Highest

^

exponentiation

-

unary minus

/, *

multiplication and division

\

integer division

mod

modulo

+, -

addition and subtraction

⁴³ LANGCHAR_DCL_Operator_Precedence

⁴⁴ Operator Precedence

⁴⁵ Operator precedence

⁴⁶ LangChar: 1060

X

=, <>, >, <, <=, >=

relational

not

logical negation

and

and

or

or

xor

exclusive or

Lowest

#47 S48 K49 +50 DCL Comments

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

{bmc bullet.bmp} All text between a single quote and the end of the line is ignored:

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

{bmc bullet.bmp} The REM statement causes the compiler to ignore the entire line:

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

{bmc bullet.bmp} 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.

⁴⁷ LANGCHAR_DCL_Comments

⁴⁸ Comments

⁴⁹ Comments

⁵⁰ LangChar: 1065

DCL Constants

{bmc bullet.bmp} 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`.

{bmc bullet.bmp} 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

⁵¹ LANGCHAR_DCL_Constants

⁵² Constants

⁵³ Constants

⁵⁴ LangChar: 1070

#55 #56 #57 #58

DCL Limitations

{bmc bullet.bmp}	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.
{bmc bullet.bmp}	Strings are limited to 32K in size (32767 bytes).
{bmc bullet.bmp}	Script source size is limited to 42K.
{bmc bullet.bmp}	Compiled code size is limited to 64K. The code segment contains the executable pcode.
{bmc bullet.bmp}	The maximum size of the symbol table (used by runtime errors and the debugger) is 64K.
{bmc bullet.bmp}	Arrays can have up to 60 dimensions.
{bmc bullet.bmp}	Variable names are limited to 40 characters.
{bmc bullet.bmp}	Labels are limited to 40 characters.
{bmc bullet.bmp}	A given subroutine or function can have up to 100 variables, including variables that are passed.
{bmc bullet.bmp}	The stack size for the runtime is 2048 bytes.
{bmc bullet.bmp}	The number of open DDE channels is unlimited (limited only by available memory and system resources).
{bmc bullet.bmp}	The number of open files is limited to 255, or the operating system limit, whichever is less.
{bmc bullet.bmp}	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.)
{bmc bullet.bmp}	Number of nesting levels is limited by compiler memory.
{bmc bullet.bmp}	Text file input/output buffer size is 512 bytes.
{bmc bullet.bmp}	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).
{bmc bullet.bmp}	Each <code>gosub</code> requires 2 bytes of the DCL runtime stack.

⁵⁵ LANGCHAR_DCL_Limitations

⁵⁶ DCL Limitations

⁵⁷ Limitations

⁵⁸ LangChar: 1075