PC-DMIS Basic

Language Reference

By Brown and Sharpe Mfg. Co.

This manual was produced using *Doc-To-Help*®, by WexTech Systems, Inc.

Contents

Cypress Enable Scripting Language Elements 17

Basic Script Editor

Introduction

The UTILITIES | SCRIPTING | BASIC SCRIPT EDITOR menu option opens the *Basic Script Editor*. The *Basic Script Editor* can be used to create and edit basic scripts that can be used in Basic Script objects during execution or from the *Basic Script toolbar*. The *Basic Script Editor* consists of the following menus:

1) File menu
2) Edit menu
3) View menu
4) Run menu
5) Help menu

File Menu

The Basic Script Editor's FILE menu gives you the following commands and options:

New

The FILE | NEW menu option opens a new *Basic Script Editor* in which you can write a new script.

Open

The FILE | OPEN menu option allows you to navigate to and open an existing script. In order for files to appear in the Basic Script Editor, files must be of file type .BAS.

Save

The FILE | SAVE menu option allows you to save a script. With a new script, the first time this option is selected, the *Save As Dialog box* will appear.

Save As

The FILE | SAVE AS menu option allows you to save a new script, or an already existing script by a new file name. The *Save As Dialog box* appears, allowing you to select the file name and the directory to which you will be saving the script.

Print

The FILE | PRINT menu option allows you to print the script in the *Basic Script Editor* from your system's printer.

Print Preview

The FILE | PRINT PREVIEW menu option allows you to preview what will be sent to the printer when PRINT is selected from the *Basic Script Editor's* FILE menu.

Exit

The FILE | EXIT menu option allows you to exit out of the *Basic Script Editor* without saving any changes you have made to any open scripts. Choosing FILE | EXIT will return you the the main user interface. The menu bar will return to normal PC-DMIS functions.

Edit Menu

The EDIT menu of the *Basic Script Editor* allows you to use basic Edit functions to manipulate the text displayed in the *Basic Script Editor*.

Undo

The EDIT | UNDO menu option allows you to undo the most recent action taken in the *Basic Script Editor*.

Cut

The EDIT | CUT menu option allows you to cut selected text from the *Basic Script Editor*. Cut text is stored in the Windows clipboard to later be pasted elsewhere.

Copy

The EDIT | COPY menu option allows you to copy selected text. Copied text is stored in the Windows clipboard to later be pasted elsewhere.

Paste

The EDIT | PASTE command allows you to paste text that is stored in the Windows clipboard.

Delete

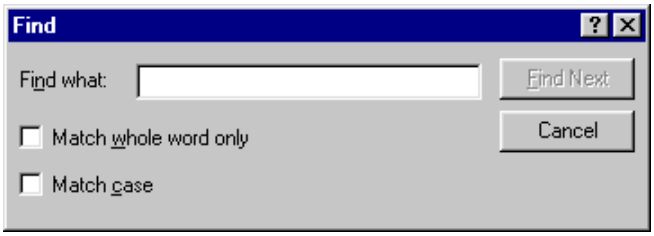
The EDIT | DELETE command allows you to delete highlighted text.

Select All

The EDIT | SELECT ALL menu option automatically selects all the text within the *Basic Script Editor*. You can then CUT, COPY, or DELETE the selected text.

Find

The EDIT | FIND menu option brings up the *Find Dialog box*.



Find Dialog box

This dialog allows you to search for a specific word, or term within the *Basic Script Editor*.

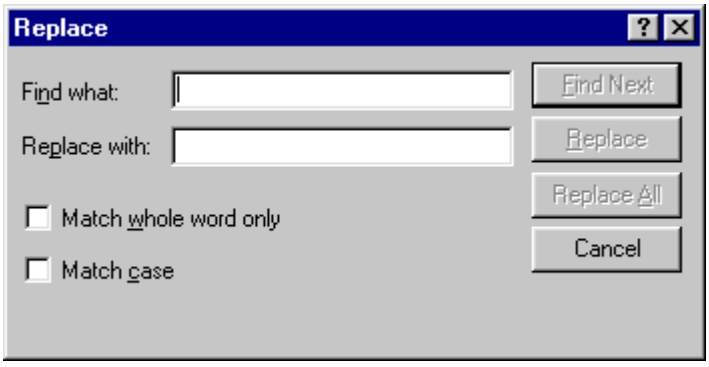
- If you choose the *Match whole word only check box* the dialog will display only those words that match the entire word.
- If you choose the *Match Case check box,* then the dialog will display only those terms that match the case (Uppercase or Lowercase) that you used in the *Find what field*.

Find Next

The EDIT | FIND NEXT will search in the *Basic Script Editor* for the next term that meets the qualifications specified in the *Find Dialog box* (See EDIT | FIND above.)

Replace

The EDIT | REPLACE menu option brings up the *Replace Dialog box*



Replace Dialog box

This dialog is an extension of the EDIT | FIND command. This allows you to search for a specific term and then replace it with the term entered in the *Replace with field.*

Find Next

The **Find Next** button searches through the *Basic Script Editor* and brings up the first instance that meets the qualifications entered in the dialog box.

Replace

The **Replace** button allows you to replace what has been found (using the **Find Next** button) with what is in the *Replace with field*.

Replace All

The **Replace All** button allows you to replace all instances in the *Basic Script Editor* that meet the search qualifications with what is in the *Replace with field*.

Cancel

The **Cancel** button closes the *Replace Dialog box*.

View Menu

The VIEW menu allows you to choose if the Basic Script Editor's Toolbar and / or Status Bar is being displayed. Select VIEW | TOOLBAR to toggle the toolbar on or off. Select VIEW | STATUS BAR to toggle the status bar on or off.

Run Menu

The RUN menu allows you to COMPILE a scriptor EXECUTE a script. Use the compile command to test the script for syntactic errors. The execute command executes the script.

Help Menu

The HELP menu allows you to access various options that aid you in using the Basic Script Editor.

Basic Help

The HELP | BASIC HELP command brings up the on-line help file created for the add on Basic Module.

Syntax Help

The HELP | SYNTAX HELP toggles the option to use the syntax help when using the Basic Script Editor. If this option is selected, a pop up scroll box appears within the *Basic Script Editor* whenever you type in a command or term used in the Basic programming language. You can use arrow keys to select the appropriate term. Once selected, hit the **TAB** key and that term will appear in the *Basic Script Editor*. Hitting the **Spacebar** displays the syntax needed for the command.

Syntax Help File

The HELP | SYNTAX HELP FILE allows you to select the syntax file used in the HELP | SYNTAX HELP command. An *Open File Dialog box* will appear. Navigate to the directory that contains PC-DMIS for Windows and select the "Pcdmis.syn" file.

Basic Script Toolbar

{bmct bstoolbr.shg}

The *Basic Editor Toolbar* supports the following functions:

New

This icon allows you to create a new basic script in the editor.

Open

This icon brings up an *Open File Dialog box* allowing you opens an existing basic script into the editor.

Save

This icon saves the current basic script. If you have not already named the current script, a *Save As Dialog box* asking for the name of the script will appear.

Print

This icon prints the current basic script.

Print Preview

This icon allows you to see the current basic script in the *Print Preview window* as it will appear when printed.

Find

This icon allows you to search for text in the current basic script.

Cut

Ths button cuts currently selected text and put text on the clipboard.

Copy

This icon copies currently selected text and put text on the clipboard.

Paste

This icon pastes text from the clipboard into the editor at the current insertion point.

Undo

This icon allows you to undo the last editing change.

Run

This icon compiles and runs the current basic script.

Note: Scripts run from the editor using the PC-DMIS basic commands can insert objects into the current part program.

Cypress Enable Scripting Language Elements

In this Section, the general elements of the Enable language are described. Enable scripts can include comments, statements, various representations of numbers, 11 variable data types including user defined types, and multiple flow of control structures. Enable is also extendable by calling external DLL's or calling functions back in the applications .exe file.

Comments

Comments are non-executed lines of code which are included for the benefit of the programmer. Comments can be included virtually anywhere in a script. Any text following an apostrophe or the word Rem is ignored by Enable. Rem and all other keywords and most names in Enable are not case sensitive

```
'                                       This whole line is a comment
rem                                     This whole line is a comment
REM                                     This whole line is a comment
Rem                                     This whole line is a comment
```

Comments can also be included on the same line as executed code:

```
MsgBox Msg                              ' Display message.
```

Everything after the apostrophe is a comment.

Statements:

In Enable there is no statement terminator. More than one statement can be put on a line if they are separated by a colon.

```
X.AddPoint( 25, 100) :  X.AddPoint(  0,  75)
```

Which is equivalent to:

```
X.AddPoint( 25, 100)
X.AddPoint(  0,  75)
```

Line Continuation Character:

The underscore is the line continuation character in Enable. There must be a space before and after the line continuation character.

```
X.AddPoint _
( 25, 100)
```

Numbers

Cypress Enable supports three representations of numbers: Decimal, Octal and Hexadecimal. Most of the numbers used in this manual are decimal or base 10 numbers. However, if you need to use Octal (base 8) or hexadecimal (base 16) numbers simply prefix the number with &O or &H respectively.

Variable and Constant Names

Variable and Constant names must begin with a letter. They can contain the letters A to Z and a to z, the underscore "_", and the digits 0 to 9. Variable and constant names must begin with a letter, be no longer than 40 characters. and cannot be reserved words. For a table of reserved words, see the Language Overview section of this manual. One exception to this rule is that object member names and property names may be reserved words.

Variable Types

Variant

As is the case with Visual Basic, when a variable is introduced in Cypress Enable, it is not necessary to declare it first (see option explicit for an exception to this rule). When a variable is used but not declared then it is implicitly declared as a **variant** data type. Variants can also be declared explicitly using "As Variant" as in Dim x As Variant. The variant data type is capable of storing numbers, strings, dates, and times. When using a variant you do not have to explicitly convert a variable from one data type to another. This data type conversion is handled automatically.

```
Sub Main
Dim x                                       'variant variable
x = 10
x = x + 8
x = "F" & x
print x       'prints F18
End Sub
```

A variant variable can readily change its type and its internal representation can be determined by using the function **VarType**. **VarType** returns a value that corresponds to the explicit data types. See VarType in A-Z Reference for return values.

When storing numbers in variant variables the data type used is always the most compact type possible. For example, if you first assign a small number to the variant it will be stored as an integer. If you then assign your variant to a number with a fractional component it will then be stored as a double.

For doing numeric operations on a variant variable it is sometimes necessary to determine if the value stored is a valid numeric, thus avoiding an error. This can be done with the **IsNumeric** function.

Variants and Concatenation

If a string and a number are concatenated the result is a string. To be sure your concatenation works regardless of the data type involved use the **&** operator. The **&** will not perform arithmetic on your numeric values it will simply concatenate them as if they were strings.

The **IsEmpty** function can be used to find out if a variant variable has been previously assigned.

Other Data Types

The twelve data types available in Cypress Enable are shown below:

Data Types

Variable	Symbol	Type Declaration	Size
Byte		Dim BVar As Byte	0 to 255
Boolean		Dim BoolVar As Boolean	True or False
String	$	Dim Str_Var As String	0 to 65,500 char
Integer	%	Dim Int_Var As Integer	2 bytes
Long	&	Dim Long_Var As Long	4 bytes
Single	!	Dim Sing_Var As Single	4 bytes
Double	#	Dim Dbl_Var As Double	8 bytes
Variant		Dim X As Any	
Currency		Dim Cvar As Currency	8 bytes
Object		Dim X As Object	4 bytes
Date		Dim D As Date	8 bytes
User Defined Types			size of each element

Scope of Variables

Cypress Enable scripts can be composed of many files and each file can have many subroutines and functions in it. Variable names can be reused even if they are contained in separate files. Variables can be local or global.

Declaration of Variables

In Cypress Enable variables are declared with the **Dim** statement. To declare a variable other than a variant the variable must be followed by **As** or appended by a type declaration character such as a % for **Integer** type.

```
Sub Main
Dim X As Integer
```

```
Dim Y As Double
Dim  Name$, Age%                          ' multiple declaration on one line Dim v
End Sub
```

Control Structures

Cypress Enable has complete process control functionality. The control structures available are **Do** loops, **While** loops, **For** loops, **Select Case**, **If Then** , and **If Then Else.** In addition, Cypress Enable has one branching statement: **GoTo**. The **Goto** Statement branches to the label specified in the **Goto** Statement.

```
Goto label1
.
.
.

label1:
```

The program execution jumps to the part of the program that begins with the label "Label1:".

Loop Structures

Do Loops

The **Do...Loop** allows you to execute a block of statements an indefinite number of times. The variations of the **Do...Loop** are **Do While**, **Do Until**, **Do Loop While**, and **Do Loop Until**.

```
Do While|Until condition
Statement(s)...
[Exit Do]
Statement(s)...
Loop

Do Until condition
Statement(s)...
Loop

Do
Statements...
Loop While condition

Do
statements...
Loop Until condition
```

Do While and **Do Until** check the condition before entering the loop, thus the block of statements inside the loop are only executed when those conditions are met. **Do Loop While** and **Do Loop Until** check the condition after having executed the block of statements thereby guaranteeing that the block of statements is executed at least once.

While Loop

The **While...Wend** loop is similar to the **Do While** loop. The condition is checked before executing the block of statements comprising the loop.

```
While condition
statements...
Wend
```

For ... Next Loop

The **For...Next** loop has a counter variable and repeats a block of statements a set number of times. The counter variable increases or decreases with each repetition through the loop. The counter default is one if the **Step** variation is not used.

```
For counter = beginning value To ending value [Step increment]
statements...
Next
```

If and Select Statements

The **If...Then** block has a single line and multiple line syntax. The condition of an **If** statement can be a comparison or an expression, but it must evaluate to True or False.

```
If condition Then Statements...          'single line syntax

If condition Then                                  'multiple line syntax
statements...
End If
```

The other variation on the **If** statement is the **If...Then...Else** statement. This statement should be used when there is different statement blocks to be executed depending on the condition. There is also the **If...Then...ElseIf...** variation, these can get quite long and cumbersome, at which time you should consider using the **Select** statement.

```
If condition Then
statements...
ElseIf condition Then
statements...
Else

End If
```

The **Select Case** statement tests the same variable for many different values. This statement tends to be easier to read, understand and follow and should be used in place of a complicated **If...Then...ElseIf** statement.

```
Select Case variable to test
Case 1
statements...
Case 2
```

```
statements...
Case 3
statements...
Case Else
statements...
End Select
```

See Language Reference A - Z for exact syntax and code examples.

Subroutines and Functions

Naming conventions

Subroutine and Function names can contain the letters A to Z and a to z, the underscore "_" and digits 0 to 9. The only limitation is that subroutine and function names must begin with a letter, be no longer than 40 characters, and not be reserved words. For a list of reserved words, see the table of reserved words in the Language Overview section of this manual.

Cypress Enable allows script developers to create their own functions or subroutines or to make DLL calls. Subroutines are created with the syntax "Sub <subname> .... End Sub". Functions are similar "Function <funcname> As <type> ... <funcname> = <value> ... End Function." DLL functions are declared via the **Declare** statement.

ByRef and ByVal

ByRef gives other subroutines and functions the permission to make changes to variables that are passed in as parameters. The keyword ByVal denies this permission and the parameters cannot be reassigned outside their local procedure. ByRef is the Enable default and does not need to be used explicitly. Because ByRef is the default all variables passed to other functions or subroutines can be changed, the only exception to this is if you use the ByVal keyword to protect the variable or use parentheses which indicate the variable is ByVal.

If the arguments or parameters are passed with parentheses around them, you will tell Enable that you are passing them ByVal

```
SubOne var1, var2, (var3)
```

The parameter var3 in this case is passed by value and cannot be changed by the subroutine SubOne.

```
Function R( X As String, ByVal n As Integer)
```

In this example the function R is receiving two parameters X and n. The second parameter n is passed by value and the contents cannot be changed from within the function R.

In the following code samples scalar variable and user defined types are passed by reference.

Scalar Variables

```
Sub Main
Dim x(5) As Integer
Dim i As Integer
```

```
for i = 0 to 5
x(i) = i
next i
Print i
Joe (i), x  ‘ The parenthesis around it turn it into an expression which passes by value
print "should be 6: "; x(2), i
End Sub

Sub Joe( ByRef j As Integer, ByRef y() As Integer )
print "Joe: "; j, y(2)
j = 345
for i = 0 to 5
print "i: "; i; "y(i): "; y(i)
next i
y(2) = 3 * y(2)
End Sub
```

Passing User Defined Types by Ref to DLL’s and Enable functions

```
' OpenFile() Structure
Type OFSTRUCT
cBytes As String * 1
fFixedDisk As String * 1
nErrCode As Integer
reserved As String * 4
szPathName As String * 128
End Type

' OpenFile() Flags
Global Const OF_READ = &H0
Global Const OF_WRITE = &H1
Global Const OF_READWRITE = &H2
Global Const OF_SHARE_COMPAT = &H0
Global Const OF_SHARE_EXCLUSIVE = &H10
Global Const OF_SHARE_DENY_WRITE = &H20
Global Const OF_SHARE_DENY_READ = &H30
Global Const OF_SHARE_DENY_NONE = &H40
Global Const OF_PARSE = &H100
Global Const OF_DELETE = &H200
Global Const OF_VERIFY = &H400
Global Const OF_CANCEL = &H800
Global Const OF_CREATE = &H1000
Global Const OF_PROMPT = &H2000
```

```
Global Const OF_EXIST = &H4000
Global Const OF_REOPEN = &H8000


Declare Function OpenFile Lib "Kernel" (ByVal lpFileName As String, lpReOpenBuff As OFSTRUCT, ByVal
    wStyle As Integer) As Integer


Sub Main
Dim ofs As OFSTRUCT
' Print OF_READWRITE
ofs.szPathName = "c:\enable\openfile.bas"
print ofs.szPathName
ofs.nErrCode = 5
print ofs.nErrCode
OpenFile "t.bas", ofs
print ofs.szPathName
print ofs.nErrCode
End Sub
```

Calling Procedures in DLLs

DLLs or Dynamic-link libraries are used extensively by Engineers to funtions and subroutines located there. There are two main ways that Enable can be extended, one way is to call functions and subroutines in DLLs and the other way is to call functions and subroutines located in the calling application. The mechanisms used for calling procedures in either place are similar. (See the Declare Statement for more deatils)

To declare a DLL procedure or a procedure located in your calling application place a declare statement in your declares file or outside the code area. All declarations in Enable are Global to the run and accesible by all subroutines and functions. If the procedure does not return a value, declare it as a subroutine. If the procedure does have a return value declare it as a function.

```
Declare Function GetPrivateProfileString Lib "Kernel32" (ByVal lpApplicationName As String, ByVal _
lpKeyName As String, ByVal lpDefault As String, ByVal lpReturnedString As String, ByVal nSize As _
Integer, ByVal lpFileName As String) As Integer


Declare Sub InvertRect Lib “User” (ByVal hDC AS Integer, aRect As Rectangle)
```

Notice the line extension character “-“ the underscore. If a piece of code is too long to fit on one line a line extension character can be used when needed.

Once a procedure is declared, you can call it just as you would another Enable Function.

It is important to note that Enable cannot verify that you are passing correct values to a DLL procedure. If you pass incorrect values, the procedure may fail.

Passing and Returning Strings

Cypress Enable maintains variable-length strings internally as BSTRs. BSTRs are defined in the OLE header files as OLECHAR FAR *. An OLECHAR is a UNICODE character in 32-bit OLE and an ANSI character in 16-bit OLE. A BSTR can contain NULL values because a length is also maintained with the BSTR. BSTRs are also NULL terminated so they can be treated as an LPSTR. Currently this length is stored immediately prior to the string. This may change in the future, however, so you should use the OLE APIs to access the string length.

You can pass a string from Cypress Enable to a DLL in one of two ways. You can pass it "by value" (ByVal) or "by reference". When you pass a string ByVal, Cypress Enable passes a pointer to the beginning of the string data (i.e. it passes a BSTR). When a string is passed byreference, Enable passes a pointer to a pointer to the string data (i.e. it passes a BSTR *).

OLE API

```
SysAllocString/SysAllocStringLen
SysAllocString/SysAllocStringLen
SysFreeString
SysStringLen
SysReAllocStringLen
SysReAllocString
```

Note:: The BSTR is a pointer to the string, so you don't need to dereference it.

File Input/Output

Enable supports full sequential and binary file I/O.

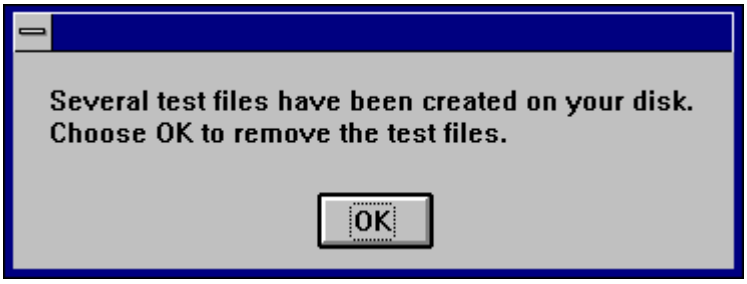
Functions and Statements that apply to file access:

Dir, EOF, FileCopy, FileLen, Seek, Open, Close, Input, Line Input, Print and Write

File I/O Examples

```
Sub Main
Open "TESTFILE" For Input As #1          ' Open file.
Do While Not EOF(1)                          ' Loop until end of file.
Line Input #1, TextLine                  ' Read line into variable.
Print TextLine                               ' Print to Debug window.
Loop
Close #1                                 ' Close file.
End Sub

Sub test
Open "MYFILE" For Input As #1            ' Open file for input.
```

```
Do While Not EOF(1)                     ' Check for end of file.
Line Input #1, InputData                ' Read line of data.
MsgBox InputData
Loop
Close #1                                ' Close file.
End Sub

Sub FileIO_Example()
Dim Msg                                 ' Declare variable.
Call Make3Files()                       ' Create data files.
Msg = "Several test files have been created on your disk. "
Msg = Msg & "Choose OK to remove the test files."
MsgBox Msg
For I = 1 To 3
Kill "TEST" & I ' Remove data files from disk.
Next I
End Sub

Sub Make3Files ()
Dim I, FNum, FName                      ' Declare variables.
For I = 1 To 3
FNum = FreeFile                         ' Determine next file number.
FName = "TEST" & FNum
Open FName For Output As FNum           ' Open file.
Print #I, "This is test #" & I          ' Write string to file.
Print #I, "Here is another ";  "line";  I
Next I
Close                                   ' Close all files.
End Sub
```



Arrays

Cypress Enable supports single and multi dimensional arrays. Using arrays you can refer to a series of variables by the same name each with a separate index. Arrays have upper and lower bounds. Enable allocates space for each index number in the array. Arrays should not be declared larger then necessary.

All the elements in an array have the same data type. Enable supports arrays of bytes, Booleans, longs, integers, singles, double, strings, variants and User Defined Types.

Ways to Declare a Fixed-Size Array

- *Global array*, use the **Dim** statement outside the procedure section of a code module to declare the array.
- To create a *local* array, use the **Dim** statement inside a procedure.
- Cypress Enable supports Dynamic arrays.

Declaring an Array

The array name must be followed by the upper bound in parentheses. The upper bound must be an integer.

```
Dim  ArrayName (10) As Interger
Dim Sum (20) As Double
```

Creating a Global Array

To create a global array, you simply use **Dim** outside the procedure:

```
Dim Counters (12) As Integer
Dim Sums (26) As Double

Sub Main () …
```

The same declarations within a procedure use **Static or Dim**:

```
Static Counters (12)  As Integer
Static Sums (22) As Double
```

The first declaration creates an array with 11 elements, with index numbers running from 0 to 10. The second creates an array with 21 elements. To change the default lower bound to 1 place an **Option Base** statement in the Declarations section of a module:

```
Option Base 1
```

Another way to specify the lower bound is to provide it explicitly (as an integer, in the range -32,768 to 32,767) using the **To** key word:

```
Dim Counters (1 To 13) As Integer
Dim Sums (100 To 126) As String
```

In the preceding declarations, the index numbers of Counters run from 1 to 13, and the index numbers of Sums run from 100 to 126.

Note: Many other versions of Basic allow you to use an array without first declaring it. Enable Basic does not allow this; you must declare an array before using it.

Manipulating Arrays

Loops often provide an efficient way to manipulate arrays. For example, the following **For** loop initializes all elements in the array to 5:

```
Static Counters (1 To 20) As Integer
Dim I As Integer
For I = 1 To 20
Counter ( I ) = 5
Next I
…
```

MultiDimensional Arrays

Cypress Enable supports multidimensional arrays. For example the following example declares a two-dimensional array within a procedure.

```
Static Mat(20, 20) As Double
```

Either or both dimensions can be declared with explicit lower bounds.

```
Static Mat(1 to 10, 1 to 10) As Double
```

You can efficiently process a multidimensional array with the use of for loops. In the following statements the elemtents in a multidimensional array are set to a value.

```
Dim L As Integer, J As Integer
Static TestArray(1 To 10, 1 to 10) As Double
For L = 1 to 10
For J = 1 to 10
TestArray(L,J) = I * 10 + J
Next J
Next L
```

Arrays can be more than two dimensional. Enable does not have an arbitrary upper bound on array dimensions.

```
Dim ArrTest(5, 3, 2)
```

This declaration creates an arrray that has three dimensions with sizes 6 by 4, by 3 unless Option Base 1 is set previously in the code. The use of Option Base 1 sets the lower bound of all arrays to 1 instead of 0.

User Defined Types

Users can define their own types that are composites of other built-in or user defined types. Variables of these new composite types can be declared and then member variables of the new type can be accessed using dot notation. Only variables of user defined types that contain simple data types can be passed to DLL functions expecting ‘C’ structures.

User Defined types are created using the type statement, which must be placed outside the procedure in your Enable Code. User defined types are global. The variables that are declared as user defined types can be either global or local. User Defined Types in Enable cannot contain arrays at this time

```
Type type1
a As Integer
d As Double
s As String
End Type

Type type2
a As Integer
o As type1
End Type

Dim type2a As type2
Dim type1a As type1

Sub TypeExample ()
a = 5
type1a.a = 7472
type1a.d = 23.1415
type1a.s = "YES"
type2a.a = 43
type2a.o.s = "Hello There"
MsgBox type1a.a
MsgBox type1a.d
MsgBox type1a.s
MsgBox type2a.a
MsgBox type2a.o.s
MsgBox a
End Sub
```

Dialog Support

Cypress Enable has support for custom dialogs. The syntax is similar to the syntax used in Microsoft Word Basic. The dialog syntax is not part of Microsoft Visual Basic or Microsoft Visual Basic For Applications (VBA). Enable has complete support for dialogs. The type of dialogs supported are outlined below.

Dialog Box controls

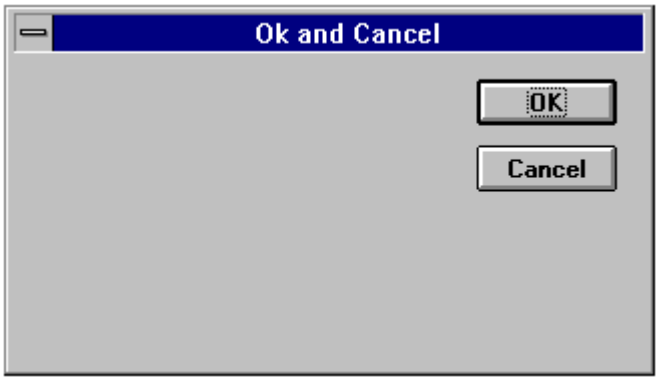
Enable Basic supports the standard Windows dialog box controls. This section introduces the controls available for custom dialog boxes and provides guidelines for using them.

The Dialog Box syntax begins with the statement “Begin Dialog”. The first two parameters of this statement are optional. If they are left off the dialog will automatically be centered.

```
Begin Dialog DialogName1 240, 184, "Test Dialog"
```

```
Begin Dialog DialogName1 60, 60,240, 184, "Test Dialog"
```

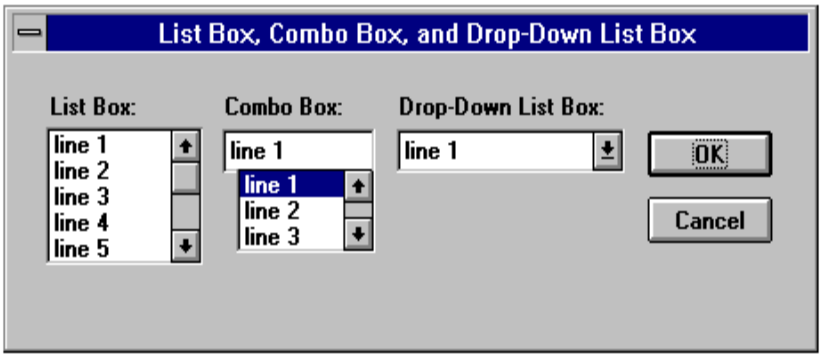
OK and Cancel Buttons



```
Sub Main
Begin Dialog ButtonSample 16,32,180,96,"OK and Cancel"
OKButton 132,8,40,14
CancelButton 132,28,40,14
End Dialog
Dim Dlg1 As ButtonSample
Button = Dialog (Dlg1)
End Sub
```

Every custom dialog box must contain at least one “command” button - a OK button or a Cancel button. Enable includes separate dialog box definition statements for each of these two types of buttons.

List Boxes, Combo Boxes and Drop-down List Boxes

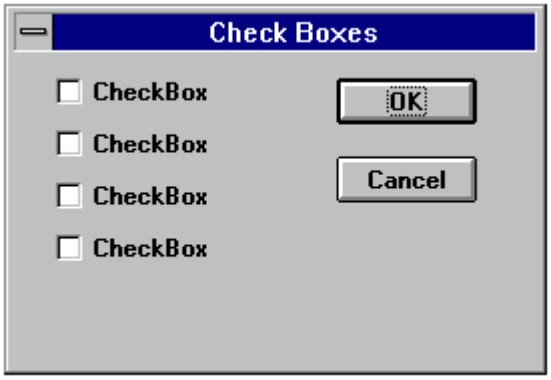



```
Sub Main
Dim MyList$ (5)
MyList (0) = "line Item 1"
MyList (1) = "line Item 2"
MyList (2) = "line Item 3"
MyList (3) = "line Item 4"
MyList (4) = "line Item 5"
MyList (5) = "line Item 6"

Begin Dialog BoxSample 16,35,256,89,"List Box, Combo Box, and Drop-Down List Box"
OKButton 204,24,40,14
CancelButton 204,44,40,14
ListBox 12,24,48,40, MyList$( ),.Lstbox
DropListBox 124,24,72,40, MyList$( ),.DrpList
ComboBox 68,24,48,40, MyList$( ),.CmboBox
Text 12,12,32,8,"List Box:"
Text 124,12,68,8,"Drop-Down List Box:"
Text 68,12,44,8,"Combo Box:"
End Dialog
Dim Dlg1 As BoxSample
```

```
Button = Dialog ( Dlg1 )


End Sub
```

You can use a list box, drop-down list box, or combo box to present a list of items from which the user can select. A drop-down list box saves space (it can drop down to cover other dialog box controls temporarily). A combo box allows the user either to select an item from the list or type in a new item. The items displayed in a list box, drop-down list box, or combo box are stored in an array that is defined before the instructions that define the dialog box.

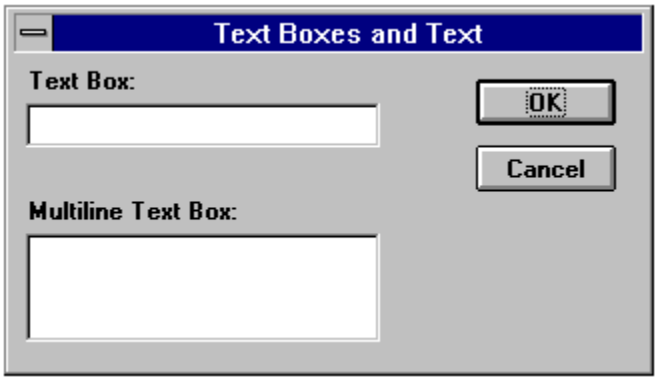
Check Boxes




```
Sub Main
Begin Dialog CheckSample15,32,149,96,"Check Boxes"
OKButton 92,8,40,14
CancelButton 92,32,40,14
CheckBox 12,8,45,8,"CheckBox",.CheckBox1
CheckBox 12,24,45,8,"CheckBox",.CheckBox2
CheckBox 12,40,45,8,"CheckBox",.CheckBox3
CheckBox 12,56,45,8,"CheckBox",.CheckBox4
End Dialog
Dim Dlg1 As CheckSample
Button = Dialog ( Dlg1 )
End Sub
```

You use a check box to make a “yes or no” or “on or off” choice. for example, you could use a check box to display or hide a toolbar in your application.

Text Boxes and Text




```
Sub Main
Begin Dialog TextBoxSample 16,30,180,96,"Text Boxes and Text"
OKButton 132,20,40,14
CancelButton 132,44,40,14
Text 8,8,32,8,"Text Box:"
TextBox 8,20,100,12,.TextBox1
Text 8,44,84,8,"Multiline Text Box:"
TextBox 8,56,100,32,.TextBox2
End Dialog
Dim Dlg1 As TextBoxSample
Button = Dialog ( Dlg1 )


End Sub
```

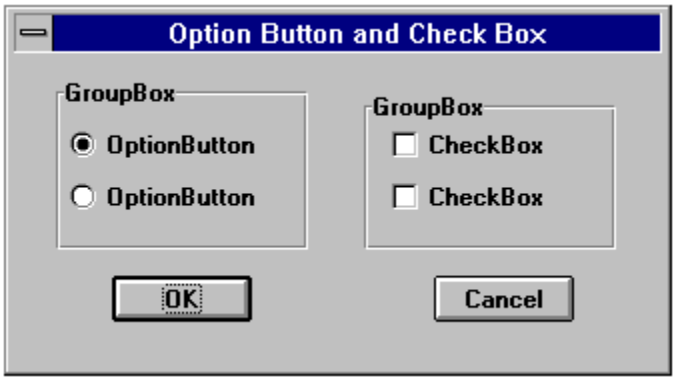
A text box control is a box in which the user can enter text while the dialog box is displayed. By default, a text box holds a single line of text. Enable support single and multi-line text boxes. The last parameter of the textbox function contains a variable to set the textbox style.

```
'=========================================================
' This sample shows how to implement a multiline textbox
'=========================================================
Const ES_LEFT             = &h0000&  'Try these different styles or-ed together
Const ES_CENTER           = &h0001&  ' as the last parameter of Textbox the change
Const ES_RIGHT            = &h0002&  ' the text box style.
Const ES_MULTILINE        = &h0004&  ' A 1 in the last parameter position defaults to
Const ES_UPPERCASE        = &h0008&  ' A multiline, Wantreturn, AutoVScroll testbox.
Const ES_LOWERCASE        = &h0010&
Const ES_PASSWORD         = &h0020&
Const ES_AUTOVSCROLL      = &h0040&
Const ES_AUTOHSCROLL      = &h0080&
Const ES_NOHIDESEL        = &h0100&
Const ES_OEMCONVERT       = &h0400&
Const ES_READONLY         = &h0800&
Const ES_WANTRETURN       = &h1000&
Const ES_NUMBER           = &h2000&

Sub Multiline
Begin Dialog DialogType 60, 60, 140, 185, "Multiline text Dialog", .DlgFunc
TextBox 10, 10, 120, 150, .joe, ES_MULTILINE Or ES_AUTOVSCROLL Or ES_WANTRETURN           '
Indicates multiline TextBox
'TextBox 10, 10, 120, 150, .joe, 1 ' indicates multi-line textbox
CancelButton 25, 168, 40, 12
OKButton 75, 168, 40, 12
End Dialog
```

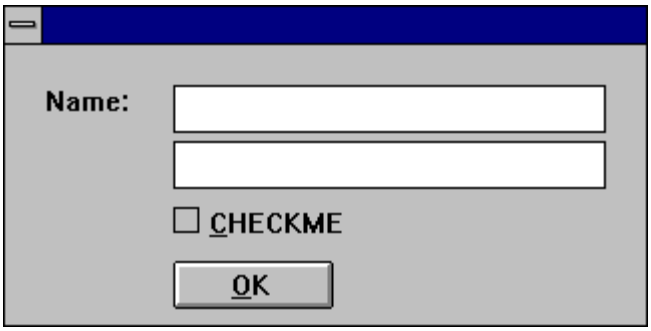
```
Dim Dlg1 As DialogType
Dlg1.joe  = "The quick brown fox jumped over the lazy dog"
' Dialog returns -1 for OK, 0 for Cancel
button = Dialog( Dlg1 )
'MsgBox "button: " & button
If button = 0 Then Exit Sub

MsgBox "TextBox: "& Dlg1.joe
End Sub
```

Option Buttons and Group Boxes

You can have option buttons to allow the user to choose one option from several. Typically, you would use a group box to surround a group of option buttons, but you can also use a group box to set off a group of check boxes or any related group of controls.




```
Begin Dialog GroupSample 31,32,185,96,"Option Button and Check Box"
OKButton 28,68,40,14
CancelButton 120,68,40,14
GroupBox 12,8,72,52,"GroupBox",.GroupBox1
GroupBox 100,12,72,48,"GroupBox",.GroupBox2
OptionGroup .OptionGroup1
OptionButton 16,24,54,8,"OptionButton",.OptionButton1
OptionButton 16,40,54,8,"OptionButton",.OptionButton2
CheckBox 108,24,45,8,"CheckBox",.CheckBox1
CheckBox 108,40,45,8,"CheckBox",.CheckBox2
End Dialog
Dim Dlg1 As GroupSample
Button = Dialog (Dlg1)
End Sub
```




```
Sub Main
Begin Dialog DialogName1 60, 60, 160, 70
TEXT 10, 10, 28, 12, "Name:"
TEXTBOX 42, 10, 108, 12, .nameStr
TEXTBOX 42, 24, 108, 12, .descStr
CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
OKBUTTON 42, 54, 40, 12
End Dialog
Dim Dlg1 As DialogName1
Dialog Dlg1

MsgBox Dlg1.nameStr
MsgBox Dlg1.descStr
MsgBox Dlg1.checkInt
End Sub
```

The Dialog Function

Cypress Enable supports the dialog function. This function is a user-defined function that can be called while a custom dialog box is displayed. The dialog function makes nested dialog boxes possible and receives messages from the dialog box while it is still active.

When the function dialog() is called in Enable it displays the dialog box and calls the dialog function for that dialog. Enable calls the dialog function to see if there are any commands to execute. Typical commands that might be used are disabling or hiding a control. By default, all dialog box controls are enabled. If you want a control to be hidden you must explicitly make it disabled during initialization. After initialization Enable displays the dialog box. When an action is taken by the user Enable calls the dialog function and passes values to the function that indicate the kind of action to take and the control that was acted upon.

The dialog box and its function are connected in the dialog definition. A “function name” argument is added to the Begin Dialog instruction and matches the name of the dialog function located in your Enable program.

```
Begin Dialog UserDialog1 60,60, 260, 188, "3", .Enable
```

The Dialog Box Controls

A dialog function needs an identifier for each dialog box control that it acts on. The dialog function uses string identifiers. String identifiers are the same as the identifiers used in the dialog record.

```
CheckBox 8, 56, 203, 16, "Check to display controls",. Chk1
```

The control's identifier and label are different. An identifier begins with a period and is the last parameter in a dialog box control instruction. In the sample code above "Check to display controls" is the label and .chk1 is the identifier.

The Dialog Function Syntax

The syntax for the dialog function is as follows:

```
Function FunctionName( ControlID$, Action%, SuppValue%)
Statement Block
FunctionName = ReturnValue
End Function
```

All parameters in the dialog function are required.

A dialog function returns a value when the user chooses a command button. Enable acts on the value returned. The default is to return 0 (zero) and close the dialog box. If a non zero is assigned the dialog box remains open. By keeping the dialog box open, the dialog function allows the user to do more than one command from the same dialog box. Dialog examples ship as part of the sample .bas programs and can be found in your install directory.

ControlID$

ControlID$ Receives the identifier of the dialog box control

Action

Action Identifies the action that calls the dialog function. There are six possibilities, Enable supports the first 4.

Action 1 The value passed before the dialog becomes visible

Action 2 The value passed when an action is taken (i.e. a button is pushed, checkbox is checked etc...) The controlID$ is the same as the identifier for the control that was chosen

Action 3 Corresponds to a change in a text box or combo box. This value is passed when a control loses the focus (for example, when the user presses the TAB key to move to a different control) or after the user clicks an item in the list of a combo box (an *Action* value of 2 is passed first). Note that if the contents of the text box or combo box do not change, an *Action* value of 3 is not passed. When *Action* is 3, *ControlID$* corresponds to the identifier for the text box or combo box whose contents were changed.

Action 4 Corresponds to a change of focus. When *Action* is 4, *ControlID$* corresponds to the identifier of the control that is gaining the focus. *SuppValue* corresponds to the numeric identifier for the control that lost the focus. A Dialog function cannot display a message box or dialog box in response to an *Action* value of 4

Supp Value

SuppValue receives supplemental information about a change in a dialog box control. The information SuppValue receives depends on which control calls the dialog function. The following *SuppValue* values are passed when *Action* is 2 or 3.

Control	SuppValue passed
ListBox, DropListBox, or ComboBox	Number of the item selected where 0 (zero) is the first item in the list box, 1 is the second item, and so on.
CheckBox	1 if selected, 0 (zero) if cleared.
OptionButton	Number of the option button selected, where 0 (zero) is the first option button within a group, 1 is the second option button, and so on.
TextBox	Number of characters in the text box.
ComboBox	If Action is 3, number of characters in the combo box.
CommandButton	A value identifying the button chosen. This value is not often used, since the same information is available from the ControlID$ value.

Statements and Functions Used in Dialog Functions

Statement or Function	Action or Result
DlgControlId	Returns the numeric equivalent of Identifier$, the string identifier for a dialog box control.
DlgEnable, DlgEnable()	The **DlgEnable** statement is used to enable or disable a dialog box control. When a control is disabled, it is visible in the dialog box, but is dimmed and not functional. **DlgEnable()** is used to determine whether or not the control is enabled.
DlgFocus, DlgFocus()	The **DlgFocus** statement is used to set the focus on a dialog box control. (When a dialog box control has the focus, it is highlighted.) **DlgFocus()** returns the identifier of the control that has the focus.
DlgListBoxArray, DlgListBoxArray()	The **DlgListBoxArray** statement is used to fill a list box or combo box with the elements of an array. It can be used to change the contents of a list box or combo box while the dialog box is displayed. **DlgListBoxArray()** returns an item in an array and the number of items in the array.
DlgSetPicture	The **DlgSetPicture** statement is used in a dialog function to set the graphic displayed by a picture control.
DlgText, DlgText	The **DlgText** statement is used to set the text or text label for a dialog box control. **TheDlgText()** function returns the label of a control.
DlgValue, DlgValue()	The **DlgValue** statement is used to select or clear a dialog box control. Then **DlgValue()** function

	returns the setting of a control.
DlgVisible, DlgVisible()	The **DlgVisible** statement is used to hide or show a dialog box control. The **DlgVisible()** function is used to determine whether a control is visible or hidden.

DlgControlId Function

`DlgControlId(`*`Identifier`*`)`

Used within a dialog function to return the numeric identifier for the dialog box control specified by *Identifier*, the string identifier of the dialog box control. Numeric identifiers are numbers, starting at 0 (zero) , that correspond to the positions of the dialog box control instructions within a dialog box definition. For example, consider the following instruction in a dialog box definition:

CheckBox 90, 50, 30, 12, “&Update”, .MyCheckBox

The instruction DlgControlId(“MyCheckBox”) returns 0 (zero) if the CheckBox instruction is the first instruction in the dialog box definition, 1 if it is the second, and so on.

In most cases, your dialog functions will perform actions based on the string identifier of the control that was selected.

DlgFocus Statement, DlgFocus() Function

```
DlgFocus Identifier
```

```
DlgFocus()
```

The DlgFocus statement is used within a dialog function to set the focus on the dialog box control identified by Identifier while the dialog box is displayed. When a dialog box control has the focus, it is active and responds to keyboard input. For example, if a text box has the focus, any text you type appears in that text box.

The DlgFocus() function returns the string identifier for the dialog box control that currently has the focus.

Example

This example sets the focus on the control “MyControl1” when the dialog box is initially displayed. (The main subroutine that contains the dialog box definition is not shown.)

```
Function MyDlgFunction( identifier, action, suppvalue)
Select Case action
Case 1                                          ‘ The dialog box is displayed
DlgFocus “MyControl1”
Case 2
‘ Statements that perform actions based on which control is selected
End Select
End Function
```

DlgListBoxArray, DlgListBoxArray()

```
DLGLISTBOXARRAY IDENTIFIER, ARRAYVARIABLE()
```

```
DLGLISTBOXARRAY(IDENTIFIER, ARRAYVARIABLE())
```

The DlgListBoxArray statement is used within a dialog function to fill a ListBox, DropListBox, or ComboBox with the contents of ArrayVariable() while the dialog box is displayed.

The DlgListBoxArray() function fills ArrayVariable() with the contents of the ListBox, DropListBox, or ComboBox specified by Identifier and returns the number of entries in the ListBox, DropListBox, or ComboBox. The ArrayVariable() parameter is optional (and currently not implemented) with the DlgListBoxArray() function; if ArrayVariable() is omitted, DlgListBoxArray() returns the number of entries in the specified control.

DlgSetPicture

```
DlgSetPicture  Identifier, PictureName
```

The DlgSetPicture function is used to set the graphic displayed by a picture control in a dialog.

The Identifier is a string or numeric representing the dialog box. The PictureName is a string that identifies the picture to be displayed.

DlgValue, DlgValue()

```
DlgValue Identifier, Value
DlgValue(Identifier)
```

The DlgValue statement is used in a dialog function to select or clear a dialog box control by setting the numeric value associated with the control specified by Identifier. For example, DlgValue “MyCheckBox”, 1 selects a check box, DlgValue “MyCHeckBox”, 0 clears a check box, and DlgValue “MyCheckBox”, -1 fills the check box with gray. An error occurs if Identifier specifies a dialog box control such as a text box or an option button that cannot be set with a numeric value.

```
The following dialog function uses a Select Case control structure to check the value of Action. The
    SuppValue is ignored in this function.


'This sample file outlines dialog capabilities, including nesting dialog boxes.


Sub Main


Begin Dialog UserDialog1 60,60, 260, 188, "3", .Enable
Text 8,10,73,13, "Text Label:"
TextBox 8, 26, 160, 18, .FText
CheckBox 8, 56, 203, 16, "Check to display controls",. Chk1
GroupBox 8, 79, 230, 70, "This is a group box:", .Group
CheckBox 18,100,189,16, "Check to change button text", .Chk2
PushButton 18, 118, 159, 16, "File History", .History
OKButton 177, 8, 58, 21
CancelButton 177, 32, 58, 21
End Dialog


Dim Dlg1 As UserDialog1
x = Dialog( Dlg1 )
End Sub
```

```
Function Enable( ControlID$, Action%, SuppValue%)

Begin Dialog UserDialog2 160,160, 260, 188, "3", .Enable
Text 8,10,73,13, "New dialog Label:"
TextBox 8, 26, 160, 18, .FText
CheckBox 8, 56, 203, 16, "New CheckBox",. ch1
CheckBox 18,100,189,16, "Additional CheckBox", .ch2
PushButton 18, 118, 159, 16, "Push Button", .but1
OKButton 177, 8, 58, 21
CancelButton 177, 32, 58, 21
End Dialog
Dim Dlg2 As UserDialog2
Dlg2.FText = "Your default string goes here"

Select Case Action%

Case 1
DlgEnable "Group", 0
DlgVisible "Chk2", 0
DlgVisible "History", 0
Case 2
If ControlID$ = "Chk1" Then
DlgEnable "Group"
DlgVisible "Chk2"
DlgVisible "History"
End If

If ControlID$ = "Chk2" Then
DlgText "History", "Push to display nested dialog"
End If

If ControlID$ = "History" Then
Enable =1
x = Dialog( Dlg2 )
End If



Case Else

End Select
Enable =1

End Function
```

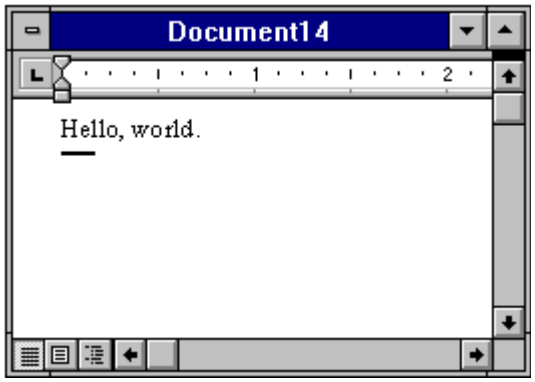
OLE Automation

What is OLE Automation?

OLE Automation is a standard, promoted by Microsoft, that applications use to expose their OLE objects to development tools, Enable Basic, and containers that support OLE Automation. A spreadsheet application may expose a worksheet, chart, cell, or range of cells all as different types of objects. A word processor might expose objects such as application, paragraph, sentence, bookmark, or selection.

When an application supports OLE Automation, the objects it exposes can be accessed by Enable Basic. You can use Enable Basic to manipulate these objects by invoking methods on the object, or by getting and setting the object's properties, just as you would with the objects in Enable Basic. For example, if you created an OLE Automation object named MyObj, you might write code such as this to manipulate the object:

```
Sub Main
Dim MyObj As Object
Set MyObj = CreateObject ("Word.Basic")
MyObj.FileNewDefault
MyObj.Insert "Hello, world."
MyObj.Bold 1
End Sub
```



The following syntax is supported for the **GetObject** function:

```
Set MyObj = GetObject ("", class)
```

Where class is the parameter representing the class of the object to retrieve. The first parameter at this time must be an empty string.

The properties and methods an object supports are defined by the application that created the object. See the application's documentation for details on the properties and methods it supports.

Accessing an Object

The following functions and properties allow you to access an OLE Automation object:

Name.	Description
CreateObject Function	Creates a new object of a specified type
GetObject Function	Retrieves an object pointer to a running application

What is an OLE Object?

An OLE *Automation Object* is an instance of a class within your application that you wish to manipulate programmatically, such as with Cypress Enable. These may be new classes whose sole purpose is to collect and expose data and functions in a way that makes sense to your customers.

The object becomes programmable when you expose those member functions. OLE Automation defines two types of members that you may expose for an object:

Methods are member functions that perform an action on an object. For example, a Document object might provide a Save method.

Properties are member function pairs that set or return information about the state of an object. For example, a Drawing object might have a style property.

For example, Microsoft suggests the following objects could be exposed by implementing the listed methods and properties for each object:

OLE Automation object	Methods	Properties
Application	Help	ActiveDocument
	Quit	Application
	Add Data	Caption
	Repeat	DefaultFilePath
	Undo	Documents
		Height
		Name
		Parent
		Path
		Printers
		StatusBar
		Top
		Value
		Visible
		Width

Document	Activate	Application
	Close	Author
	NewWindow	Comments
	Print	FullName
	PrintPreview	Keywords
	RevertToSaved	Name
	Save	Parent
	SaveAs	Path
		ReadOnly
		Saved
		Subject
		Title
		Value

To provide access to more than one instance of an object, expose a collection object. A collection object manages other objects. All collection objects support iteration over the objects they manage. For example, Microsoft suggests an application with a multiple document interface (MDI) might expose a Documents collection object with the following methods and properties:

Collection object	Methods	Properties
Documents	Add	Application
	Close	Count
	Item	Parent
	Open	

OLE Fundamentals

Object linking and embedding (OLE) is a technology that allows a programmer of Windows-based applications to create an application that can display data from many different applications, and allows the user to edit that data from within the application in which it was created. In some cases, the user can even edit the data from within their application.

The following terms and concepts are fundamental to understanding OLE.

OLE Object

An OLE object refers to a discrete unit of data supplied by an OLE application. An application can expose many types of objects. For example a spreadsheet application can expose a worksheet, macro sheet, chart, cell, or range of cells all as different types of objects. You use the OLE control to create linked and embedded objects. When a linked or embedded

object is created, it contains the name of the application that supplied the object, its data (or, in the case of a linked object, a reference to the data), and an image of the data.

OLE Automation

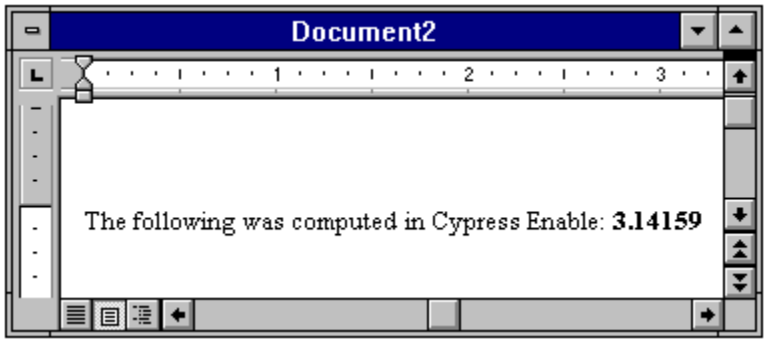
Some applications provide objects that support OLE Automation. You can use Enable Basic to programmatically manipulate the data in these objects. Some objects that support OLE Automation also support linking and embedding. You can create an OLE Automation object by using the CreateObject function.

Class

An objects class determines the application that provides the objects data and the type of data the object contains. The class names of some commonly used Microsoft applications include MSGraph, MSDraw, WordDocument, and ExcelWorksheet.

OLE Automation and Microsoft Word Example:

```
Sub OLEexample()
Dim word As Object
Dim myData As String

myData = 4 * Atn(1)                          ' Demonstrates Automatic type conversion
Set word = CreateObject("Word.Basic")
Word.AppShow
word.FileNewDefault
word.Insert "The following was computed in Cypress Enable: "
word.Bold 1                                      ' Show value in boldface
word.Insert myData
word.Bold 0

MsgBox "Done"
End Sub
```



Making Applications Work Together

Operations like linking and object embedding need applications to work together in a coordinated fashion. However, there is no way that Windows can be set up, in advance, to accommodate all the applications and dynamic link libraries that can be installed. Even within an application, the user has the ability to select various components to install.

As part of the installation process, Windows requires that applications supporting DDE/OLE features register their support by storing information in several different locations. The most important of these to cypress enable is the registration database.

WIN.INI

The win.ini file contains a special section called [embedding] that contains information about each of three applications that operate as object servers.

The Registration Database.

Starting with Windows 3.1, Each Windows system maintains a *registration database* file that records details about the DDE and OLE functions supported by the installed applications. The database is stored in file called **REG.DAT** in the \ **WINDOWS** directory.

The Registration database

The registration database is a file called **REG.DAT**. The file is a database that contains information that controls a variety of activities relating to data integration using DDE and OLE. The information contained in the **REG.DAT** database can be divided into four basic categories.

Associations.

The table contains information that associates files with specific extensions to particular applications. This is essentially the same function performed by the [extensions] section of the **WIN.INI**.

Shell Operations.

Windows contains two programs that are refered to as *Shell* programs. The term *Shell* refers to a program that organizes basic operating system tasks, like running applications, opening files, and sending files to the printer. Shell programs use list, windows, menus, and dialog boxes to perform these operations. In contrast, command systems like DOS require the entry of explicit command lines to accomplish these tasks

OLE Object Servers.

The registration database maintains a highly structured database of the details needed by programs that operate as object servers. This is by far the most complex task performed by the database. There is no **WIN.INI** equivalent for this function.

DDE/OLE Automation.

The registration database contains the details and the applications that support various types of DDE/OLE Automation operations.

It is useful to appreciate the difference in structure between the **WIN.INI** file and the **REG.DAT** database. **WIN.INI** is simply a text document. There are no special structures other than headings (simply titles enclosed in brackets) that

organize the information. If you want to locate an item in the **WIN.INI** file, you must search through the file for the specific item you want to locate. The registration database is a tree-like, structured database used for storing information relating to program and file operations, in particular, those that involve the use of DDE or OLE. The tree structure makes it easier to keep the complex set of instructions, needed to implement DDE and OLE operations, organized and accessible by the applications that need to use them. This is not possible when you are working with a text document like **WIN.INI.** The **WIN.INI** file records all sorts or information about the Windows system in a simple sequential listing.

Scripting Language Overview

Quick Reference of the Functions and Statements Available

Type/Functions/Statements

Flow of Control

Goto, End, OnError, Stop, Do...Loop, Exit Loop, For...Next, Exit For, If..Then..Else...End If, Return, Stop, While...Wend, Select Case

Converting

Chr, Hex, Oct, Str, CDbl, CInt, Clng, CSng, CStr, CVar, CVDate, Asc, Val, Date, DateSerial, DateValue, Format, Fix, Int, Day, Weekday, Month, Year, Hour, Minute, Second, TimeSerial, TimeValue

Dialog

Text, TextBox, ListBox, DropList, ComboBox, CheckBox, OKButton, BeginDialog, EndDialog, OptionGroup, OKButton, CancelButton, PushButton, Picture, GroupBox, Multi-line TextBox,

File I/O

FileCopy, ChDir, ChDrive, CurDir, CurDir, MkDir,RmDir, Open, Close, Print #, Kill, FreeFile, LOF, FileLen, Seek, EOF, Write #, Input, Line Input, Dir, Name, GetAttr, SetAttr, Dir, Get, Put

Math

Exp, Log, Sqr, Rnd, Abs, Sgn, Atn, Cos, Sin, Tan, Int, Fix

Procedures

Call, Declare, Function, End Function, Sub, End Sub, Exit, Global

Strings

Let, Len, InStr, Left, Mid, Asc, Chr, Right, LCase, Ucase, InStr, LTrim, RTrim, Trim, Option Compare, Len, Space, String, StrComp Format,

Variables and Constants

Dim, IsNull, IsNumeric,VarType, Const, IsDate, IsEmpty, IsNull, Option Explicit, Global, Static,

Error Trapping

On Error, Resume

Date/Time

Date, Now, Time, Timer

DDE

DDEInitiate, DDEExecute, DDETerminate

Arrays

Option Base, Option Explicit, Static, Dim, Global, Lbound, Ubound, Erase, ReDim

Miscellaneous

SendKeys, AppActivate, Shell, Beep, Rem, CreateObject, GetObject

Randomize

Data Types

Variable	Type Specifier	Usage
String	$	Dim Str_Var As String
Integer	%	Dim Int_Var As Integer
Long	&	Dim Long_Var As Long
Single	!	Dim Sing_Var As Single
Double	#	Dim Dbl_Var As Double
Variant		Dim X As Any
Boolean		Dim X As Boolean
Byte		Dim X As Byte
Object		Dim X As Object
Currency		(Not currently supported)

Operators

Arithmetic Operators

Operator	Function	Usage
^	Exponentiation	x = y^2
-	Negation	x = -2
*	Multiplication	x% = 2 * 3
/	division	x = 10/2
Mod	Modulo	x = y Mod z
+	Addition	x = 2 + 3
-	Subtraction	x = 6 - 4

*Arithmetic operators follow mathematical rules of precedence

* '+' or '&' can be used for string concatenation.

Relational Operators

Operator	Function	Usage
<	Less than	x < Y
<=	Less than or equal to	x <= Y
=	Equals	x = Y

>=	Greater than or equal to	x >= Y
>	Greater than	x > Y
<>	Not equal to	x <> Y

Logical Operators

Operator	Function	Usage
Not	Logical Negation	If Not (x)
And	Logical And	If (x> y) And (x < Z)
Or	Logical Or	if (x = y) Or (x = z)

Operator Precedence

Operator	Description	Order
()	Parenthesis	Highest
^	Exponentiation	
-	Unary minus	
/,*	Division / Multplication	
mod	Modulo	
+, -, &	Addition, subtraction, concatenation	
=, <>, <, >,<=,>=	Relational	
not	Logical negation	
and	Logical conjunction	
or	Logical disjunction	
Xor	Logical exclusion	
Eqv	Logical Equivalence	
Imp	Logical Implication	Lowest

Functions, Statements, Reserved words - Quick Reference

Abs, Access, Alias, And Any

App, AppActivate, Asc, Atn, As

Base, Beep, Begin, Binary, ByVal

Call, Case, ChDir, ChDrive, Choose, Chr, Const, Cos, CurDir, CDbl, CInt, CLng, CSng, CStr, CVar, CVDate,Close, CreateObject

Date, Day, Declare, Dim, Dir, Do...Loop,Dialog, DDEInitiate

DDEExecute, DateSerial, DateValue, Double

Else, ElseIf, End, EndIf, EOF, Eqv, Erase, Err, Error

Exit, Exp, Explicit

False, FileCopy, FileLen, Fix, For,

For...Next, Format, Function

Get, GetAttr, GoTo, Global, Get Object

Hex, Hour

If...Then...Else...[End If], Imp, Input, InputBox, InStr, Int, Integer, Is, IsEmpty, IsNull, IsNumeric, IsDate

Kill

LBound, LCase, Left, Len, Let, LOF,Log, Long, Loop, LTrim Line Input

Mid,Minute, MkDir, Mod, Month, MsgBox

Name, Next, Not, Now

Oct,On, Open, OKButton,Object, Option, Optional, Or, On Error

Print, Print #, Private, Put

Randomize, Rem, ReDim, RmDir, Rnd, Return, Rtrim

Seek, SendKeys, Set, SetAttr, Second, Select, Shell, Sin, Sqr, Stop,Str, Sng, Single, Space, Static, Step, Stop, Str, String, Sub, StringComp

Tan,Text, TextBox, Time, Timer, TimeSerial, TimeVale, Then, Type, Trim, True, To, Type

UBound, UCase, Ucase, Until

Val, Variant, VarType

Write #, While, Weekday, Wend, With

Xor

Year

Language Reference A - Z

Abs Function

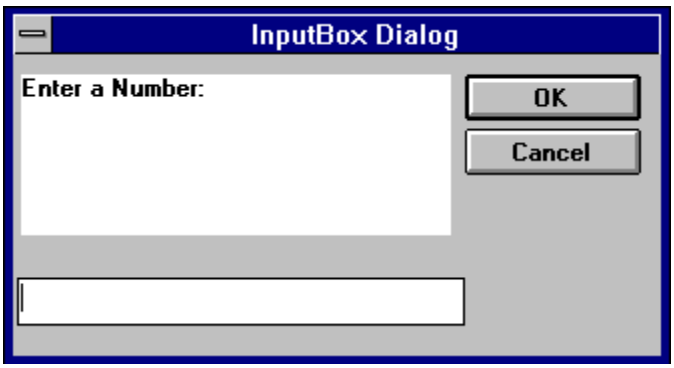
```
Abs (number)
```

Returns the absolute value of a number.

The data type of the return value is the same as that of the number argument. However, if the number argument is a Variant of VarType (String) and can be converted to a number, the return value will be a Variant of VarType (Double). If the numeric expression results in a Null, _Abs returns a Null.

Example:

```
Sub Main

Dim Msg, X, Y

X = InputBox("Enter a Number:")
Y = Abs(X)

Msg = "The number you entered is "  & X
Msg = Msg + ". The Absolute value of " & X & " is " & Y
MsgBox Msg 'Display Message.

End Sub
```



AppActivate Statement

```
AppActivate “app”
```

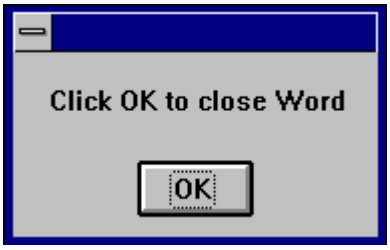
Activates an application.

The parameter *app* is a string expression and is the name that appears in the title bar of the application window to activate.

Related Topics: Shell, SendKeys

Example:

```
Sub Main ()
AppActivate "Microsoft Word"
SendKeys “%F,%N,Cypress Enable”,True
Msg = “Click OK to close Word”
MsgBox Msg
AppActivate “Microsoft Word”
SendKeys “%F,%C,N”, True
End Sub
```



Asc Function

Asc (*str*)

Returns a numeric value that is the ASCII code for the first character in a string.

Example:

```
Sub Main ()
Dim I, Msg                      ' Declare variables.
For I = Asc("A") To Asc("Z")    ' From A through Z.
Msg = Msg & Chr(I)          ' Create a string.
Next I
MsgBox Msg                      ' Display results.
End Sub
```

Atn Function

```
Atn (rad )
```

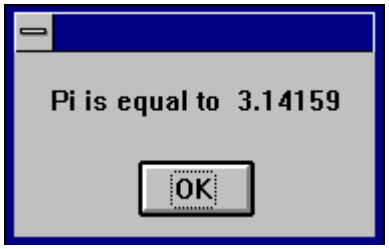
Returns the arc tangent of a number

The argument *rad* can be any numeric expression. The result is expressed in radians

Related Topics: Cos, Tan, Sin

Example:

```
Sub AtnExample ()
Dim Msg, Pi         ' Declare variables.
Pi = 4 * Atn(1)    ' Calculate Pi.
Msg = "Pi is equal to " & Str(Pi)
MsgBox Msg          ' Display results.
End Sub
```



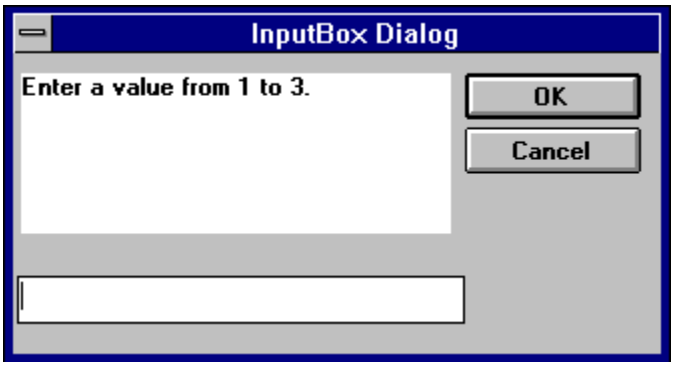
Beep Statement

```
Beep
```

Sounds a tone through the computer's speaker. The frequency and duration of the beep depends on hardware, which may vary among computers.

Example:

```
Sub BeepExample ()
Dim Answer, Msg            ' Declare variables.
Do
Answer = InputBox("Enter a value from 1 to 3.")
If Answer >= 1 And Answer <= 3 Then          ' Check range.
Exit Do                           ' Exit Do...Loop.
Else
Beep                               ' Beep if not in range.
End If
Loop
MsgBox "You entered a value in the proper range."
End Sub
```



Call Statement

```
Call funcname [(parameter(s)]
or
[parameter(s)]
```

Activates an Enable Subroutine called *name* or a DLL function with the name *name*. The first parameter is the name of the function or subroutine to call, and the second is the list of arguments to pass to the called function or subroutine.

You are never required to use the Call statement when calling an Enable subroutine or a DLL function. Parentheses must be used in the argument list if the Call statement is being used.

Example:

```
Sub Main ()
Call Beep
MsgBox "Returns a Beep"
End Sub
```

CBool Function

```
CBool (expression)
```

Converts expressions from one data type to a boolean. The parameter *expression* must be a valid string or numeric expression.

Example:

```
Sub Main

Dim A, B, Check

A = 5: B = 5
Check = CBool(A = B)
Print Check

A = 0
Check = CBool(A)
Print Check

End Sub
```

CDate Function

CVDate (*expression*)

Converts any valid expression to a Date variable with a vartype of 7.

The parameter expression must be a valid string or numeric date expression and can represent a date from January 1, 30 through December 31, 9999.

Example:

```
Sub Main

Dim MyDate, MDate, MTime, MSTime
MybDate = "May 29, 1959"                    ' Define date.
MDate = CDate(MybDate)                      ' Convert to Date data type.

MTime = "10:32:27 PM"                       ' Define time.
MSTime = CDate(MTime)                       ' Convert to Date data type.

Print MDate
Print MSTime

End Sub
```

CDbl Function

CDbl (*expression*)

Converts expressions from one data type to a double. The parameter *expression* must be a valid string or numeric expression.

Example:

```
Sub Main ()
Dim y As Integer

y = 25555                                   'the integer expression only allows for 5 digits
If  VarType(y) = 2 Then
Print y
```

```
x = CDbl(y)   'Converts the integer value of y to a double value in x
x = x * 100000 'y is now 10 digits in the form of x     '
Print x
End If

End Sub
```

ChDir Statement

ChDir *pathname*

Changes the default directory

Pathname: [*drive*:] [\ ] *dir*[*dir*]...

The parameter *pathname* is a string limited to fewer then 128 characters. The *drive* parameter is optional. The *dir* parameter is a directory name. ChDir changes the default directory on the current drive, if the drive is omitted.

Related Topics: CurDir, CurDir$, ChDrive, Dir, Dir$, MkDir, RmDir

Example:

```
Sub Main ()
Dim Answer, Msg, NL                          ' Declare variables.
NL = Chr(10)                                 ' Define newline.
CurPath = CurDir()                           ' Get current path.
ChDir "\"
Msg = "The current directory has been changed to "
Msg = Msg & CurDir() & NL & NL & "Press OK to change back "
Msg = Msg & "to your previous default directory."
Answer = MsgBox(Msg)                         ' Get user response.
ChDir CurPath                                ' Change back to user default.
Msg = "Directory changed back to " & CurPath & "."
MsgBox Msg                                   ' Display results.
End Sub
```

ChDrive Statement

ChDrive *drivename*

Changes the default drive

The parameter *drivename* is a string and must correspond to a an existing drive. If *drivename* contains more than one letter, only the first character is used.

Example:

```
Sub Main ()
Dim Msg, NL                                 ' Declare variables.
NL = Chr(10)                                ' Define newline.
CurPath = CurDir()                          ' Get current path.
ChDir "\"
ChDrive "C:"
Msg = "The current directory has been changed to "
Msg = Msg & CurDir() & NL & NL & "Press OK to change back "
Msg = Msg & "to your previous default directory."
MsgBox Msg                                  ' Get user response.
ChDir CurPath                               ' Change back to user default.
Msg = "Directory changed back to " & CurPath & "."
MsgBox Msg                                  ' Display results.
End Sub
```

Related Topics: ChDir, CurDir, CurDir$, MkDir, RmDir

CheckBox

CheckBox *starting x position, starting y position, width, height*

For selecting one or more in a series of choices

Example:

```
Sub Main ()
Begin Dialog DialogName1 60, 70, 160, 50, "ASC - Hello"

CHECKBOX 42, 10, 48, 12, "&CHECKME", .checkInt
OKBUTTON 42, 24, 40, 12
End Dialog
Dim Dlg1 As DialogName1
Dialog Dlg1
```

```
If Dlg1.checkInt = 0 Then
Q = "didn't check the box."
Else
Q = "checked the box."
End If
MsgBox "You " & Q

End Sub
```

Choose Function

Choose(*number, choice1, [choice2,] [choice3,]…*)

Returns a value from a list of arguments

Choose will return a null value if number is less than one or greater than the number of choices in the list. If *number* is not an integer it will be rounded to the nearest integer.

Example:

```
Sub Main
number = 2
GetChoice = Choose(number, "Choice1", "Choice2", "Choice3")
Print GetChoice
End Sub
```

Chr Function

Chr(*int*)

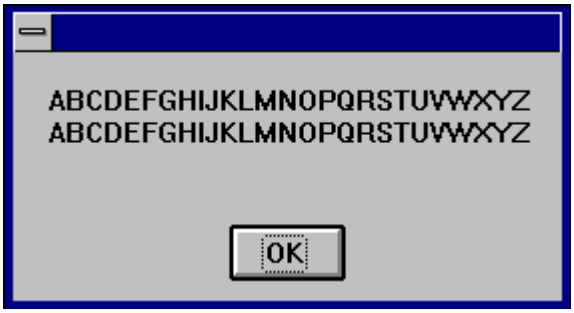
Returns a one-character string whose ASCII number is the argument

Chr returns a String

Example:

```
Sub ChrExample ()
Dim X, Y, Msg, NL
NL = Chr(10)
For X = 1 to 2
For Y = Asc("A") To Asc("Z")
```

```
Msg = Msg & Chr(Y)
Next Y
Msg = Msg & NL
Next X
MsgBox Msg
End Sub
```



CInt Function

```
CInt (expression)
```

Converts any valid expression to an integer.

Example:

```
Sub Main ()
Dim y As Long

y = 25
If  VarType(y) = 2 Then
Print y
x = CInt(y)   'Converts the long value of y to an integer value in x
Print x
End If

End Sub
```

CLng Function

CLng (*expression*)

Converts any valid expression into a long.

Example:

```
Sub Main ()
Dim y As Integer

y = 25000                                   'the integer expression can only hold five digits
If  VarType(y) = 2 Then
Print y
x = CLng(y)  'Converts the integer value of x to a long value in x
x = x * 10000 'y is now ten digits in the form of x
Print x
End If

End Sub
```

Close Statement

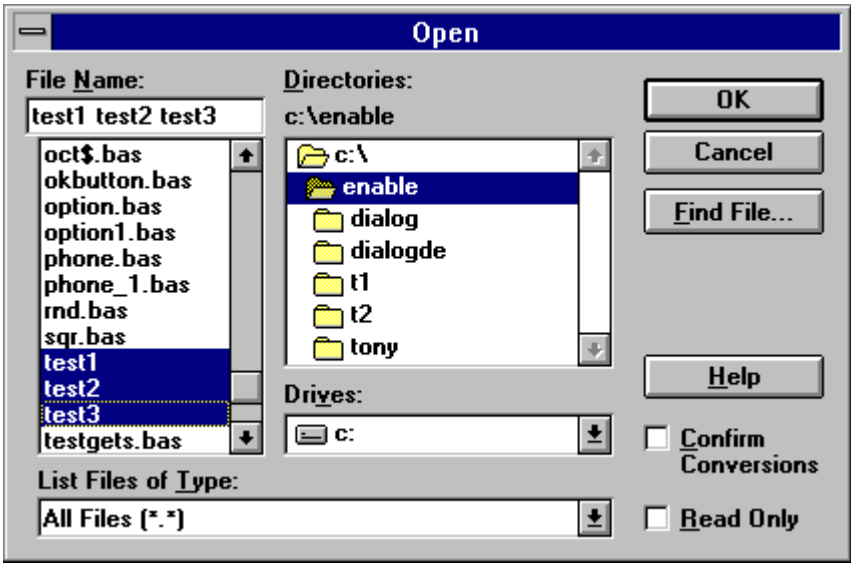
Close *[[#filenumber] [, [#]filenumber],,,*

The Close Statement takes one argument *filenumber*. *Filenumber* is the number used with the Open Statement to open the file. If the Close Statement is used without any arguments it closes all open files.

Example:

```
Sub Main
Open "c:\test.txt" For Input As #1
Do While Not EOF(1)
MyStr = Input(10, #1)
MsgBox MyStr
Loop
Close #1

End Sub

Sub Make3Files ()
Dim I, FNum, FName                          ' Declare variables.
For I = 1 To 3
FNum = FreeFile                             ' Determine next file number.
FName = "TEST" & FNum
```

```
Open FName For Output As FNum                ' Open file.
Print #I, "This is test #" & I               ' Write string to file.
Print #I, "Here is another ";  "line";  I
Next I
Close                                        ' Close all files.
End Sub
```



Const Statement

Const *name* = *expression*

Assigns a symbolic name to a constant value.

A constant must be defined before it is used.

The definition of a Const in Cypress Enable outside the procedure or at the module level is a global. The syntax Global Const and Const are used below outside the module level are identical.

A type declaration character may be used however if none is used Enable will automatically assign one of the following data types to the constant, long (if it is a long or integer), Double (if a decimal place is present), or a String (if it is a string).

Example:

```
Global Const Height = 14.4357     '
Const PI = 3.14159      'Global to all procedures in a module
Sub Main ()
```

```
Begin Dialog DialogName1 60, 60, 160,70, "ASC - Hello"
TEXT 10, 10, 100, 20, "Please fill in the radius of circle x"
TEXT 10, 40, 28, 12, "Radius"
TEXTBOX 42, 40, 28, 12, .Radius
OKBUTTON 42, 54,40, 12
End Dialog
Dim Dlg1 As DialogName1
Dialog Dlg1
CylArea = Height * (Dlg1.Radius * Dlg1.Radius) * PI
MsgBox "The volume of Cylinder x is " & CylArea
End Sub
```

Cos Function

Cos (*rad*)

Returns the cosine of an angle

The argument *rad* must be expressed in radians and must be a valid numeric expression.Cos will by default return a double unless a single or integer is specified as the return value.

Example:

```
Sub Main()
Dim J As Double
Dim I As Single                    ' Declare variables.
Dim K As Integer
For I =1 To 10    '
Msg = Msg & Cos(I) & ", "          'Cos function call
J=Cos(I)
Print J
K=Cos(I)
Print K
Next I
MsgBox Msg                      ' Display results.
MsgBox Msg1
End Sub
```

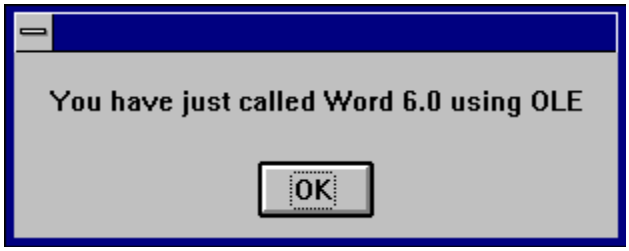
CreateObject Function

CreateObject (*class*)

Creates an OLE automation object.

```
Sub Command1_Click ()
Dim word6 As object
Set word6 = CreateObject("Word.Basic")
word6.FileNewDefault
word6.InsertPara
word6.Insert "Attn:"
word6.InsertPara
word6.InsertPara
word6.InsertPara
word6.Insert "          Vender Name: "
word6.Bold 1
name = "Some Body"
word6.Insert name
word6.Bold 0
word6.InsertPara
word6.Insert "    Vender Address:"
word6.InsertPara
word6.Insert "        Vender Product:"
word6.InsertPara
word6.InsertPara
word6.Insert "Dear Vender:"
word6.InsertPara
word6.InsertPara
word6.Insert "The letter you are reading was created with Cypress Enable."
word6.Insert " Using OLE Automation Cypress Enable can call any other OLE _ enabled "
word6.Insert "application. Enable is a Basic Scripting Language for _ applications"
word6.InsertPara
word6.InsertPara
word6.Insert "      Product Name: Cypress Enable"
word6.InsertPara
word6.Insert "      Company Name: Cypress Software Inc."
word6.InsertPara

word6.InsertPara
MsgBox "You have just called Word 6.0 using OLE"
End Sub

Vender Name: Client Name
Vender Address:
Vender Product:
```

```
Dear Vender:


The letter you are reading was created with Cypress Enable.Using OLE Automation Cypress Enable can call
    any other OLE enabled application. Enable is a Basic Scripting Language for applications


Product Name: Cypress Enable
Company Name: Cypress Software Inc.
```



CSng Function

```
CSng (expression)
```

Converts any valid expression to a Single.

Example:

```
Sub Main ()
Dim y As Integer

y = 25
If  VarType(y) = 2 Then
Print y
x = CSng(y)   'Converts the integer value of y to a single value in x
Print x
End If
```

CStr Function

```
CStr(expression)
```

Converts any valid expression to a String.

Example:

```
Sub Main
Dim Y As Integer
Y = 25
Print Y
If VarType(Y) = 2 Then
X = CStr(Y) 'converts Y To a Str
X = X + "hello"  'It is now possible to combine Y with strings
Print X
End If
End Sub
```

CurDir Function

```
CurDir (drive)
```

Returns the current path for the specified drive

CurDir returns a Variant; CurDir$ returns a String.

Example:

```
'Declare Function CurDir Lib "NewFuns.dll" ()  As String
Sub Form_Click ()
Dim Msg, NL                                  ' Declare variables.
NL = Chr(10)                                 ' Define newline.
Msg = "The current directory is: "
Msg = Msg & NL & CurDir()
MsgBox Msg                                   ' Display message.
End Sub
```

CVar Function

CVar (*expression*)

Converts any valid expression to a Variant.

Example:

```
Sub Main

Dim MyInt As Integer
MyInt = 4534
Print MyInt
MyVar = CVar(MyInt & "0.23") 'makes MyInt a Variant + 0.32
Print MyVar

End Sub
```

Date Function

```
Date, Date()
```

Returns the current system date

Date returns a Variant of VarType 8 (String) containing a date.

Example:

```
' Format Function Example
' This example shows various uses of the Format function to format values
' using both named and user-defined formats. For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running. When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used. When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale. For this example, English/United States is assumed.
```

```
' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main
x = Date()
Print Date
Print x
Print “VarType: “ & VarType(Date)
MyTime = "08:04:23 PM"
MyDate = "03/03/95"
MyDate = "January 27, 1993"

SysDate = Date
MsgBox Sysdate,0,"System Date"

MsgBox Now,0,"Now"
MsgBox MyTime,0,"MyTime"

MsgBox Second( MyTime ) & " Seconds"
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

' Returns current system time in the system-defined long time format.
MsgBox Format(Time, "Short Time") & " Short Time"
MsgBox Format(Time, "Long Time") & "Long Time"

' Returns current system date in the system-defined long date format.
MsgBox Format(Date, "Short Date") & " Short Date"
MsgBox Format(Date, "Long Date") & " Long Date"

MyDate = "30 December 91" ' use of European date
print Mydate

MsgBox MyDate,0,"MyDate International..."
MsgBox Day(MyDate),0,"day"
MsgBox Month(MyDate),0,"month"
MsgBox Year(MyDate),0,"year"

MyDate = "30-Dec-91" ' another of European date usage
```

```
print Mydate


MsgBox MyDate,0,"MyDate International..."
MsgBox Day(MyDate),0,"day"
MsgBox Month(MyDate),0," month"
MsgBox Year(MyDate),0,"year"


MsgBox Format("This is it", ">")        ' Returns "THIS IS IT".

End Sub
```

DateSerial Function

DateSerial (*year, month,day*)

Returns a variant (Date) corresponding to the year, month and day that were passed in. All three parameters for the DateSerial Function are required and must be valid.

Related Topics: DateValue, TimeSerial, TimeValue

Example:

```
Sub Main

Dim MDate
MDate = DateSerial(1959, 5, 29)
Print MDate

End Sub
```

DateValue Function

DateValue(*dateexpression*)

Returns a variant (Date) corresponding to the string date expression that was passed in. *dateexpression* can be a string or any expression that can represent a date, time or both a date and a time.

Related Topics: DateSerial, TimeSerial, TimeValue

Example:

```
Sub Main()
Dim v As Variant
Dim d As Double
d = Now
Print d
v = DateValue("1959/05/29")
MsgBox (VarType(v))
MsgBox (v)
End Sub
```

Day Function

Day(*dateexpression*)

Returns a variant date corresponding to the string date expression that was passed in. *dateexpression* can be a string or any expression that can represent a date.

Related Topics: Month, Weekday, Hour, Second

Example:

```
Sub Main

Dim MDate, MDay
MDate = #May 29, 1959#
MDay = Day(MDate)
Print "The Day listed is the " & MDay

End Sub
```

Declare Statement

Declare Sub *procedurename Lib Libname*$ [*Alias aliasname*$][(*argument list*)]

Declare Function *procedurename Lib Libname*$ [*Alias aliasname*$] [(*argument list*)][As *Type*]

The Declare statement makes a reference to an external procedure in a Dynamic Link Library (DLL).

The *procedurename* parameter is the name of the function or subroutine being called.

The *Libname* parameter is the name of the DLL that contains the procedure.

The optional Alias *aliasname* clause is used to supply the procedure name in the DLL if different from the name specified on the procedure parameter. When the optional *argument list* needs to be passed the format is as follows:

```
([ByVal] variable [As type] [,ByVal] variable [As type] ]…])
```

The optional ByVal parameter specifies that the variable is [passed by value instead of by reference (see “ByRef and ByVal” in this manual). The optional As type parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Varaint (see “Variable Types” in this manual).

If a procedure has no arguments, use double parentheses () only to assure that no arguments are passed. For example:

```
Declare Sub OntTime Lib “Check” ()
```

Cypress Enable extentions to the declare statement. The following syntax is not supported by Microsoft Visual Basic.

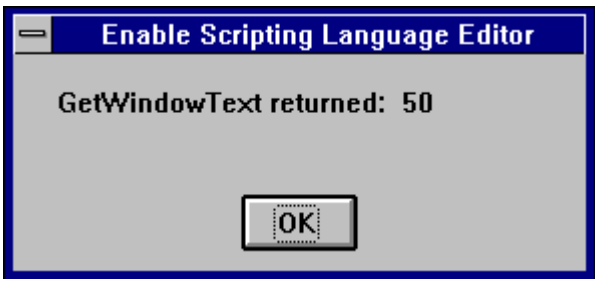
Declare Function *procedurename App* [*Alias aliasname*$] [(*argument list*)][As *Type*]

This form of the Declare statement makes a reference to a function located in the executable file located in the application where Enable is embedded.

Related Topics: Call

Example:

```
Declare Function GetFocus Lib "User" () As Integer
Declare Function GetWindowText Lib "User" (ByVal hWnd%, ByVal Mess$, ByVal cbMax%) As _ Integer

Sub Main
Dim hWindow%
Dim str1 As String *51
Dim str2 As String * 25

hWindow% = GetFocus()
print "GetWindowText returned: ", GetWindowText( hWindow%, str1,51 )
print "GetWindowText2 returned: ", GetWindowText( hWindow%, str2, 25)
print str1
print str2

End Sub
```



Dialog, Dialog Function

Dialog(*DialogRecord*)

Returns a value corresponding to the button the user chooses.

The Dialog() function is used to display the dialog box specified by *DialogRecord* . *DialogRecord* is the name of the dialog and must be defined in a preceeding Dim statement.

The return value or button:

-1 = OK button

0 = Cancel button

> 0 A command button where 1 is the first PushButton in the definition of the dialog and 2 is the second and so on.

Example:

```
' This sample shows all of the dialog controls on one dialog and how to
' vary the response based on which PushButton was pressed.

Sub Main ()
Dim MyList$(2)
MyList(0) = "Banana"
MyList(1) = "Orange"
MyList(2) = "Apple"
Begin Dialog DialogName1 60, 60, 240, 184, "Test Dialog"
Text 10, 10, 28, 12, "Name:"
TextBox 40, 10,50, 12, .joe
ListBox 102, 10, 108, 16, MyList$(), .MyList1
ComboBox 42, 30, 108, 42, MyList$(), .Combo1
DropListBox 42, 76, 108, 36, MyList$(), .DropList1$
OptionGroup .grp1
OptionButton 42, 100, 48, 12, "Option&1"
OptionButton 42, 110, 48, 12, "Option&2"
OptionGroup .grp2
```

```
OptionButton 42, 136, 48, 12, "Option&3"
OptionButton 42, 146, 48, 12, "Option&4"
GroupBox 132, 125, 70, 36, "Group"
CheckBox 142, 100, 48, 12, "Check&A", .Check1
CheckBox 142, 110, 48, 12, "Check&B", .Check2
CheckBox 142, 136, 48, 12, "Check&C", .Check3
CheckBox 142, 146, 48, 12, "Check&D", .Check4
CancelButton 42, 168, 40, 12
OKButton 90, 168, 40, 12
PushButton 140, 168, 40, 12, "&Push Me 1"
PushButton 190, 168, 40, 12, "Push &Me 2"
End Dialog
Dim Dlg1 As DialogName1
Dlg1.joe  = "Def String"
Dlg1.MyList1 = 1
Dlg1.Combo1 = "Kiwi"
Dlg1.DropList1 = 2
Dlg1.grp2 = 1
' Dialog returns -1 for OK, 0 for Cancel, button # for PushButtons
button = Dialog( Dlg1 )
'MsgBox "button: " & button 'uncomment for button return vale
If button = 0 Then Return

MsgBox "TextBox: "& Dlg1.joe
MsgBox "ListBox: " & Dlg1.MyList1
MsgBox Dlg1.Combo1
MsgBox Dlg1.DropList1
MsgBox "grp1: " & Dlg1.grp1
MsgBox "grp2: " & Dlg1.grp2
Begin Dialog DialogName2 60, 60, 160, 60, "Test Dialog 2"
Text 10, 10, 28, 12, "Name:"
TextBox 42, 10, 108, 12, .fred
OkButton 42, 44, 40, 12
End Dialog
If button = 2 Then
Dim Dlg2 As DialogName2
Dialog Dlg2
MsgBox Dlg2.fred
ElseIf button = 1 Then
Dialog Dlg1
MsgBox Dlg1.Combo1
End If
End Sub
```

Dim Statement

Dim variablename[(*subscripts*)][As *Type*][,name][As *Type*]]

Allocates storage for and declares the data type of variables and arrays in a module.

The types currently supported are integer, long, single, double and string and variant.

Example:

```
Sub Main
Dim x As Long
Dim y As Integer
Dim z As single
Dim a As double
Dim s As String
Dim v As Variant ' This is the same as Dim x or Dim x as any
End Sub
```

Dir Function

Dir[(*path,attributes*)]

Returns a file/directory name that matches the given *path* and *attributes*.

Example:

```
'=========================================================
' Bitmap sample using the Dir Function
'=========================================================

Sub DrawBitmapSample
Dim MyList()
Begin Dialog BitmapDlg 60, 60, 290, 220, "Enable bitmap sample", .DlgFunc
ListBox 10, 10, 80, 180, MyList(), .List1, 2
Picture 100, 10, 180, 180, "Forest.bmp", 0, .Picture1
CancelButton 42, 198, 40, 12
OKButton 90, 198, 40, 12
End Dialog

Dim frame As BitmapDlg
```

```
' Show the bitmap dialog
Dialog frame
End Sub

Function DlgFunc( controlID As String, action As Integer, suppValue As Integer )

DlgFunc = 1                                  ' Keep dialog active

Select Case action
Case 1 ' Initialize
temp = Dir( "c:\Windows\*.bmp" )
count = 0
While temp <> ""
count = count + 1
temp = Dir
Wend
Dim x() As String
ReDim x(count)
x(0) = Dir( "c:\Windows\*.bmp" )
For i = 1 To count
x(i) = dir
Next i
DlgListBoxArray "List1", x()
Case 2 ' Click
fileName = "c:\windows\" & DlgText("List1")
DlgSetPicture "Picture1", fileName
End Select
End Function
```

DlgEnable Statement

DlgEnable "*ControlName", Value*

This statement is used to enable or disable a particular control on a dialog box.

The parameter *ControlName* is the name of the control on the dialog box. The parameter *Value* is the value to set it to. 1 = Enable, 0 = Disable. On is equal to 1 in the example below. If the second parameter is omitted the status of the control toggles. The entire example below can be found in the dialog section of this manual and in the example .bas files that ship with Cypress Enable.

Related Topics: DlgVisible, DlgText

Example:

```
Function Enable( ControlID$, Action%, SuppValue%)
Begin Dialog UserDialog2 160,160, 260, 188, "3", .Enable
Text 8,10,73,13, "New dialog Label:"
TextBox 8, 26, 160, 18, .FText
CheckBox 8, 56, 203, 16, "New CheckBox",. ch1
CheckBox 18,100,189,16, "Additional CheckBox", .ch2
PushButton 18, 118, 159, 16, "Push Button", .but1
OKButton 177, 8, 58, 21
CancelButton 177, 32, 58, 21
End Dialog

Dim Dlg2 As UserDialog2
Dlg2.FText = "Your default string goes here"
Select Case Action%

Case 1
DlgEnable "Group", 0
DlgVisible "Chk2", 0
DlgVisible "History", 0
Case 2
If ControlID$ = "Chk1" Then
DlgEnable "Group", On
DlgVisible "Chk2"
DlgVisible "History"
End If

If ControlID$ = "Chk2" Then
DlgText "History", "Push to display nested dialog"
End If

If ControlID$ = "History" Then
Enable =1
Number = 4
MsgBox SQR(Number) & " The sqr of 4 is 2"
x = Dialog( Dlg2 )
End If

If ControlID$ = "but1" Then
```

```
End If

Case Else

End Select
Enable =1

End Function
```

DlgText Statement

DLGTEXT "*CONTROLNAME*", *STRING*

This statement is used to set or change the text of a dialog control.

The parameter *ControlName* is the name of the control on the dialog box. The parameter *String* is the value to set it to.

Related Topics: DlgEnable, DlgVisible

Example:

```
If ControlID$ = "Chk2" Then
DlgText "History", "Push to display nested dialog"
End If
```

DlgVisible Statement

DlgVisible "*ControlName*", *Value*

This statement is used to hide or make visible a particular control on a dialog box.

The parameter *ControlName* is the name of the control on the dialog box. The parameter *Value* is the value to set it to. 1 = Visible, 0 = Hidden. On is equal to 1. If the second parameter is omitted the status of the control toggles. The entire example below can be found in the dialog section of this manual and in the example .bas files that ship with Cypress Enable.

Related Topics: DlgEnable, DlgText

Example:

```
If ControlID$ = "Chk1" Then
DlgEnable "Group", On
DlgVisible "Chk2"
DlgVisible "History"
End If
```

Do...Loop Statement

Do [{While|Until} *condition*]
[*statements*]
[Exit Do]
[*statements*]
Loop

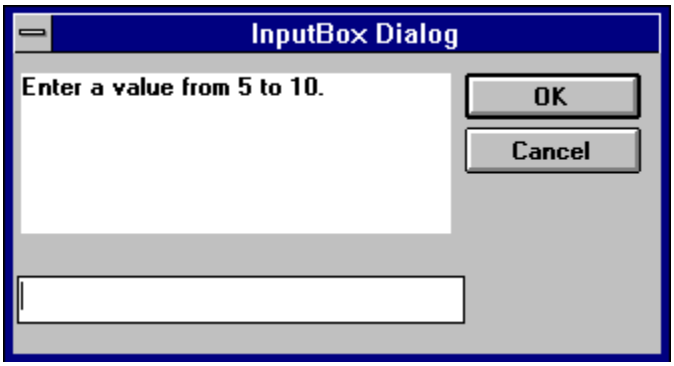
Do
[*statements*]
[Exit Do]
[*statements*]
Loop [{While|Until} *condition*]

Repeats a group of statements while a condition is true or until a condition is met.

Related Topics: While, Wend

Example:

```
Sub Main ()
Dim Value, Msg         ' Declare variables.
Do
Value = InputBox("Enter a value from 5 to 10.")
If Value >= 5 And Value <= 10 Then
Exit Do         ' Exit Do...Loop.
Else
Beep            ' Beep if not in range.
End If
Loop

End Sub
```



End Statement

End[{*Function* | *If* | *Sub*}]

Ends a program or a block of statements such as a Sub procedure or a function.

Related Topics: Exit, Function, If...Then...Else, Select Case, Stop

Example:

```
Sub Main()

Dim Var1 as String

Var1 = "hello"
MsgBox " Calling Test"
Test Var1
MsgBox Var1

End Sub

Sub Test(wvar1 as string)

wvar1 = "goodbye"
MsgBox "Use of End Statement"
End

End Sub
```

EOF Function

EOF(*Filenumber*)

Returns a value during file input that indicates whether the end of a file has been reached.

Related Topics: Open Statement

Example:

```
' Input Function Example

' This example uses the Input function to read 10 characters at a time from a ' file and display them in
   a MsgBox. This example assumes that TESTFILE is a 'text file with a few lines of 'sample data.

Sub Main
Open "TESTFILE" For Input As #1              ' Open file.
Do While Not EOF(1)                             ' Loop until end of file.
MyStr = Input(10, #1)                        ' Get ten characters.
MsgBox MyStr
Loop
Close #1                                                  ' Close file.
End Sub
```

Erase Statement

Erase *arrayname*[,*arrayname*]

Reinitializes the elements of a fixed array.

Related Topics: Dim

Example:

```
' This example demonstrates some of the  features of arrays. The lower bound
' for an array is 0 unless it is specified or option base has set it as is
' done in this example.

Option Base 1

Sub Main
```

```
' Declare array variables.
Dim Num(10) As Integer   ' Integer array.
Dim StrVarArray(10) As String   ' Variable-string array.
Dim StrFixArray(10) As String * 10   ' Fixed-string array.
Dim VarArray(10) As Variant   ' Variant array.
Dim DynamicArray() As Integer   ' Dynamic array.
ReDim DynamicArray(10)   ' Allocate storage space.
Erase Num   ' Each element set to 0.
Erase StrVarArray   ' Each element set to zero-length
' string ("").
Erase StrFixArray   ' Each element set to 0.
Erase VarArray   ' Each element set to Empty.
Erase DynamicArray   ' Free memory used by array.

End Sub
```

Exit Statement

Exit {*Do* | *For* | *Function* | *Sub* }

Exits a loop or procedure

Related Topics: End Statement, Stop Statement

Example:

```
' This sample shows Do ... Loop with Exit Do to get out.

Sub Main ()
Dim Value, Msg                              ' Declare variables.
Do
Value = InputBox("Enter a value from 5 to 10.")
If Value >= 5 And Value <= 10 Then              ' Check range.
Exit Do                                 ' Exit Do...Loop.
Else
Beep                                    ' Beep if not in range.
End If
Loop

End Sub
```

Exp

Exp(*num*)

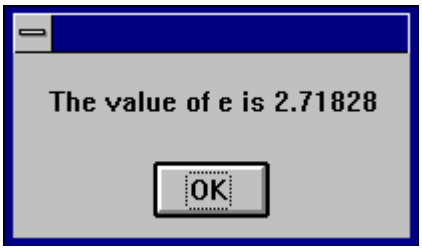
Returns the base of the natural log raised to a power (e ^ num).

The value of the constant e is approximately 2.71828.

Related Topics: Log

Example:

```
Sub ExpExample ()
' Exp(x) is e ^x so Exp(1) is e ^1 or e.
Dim Msg, ValueOfE                          ' Declare variables.
ValueOfE = Exp(1)                          ' Calculate value of e.
Msg = "The value of e is " & ValueOfE
MsgBox Msg                                 ' Display message.
End Sub
```



FileCopy Function

`FileCopy( sourcefile, destinationfile)`

Copies a file from source to destination.

The *sourcefile* and *destinationfile* parameters must be valid string expressions. *sourcefile* is the file name of the file to copy, *destinationfile* is the file name to be copied to.

Example:

```
Dim SourceFile, DestinationFile
SourceFile = "SRCFILE"   ' Define source file name.
DestinationFile = "DESTFILE"   ' Define target file name.
```

```
FileCopy SourceFile, DestinationFile   ' Copy source to target.
```

FileLen Function

FileLen(*filename*)

Returns a Long integer that is the length of the file in bytes

Related Topics: LOF Function

Example:

```
Sub Main

Dim MySize
MySize = FileLen("C:\TESTFILE")             ' Returns file length (bytes).
Print MySize

End Sub
```

Fix Function

Fix(*number*)

Returns the integer portion of a number

Related Topics: Int

Example:

```
Sub Main

Dim MySize
MySize = Fix(4.345)
Print MySize

End Sub
```

For each … Next Statement

```
For  Each  element in group
[statements]
[Exit For]
[statements]
Next [element]
```

Repeats the group of statments for each element in an array of a collection. For each … Next statements can be nested if each loop element is unique. The For Each…Next statement cannot be used with and array of user defined types.

Example:

```
Sub Main
dim z(1 to 4) as double
z(1) = 1.11
z(2) = 2.22
z(3) = 3.33
For Each v In z
Print v
Next v
End Sub
```

For...Next Statement

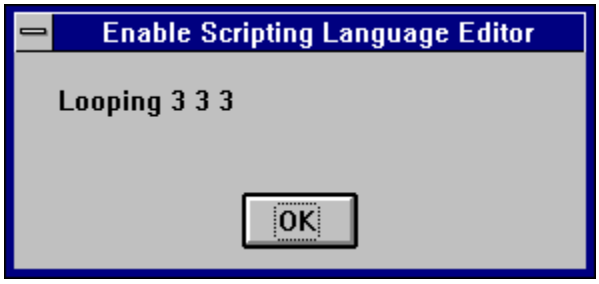
For *counter* = *expression1* to *expression2* [Step *increment]*
[*statements*]
Next [*counter*]

Repeats the execution of a block of statements for a specified number of times.

Example:

```
Sub main ()

Dim x,y,z

For x = 1 to 5
For y = 1 to 5
```

```
For z = 1 to 5
Print "Looping" ,z,y,x
Next z
Next y
Next x
End Sub
```



Format Function

Format (*expression* [,*fmt*])

Formats a string, number or variant datatype to a format expression.

Format returns returns a string

Part	Description
Expression	Expression to be formatted.
Fmt	A string of characters that specify how the expression is to displayed. or the name of a commonly-used format that has been predefined in Enable Basic. Do not mix different type format expressions in a single fmt parameter.

If the *fmt* parameter is omitted or is zero-length and the *expression* parameter is a numeric, **Format[$]** provides the same functionality as the **Str[$]** function by converting the numeric value to the appropriate return data type, Positive numbers convert to strings using **Format[$]** lack the leading space reserved for displaying the sign of the value, whereas those converted using **Str[$]** retain the leading space.

To format numbers, you can use the commonly-used formats that have been predefined in Enable Basic or you can create user-defined formats with standard characters that have special meaning when used in a format expression.

Predefined numeric format names:

Format

Name	Description
General	Display the number as is, with no thousand Separators Number.
Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Standard	Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator.

Format

Name	Description
Scientific	Use standard scientific notation.
True/False	Display False if number is 0, otherwise display True.

Characters for Creating User-Defined Number Formats

The following shows the characters you can use to create user-defined number formats.

Character	Meaning
Null string	Display the number with no formatting.
0	Digit placeholder. Display a digit or a zero. If the number being formatted has fewer digits than there are zeros (on either side of the decimal) in the format expression, leading or trailing zeros are displayed. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, the number is rounded to as many decimal places as there are zeros. If the number has more digits to left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, the extra digits are displayed without modification.
#	Digit placeholder. Displays a digit or nothing. If there is a digit in the expression being formatted in the position where the # appears in the format string, displays it; otherwise,

	nothing is displayed.
.	Decimal placeholder.The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator.

Character	Meaning	Description
%	Percentage placeholder.	The percent character (%) is inserted in the position where it appears in the format string. The expression is multiplied by 100.
,	Thousand separator.	The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Use of this separator as specified in the format statement contains a comma surrounded by digit placeholders(0 or #). Two adjacent commas or a comma immediately to the left of the decimal separator (whether or not a decimal is specified) means “scale the number by dividing it by 1000, rounding as needed.”
E-E+e-e+	Scientific format.	If the format expression contains at least one digit placeholder (0 or #) to the right of E-,E+,e- or e+, the number is displayed in scientific formatted E or e inserted between the number and its exponent. The number of digit placeholders to the right determines the number of digits in the exponent. Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a plus sign next to positive exponents.
:	Time separator.	The actual character used as the time separator depends on the Time Format specified in the International section of the Control Panel.
/	Date separator.	The actual character used as the date separator in the formatted out depends on Date Format specified in the International section of the Control Panel.

Character	Meaning
- + $ ( ) space	Display a literal character. To display a character other than one of those listed, precede it with a backslash (\).
\	Display the next character in the format string. The backslash itself isn't displayed. To display a backslash, use two backslashes (\\). Examples of characters that can't be displayed as literal characters are the date- and time- formatting characters (a,c,d,h,m,n,p,q,s,t,w,y, and /:), the numeric -formatting characters(#,0,%,E,e,comma, and period), and the string- formatting characters (@,&,<,>, and !).
"String"	Display the string inside the double quotation marks. To include a string in *fmt* from within Enable, you must use the ANSI code for a double quotation mark Chr(34) to enclose the text.
*	Display the next character as the fill character. Any empty space in a field is filled with the character following the asterisk.

Unless the *fmt* argument contains one of the predefined formats, a format expression for numbers can have from one to four sections separated by semicolons.

If you use	The result is
One section only	The format expression applies to all values.
Two	The first section applies to positive values, the second to negative sections values.
Three	The first section applies to positive values, the second to negative sections values, and the third to zeros.
Four	The first section applies to positive values, the second to negative section values, the third to zeros, and the fourth to **Null** values.

The following example has two sections: the first defines the format for positive values and zeros; the second section defines the format for negative values.

```
"$#,##0; ($#,##0)"
```

If you include semicolons with nothing between them. the missing section is printed using the format of the positive value. For example, the following format displays positive and negative values using the format in the first section and displays “Zero” if the value is zero.

```
“$#,##0;;\Z\e\r\o”
```

Sample Format Number Expressions

Some sample format expressions for numbers are shown below. (These examples all assume the Country is set to United States in the International section of the Control Panel.) The first column contains the format strings. The other columns contain the output the results if the formatted data has the value given in the column headings

Format (*fmt*)	Positive 3	Negative 3	Decimal .3	Null
Null string	3	-3	0.3	
0	3	-3	1	
0.00	3.00	-3.00	0.30	
#,##0	3	-3	1	
#,##0.00;;;Nil	3.00	-3.00	0.30	Nil
$#,##0;($#,##0)	$3	($3)	$1	
$#,##0.00;($#,##0.00) $3.00	($3.00)	$0.30		
0%	300%	-300%	30%	
0.00%	300.00%	-300.00%	30.00%	
0.00E+00	3.00E+00	-3.00E+00	3.00E-01	
0.00E-00	3.00E00	-3.00E00	3.00E-01	

Numbers can also be used to represent date and time information. You can format date and time serial numbers using date and time formats or number formats because date/time serial numbers are stored as floating-point values.

To format dates and times, you can use either the commonly used format that have been predefined or create user-defined time formats using standard meaning of each:

The following table shows the predefined data format names you can use and the meaning of each.

Format

Name	Description
General	Display a date and/or time. for real numbers, display a date and time.(e.g. 4/3/93 03:34 PM); If there is no fractional part, display only a date (e.g. 4/3/93); if there is no integer part, display time only (e.g. 03:34 PM).
Long Date	Display a Long Date, as defined in the International section of the Control Panel.
Medium	Display a date in the same form as the Short Date, as defined in the international section of the Control Panel, except spell out

	the month abbreviation.
Short Date	Display a Short Date, as defined in the International section of the Control Panel.
Long Time	Display a Long Time, as defined in the International section of the Control panel. Long Time includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minutes and the Time AM/PM designator.
Short Time	Display a time using the 24-hour format (e.g. 17:45)

This table shows the characters you can use to create user-defined date/time formats.

Character	Meaning
c	Display the date as dddd and display the time as ttttt. in the order.
d	Display the day as a number without a leading zero (1-31).
dd	Display the day as a number with a leading zero (01-31).
ddd	Display the day as an abbreviation (Sun-Sat).
ddddd	Display a date serial number as a complete date (including day , month, and year).

Character	Meaning
w	Display the day of the week as a number (1- 7).
ww	Display the week of the year as a number (1-53).
m	Display the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01-12). If mm immediately follows h or hh, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan-Dec).
mmmm	Display the month as a full month name (January-December).
q	display the quarter of the year as a number (1-4).
y	Display the day of the year as a number (1-366).
yy	Display the day of the year as a two-digit number (00-99)
yyyy	Display the day of the year as a four-digit number (100-9999).
h	Display the hour as a number without leading zeros (0-23).
hh	Display the hour as a number with leading zeros (00-23).
n	Display the minute as a number without leading zeros (0-59).
nn	Display the minute as a number with leading zeros (00-59).
s	Display the second as a number without leading zeros (0-59).
ss	Display the second as a number with leading zeros (00-59).
ttttt	Display a time serial number as a complete time (including hour, minute, and second) formatted using the time separator defined by the Time Format in the International section of the Control Panel. A leading zero is displayed if the Leading Zero option is selected and the time is before 10:00 A.M. or P.M. The default

	time format is h:mm:ss.
AM/PM	Use the 12-hour clock and display an uppercase AM/PM
am/pm	Use the 12-hour clock display a lowercase am/pm

Character	Meaning
A/P	Use the 12-hour clock display a uppercase A/P
a/p	Use the 12-hour clock display a lowercase a/p
AMPM	Use the 12-hour clock and display the contents of the 11:59 string (s1159) in the WIN.INI file with any hour before noon; display the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as it exists in the WIN.INI file. The default format is AM/PM.

The Following are examples of user-defined date and time formats:

Format	Display
m/d/yy	2/26/65
d-mmmm-yy	26-February-65
d-mmmm	26 February
mmmm-yy	February 65
hh:nn AM/PM	06:45 PM
h:nn:ss a/p	6:45:15 p
h:nn:ss	18:45:15
m/d/yy/h:nn	2/26/65 18:45

Strings can also be formatted with **Format[$].** A format expression for strings can have one section or two sections separated by a semicolon.

If you use	The result is
One section only	The format applies to all string data.
Two sections	The first section applies to string data, the second to **Null** values and zero-length strings.

The following characters can be used to create a format expression for strings:

Character	Meaning
@	Character placeholder. Displays a character or a space. Placeholders are filled from right to left unless there is an ! character in the format string.
&	Character placeholder. Display a character or nothing.
<	Force lowercase.
>	Force uppercase.
!	Force placeholders to fill from left to right instead of right to left.

Related Topics: Str, Str$ Function.

Example:

```
' Format Function Example

' This example shows various uses of the Format function to format values
' using both named and user-defined formats. For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running. When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used. When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale. For this example, English/United States is assumed.

' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main

MyTime = "08:04:23 PM"
MyDate = "03/03/95"
MyDate = "January 27, 1993"

MsgBox Now
MsgBox MyTime

MsgBox Second( MyTime ) & " Seconds"
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

' Returns current system time in the system-defined long time format.
MsgBox Format(Time, "Short Time")
MyStr = Format(Time, "Long Time")

' Returns current system date in the system-defined long date format.
MsgBox Format(Date, "Short Date")
```

```
MsgBox Format(Date, "Long Date")

MyStr Format(MyTime, "h:n:s")        ' Returns "17:4:23".
MyStr Format(MyTime, "hh:nn:ss")' Returns "20:04:22 ".
MyStr Format(MyDate, "dddd, mmm d yyyy")' Returns "Wednesday, Jan 27 1993".

' If format is not supplied, a string is returned.
MsgBox Format(23)                       ' Returns "23".

' User-defined formats.
MsgBox Format(5459.4, "##,##0.00")      ' Returns "5,459.40".
MsgBox Format(334.9, "###0.00")         ' Returns "334.90".
MsgBox Format(5, "0.00%")               ' Returns "500.00%".
MsgBox Format("HELLO", "<")             ' Returns "hello".
MsgBox Format("This is it", ">")        ' Returns "THIS IS IT".

End Sub
```

FreeFile Function

FreeFile

Returns an integer that is the next available file handle to be used by the Open Statement.

Related Topics: Open, Close, Write

Example:

```
Sub Main
Dim Mx, FileNumber
For Mx = 1 To 3
FileNumber = FreeFile
Open "c:\e1\TEST" & Mx For Output As #FileNumber
Write #FileNumber, "This is a sample."
Close #FileNumber
Next Mx

Open "c:\e1\test1" For Input As #1
Do While Not EOF(1)
MyStr = Input(10, #1)
MsgBox MyStr
```

```
Loop
Close #1
End Sub
```

Function Statement

Function *Fname* [(*Arguments*)] [As *type*]

[*statements*]

Functionname = expression

[*statements*]

Functionname = expression

End Function

Declares and defines a procedure that can receive arguments and return a value of a specified data type.

When the optional argument list needs to be passed the format is as follows:

```
([ByVal] variable [As type] [,ByVal] variable [As type] ]…])
```

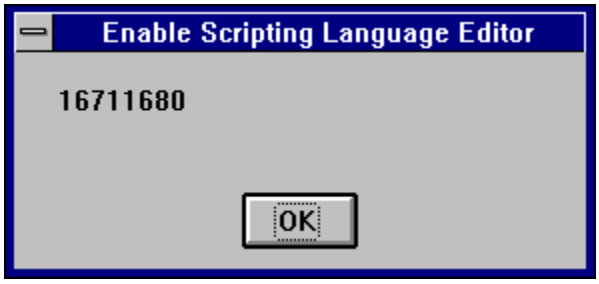
The optional ByVal parameter specifies that the variable is [passed by value instead of by reference (see “ByRef and ByVal” in this manual). The optional As type parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Varaint (see “Variable Types” in this manual).

Related Topics: Dim, End, Exit, Sub

Example:

```
Sub Main
Dim I as integer
For I = 1 to 10
Print GetColor2(I)
Next I
End Sub
Function GetColor2( c% ) As Long
GetColor2 = c% * 25
If c% > 2 Then
GetColor2 = 255                                ' 0x0000FF - Red
End If
If c% > 5 Then
GetColor2 = 65280                              ' 0x00FF00 - Green
End If
If c% > 8 Then
```

```
GetColor2 = 16711680                        ' 0xFF0000 - Blue
End If
End Function
```



Get Statement

```
GetStatement [#] filenmber,[recordnumber], variablename
```

Reads from a disk file into a varable

The Get Statement has these parts:

Filenumber The number used to Open the file with.

Recordnumber *For files opened in Binary mode* recordnumber *is the byte position where reading starts.*

VariableName The name of the variable used to receive the data from the file.

Related Topics: Open, Put

Get Object Function

```
GetObject(filename[,class])
```

The GetObject Function has two parameters a filename and a class. The filename is the name of the file containing the object to retrieve. If filename is an empty string then class is required. Class is a string containing the class of the object to retrieve.

Related Topics: CreateObject

Global Statement

```
Global Const constant
```

The Global Statement must be outside the procedure section of the script. Global variables are available to all functions and subroutines in your program

Related Topics: Dim, Const and Type Statements

Example:

```
Global Const Height = 14.4357     '
Const PI = 3.14159      'Global to all procedures in a module
Sub Main ()
Begin Dialog DialogName1 60, 60, 160,70, "ASC - Hello"
TEXT 10, 10, 100, 20, "Please fill in the radius of circle x"
TEXT 10, 40, 28, 12, "Radius"
TEXTBOX 42, 40, 28, 12, .Radius
OKBUTTON 42, 54,40, 12
End Dialog
Dim Dlg1 As DialogName1
Dialog Dlg1
CylArea = Height * (Dlg1.Radius * Dlg1.Radius) * PI
MsgBox "The volume of Cylinder x is " & CylArea
End Sub
```

GoTo Statement

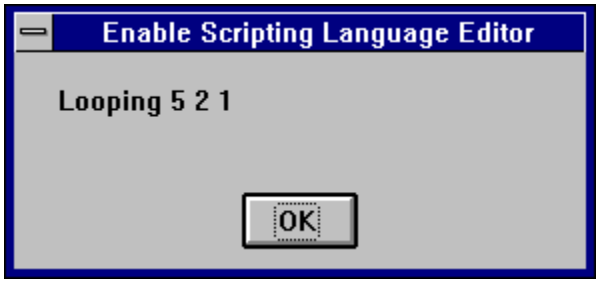
GoTo *label*

Branches unconditionally and without return to a specified label in a procedure.

Example:

```
Sub main ()

Dim x,y,z

For x = 1 to 5
For y = 1 to 5
For z = 1 to 5
Print "Looping" ,z,y,x
If y > 3 Then
GoTo Label1
End If
```

```
Next z
Next y
Next x
Label1:

End Sub
```



Hex

Hex (*num*)

Returns the hexadecimal value of a decimal parameter.

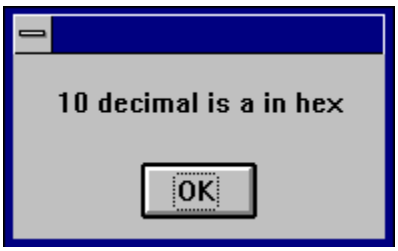
Hex returns a string

The parameter *num* can be any valid number. It is rounded to nearest whole number before evaluation.

Related Topics: Oct, Oct$

Example:

```
Sub Main ()

Dim Msg As String, x%
x% = 10
Msg =Str( x%) &  " decimal is "
Msg = Msg & Hex(x%) & " in hex "
MsgBox Msg
End Sub
```



Hour Function

Hour(*string*)

The Hour Function returns an integer between 0 and 23 that is the hour of the day indicated in the parameter *number*.

The parameter string is any number expressed as a string that can represent a date and time from January 1, 1980 through December 31, 9999.

Example:

```
' This example shows various uses of the Format function to format values
' using both named and user-defined formats. For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running. When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used. When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale. For this example, English/United States is assumed.

' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main

MyTime = "08:04:23 PM"
MyDate = "03/03/95"
MyDate = "January 27, 1993"

MsgBox Now
MsgBox MyTime

MsgBox Second( MyTime ) & " Seconds"
```

```
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

' Returns current system time in the system-defined long time format.
MsgBox Format(Time, "Short Time")
MyStr = Format(Time, "Long Time")

' Returns current system date in the system-defined long date format.
MsgBox Format(Date, "Short Date")
MsgBox Format(Date, "Long Date")

' This section not yet supported
'MyStr = Format(MyTime, "h:n:s")        ' Returns "17:4:23".
'MyStr = Format(MyTime, "hh:nn:ss AMPM")' Returns "05:04:23 PM".
'MyStr = Format(MyDate, "dddd, nnn d yyyy")' Returns "Wednesday, Jan 27 1993".

' If format is not supplied, a string is returned.
MsgBox Format(23)                       ' Returns "23".

' User-defined formats.
MsgBox Format(5459.4, "##,##0.00")      ' Returns "5,459.40".
MsgBox Format(334.9, "###0.00")         ' Returns "334.90".
MsgBox Format(5, "0.00%")               ' Returns "500.00%".
MsgBox Format("HELLO", "<")             ' Returns "hello".
MsgBox Format("This is it", ">")        ' Returns "THIS IS IT".

End Sub
```

HTMLDialog

HTMLDialog (*path*, *number*)

Runs a DHTML dialog that is specified in the path.

Example:

```
x =HtmlDialog( "c:\enable40\htmlt.htm", 57 )
```

```
‘See sample code on the samples disk htmldlg.bas
```

If...Then...Else Statement

Syntax 1

```
If condition Then thenpart [Else elsepart]
```

Syntax 2

If condition Then

```
[statement(s)]
ElseIf condition Then
[statement(s)]
Else
[statements(s)].
End If
Syntax 2
```

If conditional Then statement

Allows conditional statements to be executed in the code.

Related Topics: Select Case

Example:

```
Sub IfTest
' demo If...Then...Else
Dim msg as String
Dim nl as String
Dim someInt as Integer

nl = Chr(10)
msg = "Less"
someInt = 4

If 5 > someInt Then msg = "Greater" : Beep
MsgBox “” & msg
```

```
If 3 > someInt Then
msg = "Greater"
Beep
Else
msg = "Less"
End If
MsgBox “” & msg

If someInt = 1 Then
msg = "Spring"
ElseIf someInt = 2 Then
msg = "Summer"
ElseIf someInt = 3 Then
msg = "Fall"
ElseIf someInt = 4 Then
msg = "Winter"
Else
msg = "Salt"
End If
MsgBox “” & msg

End Sub
```

Input # Statement

```
Input   # filenumber, variablelist
```

Input # Statement reads data from a sequential file and assigns that data to variables.

The Input # Statement has two parameters filenumber and variablelist. filenumber is the number used in the open statement when the file was opened and variablelist is a Comma-delimited list of the variables that are assigned when read from the file..

Example:

```
Dim MyString, MyNumber
Open "c:\TESTFILE" For Input As #1   ' Open file for input.
Do While Not EOF(1)   ' Loop until end of file.
Input #1, MyString, MyNumber   ' Read data into two variables.
Loop
Close #1   ' Close file.
```

Input Function

Input(*n* , [#] *filenumber*)

Input returns characters from a sequential file.

The input function has two parameters *n* and *filenumber*. *n* is the number of bytes to be read from a file and *filenumber* is the number used in the open statement when the file was opened.

Example:

```
Sub Main
Open "TESTFILE" For Input As #1             ' Open file.
Do While Not EOF(1)                            ' Loop until end of file.
MyStr = Input(10, #1)                       ' Get ten characters.
MsgBox MyStr
Loop
Close #1                                                ' Close file.
End Sub
```

InputBox Function

InputBox(*prompt*[,[*title*][,[*default*][,*xpos*,*ypos*]]])

InputBox returns a String.

Prompt is string that is displayed usually to ask for input type or information.

Title is a string that is displayed at the top of the input dialog box.

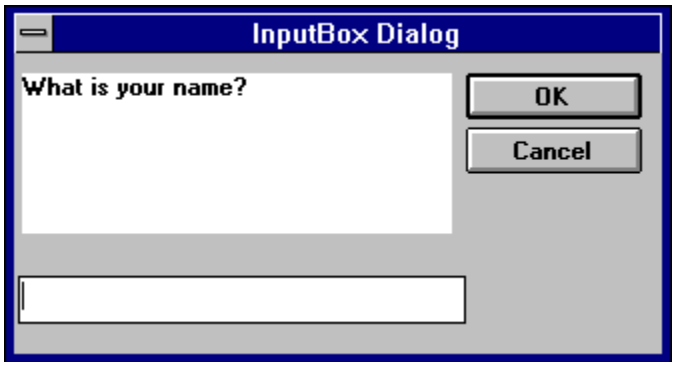
Default is a string that is displayed in the text box as the default entry.

Xpos and Ypos and the x and y coodinates of the relative location of the input dialog box.

Example:

```
Sub Main ()
Title$ = "Greetings"
Prompt$ = "What is your name?"
Default$ = ""
X% = 200
Y% = 200
```

```
N$ = InputBox$(Prompt$, Title$, Default$, X%, Y%)
End Sub
```



InStr

InStr(*numbegin*, *string1*, *string2*)

Returns the character position of the first occurrence of *string2* within *string1*.

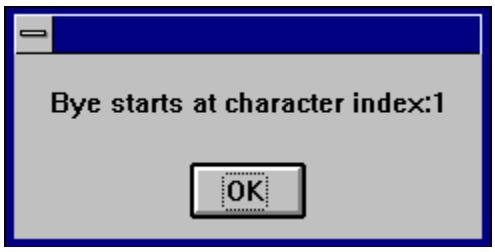
The *numbegin* parameter is not optional and sets the starting point of the search. *numbegin* must be a valid positive integer no greater than 65,535.

string1 is the string being searched and *string2* is the string we are looking for.

Related Topics: Mid Function

Example:

```
Sub Main ()
B$ = "Good Bye"
A% = InStr(2, B$, "Bye")
C% = Instr(3, B$, "Bye")
End Sub
```



Int Function

```
Int(number )
```

Returns the integer portion of a number

Related Topics: Fix

IsArray Function

```
IsArray(variablename )
```

Returns a boolean value True or False indicating whether the parameter vaiablename is an array.

Related Topics: IsEmpty, IsNumeric, VarType, IsObject

Example:

```
Sub Main

Dim MArray(1 To 5) As Integer, MCheck

MCheck = IsArray(MArray)
Print MCheck

End Sub
```

IsDate

```
IsDate(variant )
```

Returns a value that indicates if a variant parameter can be converted to a date.

Related Topics: IsEmpty, IsNumeric, VarType

Example:

```
Sub Main
Dim x As String

Dim MArray As Integer, MCheck
MArray = 345
x = "January 1, 1987"
MCheck = IsDate(MArray)
MChekk = IsDate(x)
MArray1 = CStr(MArray)
MCheck1 = CStr(MCheck)
Print MArray1 & " is a date " & Chr(10) & MCheck
Print x & " is a date" & Chr(10) & MChekk
End Sub
```

IsEmpty

IsEmpty(*variant*)

Returns a value that indicates if a variant parameter has been initialized.

Related Topics: IsDate, IsNull, IsNumeric, VarType

Example:

```
' This sample explores the concept of an empty variant

Sub Main
Dim x       ' Empty
x = 5       ' Not Empty - Long
x = Empty   ' Empty
y = x       ' Both Empty
MsgBox “x” & " IsEmpty: " & IsEmpty(x)
End Sub
```

IsNull

IsNull(*v*)

Returns a value that indicates if a variant contains the NULL value.

The parameter *v* can be any variant. IsNull returns a TRUE if *v* contains NULL. If isNull returns a FALSE the variant expression is not NULL.

The NULL value is special because it indicates that the *v* parameter contains no data. This is different from a null-string, which is a zero length string and an empty string which has not yet been initialized.

Related Topics: IsDate, IsEmpty, IsNumeric, VarType

IsNumeric

IsNumeric(*v*)

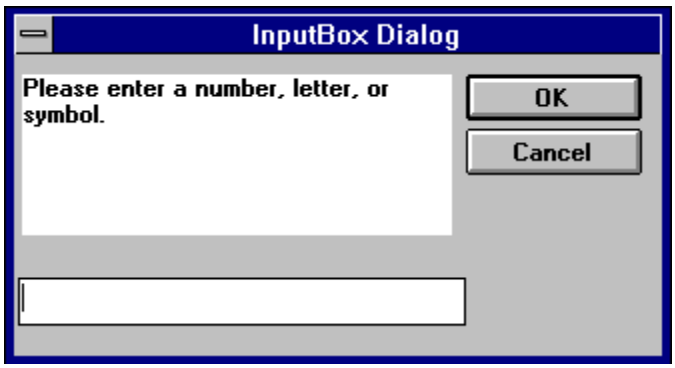
Returns a TRUE or FALSE indicating if the *v* parameter can be converted to a numeric data type.

The parameter *v* can be any variant, numeric value, Date or string (if the string can be interpreted as a numeric).

Related topics: IsDate, IsEmpty, IsNull, VarType

Example:

```
Sub Form_Click ()
Dim TestVar                                ' Declare variable.
TestVar = InputBox("Please enter a number, letter, or symbol.")
If IsNumeric(TestVar) Then                 ' Evaluate variable.
MsgBox "Entered data is numeric." ‘ Message if number.
Else
MsgBox "Entered data is not numeric."      ' Message if not.
End If
End Sub
```



IsObject Function

IsObject(*objectname*)

Returns a boolean value True or False indicating whether the parameter objectname is an object.

Related Topics: IsEmpty, IsNumeric, VarType, IsObject

Example:

```
Sub Main

Dim MyInt As Integer, MyCheck
Dim MyObject As Object
Dim YourObject As Object
Set MyObject = CreateObject("Word.Basic")

Set YourObject = MyObject
MyCheck = IsObject(YourObject)

Print MyCheck

End Sub
```

Kill Statement

Kill *filename*

Kill will only delete files. To remove a directory use the RmDir Statement

Related Topics: RmDir

Example:

```
Const NumberOfFiles = 3

Sub Main ()
Dim Msg                                              ' Declare variable.
Call MakeFiles()                                  ' Create data files.
Msg = "Several test files have been created on your disk. You may see "
Msg = Msg & "them by switching tasks. Choose OK to remove the test files."
MsgBox Msg
For I = 1 To NumberOfFiles
Kill "TEST" & I                                   ' Remove data files from disk.
Next I
End Sub

Sub MakeFiles ()
Dim I, FNum, FName                                   ' Declare variables.
For I = 1 To NumberOfFiles
FNum = FreeFile                                   ' Determine next file number.
FName = "TEST" & I
Open FName For Output As FNum               ' Open file.
Print #FNum, "This is test #" & I            ' Write string to file.
Print #FNum, "Here is another ";  "line";  I
Next I
Close                                                              ' Close all files.
Kill FName
End Sub
```

LBound Function

LBound(*array [,dimension]*)

Returns the smallest available subscript for the dimension of the indicated array.

Related Topics: UBound Function

Example:

```
' This example demonstrates some of the  features of arrays. The lower bound
' for an array is 0 unless it is specified or option base has set as is
' done in this example.

Option Base 1

Sub Main
Dim a(10) As Double
MsgBox "LBound: " & LBound(a) & " UBound: " & UBound(a)
Dim i As Integer
For i = 0 to 3
a(i) = 2 + i * 3.1
Next i
Print a(0),a(1),a(2), a(3)
End Sub
```

LCase, Function

Lcase[$](*string*)

Returns a string in which all letters of the string parameter have been converted to upper case.

Related Topics: Ucase Function

Example:

```
' This example uses the LTrim and RTrim functions to strip leading and
' trailing spaces, respectively, from a string variable. It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Sub Main
MyString = "  <-Trim->  "                  ' Initialize string.
TrimString = LTrim(MyString)               ' TrimString = "<-Trim->  ".
MsgBox "|" & TrimString & "|"
TrimString = LCase(RTrim(MyString))        ' TrimString = "  <-trim->".
MsgBox "|" & TrimString & "|"
TrimString = LTrim(RTrim(MyString))        ' TrimString = "<-Trim->".
MsgBox "|" & TrimString & "|"
```

```
' Using the Trim function alone achieves the same result.
TrimString = UCase(Trim(MyString))          ' TrimString = "<-TRIM->".
MsgBox "|" & TrimString & "|"
End Sub
```

Left

Left(*string, num*)

Returns the left most *num* characters of a string parameter.

Left returns a Variant, Left$ returns a String

Example:

```
Sub Main ()
Dim LWord, Msg, RWord, SpcPos, UsrInp       ' Declare variables.
Msg = "Enter two words separated by a space."
UsrInp = InputBox(Msg)                      ' Get user input.
print UsrInp
SpcPos = InStr(1, UsrInp, " ")              ' Find space.
If SpcPos Then
LWord = Left(UsrInp, SpcPos - 1)            ' Get left word.
print "LWord: ";  LWord
RWord = Right(UsrInp, Len(UsrInp) - SpcPos)  ' Get right word.
Msg = "The first word you entered is " & LWord
Msg = Msg & "." & " The second word is "
Msg = "The first word you entered is <" & LWord & ">"
Msg = Msg & RWord & "."
Else
Msg = "You didn't enter two words."
End If
MsgBox Msg                                  ' Display message.
MidTest = Mid("Mid Word Test", 4, 5)
Print MidTest
End Sub
```

Len

Len(*string*)

Returns the number of characters in a string.

Related Topics: InStr

Example:

```
Sub Main ()
A$ = "Cypress Enable"
StrLen% = Len(A$)                          'the value of StrLen is 14
MsgBox StrLen%
End Sub
```

Let Statement

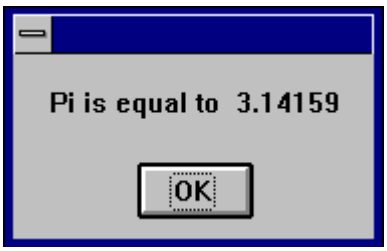
[Let] *variablename* = *expression*

Let assigns a value to a variable.

Let is an optional keyword that is rarely used. The Let statement is required in older versions of BASIC.

Example:

```
Sub Form_Click ()
Dim Msg, Pi         ' Declare variables.
Let Pi = 4 * Atn(1)    ' Calculate Pi.
Msg = "Pi is equal to " & Str(Pi)
MsgBox Msg          ' Display results.
End Sub
```



Line Input # Statement

`Line Input # filenumber and name`

Reads a line from a sequential file into a String or Variant variable.

The parameter *filenumber* is used in the open statement to open the file. The parameter name is the name of a variable used to hold the line of text from the file.

Related Topics: Open

Example:

```
' Line Input # Statement Example:
' This example uses the Line Input # statement to read a line from a
' sequential file and assign it to a variable. This example assumes that
' TESTFILE is a text file with a few lines of sample data.

Sub Main
Open "TESTFILE" For Input As #1             ' Open file.
Do While Not EOF(1)                            ' Loop until end of file.
Line Input #1, TextLine                     ' Read line into variable.
Print TextLine                                 ' Print to Debug window.
Loop
Close #1                                    ' Close file.

End Sub
```

LOF

LOF(*filenumber*)

Returns a long number for the number of bytes in the open file.

The parameter filenumber is required and must be an integer.

Related Topics: FileLen

Example:

```
Sub Main

Dim FileLength
Open "TESTFILE" For Input As #1
FileLength = LOF(1)
Print FileLength
Close #1

End Sub
```

Log

Log(*num*)

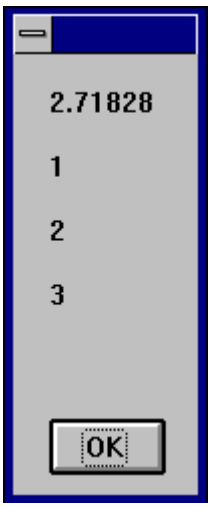
Returns the natural log of a number

The parameter *num* must be greater than zero and be a valid number.

Related Topics: Exp, Sin, Cos

Example:

```
Sub Form_Click ( )
Dim I, Msg, NL
NL = Chr(13) & Chr(10)
Msg = Exp(1) & NL
For I = 1 to 3
Msg = Msg & Log(Exp(1) ^ I ) & NL
Next I
MsgBox Msg
End Sub
```



Mid Function

string = Mid(*strgvar*,*begin*,*length*)

Returns a substring within a string.

Example:

```
Sub Main ()
Dim LWord, Msg, RWord, SpcPos, UsrInp        ' Declare variables.
Msg = "Enter two words separated by a space."
UsrInp = InputBox(Msg)                       ' Get user input.
print UsrInp
SpcPos = InStr(1, UsrInp, " ")               ' Find space.
If SpcPos Then
LWord = Left(UsrInp, SpcPos - 1)             ' Get left word.
print "LWord: ";  LWord
RWord = Right(UsrInp, Len(UsrInp) - SpcPos)  ' Get right word.
Msg = "The first word you entered is " & LWord
Msg = Msg & "." & " The second word is "
Msg = "The first word you entered is <" & LWord & ">"
Msg = Msg & RWord & "."
Else
Msg = "You didn't enter two words."
End If
MsgBox Msg                                   ' Display message.
MidTest = Mid("Mid Word Test", 4, 5)
Print MidTest
```

```
End Sub
```

Minute Function

`Minute(`*`string`*`)`

Returns an integer between 0 and 59 representing the minute of the hour.

Example:

```
' Format Function Example

' This example shows various uses of the Format function to format values
' using both named and user-defined formats. For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running. When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used. When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale. For this example, English/United States is assumed.

' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main

MyTime = "08:04:23 PM"
MyDate = "03/03/95"
MyDate = "January 27, 1993"

MsgBox Now
MsgBox MyTime

MsgBox Second( MyTime ) & " Seconds"
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

End Sub
```

MkDir

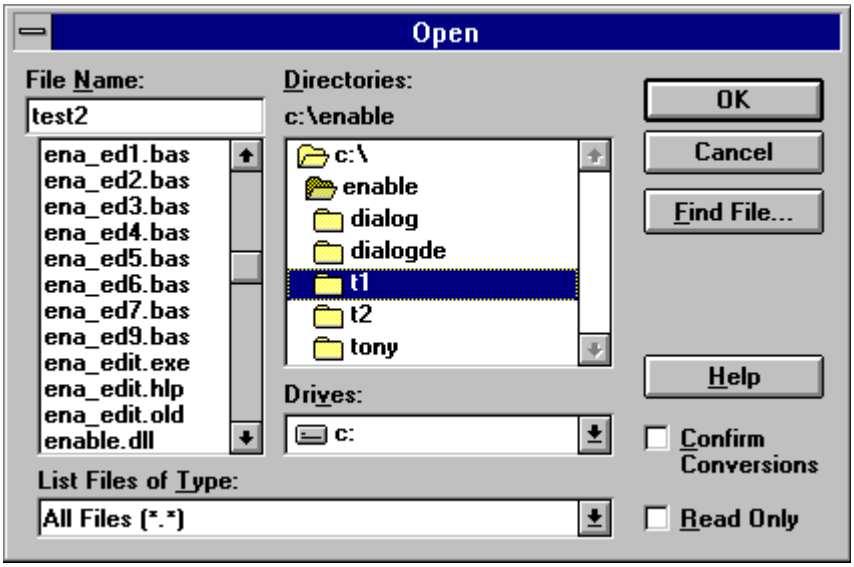
```
MkDir  path
```

Creates a new directory.

The parameter *path* is a string expression that must contain fewer than 128 characters.

Example:

```
Sub Main
Dim DST As String

DST = "t1"
mkdir DST
mkdir "t2"
End Sub
```



Month Function

Month(*number*)

Returns an integer between 1 and 12, inclusive, that represents the month of the year.

Related Topics: Day, Hour, Weekday, Year

Example:

```
Sub Main
MyDate = "03/03/96"
print MyDate
x = Month(MyDate)
print x

End Sub
```

MsgBox Function MsgBox Statement

MsgBox (*msg, [type] [, title]*)

Displays a message in a dialog box and waits for the user to choose a button.

The first parameter *msg* is the string displayed in the dialog box as the message. The second and third parameters are optional and respectively designate the type of buttons and the title displayed in the dialog box.

MsgBox Function returns a value indicating which button the user has chosen; the MsgBox statement does not.

Value	Meaning
Group 1	
0	Display OK button only
1	Display OK and Cancel buttons
2	Display Abort, Retry, and Ignore buttons
3	Display Yes, No, and Cancel buttons
4	Display Yes and No buttons
5	Display Retry and Cancel buttons
Group 2	
16	Stop Icon
32	Question Icon
48	Exclamation Icon
64	Information Icon
Group 3	
0	First button is default
256	Second button is default
512	Third button is default
Group 4	
768	Fourth button is default
0	Application modal

4096	System modal

The first group of values (1-5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512) determines which button is the default; and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to create a final value for the argument type, use only one number from each group. If omitted, the default value for type is 0.

title:

String expression displayed in the title bar of the dialog box. If you omit the argument title, MsgBox has no default title.

The value returned by the MsgBox function indicates which button has been selected, as shown below:

Value	Meaning
1	OK button selected.
2	Cancel button selected.
3	Abort button selected.
4	Retry button selected.
5	Ignore button selected.
6	Yes button selected.
7	No button selected.

If the dialog box displays a Cancel button, pressing the Esc key has the same effect as choosing Cancel.

MsgBox Function, MsgBox Statement Example

The example uses MsgBox to display a close without saving message in a dialog box with a Yes button a No button and a Cancel button. The Cancel button is the default response. The MsgBox function returns a value based on the button chosen by the user. The MsgBox statement uses that value to display a message that indicates which button was chosen.

Related Topics: InputBox, InputBox$ Function

Example:

```
Dim Msg, Style, Title, Help, Ctxt, Response, MyString
Msg = "Do you want to continue ?"   ' Define message.
'Style = vbYesNo + vbCritical + vbDefaultButton2   ' Define buttons.
Style = 4 + 16 + 256   ' Define buttons.
Title = "MsgBox Demonstration"   ' Define title.
Help = "DEMO.HLP"   ' Define Help file.
Ctxt = 1000   ' Define topic
```

```
' context.
' Display message.
Response = MsgBox(Msg, Style, Title, Help, Ctxt)
If Response = vbYes Then   ' User chose Yes.
MyString = "Yes"   ' Perform some action.
Else   ' User chose No.
MyString = "No"   ' Perform some action.
End If
```

Name Statement

Name *oldname* As *newname*

Changes the name of a directory or a file.

The parameters *oldname* and *newname* are strings that can optionally contain a path.

Related Topics: Kill, ChDir

Now Function

Now

Returns a date that represents the current date and time according to the setting of the computer's system date and time

The Now function returns a Variant data type containing a date and time that are stored internally as a double. The number is a date and time from January 1, 100 through December 31, 9999, where January 1, 1900 is 2. Numbers to the left of the decimal point represent the date and numbers to the right represent the time.

Example:

```
Sub Main ()
Dim Today
Today = Now
End Sub
```

Oct Function

Oct (*num*)

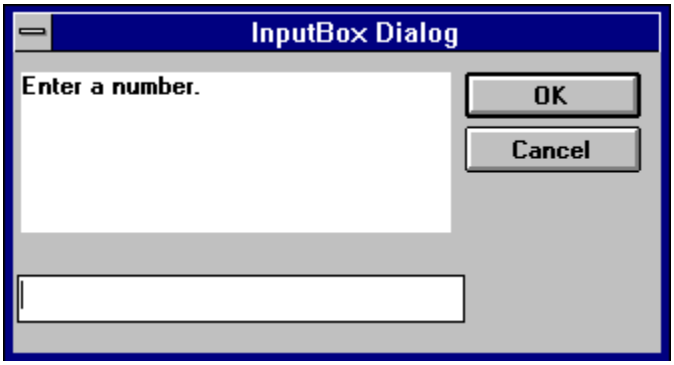
Returns the octal value of the decimal parameter

Oct returns a string

Related Topics: Hex

Example:

```
Sub Main ()
Dim Msg, Num                                ' Declare variables.
Num = InputBox("Enter a number.")           ' Get user input.
Msg = Num & " decimal is &O"
Msg = Msg & Oct(Num) & " in octal notation."
MsgBox Msg                                  ' Display results.
End Sub
```



OKButton

OKBUTTON starting x position, starting y position, width, Height

For selecting options and closing dialog boxes

Example:

```
Sub Main ()
Begin Dialog DialogName1 60, 60, 160, 70, "ASC - Hello"
TEXT 10, 10, 28, 12, "Name:"
TEXTBOX 42, 10, 108, 12, .nameStr
TEXTBOX 42, 24, 108, 12, .descStr
CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
OKBUTTON 42, 54, 40, 12
End Dialog
Dim Dlg1 As DialogName1
Dialog Dlg1

MsgBox Dlg1.nameStr
MsgBox Dlg1.descStr
MsgBox Dlg1.checkInt
End Sub
```

On Error

On Error { *GoTo line* | *Resume Next* | *GoTo 0* }

Enables error-handling routine and specifies the line label of the error-handling routine.

Related Topics: Resume

The line parameter refers to a label. That label must be present in the code or an error is generated.

Example:

```
Sub Main
```

```
On Error GoTo dude
Dim x as object
x.draw        ' Object not set
jpe           ' Undefined function call
print 1/0     ' Division by zero
Err.Raise 6                                ' Generate an "Overflow" error
MsgBox "Back"
MsgBox "Jack"
Exit Sub
dude:
MsgBox "HELLO"
Print Err.Number, Err.Description
Resume Next
MsgBox "Should not get here!"
MsgBox "What?"
End Sub
```

Errors can be raised with the syntax:

```
Err.Raise x
```

Defined x Value Descriptions

The list below shows the corresponding descriptions for the defined values of x:

```
3:  "Return without GoSub";
5:  "Invalid procedure call";
6:  "Overflow";
7:  "Out of memory";
9:  "Subscript out of range";
10: "Array is fixed or temporarily locked";
11: "Division by zero";
13: "Type mismatch";
14: "Out of string space";
16: "Expression too complex";
17: "Can't perform requested operation";
18: "User intrrupt occurred";
20: "Resume without error";
28: "Out of stack space";
35: "Sub, Function, or Property not defined";
47: "Too many DLL application clients";
48: "Error in loading DLL";
49: "Bad DLL calling convention";
```

```
51: "Internal error";
52: "Bad file name or number";
53: "File not found";
54: "Bad file mode";
55: "File already open";
57: "Device I/O error";
58: "File already exists";
59: "Bad record length";
60: "Disk full";
62: "Input past end of file";
63: "Bad record number";
67: "Too many files";
68: "Device unavailable";
70: "Permission denied";
71: "Disk not ready";
74: "Can't rename with different drive";
75: "Path/File access error";
76: "Path not found";
91: "Object variable or With block variable not set";
92: "For loop not initialized";
93: "Invalid pattern string";
94: "Invalid use of Null";
// OLE Automation Messages
429: "OLE Automation server cannot create object";
430: "Class doesn't support OLE Automation";
432: "File name or class name not found during OLE Automation operation";
438: "Object doesn't support this property or method";
440: "OLE Automation error";
443: "OLE Automation object does not have a default value";
445: "Object doesn't support this action";
446: "Object doesn't support named arguments";
447: "Object doesn't support current local setting";
448: "Named argument not found";
449: "Argument not optional";
450: "Wrong number of arguments";
451: "Object not a collection";
// Miscellaneous Messages
444: "Method not applicable in this context";
452: "Invalid ordinal";
453: "Specified DLL function not found";
457: "Duplicate Key";
460: "Invalid Clipboard format";
461: "Specified format doesn't match format of data";
480: "Can't create AutoRedraw image";
```

```
481: "Invalid picture";
482: "Printer error";
483: "Printer driver does not supported specified property";
484: "Problem getting printer information from from the system.";
// Make sure the printer is setp up correctly.
485: "invalid picture type";
520: "Can't empty Clipboard";
521: "Can't open Clipboard";
```

Open Statement

```
Open filename$ [For mode] [Access access] As [#]filenumber
```

Opens a file for input and output operations.

You must open a file before any I/O operation can be performed on it.

The Open statement has these parts:

Part	Description
file	File name or path.
*mod*e	Reserved word that specifies the file mode: **Append, Binary Input, Output**
Access	Reserved word that specifies which operations are permitted on the **open file: Read, Write.**
filenumber	Integer expression with a value between 1 and 255, inclusive. When an Open statement is executed, filenumber is associated with the file as long as it is open. Other I/O statements can use the number to refer to the file.

If file doesn't exist, it is created when a file is opened for Append, Binary or Output modes.

The argument mode is a reserved word that specifies one of the following file modes.

Mode	Description
Input	Sequential input mode.
Output.	Sequential output mode

Append Sequential output mode. Append sets the file pointer to the end of the file. A Print # or Write # statement then extends (appends to) the file.

The argument access is a reserved word that specifies the operations that can be performed on the opened file. If the file is already opened by another process and the specified type of access is not allowed, the Open operation fails and a Permission denied error occurs. The Access clause works only if you are using a version of MS-DOS that supports networking (MS-DOS version 3.1 or later). If you use the Access clause with a version of MS-DOS that doesn't support networking, a feature unavailable error occurs. The argument access can be one of the following reserved words.

Access type	Description
Read	Opens the file for reading only.
Write	Opens the file for writing only.
Read Write	Opens the file for both reading and riting. This mode is valid only for Random and Binary files and files opened for Append mode.

The following example writes data to a test file and reads it back.

Example:

```
Sub Main ()

Open "TESTFILE" For Output As #1            ' Open to write file.
userData1$ = InputBox("Enter your own text here")
userData2$ = InputBox("Enter more of your own text here")
Write #1, "This is a test of the Write # statement."
Write #1,userData1$, userData2
Close #1

Open "TESTFILE" for Input As #2             ' Open to read file.
Do While Not EOF(2)
Line Input #2, FileData                     ' Read a line of data.
PRint FileData                                 ' Construct message.

Loop
Close #2                                       ' Close all open files.
MsgBox "Testing Print Statement"            ' Display message.
Kill "TESTFILE"                             ' Remove file from disk.
End Sub
```

Option Base Statement

```
Option Base number
```

Declares the default lower bound for array subscripts.

The Option Base statement is never required. If used, it can appear only once in a module, it can occur only in the Declarations section, and must be used before you declare the dimensions of any arrays.

The value of number must be either 0 or 1. The default base is 0.

The To clause in the Dim, Global, and Static statements provides a more flexible way to control the range of an array's subscripts. However, if you don't explicitly set the lower bound with a To clause, you can use Option Base to change the default lower bound to 1.

The example uses the Option Base statement to override the default base array subscript value of 0.

Related Topics: Dim, Global and Lbound Statements

Example:

```
Option Base 1                                ' Module level statement.
Sub Main
Dim A(), Msg, NL                             ' Declare variables.
NL = Chr(10)                                 ' Define newline.
ReDim A(20)                                  ' Create an array.
Msg = "The lower bound of the A array is " & LBound(A) & "."
Msg = Msg & NL & "The upper bound is " & UBound(A) & "."
MsgBox Msg                                   ' Display message.
End Sub
```

Option Explicit Statement

```
Option Explicit
```

Forces explicit declaration of all variables.

The Option explicit statement is used outside of the script in the declarations section. This statement can be contained in a declare file or outside of any script in a file or buffer. If this statement is contained in the middle of a file the rest of the compile buffer will be affected.

Related Topics: Const and Global Statements

Example:

```
Option Explicit
Sub Main
Print y    ‘because y is not explicitly dimmed an error will occur.

End Sub
```

Print Method

Print [*expr, expr...*] Print a string to an object.

Example:

```
Sub PrintExample ()
Dim Msg, Pi                                        ' Declare variables.
Let Pi = 4 * _Atn(1)                          ' Calculate Pi.
Msg = "Pi is equal to " & Str(Pi)
MsgBox Msg                                                ' Display results.
Print Pi                                              ‘Pints the results in the
' compiler messages window
End Sub
```

Print # Statement

Print # *filenumber*, [[{Spc(*n*) | Tab(*n*)}][*expressionlist*] [{; | ,}]]

Writes data to a sequential file.

Print statement Description:

filenumber:

Number used in an Open statement to open
a sequential file. It can be any

number of an open file. Note that the

number sign (#) preceding filenumber is not optional.

Spc(*n*):

Name of the Basic function optionally used to insert *n* spaces into the printed

output. Multiple use is permitted.

Tab(*n*):

Name of the Basic function optionally used to tab to the *nth* column before printing

expressionlist. Multiple use is permitted.

expressionlist :

Numeric and/or string expressions to be written to the file.

{;|,}

Character that determines the position of the next character printed. A semicolon means the next character is printed immediately after the last character; a comma means the next character is printed at the start of the next print zone. Print zones begin every 14 columns. If neither character is specified, the next character is printed on the next line.

If you omit expressionlist, the Print # statement prints a blank line in the file, but you must include the comma. Because Print # writes an image of the data to the file, you must delimit the data so it is printed correctly. If you use commas as delimiters, Print # also writes the blanks between print fields to the file.

The Print # statement usually writes Variant data to a file the same way it writes any other data type. However, there are some exceptions:

If the data being written is a Variant of VarType 0 (Empty), Print # writes nothing to the file for that data item.

If the data being written is a Variant of VarType 1 (Null), Print # writes the literal #NULL# to the file.

If the data being written is a Variant of VarType 7 (Date), Print # writes the date to the file using the Short Date format defined in the WIN.INI file. When either the date or the time component is missing or zero, Print # writes only the part provided to the file.

The following example writes data to a test file.

Example:

```
Sub Main
Dim I, FNum, FName                          ' Declare variables.
For I = 1 To 3
FNum = FreeFile                             ' Determine next file number.
FName = "TEST" & FNum
Open FName For Output As FNum               ' Open file.
Print #I, "This is test #" & I              ' Write string to file.
Print #I, "Here is another ";  "line";  I
Next I
Close                                       ' Close all files.
End Sub

The following example writes data to a test file and reads it back.

Sub Main ()
Dim FileData, Msg, NL                       ' Declare variables.
NL = Chr(10)                                ' Define newline.
Open "TESTFILE" For Output As #1            ' Open to write file.
Print #2, "This is a test of the Print # statement."
Print #2                                       ' Print blank line to file.
Print #2, "Zone 1", "Zone 2"                ' Print in two print zones.
Print #2, "With no space between"  ;  "." ' Print two strings together.
Close
Open "TESTFILE" for Input As #2             ' Open to read file.
Do While Not EOF(2)
Line Input #2, FileData                     ' Read a line of data.
Msg = Msg & FileData & NL                   ' Construct message.
MsgBox Msg
Loop
Close                                          ' Close all open files.
MsgBox "Testing Print Statement"            ' Display message.
Kill "TESTFILE"                             ' Remove file from disk.
End Sub
```

Randomize Statement

```
Randomize[number]
```

Used to Initialize the random number generator.

The Randomize statement has one optional parameter *number*. This parameter can be any valid number and is used to initialize the random number generator. If you omit the parameter then the value returned by the Timer function is used as the default parameter to seed the rando number generator.

Example:

```
Sub Main

Dim MValue

Randomize   ' Initialize random-number generator.
MValue = Int((6 * Rnd) + 1)
Print MValue

End Sub
```

ReDim Statement

```
ReDim varname(subscripts)[As Type][,varname(subscripts)]
```

Used to declare dynamic arrays and reallocate storage space.

The ReDim statement is used to size or resize a dynamic array that has already been declared using the Dim statement with empty parentheses. You can use the ReDim statement to repeatedly change the number of elements in and array but not to change the number of dimensions in an array or the type of the elements in the array.

Example:

```
Sub Main

Dim TestArray() As Integer
Dim I
ReDim TestArray(10)
For I = 1 To 10
TestArray(I) = I + 10
Print TestArray(I)
Next I
```

```
End Sub
```

Rem Statement

```
Rem remark ‘remark
```

Used to include explanatory remarks in a program.

The parameter *remark* is the text of any comment you wish to include in the code.

Example:

```
Rem This is a remark

Sub Main()

Dim Answer, Msg                                ' Declare variables.
Do
Answer = InputBox("Enter a value from 1 to 3.")
Answer = 2
If Answer >= 1 And Answer <= 3 Then              ' Check range.
Exit Do                                    ' Exit Do...Loop.
Else
Beep                                       ' Beep if not in range.
End If
Loop
MsgBox "You entered a value in the proper range."
End Sub
```

Right Function

Right (*stringexpression, n*)

Returns the right most n characters of the string parameter.

The parameter *stringexpression* is the string from which the rightmost characters are returned.

The parameter *n* is the number of characters that will be returned and must be a long integer.

Related Topics: Len, Left, Mid Functions.

Example:

```
' The example uses the Right function to return the first of two words
' input by the user.

Sub Main ()
Dim LWord, Msg, RWord, SpcPos, UsrInp        ' Declare variables.
Msg = "Enter two words separated by a space."
UsrInp = InputBox(Msg)                       ' Get user input.
print UsrInp
SpcPos = InStr(1, UsrInp, " ")               ' Find space.
If SpcPos Then
LWord = Left(UsrInp, SpcPos - 1)             ' Get left word.
print "LWord: ";  LWord
RWord = Right(UsrInp, Len(UsrInp) - SpcPos)  ' Get right word.
Msg = "The first word you entered is " & LWord
Msg = Msg & "." & " The second word is "
Msg = "The first word you entered is <" & LWord & ">"
Msg = Msg & RWord & "."
Else
Msg = "You didn't enter two words."
End If
MsgBox Msg                                   ' Display message.
End Sub
```

RmDir Statement

RmDir *path*

Removes an existing directory.

The parameter *path* is a string that is the name of the directory to be removed.

Related Topics: ChDir, CurDir

Example:

```
' This sample shows the functions mkdir (Make Directory)
' and rmdir (Remove Directory)

Sub Main
Dim dirName As String
```

```
dirName = "t1"
mkdir dirName
mkdir "t2"
MsgBox "Directories: t1 and t2 created. Press OK to remove them"
rmdir "t1"
rmdir "t2"
End Sub
```

Rnd Function

Rnd (*number*)

Returns a random number.

The parameter *number* must be a valid numeric expression.

Example:

```
'Rnd Function Example

'The example uses the Rnd function to simulate rolling a pair of dice by
'generating random values from 1 to 6. Each time this program is run,


Sub Main ()
Dim Dice1, Dice2, Msg                       ' Declare variables.
Dice1 = CInt(6 * Rnd() + 1)                 ' Generate first die value.
Dice2 = CInt(6 * Rnd() + 1)                 ' Generate second die value.
Msg = "You rolled a " & Dice1
Msg = Msg & " and a " & Dice2
Msg = Msg & " for a total of "
Msg = Msg & Str(Dice1 + Dice2) & "."
MsgBox Msg                                  ' Display message.
End Sub
```

Second Function

Second (*number*)

Returns an integer that is the second portion of the minute in the time parameter.

The parameter *number* must be a valid numeric expression.

Related Topics: Day, Hour, Minute, Now.

Example:

```
' Format Function Example

' This example shows various uses of the Format function to format values
' using both named and user-defined formats. For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running. When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used. When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale. For this example, English/United States is assumed.

' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main

MyTime = "08:04:23 PM"
MyDate = "03/03/95"
MyDate = "January 27, 1993"

MsgBox Now
MsgBox MyTime

MsgBox Second( MyTime ) & " Seconds"
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

' Returns current system time in the system-defined long time format.
MsgBox Format(Time, "Short Time")
MyStr = Format(Time, "Long Time")
```

```
' Returns current system date in the system-defined long date format.
MsgBox Format(Date, "Short Date")
MsgBox Format(Date, "Long Date")

'This section not yet supported
MsgBox Format(MyTime, "h:n:s")        ' Returns "17:4:23".
MsgBox Format(MyTime, "hh:nn:ss")' Returns "05:04:23".
MsgBox Format(MyDate, "dddd, mmm d yyyy")' Returns "Wednesday, Jan 27 1993".

' If format is not supplied, a string is returned.
MsgBox Format(23)                      ' Returns "23".

' User-defined formats.
MsgBox Format(5459.4, "##,##0.00")      ' Returns "5,459.40".
MsgBox Format(334.9, "###0.00")         ' Returns "334.90".
MsgBox Format(5, "0.00%")               ' Returns "500.00%".
MsgBox Format("HELLO", "<")             ' Returns "hello".
MsgBox Format("This is it", ">")        ' Returns "THIS IS IT".

End Sub
```

Seek Function

Seek (*filenumber*)

The parameter *filenumber* is used in the open statement and must be a valid numeric expression.

Seek returns a number that represents the byte position where the next operation is to take place. The first byte in the file is at position 1.

Related Topics: Open

Example:

```
Sub Main
Open "TESTFILE" For Input As #1             ' Open file for reading.
Do While Not EOF(1)                            ' Loop until end of file.
MyChar = Input(1, #1)                       ' Read next character of data.
Print Seek(1)                                  ' Print byte position  .
Loop
```

```
Close #1                                        ' Close file.
End Sub
```

Seek Statement

Seek *filenumber, position*

The parameter *filenumber* is used in the open statement and must be a valid numeric expression, the parameter *position* is the number that indicates where the next read or write is to occur. In Cypress Enable Basic position is the byte position relative to the beginning of the file.

Seek statement sets the position in a file for the next read or write

Related Topics: Open

Example:

```
Sub Main
Open "TESTFILE" For Input As #1             ' Open file for reading.
For i = 1 To 24 Step 3                          ' Loop until end of file.
Seek #1, i                                      ' Seek to byte position
MyChar = Input(1, #1)                       ' Read next character of data.
Print MyChar                                    'Print character of data
Next i
Close #1                                            ' Close file.
End Sub
```

Select Case Statement

Executes one of the statement blocks in the case based on the test variable

```
Select Case testvar
Case var1
Statement Block
Case var2
Statement Block
Case Else
Statement Block
End Select
```

The syntax supported by the Select statement includes the “To” keyword, a coma delimited list and a constant or variable.

Select Case Number ' Evaluate Number.

Case 1 To 5 ' Number between 1 and 5, inclusive.

…

' The following is the only Case clause that evaluates to True.

Case 6, 7, 8 ' Number between 6 and 8.

…

Case 9 To 10 ' Number is 9 or 10.

…

Case Else ' Other values.

…

End Select

Related Topics: If...Then...Else

Example:

```
' This rather tedious test shows nested select statements and if uncommented,
' the exit for statement

Sub Test ()
For x = 1 to 5
print x
Select Case x
Case 2
Print "Outer Case Two"
Case 3
Print "Outer Case Three"
'            Exit For
Select Case x
Case 2
Print "Inner Case Two"
Case 3
Print "Inner Case Three"
'               Exit For
Case Else                                  ' Must be something else.
Print "Inner Case Else:", x
End Select
```

```
Print "Done with Inner Select Case"
Case Else                                ' Must be something else.
Print "Outer Case Else:",x
End Select
Next x
Print "Done with For Loop"
End Sub
```

SendKeys Function

SendKeys (*Keys, [wait]*)

Sends one or more keystrokes to the active window as if they had been entered at the keyboard

The SendKeys statement has two parameters. The first parameter *keys* is a string and is sent to the active window. The second parameter *wait* is optional and if omitted is assumed to be false. If wait is true the keystrokes must be processed before control is returned to the calling procedure.

Example:

```
Sub Main ()
Dim I, X, Msg                            ' Declare variables.
X = Shell("Calc.exe", 1)                 ' Shell Calculator.
For I = 1 To 5                           ' Set up counting loop.
SendKeys I & "{+}", True                 ' Send keystrokes to Calculator
Next I                                   ' to add each value of I.
AppActivate "Calculator"                 ' Return focus to Calculator.
SendKeys "%{F4}", True                   ' Alt+F4 to close Calculator.
End Sub
```

Set Statement

Set *Object* = {[New] *objectexpression* | Nothing}

Assigns an object to an object variable.

Related Topics: Dim, Global, Static

Example:

```
Sub Main
```

```
Dim visio As Object
Set visio = CreateObject( "visio.application" )
Dim draw As Object
Set draw = visio.Documents
draw.Open "c:\visio\drawings\Sample1.vsd"
MsgBox "Open docs: " & draw.Count
Dim page As Object
Set page = visio.ActivePage
Dim red As Object
Set red = page.DrawRectangle (1, 9, 7.5, 4.5)
red.FillStyle = "Red fill"

Dim cyan As Object
Set cyan = page.DrawOval (2.5, 8.5, 5.75, 5.25)
cyan.FillStyle = "Cyan fill"

Dim green As Object
Set green = page.DrawOval (1.5, 6.25, 2.5, 5.25)
green.FillStyle = "Green fill"

Dim DarkBlue As Object
set DarkBlue = page.DrawOval (6, 8.75, 7, 7.75)
DarkBlue.FillStyle = "Blue dark fill"

visio.Quit
End Sub
```

Shell Function

Shell (*app [, style]*)

Runs an executable program.

The shell function has two parameters. The first one, *app* is the name of the program to be executed. The name of the program in *app* must include a .PIF, .COM, .BAT, or .EXE file extension or an error will occur. The second argument, *style* is the number corresponding to the style of the window . It is also optional and if omitted the program is opened minimized with focus.

Window styles:

Normal with focus 1,5,9

Minimized with focus (default) 2

Maximized with focus 3

normal without focus 4,8

minimized without focus 6,7

Return value: ID, the task ID of the started program.

Example:

```
' This example uses Shell to leave the current application and run the
' Calculator program included with Microsoft Windows; it then
' uses the SendKeys statement to send keystrokes to add some numbers.

Sub Main ()
Dim I, X, Msg                          ' Declare variables.
X = Shell("Calc.exe", 1)               ' Shell Calculator.
For I = 1 To 5                         ' Set up counting loop.
SendKeys I & "{+}", True               ' Send keystrokes to Calculator
Next I                                 ' to add each value of I.
AppActivate "Calculator"               ' Return focus to Calculator.
SendKeys "%{F4}", True                 ' Alt+F4 to close Calculator.
End Sub
```

Sin Function

Sin (*rad*)

Returns the sine of an angle that is expressed in radians

Example:

```
Sub Main ()
pi = 4 * Atn(1)
```

```
rad = 90 * (pi/180)
x = Sin(rad)
print x
End Sub
```

Space Function

Space[$] (*number*)

Skips a specified number of spaces in a print# statement.

The parameter number can be any valid integer and determines the number of blank spaces.

Example:

```
' This sample shows the space function

Sub Main

MsgBox "Hello" & Space(20) & "There"

End Sub
```

Sqr Function

Sqr(*num*)

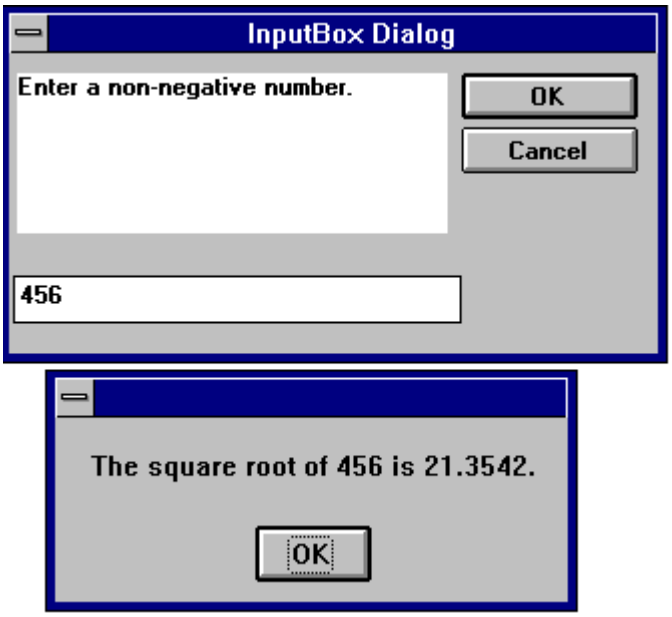
Returns the square root of a number.

The parameter *num* must be a valid number greater than or equal to zero.

Example:

```
Sub Form_Click ()
Dim Msg, Number                          ' Declare variables.
Msg = "Enter a non-negative number."
Number = InputBox(Msg)                   ' Get user input.
If Number < 0 Then
```

```
Msg = "Cannot determine the square root of a negative number."
Else
Msg = "The square root of " & Number & " is "
Msg = Msg & Sqr(Number) & "."
End If
MsgBox Msg                                    ' Display results.

End Sub
```



Static Statement

Static *variable*

Used to declare variables and allocate storage space. These variables will retain their value through the program run

Related Topics: Dim, Function, Sub

Example:

```
' This example shows how to use the static keyword to retain the value of
' the variable i in sub Joe. If Dim is used instead of Static then i
' is empty when printed on the second call as well as the first.

Sub Main
```

```
For i = 1 to 2
Joe 2
Next i
End Sub

Sub Joe( j as integer )
Static i
print i
i = i + 5
print i
End Sub
```

Stop Statement

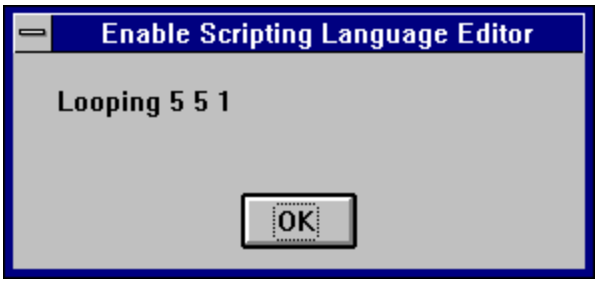
```
Stop
```

Ends execution of the program

The Stop statement can be placed anywhere in your code.

Example:

```
Sub main ()

Dim x,y,z

For x = 1 to 5
For y = 1 to 5
For z = 1 to 5
Print "Looping" ,z,y,x
Next z
Next y
Stop
Next x
End Sub
```



Str Function

Str(*numericexpr*)

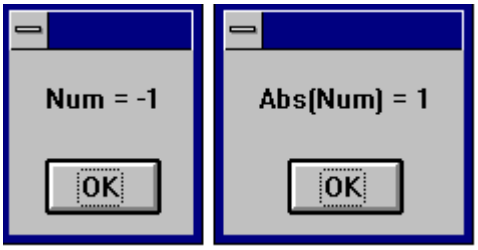
Returns the value of a numeric expression.

Str returns a String.

Related topics: Format, Val

Example:

```
Sub main ()
Dim msg
a = -1
msgBox "Num = " & Str(a)
MsgBox "Abs(Num) =" & Str(Abs(a))

End Sub
```



StrComp Function

StrComp(*nstring1,string2, [compare]*)

Returns a variant that is the result of the comparison of two strings

Example:

```
Sub Main

Dim MStr1, MStr2, MComp
MStr1 = "ABCD": MStr2 = "today"             ' Define variables.
print MStr1, MStr2
MComp = StrComp(MStr1, MStr2)               ' Returns -1.
print MComp
MComp = StrComp(MStr1, MStr2)               ' Returns -1.
print MComp
MComp = StrComp(MStr2, MStr1)               ' Returns 1.
print MComp
End Sub
```

String Function

String (*numeric*, *charcode*)

String returns a string.

String is used to create a string that consists of one character repeated over and over.

Related topics: Space Function

Example:

```
Sub Main

Dim MString
MString = String(5, "*")                    ' Returns "*****".
MString = String(5, 42)                     ' Returns "44444".
MString = String(10, "Today")               ' Returns "TTTTTTTTTT".
Print MString
End Sub
```

Sub Statement

Sub *SubName* [(*arguments*)]

Dim [*variable*(s)]

[*statementblock*]

[Exit Function]

End Sub

Declares and defines a Sub procedures name, parameters and code.

When the optional argument list needs to be passed the format is as follows:

```
([ByVal] variable [As type] [,ByVal] variable [As type] ]…])
```

The optional ByVal parameter specifies that the variable is [passed by value instead of by reference (see “ByRef and ByVal” in this manual). The optional As type parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Varaint (see “Variable Types” in this manual).

Related Topics: Call, Dim, Function

Example:

```
Sub Main
Dim DST As String

DST = "t1"
mkdir DST
mkdir "t2"
End Sub
```

Tan Function

Tan(*angle*)

Returns the tangent of an angle as a double.

The parameter *angle* must be a valid angle expressed in radians.

Related Topic: Atn, Cos, Sin

Example:

```
' This sample program show the use of the Tan function

Sub Main ()
Dim Msg, Pi         ' Declare variables.
Pi = 4 * Atn(1)     ' Calculate Pi.
Msg = "Pi is equal to " & Pi
MsgBox Msg          ' Display results.
x = Tan(Pi/4)
MsgBox  x & " is the tangent of Pi/4"
End Sub
```

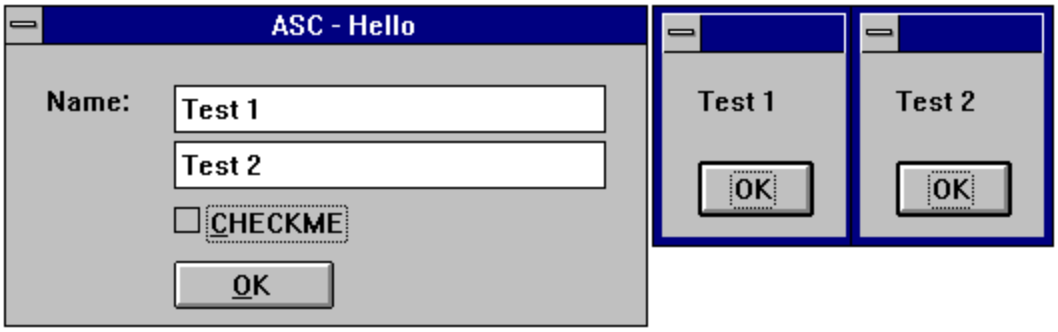
Text Statement

```
Text Starting X position, Starting Y position, Width, Height, Label
```

Creates a text field for titles and labels.

Example:

```
Sub Main ()
Begin Dialog DialogName1 60, 60, 160, 70, "ASC - Hello"
TEXT 10, 10, 28, 12, "Name:"
TEXTBOX 42, 10, 108, 12, .nameStr
TEXTBOX 42, 24, 108, 12, .descStr
CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
OKBUTTON 42, 54, 40, 12
End Dialog
Dim Dlg1 As DialogName1
Dialog Dlg1

MsgBox Dlg1.nameStr
MsgBox Dlg1.descStr
MsgBox Dlg1.checkInt
End Sub
```



TextBox Statement

```
TextBox Starting X position, Starting Y position, Width, Height, Default String
```

Creates a Text Box for typing in numbers and text

Example:

```
Sub Main ()
Begin Dialog DialogName1 60, 60, 160, 70, "ASC - Hello"
TEXT 10, 10, 28, 12, "Name:"
TEXTBOX 42, 10, 108, 12, .nameStr
TEXTBOX 42, 24, 108, 12, .descStr
CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
OKBUTTON 42, 54, 40, 12
End Dialog
Dim Dlg1 As DialogName1
Dialog Dlg1

MsgBox Dlg1.nameStr
MsgBox Dlg1.descStr
MsgBox Dlg1.checkInt
End Sub
```

Time Function

```
Time[()]
```

Returns the current system time.

Related topics: To set the time use the TIME$ statement.

Example:

```
Sub Main
x = Time$(Now)
Print x
End Sub
```

Timer Event

```
Timer
```

Timer Event is used to track elapsed time or can be display as a stopwatch in a dialog. The timers value is the number of seconds from midnight.

Related topics: DateSerial, DateValue, Hour, Minute, Now, Second, TimeValue.

Example:

```
Sub Main

Dim TS As Single
Dim TE As Single
Dim TEL As Single


TS = Timer

MsgBox "Starting Timer"

TE = Timer

TT = TE - TS
Print TT


End Sub
```

TimeSerial - Function

```
TimeSerial ( hour, minute, second   )
```

Returns the time serial for the supplied parameters *hour*, *minute*, *second*.

Related topics: DateSerial, DateValue, Hour, Minute, Now, Second, TimeValue.

Example:

```
Sub Main

Dim MTime
MTime = TimeSerial(12, 25, 27)
Print MTime

End Sub
```

TimeValue - Function

```
TimeValue ( TimeString  )
```

Returns a double precision serial number based of the supplied string parameter.

```
Midnight = TimeValue(“23:59:59”)
```

Related topics: DateSerial, DateValue, Hour, Minute, Now, Second, TimeSerial.

Example:

```
Sub Main

Dim MTime
MTime = TimeValue("12:25:27 PM")
Print MTime

End Sub
```

Trim, LTrim, RTrim Functions

[L| R] Trim (*String*)

Ltrim, Rtrim and Trim all Return a copy of a string with leading, trailing or both leading and trailing spaces removed.

Ltrim, Rtrim and Trim all return a string

Ltrim removes leading spaces.

Rtrim removes trailing spaces.

Trim removes leading and trailing spaces.

Example:

```
' This example uses the LTrim and RTrim functions to strip leading and
' trailing spaces, respectively, from a string variable. It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Sub Main
MyString = "  <-Trim->  "                   ' Initialize string.
TrimString = LTrim(MyString)                ' TrimString = "<-Trim->  ".
MsgBox "|" & TrimString & "|"
TrimString = LCase(RTrim(MyString))         ' TrimString = "  <-trim->".
MsgBox "|" & TrimString & "|"
TrimString = LTrim(RTrim(MyString))         ' TrimString = "<-Trim->".
MsgBox "|" & TrimString & "|"
' Using the Trim function alone achieves the same result.
TrimString = UCase(Trim(MyString))          ' TrimString = "<-TRIM->".
MsgBox "|" & TrimString & "|"
End Sub
```

Type Statement

```
Type usertype                                 elementname As typename
[ elementname As typename]
. . .
```

```
End Type
```

Defines a user-defined data type containing one or more elements.

The **Type** statement has these parts:

Part	Description
Type	Marks the beginning of a user-defined type.
usertype	Name of a user-defined data type. It follows standard variable naming conventions.
elementname	Name of an element of the user-defined data type. It follows standard variable-naming conventions.
subscripts	Dimensions of an array element. You can declare multiple dimensions. (**not currently implemented**)
typename	One of these data types: Integer, Long, Single, Double, String (for variable-length strings), String * length (for fixed-length strings), Variant, or another user-defined type. The argument typename can't be an object type. End Type Marks the end of a user-defined type.

Once you have declared a user-defined type using the Type statement, you can declare a variable of that type anywhere in your script. Use Dim or Static to declare a variable of a user-defined type. Line numbers and line labels aren't allowed in Type...End Type blocks.

User-defined types are often used with data records because data records frequently consist of a number of related elements of different data types. Arrays cannot be an element of a user defined type in Enable.

Example:

```
' This sample shows some of the features of user defined types

Type type1
a As Integer
d As Double
s As String
End Type

Type type2
a As String
o As type1
End Type

Type type3
b As Integer
c As type2
End Type
```

```
Dim type2a As type2
Dim type2b As type2
Dim type1a As type1
Dim type3a as type3

Sub Form_Click ()
a = 5
type1a.a = 7472
type1a.d = 23.1415
type1a.s = "YES"
type2a.a = "43 - forty three"
type2a.o.s = "Yaba Daba Doo"
type3a.c.o.s = "COS"
type2b.a = "943 - nine hundred and forty three"
type2b.o.s = "Yogi"
MsgBox type1a.a
MsgBox type1a.d
MsgBox type1a.s
MsgBox type2a.a
MsgBox type2a.o.s
MsgBox type2b.a
MsgBox type2b.o.s
MsgBox type3a.c.o.s
MsgBox a
End Sub
```

UBound Function

Ubound(*arrayname[,dimension*])

Returns the value of the largest usable subscript for the specified dimension of an array.

Related Topics: Dim, Global, Lbound, and Option Base

Example:

```
' This example demonstrates some of the  features of arrays. The lower bound
' for an array is 0 unless it is specified or option base is set it as is
' done in this example.
```

```
Option Base 1

Sub Main
Dim a(10) As Double
MsgBox "LBound: " & LBound(a) & " UBound: " & UBound(a)
Dim i As Integer
For i = 1 to 3
a(i) = 2 + i
Next i
Print a(1),a(1),a(2), a(3)
End Sub
```

UCase Function

Ucase (*String*)

Returns a copy of *String* in which all lowercase characters have been converted to uppercase.

Related Topics: Lcase, Lcase$ Function

Example:

```
' This example uses the LTrim and RTrim functions to strip leading and
' trailing spaces, respectively, from a string variable. It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Sub Main
MyString = "  <-Trim->  "                   ' Initialize string.
TrimString = LTrim(MyString)                ' TrimString = "<-Trim->  ".
MsgBox "|" & TrimString & "|"
TrimString = LCase(RTrim(MyString))         ' TrimString = "  <-trim->".
MsgBox "|" & TrimString & "|"
TrimString = LTrim(RTrim(MyString))         ' TrimString = "<-Trim->".
MsgBox "|" & TrimString & "|"
' Using the Trim function alone achieves the same result.
TrimString = UCase(Trim(MyString))          ' TrimString = "<-TRIM->".
MsgBox "|" & TrimString & "|"
End Sub
```

Val

Val(*string*)

Returns the numeric value of a string of characters.

Example:

```
Sub main
Dim Msg
Dim YourVal As Double
YourVal = Val(InputBox$("Enter a number"))
Msg = "The number you enered is: " & YourVal
MsgBox Msg
End Sub
```

VarType

VarType(*varname*)

Returns a value that indicates how the parameter *varname* is stored internally.

The parameter *varname* is a variant data type.

VarType	return values:
Empty	0
Null	1
Integer	2
Long	3
Single	4
Double	5
Currency	6 (not available at this time)
Date/Time	7
String	8

Related Topics: IsNull, IsNumeric

Example:

```
If VarType(x) = 5 Then Print "Vartype is Double"   'Display variable type
```

Weekday Function

Weekday(*date,firstdayof week*)

Returns a integer containing the whole number for the weekday it is representing.

Related Topics: Hour, Second, Minute, Day

Example:

```
Sub Main

x = Weekday(#5/29/1959#)
Print x

End Sub
```

While...Wend Statement

While condition
.
.
.
[StatementBlock]
.
.
.
Wend

While begins the while...Wend flow of control structure. Condition is any numeric or expression that evaluates to true or false. If the condition is true the statements are executed. The statements can be any number of valid Enable Basic statements. Wend ends the While...Wend flow of control structure.

Related Topics: Do...Loop Statement

Example:

```
Sub Main
Const Max = 5
Dim A(5) As String
A(1) = "Programmer"
A(2) = "Engineer"
A(3) = "President"
A(4) = "Tech Support"
A(5) = "Sales"
Exchange = True

While Exchange
Exchange = False
For I = 1 To Max
MsgBox A(I)
Next                                    I
Wend
```

With Statement

With object
[STATEMENTS]
End With

The With statement allows you to proeform a series of commands or statements on a particular object without again refering to the name of that object. With statements can be nested by putting one With block within another With block. You will need to fully specify any object in an inner With block to any memeber of an object in an outer With block.

Related Topics: While Statement and Do Loop

Example:

```
' This sample shows some of the features of user defined types and the with
' statement

Type type1
a As Integer
d As Double
s As String
End Type
```

```
Type type2
a As String
o As type1
End Type

Dim type1a As type1
Dim type2a As type2

Sub Main ()

With type1a
.a = 65
.d = 3.14
End With
With type2a
.a = "Hello, world"
With .o
.s = "Goodbye"
End With
End With
type1a.s = "YES"
MsgBox type1a.a
MsgBox type1a.d
MsgBox type1a.s
MsgBox type2a.a
MsgBox type2a.o.s

End Sub
```

Write # - Statement

Write #*filenumber* [,*parameterlist*]

Writes and formats data to a sequential file that must be opened in output or append mode.

A comma delimited list of the supplied parameters is written to the indicated file. If no parameters are present, the newline character is all that will be written to the file.

Related Topics: Open and Print# Statements

Example:

```
Sub Main ()

Open "TESTFILE" For Output As #1            ' Open to write file.
userData1$ = InputBox ("Enter your own text here")
userData2$ = InputBox ("Enter more of your own text here")
Write #1, "This is a test of the Write # statement."
Write #1,userData1$, userData2
Close #1

Open "TESTFILE" for Input As #2             ' Open to read file.
Do While Not EOF(2)
Line Input #2, FileData                     ' Read a line of data.
PRint FileData                                 ' Construct message.

Loop
Close #2                                       ' Close all open files.
MsgBox "Testing Print Statement"            ' Display message.
Kill "TESTFILE"                             ' Remove file from disk.
End Sub
```

Year Function

Year(*serial#*)

Returns an integer representing a year between 1930 and 2029, inclusive. The returned integer represents the year of the serial parameter.

The parameter *serial#* is a string that represents a date.

If *serial* is a Null, this function returns a Null.

Related Topics: Date, Date$ Function/Statement, Day, Hour, Month, Minute, Now, Second.

Example:

```
Sub Main
MyDate = "11/11/94"
x = Year(MyDate)
print x
End Sub
PC-DMIS
```

Automation

Introduction

This section contains a detailed list of methods and properties for PC-DMIS Automation Objects. The various objects are listed in alphabetical order.

A **bold** item is the default property or method for this object.

Note: For information on when to use or ommit parentheses, please refer to your Basic Language documentation.

Active Tip Object Overview

The Active Tip object gives access to the properties of the PC-DMIS Set Active Tip command.

? Active Tip Members

Active Tip Members

Properties:

ActiveTip.Angle

DOUBLE value representing the rotation angle of the tip transformation matrix.

Read/Write **Double**

? Active Tip Object Overview

ActiveTip.TipID

STRING value representing the ID of the tip to be made active.

Read/Write **String**

? Active Tip Object Overview

Methods:

ActiveTip.GetShankVector

Syntax:

expression.GetOrigin (I, J, K)

Return Value: *Boolean* value representing whether the call successfully retrieved the values or not.

expression: Required expression that evaluates to a PC-DMIS **ActiveTip** object.

I: Required **Long** variable that receives the I component of the shank vector.

J: Required **Long** variable that receives the J component of the shank vector.

K: Required **Long** variable that receives the K component of the shank vector.

? Active Tip Object Overview

ActiveTip.SetShankVector

Syntax:

expression.SetOrigin (I, J, K)

Return Value: *Boolean* value representing whether the call successfully set the shank vector values.

expression: Required expression that evaluates to a PC-DMIS **ActiveTip** object.

I: Required **Long** used to set the I component of the shank vector.

J: Required **Long** used to set the J component of the shank vector.

K: Required **Long** used to set the K component of the shank vector.

? Active Tip Object Overview

AlignCommand Object Overview

Objects of type **AlignCommand** are created from more generic **Command** objects to pass alignment information back and forth.

? Command.AlignmentCommand

? AlignmentCommand Members

AlignCommand Members

Properties:

AlignCommand.Angle

Represents the offset angles of a 3D or 2D alignment. Read/write **PointData**. If used on an object other than a 3D or 2D alignment, setting this variable will do nothing, and getting this variable will return **Nothing**.

? Command.Type Property

? PointData Overview

? AlignmentCommand Object Overview

AlignCommand.AboutAxis

Represents the axis about which the alignment object rotates. Read/write **Long**.

Remarks

This function only works for objects of type ROTATE_ALIGN, ROTATE_CIRCLE_ALIGN, and ROTATEOFF_ALIGN. For other object types, trying to set this property does nothing, and trying to get this property always returns PCD_ZPLUS.

Valid Settings to set this property to are as follows:

```
PCD_XPLUS
PCD_XMINUS
PCD_YPLUS
PCD_YMINUS
PCD_ZPLUS
PCD_ZMINUS
```

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.AverageError

Represents whether or not error averaging is used during the iterative alignment. Read/write **Boolean**.

Remarks

This property is only valid for objects of type ITER_ALIGN. For other objects, getting this property always returns FALSE, and setting it does nothing.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.Axis

Represents the axis that the alignment object uses. Read/write **Long**.

Remarks

This function only works for objects of type ROTATE_ALIGN, ROTATE_CIRCLE_ALIGN, TRANS_ALIGN, and TRANSOFF_ALIGN. For other object types, trying to set this property does nothing, and trying to get this property always returns PCD_ZPLUS.

Valid Settings to set this property to are as follows:

```
PCD_XPLUS
PCD_XMINUS
PCD_YPLUS
PCD_YMINUS
PCD_ZPLUS
PCD_ZMINUS
```

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.BFOffset

Represents the offsets of a 3D or 2D alignment. Read/write **PointData**. If used on an object other than a 3D or 2D alignment, setting this variable will do nothing, and getting this variable will return **Nothing**.

? Command.Type Property

? PointData Overview

? AlignmentCommand Object Overview

AlignCommand.CadToPartMatrix

Represents the matrix used to transform points between the cad and part alignment systems. Read only **DmisMatrix**.

If used on an object other than a start alignment or a recall alignment, the identity matrix will be returned.

? Command.Type Property

? DmisMatrix Overview

? AlignmentCommand Object Overview

AlignCommand.ExternalID

Represents the external ID. Read/write **String**.

Remarks

This function only works for objects of type RECALL_ALIGN and SAVE_ALIGN. If used on an object other than a RECALL_ALIGN or SAVE_ALIGN, setting this variable will do nothing, and getting this variable will return the empty string.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.FeatID

Represents the first (or only) feature ID used by this alignment object. Read/write **String**.

Remarks

This function only works for objects of type LEVEL_ALIGN, ROTATE_ALIGN, ROTATE_CIRCLE_ALIGN, TRANS_ALIGN, and EQUATE_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return the empty string.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.FeatID2

Represents the second feature ID used by this alignment object. Read/write **String**.

Remarks

This function only works for objects of type ROTATE_CIRCLE_ALIGN and EQUATE_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return the empty string.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.FindCad

Represents the Find Cad property status of this best fit alignment object. Read/write **Boolean**.

Remarks

This function only works for objects of type BF2D_ALIGN and BF3D_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return FALSE.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.ID

Represents the ID of this alignment object. Read/write **String**.

Remarks

This function only works for objects of type START_ALIGN and RECALL_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return the empty string.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.InitID

Represents the intial ID of this alignment object. The intial ID is the ID of the alignment to recall before modifying it with this alignment. Read/write **String**.

Remarks

This function only works for objects of type START_ALIGN and RECALL_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return the empty string.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.MachineToPartMatrix

Represents the matrix used to transform points between the machine and part alignment systems. Read only **DmisMatrix**.

If used on an object other than a start alignment or a recall alignment, the identity matrix will be returned.

? Command.Type Property

? DmisMatrix Overview

? AlignmentCommand Object Overview

AlignCommand.MeasAllFeat

Represents the “Measure All Features” property of this iterative alignment object. Read/write **Boolean**.

Remarks

This function only works for objects of type ITER_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return FALSE.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.NumInputs

Returns the number of inputs to this alignment object. Read-only **Long**.

Remarks

This function only works for objects of type ITER_ALIGN, BF2D_ALIGN, and BF3D_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return zero.

? Command.Type Property

AlignmentCommand Object Overview

AlignCommand.Offset

Represents the offset property of this offset alignment object. For objects of type TRANSOFF_ALIGN, it is the number of MM or inches to offset the alignment. For objects of type ROTATEOFF_ALIGN, it is the number of radians to offset the alignment. Read/write **Double**.

Remarks

This function only works for objects of type TRANSOFF_ALIGN and ROTATEOFF_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return zero.

Command.Type Property

AlignmentCommand Object Overview

AlignCommand.Parent

Returns the parent **Command** object. Read-only.

Remarks

The parent of an **AlignCommand** object is the same underlying PC-DMIS object as the **AlignCommand** object itself. Getting the parent allows you to access the generic **Command** properties and methods of a given object.

Command.Type Property

AlignmentCommand Object Overview

AlignCommand.PointTolerance

Represents the “Point Tolerance” property of this alignment object. Read/write **Double**.

Remarks

This function only works for objects of type ITER_ALIGN, BF2D_ALIGN, and BF3D_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return zero.

Command.Type Property

AlignmentCommand Object Overview

AlignCommand.RepierceCad

Represents whether or not to repierce the cad model during the execution of this iterative alignment object. Read/write **Boolean**.

Remarks

This function only works for objects of type ITER_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return FALSE.

Command.Type Property

AlignmentCommand Object Overview

AlignCommand.UseBodyAxis

Represents whether or not to use the “Body Axis” method during the calculation of this iterative alignment object. Read/write **Boolean**.

Remarks

This function only works for objects of type ITER_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return FALSE.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.Workplane

Represents the workplane of this alignment object. It can take the values PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS, PCD_ZPLUS, and PCD_ZMINUS.Read/write **Long**.

Remarks

This function only works for objects of type ITER_ALIGN. If used on any other object type, setting this variable will do nothing, and getting this variable will return PCD_ZPLUS.

? Command.Type Property

? AlignmentCommand Object Overview

Methods:

AlignCommand.AddBestFitFeat

Syntax

Return Value=expression.AddBestFitFeat(*ID*, *tolerance*)

expression: Required expression that evaluates to a PC-DMIS **AlignCommand** object.

ID: Required **String** that is the ID of the feature to add to the level set.

tolerance: Required **Double** that is the tolerance to associate with *ID*.

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

Remarks

This function only has an effect on objects of type BF2D_ALIGN and BF3D_ALIGN. On objects of these types, it adds the feature with the ID *ID* to the set of best fit features with tolerance *tolerance*. On objects of other types, it does nothing.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.AddLevelFeat

Syntax

Return Value=expression.AddLevelFeat(*ID*)

expression: Required expression that evaluates to a PC-DMIS **AlignCommand** object.

ID: Required **String** that is the ID of the feature to add to the level set.

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

Remarks

This function only has an effect on objects of type ITER_ALIGN. On objects of this type, it adds the feature with the ID *ID* to the set of level features. On objects of other types, it does nothing.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.AddOriginFeat

Syntax

Return Value=expression.AddOriginFeat(*ID*)

expression: Required expression that evaluates to a PC-DMIS **AlignCommand** object.

ID: Required **String** that is the ID of the feature to add to the origin set.

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

Remarks

This function only has an effect on objects of type ITER_ALIGN. On objects of this type, it adds the feature with the ID *ID* to the set of origin features. On objects of other types, it does nothing.

? Command.Type Property

? AlignmentCommand Object Overview

AlignCommand.AddRotateFeat

Syntax

Return Value=expression.AddRotateFeat(*ID*)

expression: Required expression that evaluates to a PC-DMIS **AlignCommand** object.

ID: Required **String** that is the ID of the feature to add to the Rotate set.

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

Remarks

This function only has an effect on objects of type ITER_ALIGN. On objects of this type, it adds the feature with the ID *ID* to the set of rotate features. On objects of other types, it does nothing.

? Command.Type Property

? AlignmentCommand Object Overview

Application Object Overview

The Application object represents the PC-DMIS application.

To start PC-DMIS using Automation from another application, use CreateObject or GetObject to return an **Application** object.

Example:

```
Dim App as Object.
Set App = CreateObject(“Pcdlrn.Application”)
```

? Application Members

Application Members

Properties:

Application.ActivePartProgram

Represents the currently active part program. Read/Write **PartProgram**.

? PartProgram Overview

? Application Overview

Application.Caption

The text in the title bar of the application. Read/Write **String**.

? Application Overview

Application.DefaultFilePath

The directory in which the File Open dialog starts. Read/Write **String.**

? Application Overview

Application.DefaultProbeFile

The name of the last chosen probe file used when creating a new part program. Read Only **String**

? Application Overview

Application.DefaultMachineName

The name of the next available machine for attaching to a part program. Read Only **String**

? Application Overview

Application.FullName

The fully qualified path name of the PC-DMIS executable. Read-only **String**.

Example: If the PC-DMIS executable is C:\PCDMISW\PCDLRN.EXE, the FullName property is "C:\PCDMISW\PCDLRN.EXE".

? Application Overview

Application.Height

The height of the PC-DMIS window in screen pixels. Read/Write **Long**.

? Application Overview

Application.Left

The left edge of the PC-DMIS window, measured from the left edge of the Windows Desktop. Read/Write **Long**.

Remarks

The Left property is measured in screen pixels.

? Application Overview

Application.Machines

Returns the read-only **Machines** collection object.

? Machines Overview

? Application Overview

Application.Name

The file name of the PC-DMIS executable. Read-only **String**.

Remarks

The Name property is the default property for the **Application** object. If the PC-DMIS executable is C:\PCDMISW\PCDLRN.EXE, the FullName property is “PCDLRN.EXE”.

? Application Overview

Application.OperatorMode

Represents whether or not you are in operator mode. TRUE when in operator mode, FALSE otherwise. Read/Write Boolean.

Remarks

Changing into or out of operator mode makes significant changes to the appearance and utility of PC-DMIS.

? Application Overview

Application.PartPrograms

Returns the collection of part programs currently active in PC-DMIS. Read-only **PartPrograms** collection.

? PartPrograms Overview

? Application Overview

Application.Path

Returns the directory in which the PC-DMIS executable resides. Read-only **String**.

Remarks

If the PC-DMIS executable is C:\PCDMISW\PCDLRN.EXE, the FullName property is “C:\PCDMISW\”.

? Application Overview

Application.StatusBar

The text on the status bar of the main PC-DMIS window. Read/Write **String**.

? Application Overview

Application.Top

The top edge of the PC-DMIS window, measured from the top edge of the Windows Desktop. Read/Write **Long**.

Remarks

The Top property is measured in screen pixels.

? Application Overview

Application.UserExit

TRUE if the PC-DMIS automation engine is will shut down when the user exits PC-DMIS, otherwise FALSE. Read/Write **Boolean**.

? Application Overview

Application.Visible

TRUE if PC-DMIS is visible, otherwise FALSE. Read/Write **Boolean**.

? Application Overview

Application.Width

The width of the PC-DMIS window in screen pixels. Read/Write **Long**.

? Application Overview

Methods:

Application.Help

Syntax:

expression.Help *HelpFile*, *HelpContext*, *HelpString*

expression: Required expression that evaluates to a PC-DMIS **Application** object.

HelpFile: Required **String** parameter that indicates what help file to open.

HelpContext: Optional **Long** parameter that indicates which Context ID number in *HelpFile* to open.

HelpString: Optional **String** parameter that indicates a string to match among *HelpFile*'s topics.

Remarks

If both the HelpContext and HelpString are provided, the HelpString will be ignored. If neither is provided, the first help page is shown.

? Application Overview

Application.Minimize

Syntax:

expression.Minimize

The Minimize subroutine reduces the PC-DMIS window to the taskbar.

expression: Required expression that evaluates to a PC-DMIS **Application** object.

? Application Overview

Application.Maximize

Syntax:

expression.Maximize

The Maximize Subroutine expands the PC-DMIS window to full-screen size.

expression: Required expression that evaluates to a PC-DMIS **Application** object.

? Application Overview

Application.Post

Syntax:

Return Value=expression.Post(*Source*, *Destination*)

expression: Required expression that evaluates to a PC-DMIS **Application** object.

Source: Required **String** that indicates the file from which to import or export.

Destination: Required **String** that indicates the file into which to import or export.

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

The Post function tells PC-DMIS to import or export *Source* into *Destination*. It returns TRUE if the import or export process is successful, FALSE otherwise.

Exactly one of *Source* and *Destination* must be a PC-DMIS .prg or .cad file. If it is *Source*, then PC-DMIS will export based on the name of the *Destination* file. If the *Destination* file is a PC-DMIS .prg or .cad file, then PC-DMIS will import based on the name of the *Source* file.

The *Source* file must already exist, but the *Destination* file need not already exist.

? Application Overview

Application.Quit

Syntax:

expression.Quit

The Quit function tells PC-DMIS to close. It always returns TRUE.

expression: Required expression that evaluates to a PC-DMIS **Application** object.

? Application Overview

Application.Restore

Syntax:

expression.Restore

The Restore subroutine makes the PC-DMIS window open and neither maximized nor minimized.

expression: Required expression that evaluates to a PC-DMIS **Application** object.

? Application Overview

Application.SetActive

Syntax:

Return Value=expression.SetActive

expression: Required expression that evaluates to a PC-DMIS **Application** object.

Brings PC-DMIS to the foreground, making it the active application.

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

? Application Overview

Array Index Object Overview

The Array Index Object is used to set up multi-dimensional feature arrays in PC-DMIS. Methods are provided to add, remove, or edit array upper and lower bounds for array indices.

? Array Index Members

Array Index Members

Methods:

ArrayIndex.AddIndexSet

Syntax:

expression.AddIndexSet (LowerBound, UpperBound)

expression: Required expression that evaluates to a PC-DMIS **ArrayIndex** object.

LowerBound: Required **Long** parameter representing the lower bound of the index set to be added.

UpperBound: Required **Long** parameter representing the upper bound of the index set to be added.

Remarks

Adds the supplied index set to the array index command.

? Array Index Overview

ArrayIndex.GetLowerBound

Syntax:

expression.GetLowerBound (Index)

Return Value: ***Long*** representing the lower bound of the specified index set.

expression: Required expression that evaluates to a PC-DMIS **ArrayIndex** object.

Index: Required **Long** parameter that specifies which index set to use in retrieving the lower bound.

Remarks

Retrieves the lower bound of the specified index set.

? Array Index Overview

ArrayIndex.GetUpperBound

Syntax:

expression.GetUpperBound (Index)

Return Value: ***Long*** representing the upper bound of the specified index set.

expression: Required expression that evaluates to a PC-DMIS **ArrayIndex** object.

Index: Required **Long** parameter that specifies which index set to use in retrieving the upper bound.

Remarks

Retrieves the upper bound of the specified index set.

? Array Index Overview

ArrayIndex.RemoveIndexSet

Syntax:

expression.RemoveIndexSet (Index)

expression: Required expression that evaluates to a PC-DMIS **ArrayIndex** object.

Index: Required **Long** parameter that specifies which index set to remove.

Remarks

Removes the index set specified by index from the array index object.

? Array Index Overview

ArrayIndex.SetLowerBound

Syntax:

expression.SetLowerBound (Index)

expression: Required expression that evaluates to a PC-DMIS **ArrayIndex** object.

Index: Required **Long** parameter that specifies which index set to use in setting the lower bound.

Remarks

Sets the lower bound of the specified index set.

? Array Index Overview

ArrayIndex.SetUpperBound

Syntax:

expression.SetUpperBound (Index)

expression: Required expression that evaluates to a PC-DMIS **ArrayIndex** object.

Index: Required **Long** parameter that specifies which index set to use in setting the upper bound.

Remarks

Setting the upper bound of the specified index set.

? Array Index Overview

Attach Object Overview

The attach command object attaches part programs to the current part program. The current part program can then access objects from the attached part programs.

Attach Members

Properties:

Attach.AttachedAlign

ID associated with an alignment in the attached program that corresponds with an alignment in the attaching program.
Read/Write **String**

? LocalAlign Property

? Attach Object Overview

Attach.Execute

BOOLEAN value that determines whether or not the attached part program should be executed when PC-DMIS encounters the attached program.

Read/Write **Boolean**

? Attach Object Overview

Attach.ID

ID associated with the attached part program. This ID identies items in the attached part program. For example, if the ID for the attach statement is “PART2”, then feature “F1” in the attached program can be referred to as “F1:PART2”.

Read/Write **String**

? Attach Object Overview

Attach.LocalAlign

ID associated with an alignment in the attaching program that corresponds to an alignment in the attached program.
Read/Write **String**

? AttachedAlign Property

? Attach Object Overview

Attach.PartName

File name of the attached part program.

Read/Write **String**

? Attach Object Overview

BasicScanCommand Object Overview

Objects of type **BasicScanCommand** are created from more generic **Command** objects to pass information specific to the scan command back and forth. At present only DCC basic scans are user accessible.

? Command.BasicScanCommand

? BasicScanCommand Members

BasicScanCommand Members

Properties

BasicScan.AutoClearPlane

Determines whether auto clearance planes mode is on or off. Read/Write BOOLEAN.

? BasicScanCommand Overview

BasicScan.BoundaryCondition

Represents the boundary condition type. Read/write of enumeration BSBOUNDCOND_ENUM.

The allowable values have the following meaning:

BSBOUNDCOND_SPHENTRY: Represents a Spherical Boundary Condition. This Boundary condition requires the following parameters to be set by you using Automation Properties and/or Automation Methods : BoundaryConditionCenter, BoundaryConditionEndApproach, Diameter, number of Crossings.

BSBOUNDCOND_PLANECROSS: Represents a Planar Boundary Condition. This Boundary condition requires the following parameters to be set by you using Automation Properties and/or Automation Methods : BoundaryConditionCenter, BoundaryConditionEndApproach, BoundaryConditionPlaneV, number of Crossings.

BSBOUNDCOND_CYLINDER: Represents a Cylindrical Boundary Condition. This Boundary condition requires the following parameters to be set by you using Automation Properties and/or Automation Methods : BoundaryConditionCenter, BoundaryConditionEndApproach, BoundaryConditionAxisV, Diameter, number of Crossings.

BSBOUNDCOND_CONE: Represents a Conical Boundary Condition. This Boundary condition requires the following parameters to be set you user using Automation Properties and/or Automation Methods : BoundaryConditionCenter, BoundaryConditionEndApproach, BoundaryConditionAxisV, HalfAngle, number of Crossings.

The SetBoundaryConditionParams method should be used to set the values for:

- `HalfAngle`
- `Number of Crossings`
- `Diameter`

? BasicScanCommand Overview

BasicScan.BoundaryConditionAxisV

Represents the boundary condition axis vector. Read/write **PointData** object. This vector is used as the axis of the Cylindrical and Conical BoundaryConditions.

? BasicScanCommand Overview

BasicScan.BoundaryConditionCenter

Represents the boundary condition center. Read/write **PointData** object.

This Point is used by all Boundary Conditions and is the location of the Boundary Condition.

? BasicScanCommand Overview

BasicScan.BoundaryConditionEndApproach

Represents the boundary condition end approach vector. Read/write **PointData** object.

This vector is used by all Boundary Conditions and is the Approach Vector of the Probe as it crosses the Boundary condition.

BasicScanCommand Overview

BasicScan.BoundaryConditionPlaneV

Represents the boundary condition plane vector. Read/write **PointData** object.

This vector is the normal vector of the Plane used by the Plane and OldStyle Boundary Conditions.

Boundary Condition	Properties Required
Plane	BoundaryConditionCenter BoundaryConditionEndApproach BoundaryConditionPlaneV
Cone	BoundaryConditionCenter BoundaryConditionEndApproach BoundaryConditionAxisV
Cylinder	BoundaryConditionCenter BoundaryConditionEndApproach BoundaryConditionAxisV
Sphere	BoundaryConditionCenter BoundaryConditionEndApproach

BasicScanCommand Overview

BasicScan.BoundaryPointCount

Indicates the number of boundary points to used in a patch scan. Read/Write LONG.

Individual boundary points can be set or retrieved via the "BasicScan.GetBoundaryPoint" and "BasicScan.SetBoundaryPoint" methods on page 188.

BasicScanCommand Overview

BasicScan.DisplayHits

Determines whether hits of the scan are displayed in the Edit window or not. Read/Write BOOLEAN.

BasicScanCommand Overview

BasicScan.Filter

Represents the filter type. Read/write of enumeration BSF_ENUM.

The allowable values have the following meaning:

BSF_DISTANCE: PC-DMIS determines each hit based on the set increment and the last two measured hits. The approach of the probe is perpendicular to the line between the last two measured hits. The probe will stay on the cut plane. PC-DMIS will start at the first boundary point and continue taking hits at the set increment, stopping when it satisfies the Boundary Condition. In the case of a continous scan, PC-DMIS would filter the data from the CMM and keep only the hits that are apart by at least the increment. Both DCC and Manual scans can use this filter.

BSF_BODYAXISDISTANCE: PC-DMIS will take hits at the set increment along the current part's coordinate system. The approach of the probe is perpendicular to the indicated axis. The probe will stay on the cut plane. The approach vector will be normal to the selected axis and on the cut plane. This technique uses the same approach for taking each hit

(unlike the previous technique which adjusts the approach to be perpendicular to the line between the previous two hits). Only DCC scans should use this filter.

BSF_VARIABLEDISTANCE: This technique allows you to set specific maximum and minimum angle and increment values that will be used in determining where PC-DMIS will take a hit. The probe's approach is perpendicular to the line between the last two measured hits. You should provide the maximum and minimum values that will be used to determine the increments between hits. You also must enter the desired values for the maximum and minimum angles. PC-DMIS will take three hits using the minimum increment. It will then measure the angle between hit's 1-2 and 2-3.

- `If the measured angle is between the maximum and minimum values defined, PC-DMIS will continue to take hits at the current increment.`
- `If the angle is greater than the maximum value, PC-DMIS will erase the last hit and measure it again using one quarter of the current increment value.`
- `If the angle is less than the minimum increment, PC-DMIS will take the hit at the minimum increment value.`

PC-DMIS will again measure the angle between the newest hit and the two previous hits. It will continue to erase the last hit and drop the increment value to one quarter of the increment until the measured angle is within the range defined, or the minimum value of the increment is reached.

If the measured angle is less than the minimum angle, PC-DMIS will double the increment for the next hit. (If this is greater than the maximum increment value it will take the hit at the maximum increment.) PC-DMIS will again measure the angle between the newest hit and the two previous hits. It will continue to double the increment value until the measured angle is within the range defined, or the maximum increment is reached. Only DCC scans should use this filter.

? BasicScanCommand Overview

BasicScan.HitType

Represents the type of hit to use. Read/write of enumeration BSCANHIT_ENUM.

The allowable values have the following meaning:

BSCANHIT_VECTOR – use vector hits for this scan

BSCANHIT_SURFACE – use surface hits for this scan

BSCANHIT_EDGE – use edge hits for this scan.

BSCANHIT_BASIC – use basic hits for this scan. Only Manual scans use this hit type. Currently there are no Manual BasicScans.

Remarks

Not every hit type can be used with every method and filter combination.

Method	EdgeHit	Vector Hit	Surface Hit	Basic Hit
Linear	-	Y	Y	-
Edge	Y	-	-	-
Circle	-	Y	-	-
Cylinder	-	Y	-	-
Str Line	-	Y	-	-
Center	-	Y	-	-

? BasicScanCommand Overview

BasicScan.Method

Represents the method type for this scan. Read/write of enumeration BSMETHOD_ENUM.

The allowable values have the following meaning:

BSCANMETH_LINEAR: This method will scan the surface along a line. This procedure uses the starting and ending point for the line, and also includes a direction point. The probe will always remain within the cut plane while doing the scan.

BSCANMETH_EDGE: This method will scan the Edge of the Surface in a Touch Trigger mode.

BSCANMETH_CIRCLE: This method will scan around a Circle in High Speed, Continous contact mode.

BSCANMETH_CYLINDER: This method will scan around a Cylinder in High Speed, Continous contact mode.

BSCANMETH_STRAIGHTLINE: This method will scan a straight line in a plane in High Speed , Continous contact mode.

BSCANMETH_CENTER: This method will find a Low Point on a surface.

Remarks

The Method type defines the geometry of the feature to be scanned and has parameters that need to be set properly before scanning. The parameters can be set using the SetMethodParams method.

? BasicScanCommand Overview

BasicScan.MethodCutPlane

Represents the method's cut plane vector. Read/write **PointData** object.

? BasicScanCommand Overview

BasicScan.MethodEnd

Represents the scan's end point. Read/write **PointData** object.

? BasicScanCommand Overview

BasicScan.MethodEndTouch

Represents the method's end touch vector. Read/write **PointData** object.

? BasicScanCommand Overview

BasicScan.MethodInitDir

Represents the method's initial direction vector. Read/write **PointData** object.

? BasicScanCommand Overview

BasicScan.MethodInitTopSurf

Represents the initial Surface Vector for the Edge method. Read/write **PointData** object.

? BasicScanCommand Overview

BasicScan.MethodInitTouch

Represents the method's initial touch vector. Read/write **PointData** object.

? BasicScanCommand Overview

BasicScan.MethodStart

Represents the scan's start point. Read/write **PointData** object.

Method	Method Start	Method End	Method CutPlane	Method InitDir	Method InitTouch	Method InitTopSurf	Method EndTouch
Linear	Y	Y	Y	Y	Y	-	Y
Edge	Y	Y	-	Y	Y	Y	Y
Circle	Y	-	Y	-	Y	-	-
Cylinder	Y	-	Y	-	Y	-	-
Str Line	Y	Y	Y	-	-	-	-
Center	Y	Y	Y	-	Y		-

? ? BasicScanCommand Overview

BasicScan.NominalMode

Represents how to determine the nominals for this scan. Read/write of enumeration BSCANNMODE_ENUM.

The allowable values have the following meaning:

BSCANNMODE_FINDCADNOMINAL: This mode would find the Nominal data from CAD after scanning. This mode is useful only when CAD surface data is available.

SCANNMODE_MASTERDATA: This mode keeps the data scanned the first time as Master data.

? BasicScanCommand Overview

BasicScan.OperationMode

Represents mode of operation of the scan . Read/write of enumeration BSOPMODE_ENUM.

The allowable values have the following meaning:

BSCANOPMODE_REGULARLEARN: When this mode is used, PC-DMIS will execute the scan as though it is learning it. All learned measured data will replace the new measured data. The nominal will be re-calculated depending on the Nominals mode.

BSCANOPMODE_DEFINEPATHFROMHITS: This mode is available only when using analog probe heads that can do continuous contact scanning. When this option is selected, PC-DMIS allows the controller to 'define' a scan. PC-DMIS gathers all hit locations from the editor and passes them onto the controller for scanning. The controller will then adjust the path allowing the probe to pass through all the points. The data is then reduced according to the increment provided and the new data will replace any old measured data. Currently, this value cannot be used through Automation since there is no method provided to define a path.

BSCANOPMODE_HIGHSPEEDFEATUREBASED: This execute mode is available only for Analog Probe Heads. When this is selected, PC-DMIS uses the built-in High Speed scanning capability of the controller to execute a scan.

Example: If you selected a Circle scan, PC-DMIS would use a corresponding Circle scanning command in the controller and pass on the parameters to the controller to execute. In this case, PC-DMIS does not control execution of the scans.

BSCANOPMODE_NORMALEXECUTION: If a DCC scan is executed, PC-DMIS will take hits at each of the learned locations in Stitch scanning mode, storing the newly measured data.

Method	Regular Learn	Defined Path	Feature Based	Normal
Linear	Y	-	-	Y

Edge	Y	-	-	Y
Circle	-	-	Y	Y
Cylinder	-	-	Y	Y
Str Line	-	-	Y	Y
Center	Y	-	-	Y

? BasicScanCommand Overview

BasicScan.SinglePoint

Determines whether single point mode is on or off. Read/Write BOOLEAN.

When on, each point will be considered as a single measured point.

? BasicScanCommand Overview

Methods:

BasicScan.GetBoundaryConditionParams

Syntax

Return Value=expression. GetBoundaryConditionParams (*nCrossings, dRadius, dHalfAngle*)

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

nCrossing: Required **Long** variable that gets the number of crossings for this boundary condition. The scan would stop after the probe crosses (breaks) the Boundary Condition like a Sphere, Cylinder, Cone, or a Plane the given number of times.

dRadius: Required **Double** variable that gets the radius of the boundary condition. This is used by the Spherical and Cylindrical Boundary Conditions.

dHalfAngle: Required **Double** variable that gets the half-angle of the cone-type boundary condition, or gets zero if the boundary condition is not of cone type.

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

Remarks

Boundary Condition	**GetBoundaryConditionParams (*nCrossings, dRadius, dHalfAngle*)**
Plane	*Ncrossings*
Cone	*NCrossings, ,dHalfAngle*
Cylinder	*NCrossings, dRadius*
Sphere	*NCrossings, dRadius*

? BasicScanCommand Overview

BasicScan.GetBoundaryPoint

Syntax

Return Value=expression. GetBoundaryPoint (Index, X,Y, Z)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

Index: Required **Long** which indicates which boundary point to get.

X: Required **Long** variable that will hold the X value of the bounday point.

Y: Required **Long** variable that will hold the Y value of the bounday point.

Z: Required **Long** variable that will hold the Z value of the bounday point.

Remarks

This function works with patch scans. Use the boundarypointcount property to determine how many boundary points are available.

? BasicScanCommand Overview

BasicScan.GetFilterParams

Syntax

Return Value=expression. GetFilterParams (*dCutAxisLocation*, *nAxis*, *dMaxIncrement*, *dMinIncrement*, *dMaxAngle*, *dMinAngle*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

dCutAxisLocation: Not used.

nAxis: Required **Long** variable that gets the cut axis. Returns non-zero only for axis filters. For axis filters, 0 means the X axis, 1 means the Y-axis, and 2 means the Z-axis.

dMaxIncrement: Required **Double** variable that gets the maximum increment. For fixed-length filters, this is simply the fixed increment for Variable Distance Filters.

dMinIncrement: Required **Double** variable that gets the minimum increment.

dMaxAngle: Required **Double** variable that gets the maximum angle used in Variable Distance Filters.

dMinAngle: Required **Double** variable that gets the minimum angle

used in Variable Distance Filters.

Remarks

Filter	**GetFilterParams (*dCutAxisLocation, nAxis, dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*)**
Distance	*,,dMaxIncrement*
BodyAxisDistance	*,nAxis, dMaxIncrement*
VariableDistance	*,,dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*

? BasicScanCommand Overview

BasicScan.GetHitParams

Syntax

Return Value=expression. GetHitParams (*nInitSamples*, *nPermSamples*, *dSpacer*, *dIndent*, *dDepth*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

nInitSamples: Required **Long** variable that gets the number of initial sample hits for the hits in this scan. It always returns zero for basic hits and vector hits.

nPermSamples: Required **Long** variable that gets the number of permanent sample hits for the hits in this scan. It always returns zero for basic hits and vector hits.

dSpacer: Required **Double** variable that gets the spacing of the sample hits from the hit center. It always returns zero for basic hits and vector hits.

dIndent: Required **Double** variable that gets the indent of the sample hits from the hit center. It always returns zero for basic hits, vector hits, and surface.

dDepth: Required **Double** variable that gets the depth of the sample hits from the hit center. It always returns zero for basic hits, vector hits, and surface.

? BasicScanCommand Overview

BasicScan.GetMethodParams

Syntax

Return Value=*expression*. GetMethodParams (*bIn*, *bCenteringType*, *nCenteringDirection*, *dDiameter*, *dArcAngle*, *dDepth*, *dPitch*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

bIn: Required variable that gets 0 for Inside scans, 1 for Outside scans, and 2 for Plane Circle scans.

bCenteringType: Required Variable for Centering Scans that gets 0 for Axis Centering and 1 for Plane centering.

nCenteringDirection: Required **Long** variable that takes a +1 for measurement with the direction of the probe and –1 for against the direction of probe.

dDiameter: Required **Double** variable that gets the diameter of the circle or cylinder scan, and zero otherwise.

dArcAngle: Required **Double** variable that gets arc angle for circle and cylinder scans.

dDepth: Required **Double** variable that gets the depth for cylinder scans, and zero otherwise.

dPitch: Required **Double** variable that gets a Pitch for Cylinder scans.

Remarks

Method	**GetMethodParams (*bIn, bCenteringType, nCenteringDirection, dDiameter, dArcAngle, dDepth, dPitch)***
Linear	None
Edge	None
Circle	*bIn*, , , *dDiameter*, *dArcAngle*, *dDepth*
Cylinder	*bIn*, , , *dDiameter*, *dArcAngle*, *dDepth*, *dPitch*
Str Line	None
Center	, *bCenteringType*, *nCenteringDirection*

? BasicScanCommand Overview

BasicScan.GetMethodPointData

Syntax

```
Return Value=expression. GetMethodPointData (MethodStart, MethodEnd, MethodInitTouch, MethodEndTouch,
MethodInitDir, MethodCutPlane)
```

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

MethodStart: Required **PointData** object that gets the MethodStart property.

MethodEnd: Required **PointData** object that gets the MethodEnd property.

MethodInitTouch: Required **PointData** object that gets the MethodInitTouch property.

MethodEndTouch: Required **PointData** object that gets the MethodEndTouch property.

MethodInitDir: Required **PointData** object that gets the MethodInitDir property.

MethodCutPlane: Required **PointData** object that gets the MethodCutPlane property.

Remarks

If scan is a **BasicScanCommand** object, and MS, ME, MIT, MET, MID,and MCP are all **Dim**ensioned as **Object**, the following are equivalent:

scan.GetMethodParams MS,ME,MIT,MET,MID,MCP

```
set MS = scan.MethodStart
set ME = scan.MethodEnd
set MIT = scan.MethodInitTouch
set MET = scan.MethodEndTouch
set MID = scan.MethodInitDir
set MCP = scan.MethodCutPlane
```

This method is provided as a shortcut to getting these commonly used properties all at once.

? BasicScanCommand.MethodStart

? BasicScanCommand.MethodEnd

? BasicScanCommand.MethodInitTouch

? BasicScanCommand.MethodEndTouch

? BasicScanCommand.MethodInitDir

? BasicScanCommand.MethodCutPlane

? BasicScanCommand Overview

BasicScan.GetNomsParams

Syntax

Return Value=*expression*. GetNomsParams (*dFindNomsTolerance*, *dSurfaceThickness*, *dEdgeThickness*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

dFindNomsTolerance: Required **Double** variable that gets the Find Noms tolerance and is used only when the **NominalMode** property is BSCANNMODE_FINDCADNOMINAL.

dSurfaceThickness: Required **Double** variable that gets the surface thickness and is used only when the **NominalMode** property is BSCANNMODE_FINDCADNOMINAL.

dEdgeThickness: Required **Double** variable that gets the edge thickness and is used only when the **NominalMode** property is BSCANNMODE_FINDCADNOMINAL and when the **Method** property is BSCANMETH_EDGE.

? BasicScanCommand Overview

BasicScan.GetParams

Syntax

Return Value=expression. GetParams *(Method, Filter, OperationMode, HitType, NominalMode, BoundaryCondition)*

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

Method: Required **Long** variable that gets the Method property.

Filter: Required **Long** variable that gets the Filter property.

OperationMode: Required **Long** variable that gets the OperationMode property.

HitType: Required **Long** variable that gets the HitType property.

NominalMode: Required **Long** variable that gets the NominalMode property.

BoundaryCondition: Required **Long** variable that gets the BoundaryCondition property.

Remarks

If scan is a **BasicScanCommand** object, and M, F, O, H, N,and B are all **Dim**ensioned as **Object**, the following are equivalent:

scan.GetParams M, F, O, H, N, B

```
M = scan.Method
F = scan.Filter
O = scan.OperationMode
H = scan.HitType
N = scan.NominalMode
B = scan.BoundaryCondition
```

This method is provided as a shortcut to getting these commonly used properties all at once.

? BasicScanCommand.Method Property

? BasicScanCommand.Filter Property

? BasicScanCommand.OperationMode Property

? BasicScanCommand.HitType Property

? BasicScanCommand.NominalMode Property

? BasicScanCommand.BoundaryCondition Property

? BasicScanCommand Overview

BasicScan.SetBoundaryConditionParams

Syntax

Return Value=expression.SetBoundaryConditionParams (*nCrossings*, *dRadius*, *dHalfAngle*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

nCrossing: Required **Long** that sets the number of crossings for this boundary condition.

dRadius: Required **Double** that sets the radius of the boundary condition.

dHalfAngle: Required **Double** that sets the half-angle of the cone-type boundary condition, or is ignored if the boundary condition is not of cone type.

Remarks

Boundary Condition	SetBoundaryConditionParams (*nCrossings, dRadius, dHalfAngle*)
Plane	*Ncrossings*
Cone	*NCrossings,, dHalfAngle*
Cylinder	*NCrossings, dRadius*
Sphere	*NCrossings, dRadius*

? BasicScanCommand Overview

BasicScan.SetBoundaryPoint

Syntax

Return Value=expression.SetBoundaryPoint (Index, X,Y, Z)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

Index: Required **Long** which indicates which boundary point to set.

X: Required **Long** that indicates the X value of the bounday point.

Y: Required **Long** that indicates the Y value of the bounday point.

Z: Required **Long** that indicates the Z value of the bounday point.

Remarks

This function works with patch scans. Use the boundarypointcount property to set the number of boundary points.

? BasicScanCommand Overview

BasicScan.SetFilterParams

Syntax

Return Value=expression.SetFilterParams (*dCutAxisLocation, nAxis, dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

dCutAxisLocation: Not used

nAxis: Required **Long** that sets the cut axis. It is used only for axis filters. For axis filters, 0 means the X axis, 1 means the Y-axis, and 2 means the Z-axis.

dMaxIncrement: Required **Double** that sets the maximum increment. For fixed-length filters, this is simply the fixed increment

dMinIncrement:. Required **Double** that sets the minimum increment.

dMaxAngle: . Required **Double** that sets the maximum angle.

dMinAngle: . Required **Double** that sets the minimum angle.

Remarks

Filter	SetFilterParams (*dCutAxisLocation, nAxis, dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*)
Distance	*,,dMaxIncrement*
BodyAxisDistance	*,nAxis, dMaxIncrement*
VariableDistance	*,,dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*

? BasicScanCommand Overview

BasicScan.SetHitParams

Syntax

Return Value=expression.SetHitParams (*nInitSamples*, *nPermSamples*, *dSpacer*, *dIndent*, *dDepth*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

nInitSamples: Required **Long** that sets the number of initial sample hits for the hits in this scan. It is ignored for basic hits and vector hits.

nPermSamples: Required **Long** that sets the number of permanent sample hits for the hits in this scan. It is ignored for basic hits and vector hits.

dSpacer: Required **Double** that sets the spacing of the sample hits from the hit center. It is ignored for basic hits and vector hits.

dIndent: Required **Double** that sets the indent of the sample hits from the hit center. It is ignored for basic hits, vector hits, and surface.

dDepth: Required **Double** that sets the depth of the sample hits from the hit center. It is ignored for basic hits, vector hits, and surface.

? BasicScanCommand Overview

BasicScan.SetMethodParams

Syntax

Return Value=expression.SetMethodParams (*bIn*, *bCenteringType*, *nCenteringDirection*, *dDiameter*, *dArcAngle*, *dDepth*, *dPitch*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

bIn: Required variable that sets 0 for Inside scans, 1 for Outside scans, nd 2 for Plane Circle scans.

bCenteringType: Required Variable for Centering Scans that sets 0 for Axis Centering and 1 for Plane centering.

nCenteringDirection: Required **Long** variable that sets +1 for measurement with the direction of the probe and –1 for against the direction of probe.

dDiameter: Required **Double** variable that sets the diameter of the circle or cylinder scan, and zero otherwise.

dArcAngle: Required **Double** variable that sets arc angle for circle and cylinder scans.

dDepth: Required **Double** variable that sets the depth for circle and cylinder scans, and zero otherwise.

dPitch: Required **Double** variable that sets Pitch for Cylinder scans.

Remarks

Method	SetMethodParams (*bIn, bCenteringType, nCenteringDirection, dDiameter, dArcAngle, dDepth, dPitch)*
Linear	None
Edge	None
Circle	*bIn, , , dDiameter, dArcAngle, dDepth*
Cylinder	*bIn, , , dDiameter, dArcAngle, dDepth, dPitch*
Str Line	None
Center	*, bCenteringType, nCenteringDirection*

? BasicScanCommand Overview

BasicScan.SetMethodPointData

Syntax

Return Value=expression.SetMethodPointData *(MethodStart, MethodEnd, MethodInitTouch, MethodEndTouch, MethodInitDir, MethodCutPlane)*

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

MethodStart: Required **PointData** object that sets the MethodStart property.

MethodEnd: Required **PointData** object that sets the MethodEnd property.

MethodInitTouch: Required **PointData** object that sets the MethodInitTouch property.

MethodEndTouch: Required **PointData** object that sets the MethodEndTouch property.

MethodInitDir: Required **PointData** object that sets the MethodInitDir property.

MethodCutPlane: Required **PointData** object that sets the MethodCutPlane property.

Remarks

If scan is a **BasicScanCommand** object, and MS, ME, MIT, MET, MID,and MCP are all **Dim**ensioned as **Object**, the following are equivalent:

scan.SetMethodParams MS,ME,MIT,MET,MID,MCP

```
set scan.MethodStart = MS
set scan.MethodEnd = ME
set scan.MethodInitTouch = MIT
set scan.MethodEndTouch = MET
set scan.MethodInitDir = MID
set scan.MethodCutPlane = MCP
```

This method is provided as a shortcut to setting these commonly used properties all at once.

? BasicScanCommand.MethodStart

? BasicScanCommand.MethodEnd

? BasicScanCommand.MethodInitTouch

? BasicScanCommand.MethodEndTouch

? BasicScanCommand.MethodInitDir

? BasicScanCommand.MethodCutPlane

? BasicScanCommand Overview

BasicScan.SetNomsParams

Syntax

Return Value=expression.SetNomsParams (*dFindNomsTolerance*, *dSurfaceThickness*, *dEdgeThickness*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

dFindNomsTolerance: Required **Double** that sets the Find Noms tolerance.

dSurfaceThickness: Required **Double** that sets the surface thickness.

dEdgeThickness: Required **Double** that sets the edge thickness.

Remarks

? BasicScanCommand Overview

BasicScan.SetParams

Syntax

Return Value=expression.SetParams (*Method*, *Filter*, *OperationMode*, *HitType*, *NominalMode*, *BoundaryCondition*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

Method: Required **Long** that sets the Method property.

Filter: Required **Long** that sets the Filter property.

OperationMode: Required **Long** that sets the OperationMode property.

HitType: Required **Long** that sets the HitType property.

NominalMode: Required **Long** that sets the NominalMode property.

BoundaryCondition: Required **Long** that sets the BoundaryCondition property.

Remarks

If scan is a **BasicScanCommand** object, and M, F, O, H, N,and B are all **Dim**ensioned as **Object**, the following are equivalent:

scan.SetParams M, F, O, H, N, B

```
scan.Method = M
scan.Filter = F
scan.OperationMode = O
scan.HitType = H
scan.NominalMode = N
scan.BoundaryCondition = B
```

This method is provided as a shortcut to setting these commonly used properties all at once.

? BasicScanCommand Overview

Basic Scan Object Combinations

The tables below describes the different combination of Objects that can be used to create and execute a Basic Scan. The Methods will only work with the combination of different of Objects selected from this table (i.e. if you decide to set a method type of BSCANMETH_CIRCLE, then you have to use a Filter type of BSF_DISTANCE etc).

Table 1

Method	Filters
BSCANMETH_LINEAR	BSF_DISTANCE BSF_BODYAXISDISTANCE BSF_VARIABLEDISTANCE
BSCANMETH_EDGE	BSF_DISTANCE BSF_VARIABLEDISTANCE
BSCANMETH_CIRCLE	BSF_DISTANCE
BSCANMETH_CYLINDER	BSF_DISTANCE
BSCANMETH_STRAIGHTLINE	BSF_DISTANCE
BSCANMETH_CENTER	BSF_DISTANCE

? BasicScanCommand.Method Property

? BasicScanCommand.Filter Property

? BasicScanCommand.OperationMode Property

? BasicScanCommand.HitType Property

? BasicScanCommand.NominalMode Property

? BasicScanCommand.BoundaryCondition Property

? BasicScanCommand Overview

Table 2

Method	NominalMode
BSCANMETH_LINEAR	BSCANNMODE_FINDCADNOMINAL BSCANNMODE_MASTERDATA
BSCANMETH_EDGE	BSCANNMODE_FINDCADNOMINAL BSCANNMODE_MASTERDATA
BSCANMETH_CIRCLE	BSCANNMODE_FINDCADNOMINAL BSCANNMODE_MASTERDATA
BSCANMETH_CYLINDER	BSCANNMODE_FINDCADNOMINAL BSCANNMODE_MASTERDATA
BSCANMETH_STRAIGHTLINE	BSCANNMODE_FINDCADNOMINAL BSCANNMODE_MASTERDATA
BSCANMETH_CENTER	BSCANNMODE_FINDCADNOMINAL BSCANNMODE_MASTERDATA

? BasicScanCommand.Method Property

? BasicScanCommand.Filter Property

? BasicScanCommand.OperationMode Property

? BasicScanCommand.HitType Property

? BasicScanCommand.NominalMode Property

? BasicScanCommand.BoundaryCondition Property

? BasicScanCommand Overview

Table 3

Method	OperationMode
BSCANMETH_LINEAR	BSCANOPMODE_REGULARLEARN BSCANOPMODE_DEFINEPATHFROMHITS BSCANOPMODE_NORMALEXECUTION
BSCANMETH_EDGE	BSCANOPMODE_REGULARLEARN BSCANOPMODE_NORMALEXECUTION
BSCANMETH_CIRCLE	BSCANOPMODE_HIGHSPEEDFEATUREBASED BSCANOPMODE_NORMALEXECUTION
BSCANMETH_CYLINDER	BSCANOPMODE_HIGHSPEEDFEATUREBASED BSCANOPMODE_NORMALEXECUTION
BSCANMETH_STRAIGHTLINE	BSCANOPMODE_HIGHSPEEDFEATUREBASED BSCANOPMODE_NORMALEXECUTION
BSCANMETH_CENTER	BSCANOPMODE_REGULARLEARN BSCANOPMODE_NORMALEXECUTION

? BasicScanCommand.Method Property

? BasicScanCommand.Filter Property

? BasicScanCommand.OperationMode Property

? BasicScanCommand.HitType Property

? BasicScanCommand.NominalMode Property

? BasicScanCommand.BoundaryCondition Property

? BasicScanCommand Overview

Table 4

Method	HitType
BSCANMETH_LINEAR	BSCANHIT_VECTOR BSCANHIT_SURFACE
BSCANMETH_EDGE	BSCANHIT_EDGE
BSCANMETH_CIRCLE	BSCANHIT_VECTOR
BSCANMETH_CYLINDER	BSCANHIT_VECTOR
BSCANMETH_STRAIGHTLINE	BSCANHIT_VECTOR
BSCANMETH_CENTER	BSCANHIT_VECTOR

? BasicScanCommand.Method Property

? BasicScanCommand.Filter Property

? BasicScanCommand.OperationMode Property

? BasicScanCommand.HitType Property

? BasicScanCommand.NominalMode Property

? BasicScanCommand.BoundaryCondition Property

? BasicScanCommand Overview

Table 5

Method	BoundaryCondition
BSCANMETH_LINEAR	BSBOUNDCOND_SPHENTRY BSBOUNDCOND_PLANECROSS BSBOUNDCOND_CYLINDER BSBOUNDCOND_CONE
BSCANMETH_EDGE	BSBOUNDCOND_SPHENTRY BSBOUNDCOND_PLANECROSS BSBOUNDCOND_CYLINDER BSBOUNDCOND_CONE
BSCANMETH_CIRCLE	None
BSCANMETH_CYLINDER	None
BSCANMETH_STRAIGHTLINE	None
BSCANMETH_CENTER	None

? BasicScanCommand.Method Property

? BasicScanCommand.Filter Property

? BasicScanCommand.OperationMode Property

? BasicScanCommand.HitType Property

? BasicScanCommand.NominalMode Property

? BasicScanCommand.BoundaryCondition Property

? BasicScanCommand Overview

CadWindow Object Overview:

The **CadWindow** object is the one and only cad window for a part program.

CadWindow Members

Properties:

CadWindow.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the Active**PartProgram** property returns a **PartProgram** object.

? Application Overview

? CadWindow Overview

CadWindow.Height

The height of the Cad window in screen pixels. Read/Write **Long**.

? CadWindow Overview

CadWindow.Left

The left edge of the Cad window, measured from the left edge of the Windows Desktop. Read/Write **Long**.

Remarks

The Left property is measured in screen pixels.

? CadWindow Overview

CadWindow.Parent

Returns the parent **CadWindows** object. Read-only.

? CadWindows Overview

? CadWindow Overview

CadWindow.Top

The top edge of the Cad window, measured from the top edge of the Windows Desktop. Read/Write **Long**.

Remarks

The Top property is measured in screen pixels.

? CadWindow Overview

CadWindow.Visible

This property is TRUE if the Cad window is visible, FALSE otherwise. Read/write **Boolean**.

If you make the Cad window invisible, the only way to make it visible again is to set this property to TRUE.

? CadWindow Overview

CadWindow.Width

The width of the Cad window in screen pixels. Read/Write **Long**.

? CadWindow Overview

Methods:

CadWindow.Print

Syntax

Return Value=expression.Print(long *Option*, BOOL *DrawRuler*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to **CadWindow** object.

Option: Required **Long** that indicates the type of printing to occur. Options include Scale to Fit on Single Page, Print Visible Screen Area, Print Complete Views, and Print Complete View w/ Current Scale. Print Visible Screen Area is only available one of the views are zoomed. Print Complete Views is only available when multiple views exist.

DrawRuler: Required **BOOL** that indicates whether rulers should be included on the printout. This option is only available if rulers are currently turned on in the cad drawing.

Prints the Cad window

? CadWindow Overview

CadWindows Object Overview

The CadWindows object is an object containing a collection of CadWindow objects currently available to a part program.

Currently, there is exactly one CadWindow object associated with each part program, but the CAD Windows object class is made available for future changes.

CadWindows Members

Properties:

CadWindows.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the Active**PartProgram** property returns a **PartProgram** object.

? Application Overview

? CadWindows Overview

CadWindows.Count

Returns the number of CadWindow objects active in this part program. Read-only **Long**.

Currently, this property always returns one.

? CadWindows Overview

CadWindows.Parent

Represents the parent **PartProgram** object. Read-only.

? PartProgram Overview

? CadWindows Overview

Methods:

CadWindows.Item

Syntax

Return Value=expression.Item(*Item*)

Return Value: This method returns the **CadWindow** object from the parent **CadWindows** object. Read-only.

expression: Required expression that evaluates to **FlowControlCommand** object.

Item: Required **Variant** that denotes which **CadWindow** object to return.

Since there is only and exactly one **CadWindow** object, it does not matter what you pass into the *Item* argument. For the sake of future compatibility, you should pass 1.

? CadWindows Overview

Calibration Object Overview

The calibration object allows for tip calibration during part program execution. This object is placed into a part program through the add method of the commands object and obtained from the command object via the CalibrationCommand property.

Calibration Members

Properties:

Calibration.Moved

BOOLEAN value that represents whether the sphere used as the calibration tool has moved since the last tip calibration.

- If this value is true, then the tool's (identified by ToolID) calibration data is reset using the data from the sphere (identified by SphereID) that was just measured.
- If this value is false, then the current tool calibration data is used to calibrate the active tip.

Read/Write **Boolean**

? Calibration Overview

Calibration.SphereID

ID of a sphere command that occurs prior to the calibration commmand. The sphere should have identical characteristics with the tool identified by ToolID.

Read/Write **String**

? Calibration Overview

Calibration.ToolID

ID of a previously defined calibration tool that is similar to the sphere identified by SphereID. The tool data is used in the tip calibration or reset depending on the value of the moved data member.

? Calibration Overview

Command Object Overview

The **Command** object represents a single command in PC-DMIS.

? Command Members

Command Members

The **Command** object represents a single command in PC-DMIS. Examples of single commands in PC-DMIS are the start of a feature, a hit, the end of a feature, a single X dimension line, an auto feature, etc.

It is also a collection object representing the collection of executions of this object so far in the current execution or the collection of executions of this object in the previous execution.

Properties:

Command.ActiveTipCommand

Returns an ActiveTip Command object if Command is of *Type* SET_ACTIVE_TIP.

? Active Tip Object

Command.AlignmentCommand

Returns this **Command** object as an **AlignCommand** object if it can, **Nothing** otherwise.

The **Commands** that have the following *Type* can become **AlignCommand** objects are as follows:

```
START_ALIGN
LEVEL_ALIGN
ROTATE_ALIGN
TRANS_ALIGN
TRANSOFF_ALIGN
ROTATEOFF_ALIGN
SAVE_ALIGN
RECALL_ALIGN
EQUATE_ALIGN
ITER_ALIGN
BF2D_ALIGN
ROTATE_CIRCLE_ALIGN
BF3D_ALIGN
```

? AlignCommand Overview

? Command.Type Property

? Command Overview

Command.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the Active**PartProgram** property returns a **PartProgram** object.

? Application Overview

? Command Overview

Command.ArrayIndexCommand

Returns an ArrayIndex Command object if Command is of *Type* ARRAY_INDEX.

? ArrayIndex Overview

Command.AttachCommand

Returns an Attach Command object if Command is of *Type* ATTACH_PROGRAM.

? Attach Overview

Command.BasicScanCommand

Returns this **Command** object as an **BasicScanCommand** object if it can, **Nothing** otherwise. Read-only.

Only **Command** objects of type BASIC_SCAN_OBJECT can become **BasicScanCommand** objects.

? BasicScanCommand Overview

? Command Overview

Command.CalibrationCommand

Returns a Calibration Command object if Command is of *Type* CALIB_SPHERE.

? Calibration Overview

Command.Count

Represents the number of copies of this **Command** are available. If the part program is currently being executed, it is the number of times it has been executed so far. If the part program is not currently being executed, it is the number of times it was executed during the previous execution cycle. If **Command** has never been executed, *Count* has the value one. Read-only **Long**.

? Command Overview

Command.DimensionCommand

Returns this **Command** object as an **DimensionCommand** object if it can, **Nothing** otherwise. Read-only.

The **Command** objects that have the following *Type* can become **DimensionCommand** objects:

```
DIMENSION_START_LOCATION
DIMENSION_X_LOCATION
DIMENSION_Y_LOCATION
DIMENSION_Z_LOCATION
DIMENSION_D_LOCATION
DIMENSION_R_LOCATION
DIMENSION_A_LOCATION
DIMENSION_T_LOCATION
DIMENSION_V_LOCATION
DIMENSION_L_LOCATION
DIMENSION_H_LOCATION
DIMENSION_PR_LOCATION
DIMENSION_PA_LOCATION
DIMENSION_PD_LOCATION
DIMENSION_RT_LOCATION
DIMENSION_S_LOCATION
DIMENSION_RS_LOCATION
DIMENSION_STRAIGHTNESS
DIMENSION_ROUNDNESS
DIMENSION_FLATNESS
DIMENSION_PERPENDICULARITY
DIMENSION_PARALLELISM
DIMENSION_PROFILE
DIMENSION_3D_DISTANCE
```

```
DIMENSION_2D_DISTANCE
DIMENSION_3D_ANGLE
DIMENSION_2D_ANGLE
DIMENSION_RUNOUT
DIMENSION_CONCENTRICITY
DIMENSION_ANGULARITY
DIMENSION_KEYIN
DIMENSION_TRUE_START_POSITION
DIMENSION_TRUE_X_LOCATION
DIMENSION_TRUE_Y_LOCATION
DIMENSION_TRUE_Z_LOCATION
DIMENSION_TRUE_DD_LOCATION
DIMENSION_TRUE_DF_LOCATION
DIMENSION_TRUE_PR_LOCATION
DIMENSION_TRUE_PA_LOCATION
DIMENSION_TRUE_DIAM_LOCATION
```

? DimensionCommand Overview

? Command.Type Property

? Command Overview

Command.DimFormatCommand

Returns a DimFormat Command object if Command is of *Type* DIMENSION_FORMAT.

? Dimension Format Overview

Command.DimInfoCommand

Returns a DimInfo Command object if Command is of *Type* DIMENSION_INFORMATION.

? Dimension Format Overview

Command.DisplayMetaFileCommand

Returns a DispMetaFile Command object if Command is of *Type* DISPLAY_METAFILE.

? Display Metafile Overview

Command.ExternalCommand

Returns an ExternalCommand Command object if Command is of *Type* EXTERNAL_COMMAND.

? ExternalCommand Overview

Command.Feature

Represents the kind of feature that this **Command** object is. If it is not a feature it will return zero. Otherwise it will return a value from the following list. Read-only Long.

Type of Feature	**Return Value**
POINT	1
CIRCLE	2
SPHERE	3
LINE	4
CONE	5
CYLINDER	6
PLANE	7

CURVE	8
SLOT	9
SET	10
ELLIPSE	11
SURFACE	12

Command Overview

Command.FeatureCommand

Returns this **Command** object as an **FeatureCommand** object if it can, **Nothing** otherwise. Read-only.

The **Commands** that have the following *Type* can become **FeatureCommand** objects are as follows:

```
ANGLE_HIT
AUTO_ANGLE_FEATURE
AUTO_CIRCLE
AUTO_CORNER_FEATURE
AUTO_CYLINDER
AUTO_EDGE_FEATURE
AUTO_ELLIPSE
AUTO_HIGH_FEATURE
AUTO_NOTCH
AUTO_ROUND_SLOT
AUTO_SPHERE
AUTO_SQUARE_SLOT
AUTO_SURFACE_FEATURE
AUTO_VECTOR_FEATURE
BASIC_HIT
CONST_ALN_LINE
CONST_ALN_PLANE
CONST_BF_CIRCLE
CONST_BF_CONE
CONST_BF_CYLINDER
CONST_BF_LINE
CONST_BF_PLANE
CONST_BF_SPHERE
CONST_BFRE_CIRCLE
CONST_BFRE_CONE
CONST_BFRE_CYLINDER
CONST_BFRE_LINE
CONST_BFRE_PLANE
CONST_BFRE_SPHERE
CONST_CAST_CIRCLE
CONST_CAST_CONE
CONST_CAST_CYLINDER
CONST_CAST_LINE
CONST_CAST_PLANE
CONST_CAST_POINT
CONST_CAST_SPHERE
CONST_CONE_CIRCLE
CONST_CORNER_POINT
CONST_DROP_POINT
CONST_HIPNT_PLANE
CONST_INT_CIRCLE
CONST_INT_LINE
CONST_INT_POINT
CONST_MID_LINE
CONST_MID_PLANE
CONST_MID_POINT
CONST_OFF_LINE
CONST_OFF_PLANE
CONST_OFF_POINT
CONST_ORIG_POINT
CONST_PIERCE_POINT
CONST_PLTO_LINE
```

```
    CONST_PLTO_PLANE
    CONST_PROJ_CIRCLE
    CONST_PROJ_CONE
    CONST_PROJ_CYLINDER
    CONST_PROJ_LINE
    CONST_PROJ_POINT
    CONST_PROJ_SPHERE
    CONST_PRTO_LINE
    CONST_PRTO_PLANE
    CONST_REV_CIRCLE
    CONST_REV_CONE
    CONST_REV_CYLINDER
    CONST_REV_LINE
    CONST_REV_PLANE
    CONST_REV_SPHERE
    CONST_ROUND_SLOT
    CONST_SET
    CORNER_HIT
    EDGE_HIT
    GENERIC_CONSTRUCTION
    MEASURED_CIRCLE
    MEASURED_CONE
    MEASURED_CYLINDER
    MEASURED_LINE
    MEASURED_PLANE
    MEASURED_POINT
    MEASURED_SET
    MEASURED_SPHERE
    SURFACE_HIT
    VECTOR_HIT
```

? FeatureCommand Overview

? Command.Type Property

? Command Overview

Command.FileIOCommand

Returns a FileIO Command object if Command is of *Type* FILE_IO_OBJECT.

? FileIO Overview

Command.FlowControlCommand

Returns this **Command** object as an **FlowControlCommand** object if it can, **Nothing** otherwise. Read-only.

The **Commands** that have the following *Type* can become **FlowControlCommand** objects are as follows:

```
LOOP_START
    START_SUBROUTINE
    CALL_SUBROUTINE
    LABEL
    GOTO
    IF_GOTO_COMMAND
    BASIC_SCRIPT
    ONERROR
    WHILE_COMMAND
    ENDWHILE_COMMAND
    IF_BLOCK_COMMAND
    END_IF_COMMAND
    IF_ELSE_COMMAND
    END_IF_ELSE_COMMAND,
    END_ELSE_COMMAND
    DO_COMMAND
    UNTIL_COMMAND
    CASE_COMMAND
    END_CASE_COMMAND
    DEFAULT_CASE_COMMAND
    END_DEFAULT_CASE_COMMAND
```

```
SELECT_COMMAND
END_SELECT_COMMAND
```

? FlowCommand Overview

? Command.Type Property

? Command Overview

Command.ID

Represents the ID of the command. Read/write **String**.

Remarks

Only objects that have ID strings can be set. If a object does not have a string, this property is the zero-length string "".

? Command Overview

Command.IsAlignment

Returns TRUE if the command is an alignment command type. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve an Alignment Command object using the AlignmentCommand Property.

? Command Overview

Command.IsActiveTip

Returns TRUE if the command is an ActiveTip command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve an ActiveTip Command object using the ActiveTipCommand Property.

Command.IsAttach

Returns TRUE if the command is an Attach command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve an Attach Command object using the AttachCommand Property.

Command.IsArrayIndex

Returns TRUE if the command is an ArrayIndex command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve an ArrayIndex Command object using the ArrayIndexCommand Property.

Command.IsBasicScan

Returns TRUE if the command is a basic scan command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a Basic Scan Command object using the BasicScanCommand Property.

Command.IsCalibration

Returns TRUE if the command is a Calibration command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a Calibration Command object using the CalibrationCommand Property.

Command.IsComment

Returns TRUE if the command is a Comment command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a Comment Command object using the CommentCommand Property.

? Command Overview

Command.IsConstructedFeature

Returns TRUE if the command is a constructed feature. Read only **BOOL**.

? Command Overview

Command.IsDCCFeature

Returns TRUE if the command is a DCC (Auto) Feature. Read only **BOOL**.

? Command Overview

Command.IsDimension

Returns TRUE if the command is a dimension command type. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a Dimension Command object using the DimensionCommand Property.

? Command Overview

Command.IsDimFormat

Returns TRUE if the command is a DimFormat command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a DimFormat Command object using the DimFormatCommand Property.

Command.IsDimInfo

Returns TRUE if the command is a DimInfo command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a DimInfo Command object using the DimInfoCommand Property.

Command.IsDisplayMetaFile

Returns TRUE if the command is a DispMetaFile command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a DispMetaFile Command object using the DisplayMetaFileCommand Property.

Command.IsExternalCommand

Returns TRUE if the command is an externalcommand command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve an External Command object using the ExternalCommand Property.

Command.IsFileIOCommand

Returns TRUE if the command is a FileIO command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a FileIO command object using the FileIOCommand Property.

Command.IsFeature

Returns TRUE if the command is a feature command type. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a Feature Command object using the FeatureCommand Property.

Command Overview

Command.IsFlowControl

Returns TRUE if the command is a flow control command type. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a Flow Control Command object using the FlowContorlCommand Property.

Command Overview

Command.IsHit

Returns TRUE if the command is a one of the hit command types. Read only **BOOL**.

Command Overview

Command.IsLeitzMotion

Returns TRUE if the command is a LeitzMotion command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a LetizMotion Command object using the LeitzMotionCommand Property.

Command.IsLoadMachine

Returns TRUE if the command is a LoadMachine command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a LoadMachine Command object using the LoadProbeCommand Property.

Command.IsLoadProbe

Returns TRUE if the command is a LoadProbe command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a LoadProbe Command object using the LoadProbeCommand Property.

Command.IsModal

Returns TRUE if the command is a modal command type. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a Modal Command object using the ModalCommand Property.

? Command Overview

Command.IsMeasuredFeature

Returns TRUE if the command is a measured feature command. Read only **BOOL**.

? Command Overview

Command.IsMove

Returns TRUE if the command is a move command type. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a Move Command object using the MoveCommand Property.

? Command Overview

Command.IsOptMotion

Returns TRUE if the command is an OptMotion command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve an OptMotion Command object using the OptMotionCommand Property.

Command.IsStatistic

Returns TRUE if the command is a Statistics command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a Statistics Command object using the StatisticCommand Property.

Command.IsScan

Returns TRUE if the command is a Scan command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a Scan Command object using the ScanCommand Property.

Command.IsTempComp

Returns TRUE if the command is a TempComp command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a TempComp Command object using the TempCompCommand Property.

Command.IsTraceField

Returns TRUE if the command is a TraceField command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a TraceField Command object using the TraceFieldCommand Property.

Command.LeitzMotionCommand

Returns a LietzMotion Command object if Command is of *Type* OPTIONPROBE.

? Lietz Motion Overview

Command.LoadMachineCommand

Returns a LoadMachine Command object if Command is of *Type* GET_MACHINE_DATA.

? Load Machine Overview

Command.LoadProbeCommand

Returns a LoadProbe Command object if Command is of *Type* GET_PROBE_DATA.

? Load Probes Overview

Command.ModalCommand

Returns this **Command** object as a **ModalCommand** object if it can, **Nothing** otherwise. Read-only.

The **Command** objects that have the following *Type* can become **ModalCommand** objects are as follows:

```
CLAMP
PREHIT
RETRACT
CHECK
MOVE_SPEED
TOUCH_SPEED
SCAN_SPEED
CLEARANCE_PLANES
MAN_DCC_MODE
DISPLAYPRECISION
PROBE_COMPENSATION
POLARVECTORCOMP
SET_WORKPLANE
RMEAS_MODE
GAP_ONLY
RETROLINEAR_ONLY
FLY_MODE
COLUMN132
```

? FeatureCommand Overview

? Command.Type Property

? Command Overview

Command.MoveCommand

Returns this **Command** object as a **MoveCommand** object if it can, **Nothing** otherwise. Read-only.

The **Command** objects that have the following *Type* can become **MoveCommand** objects are as follows:

```
MOVE_POINT = 150,
MOVE_ROTAB = 153,
MOVE_INCREMENT = 154,
MOVE_CIRCULAR = 155,
MOVE_PH9_OFFSET = 156,
```

? MoveCommand Overview

? Command.Type Property

? Command Overview

Command.OptMotionCommand

Returns an OptMotion Command object if Command is of *Type* OPTIONMOTION.

? OptMotion Object Overview

Command.Parent

Returns the parent **Commands** collection object. Read-only.

? Command Overview

Command.ScanCommand

Returns a Scan Command object if Command is of *Type* DCCSCAN_OBJECT or *Type* MANSCAN_OBJECT.

? ScanCommand Overview

Command.ShowIDOnCad

Property used to indicate/set whether the command ID should be displayed in the CAD window. Read/Write **Boolean**

? Command Overview

Command.SlaveArm

Property used to indicate/set whether command is a slave arm object. Read/Write **Boolean**

? Command Overview

Command.StatisticCommand

Returns a Statistics Command object if Command is of *Type* STATISTICS.

? StatisticsOverview

Command.TempCompCommand

Returns a TempComp Command object if Command is of *Type* TEMP_COMP.

? Temperature Compensation Overview

Command.TraceFieldCommand

Returns a TraceField Command object if Command is of *Type* TRACEFIELD.

? TraceField Overview

? Command Overview

Command.Type

Returns the type of the **Command**. Read-only **OBTYPE**.

Remarks

The returned type is the same as the type argument to Commands.Add.

? Commands.Add

? Command Overview

Methods:

Command.Execute

Syntax

Return Value=expression.Execute

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **Command** object.

Executes the command if the command is immediately executable.

? Command Overview

Command.Dialog

Syntax

Return Value=expression.Dialog

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **Command** object.

Opens the PC-DMIS dialog for the corresponding command.

? Command Overview

Command.Dialog2

Syntax

Return Value=expression.Dialog2(*Object* *Dialog)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **Command** object.

Object: Dmis dialog command object returned if the dialog is a modeless dialog.

Opens the PC-DMIS dialog for for the corresponding command.

? Command Overview

? DmisDialog Overview

Command.GetExpression

Syntax

expression.GetExpression(*FieldType, TypeIndex)*

Return Value: **String** which is the expression on the given field if it has an expression. Otherwise, the string will be empty.

expression: Required expression that evaluates to a PC-DMIS **Command** object.

FieldType: Used to indicate which field the expression is being set for. Type **ENUM_FIELD_TYPES enumeration**.

TypeIndex: **Long** value used to indicate which instance of the supplied field type to use when an object has more than one instance of a field type.

Gets the expression of the indicated field of the command.

Remarks

Use this command to get expressions for different object fields. The ENUM_FIELD_TYPES enumeration is a large enumeration. Documentation for which field types go with which objects is not given here. You can find this information by creating the desired object in PC-DMIS, inserting the desired expression in the desired field, and exporting (posting out) the containing part program to BASIC.

? Command Overview

Command.Item

Syntax 1

Return value=expression.Item(*Num*)

Return Value: The Item function returns a **Command** object.

expression: Required expression that evaluates to a **Machines** object.

Num: Required **Long** that indicates which **Command** object to return. It is the index number of the execution in the current or previous execution

? Command Overview

Command.Mark

Syntax

expression.Mark *SameAlign*

expression: Required expression that evaluates to a PC-DMIS **Command** object.

SameAlign: Required Boolean. If *SameAlign* is FALSE, the features that are a part of the alignment for this **Command** will be marked. Otherwise, they will not be marked.

Marks the current object and all objects that depend on it. Optionally the features of the current alignment are also marked.

Remarks

If the object is a measured feature, its hits are marked. If the object is a constructed feature, the features on which it depends are marked. If the object is a dimension, the dimension feature(s) being dimensioned are marked.

? Command Overview

Command.Next

Syntax

Return Value=expression.Next

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **Command** object.

Sets *expression* to the next command in the parent **Commands** list. If *expression* is the last command, it remains unchanged. This function returns FALSE if *expression* is the last command in the parent **Commands** list, TRUE otherwise.

? Command Overview

Command.Prev

Syntax

Return Value=expression.Prev

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **Command** object.

Sets *expression* to the previous command in the parent **Commands** list. If *expression* is the first command, it remains unchanged. This function returns FALSE if *expression* is the first command in the parent **Commands** list, TRUE otherwise.

? Command Overview

Command.Remove

Syntax

expression.Remove

expression: Required expression that evaluates to a PC-DMIS **Command** object.

Removes *expression* from the part **Commands** list.

Remarks

If there are other objects which depend on *expression*, they are also removed. For example, if *expression* is a measured feature, its hits are removed as well.

? Command Overview

Commands Object Overview

The Commands collection object contains all the command objects in a part program. Use **Commands**(*index*) where *index* is the index number to return a single **Command** object.

Commands Members

Properties:

Commands.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? Command Overview

? Commands Overview

Commands.Count

Represents the number of Command objects in the parent **PartProgram** object. Read-only **Long**.

? Command Overview

? Commands Overview

Commands.Parent

Returns the parent **PartProgram** object. Read-only.

? PartProgram Overview

? Command Overview

? Commands Overview

Methods:

Commands.Add

Syntax

Return Value=expression.Add(*Type, AutoPosition*)

Return Value: This function returns the Command object added.

expression: Required expression that evaluates to a PC-DMIS **Commands** object.

Type: Required LONG in the **OBTYPE** enumeration that denotes what type of object to create.

AutoPosition: Required **Boolean** that determines what should happen when the new **Command** object is being inserted in an inappropriate place in the part program. If *AutoPosition* is FALSE, it will not be inserted at all. If it is TRUE, the new **Command** will be inserted at the new appropriate position.

? Command Overview

? Commands Overview

Commands.ClearMarked

Syntax

Return Value=expression.ClearMarked

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **Commands** object.

Clears all marked **Command** objects in this part program. ClearMarked always returns TRUE.

? Command Overview

? Commands Overview

Commands.InsertionPointAfter

Syntax

Return Value=expression.InsertionPointAfter(*Cmd)*

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **Commands** object.

Cmd: Required **Command** object that indicates which command after which to set the insertion point.

This function returns TRUE if the insertion point was successfully set, FALSE otherwise.

? Command Overview

? Commands Overview

Commands.Item

Syntax 1

Return Value=expression.Item(*NameOrNum*)

Syntax 2

expression(*NameOrNum*)

Return Value: The Item function returns a **Command** object.

expression: Required expression that evaluates to a **Commands** object.

Identifier: Required **Long** that indicates which **Command** object to return. It is the index number of the desired **Command** in the **Commands** collection denoted by *expression*.

? Command Overview

? Commands Overview

Commands.MarkAll

Syntax

Return Value=expression.MarkAll(*MarkManual*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **Commands** object.

MarkManual: Required **Boolean** that indicates whether or not to mark manual alignment features.

This function always returns TRUE

? Command Overview

? Commands Overview

Comment Object Overview

The Comment Automation object gives access to the properties of the PC-DMIS Comment command.

? Comment Members

Comment Members

Properties:

Comment.Comment

STRING value representing the comment text. Since comments in PC-DMIS can be multi-line comments, this property represents the full text of all the lines. Each line is separated by ASCII character 13 and ASCII character 10 in that order. This is a read only property. To set individual lines of the comment use the SetLine method. To get individual lines of the comment use the GetLine method.

Read Only **String**

? Comment Object Overview

Comment.CommentType

ENUM_PCD_COMMENT_TYPES enumeration type value representing the type of comment. The following enumeration values are available:

```
PCD_COMMENT_OPER = 0
PCD_COMMENT_REPORT = 1
PCD_COMMENT_INPUT = 2
PCD_COMMENT_DOCUMENTATION = 3
PCD_COMMENT_YESNO = 4
```

Read/Write **ENUM_PCD_COMMENT_TYPES enumeration type**

? Comment Object Overview

Comment.ID

STRING value representing the ID of the comment. The ID is only used for comments of type INPUT and type YESNO.

Read/Write **String**

? Comment Object Overview

Comment.Input

STRING value representing the text input by the user for comments of type INPUT or YESNO.

Read/Write **String**

Methods:

Comment.AddLine

Syntax:

expression.AddLine (Text)

Return Value: Boolean value indicating success or failure of call to method.

expression: Required expression that evaluates to a PC-DMIS **Comment** object.

Text: Required **String** representing the line of text to be added to the comment.

? Comment Object Overview

Comment.GetLine

Syntax:

expression.GetLine (Line)

Return Value: String text of the line of the comment specified by the line paramter. If Line is greater than the number of current lines in the comment, the string will be empty.

expression: Required expression that evaluates to a PC-DMIS **Comment** object.

Line: Required **Long** representing the line of text to be retrieved.

? Comment Object Overview

Comment.RemoveLine

Syntax:

expression.RemoveLine (Line)

Return Value: Boolean value indicating success or failure of call to remove a line of text from the comment. If Line is greater than the number of current lines in the comment, the call will fail.

expression: Required expression that evaluates to a PC-DMIS **Comment** object.

Line: Required **Long** representing the line of text to be removed.

? Comment Object Overview

Comment.SetLine

Syntax:

expression.SetLine (Line, Text)

Return Value: Boolean value indicating success or failure of call to set the line of text. If Line is greater than the number of current lines in the comment, the call will fail.

expression: Required expression that evaluates to a PC-DMIS **Comment** object.

Line: Required **Long** representing the line of text to be set.

Text: Required **String** which is the text to be used to set the text for the line of the comment.

? Comment Object Overview

DimData Object Overview

The DimData object is similar to a type define as follows:

```
Type DimData
        Bonus as Double
        Dev as Double
        DevAngle as Double
        Max as Double
        Meas as Double
        Min as Double
        Minus as Double
        Out as Double
        Nom as Double
        Plus as Double
End Type
```

It is be used to pass dimension information in automation functions that accept this type

? DimData Object Members

DimData Members

Properties

DimData.Bonus

Represents the Bonus member of this object. Read/write **Double**.

? DimData Object Overview

DimData.Dev

Represents the Dev member of this object. Read/write **Double**.

Remarks

The Dev member is the default property.

? DimData Object Overview

DimData.DevAngle

Represents the DevAngle member of this object. Read/write **Double**.

? DimData Object Overview

DimData.Max

Represents the Max member of this object. Read/write **Double**.

? DimData Object Overview

DimData.Meas

Represents the Meas member of this object. Read/write **Double**.

? DimData Object Overview

DimData.Min

Represents the Min member of this object. Read/write **Double**.

? DimData Object Overview

DimData.Minus

Represents the Minus member of this object. Read/write **Double**.

? DimData Object Overview

DimData.Out

Represents the Out member of this object. Read/write **Double**.

? DimData Object Overview

DimData.Nom

Represents the Nom member of this object. Read/write **Double**.

? DimData Object Overview

DimData.Plus

Represents the Plus member of this object. Read/write **Double**.

? DimData Object Overview

DimensionCommand Object Overview

Objects of type **DimensionCommand** are created from more generic **Command** objects to pass information specific to the dimension command back and forth.

? Command.DimensionCommand

? DimensionCommand Members

DimensionCommand Members

Properties:

DimensionCommand.Angle

Represents the theoretical angle of a DIMENSION_ANGULARITY dimension. Read/Write **Double**.

Remarks

This function only works for objects of type DIMENSION_ANGULARITY. If used on any other object type, setting this variable will do nothing, and getting this variable will return zero.

? Command.Type Property

? DimensionCommand Overview

DimensionCommand.ArrowMultiplier

Multiplier for display arrows of dimension. Read/Write **Double.**

DimensionCommand.Axis

Axis used with dimension. Possible values include the following:

```
DIMAXIS_NONE
DIMAXIS_XAXIS
DIMAXIS_YAXIS
DIMAXIS_ZAXIS
```

Read/Write **Enum_Dim_AxisType Enumeration**.

Remarks

This function only works with dimensions that can accept an axis as one of the inputs.

DimensionCommand.AxisLetter

Axis letter used to describe the axis or type of the dimension. Read only **String**.

DimensionCommand.Bonus

Returns the bonus tolerance of a true position dimension. Read-only **Double**.

Remarks

This function only works for single true position objects, i.e., DIMENSION_TRUE_Z_LOCATION, but not DIMENSION_TRUE_START_POSITION or DIMENSION_TRUE_END_POSITION. If used on any other object type, getting this variable will return zero.

? Command.Type Property

? DimensionCommand Overview

DimensionCommand.Deviation

Returns the deviation of a dimension. Read/Write **Double**.

? DimensionCommand Overview

DimensionCommand.DevAngle

Returns the deviation angle of a dimension. Read/Write **Double**.

? DimensionCommand Overview

DimensionCommand.GraphicalAnalysis

Flag indicating whether graphical analysis is ON for the dimension. Read/Write **Boolean.**

DimensionCommand.ID

Returns the ID of a dimension. Read/Write **String**.

Remarks

For location and true position dimensions, only the start object has an id. For single location or true position object, i.e., DIMENSION_TRUE_Z_LOCATION or DIMENSION_Y_LOCATION, setting the *ID* property has no afffect and getting it returns the empty string.

? Command.Type Property

? DimensionCommand Overview

DimensionCommand.Feat1

Returns the ID of the first feature associated with a dimension. Read/Write **String**.

Remarks

For location and true position dimensions, only the start object has an associated feature. For single location or true position object, i.e., DIMENSION_TRUE_Z_LOCATION or DIMENSION_Y_LOCATION, setting the Feat1 property has no afffect and getting it returns the empty string. Also, objects of type DIMENSION_KEYIN have no associated features.

? Command.Type Property

? DimensionCommand Overview

DimensionCommand.Feat2

Returns the ID of the second feature associated with a dimension. Read/Write **String**.

Remarks

Not every dimension type has two features associated with it. Trying to set the Feat2 property of one of these types has no effect, and getting it returns the empty string.

? Command.Type Property

? DimensionCommand Overview

DimensionCommand.Feat3

Returns the ID of the second feature associated with a dimension. Read/Write **String**.

Remarks

Not every dimension type has three features associated with it. Trying to set the Feat3 property of one of these types has no effect, and getting it returns the empty string.

? Command.Type Property

? DimensionCommand Overview

DimensionCommand.Length

Returns the length associated with a dimension. Read/Write **Double**.

Remarks

Only object of type DIMENSION_ANGULARITY, DIMENSION_ANGULARITY, DIMENSION_PERPENDICULARITY, and DIMENSION_PROFILE have a useful length property. For all other types, setting the property has no effect, and getting it always returns zero.

? Command.Type Property

? DimensionCommand Overview

DimensionCommand.Nominal

Returns the nominal associated with a dimension. Read/Write **Double**.

Remarks

Only object of type DIMENSION_START_LOCATION, DIMENSION_TRUE_START_POSITION do not have a useful nominal property. For these types, setting the property has no effect, and getting it always returns zero.

? Command.Type Property

? DimensionCommand Overview

DimensionCommand.Max

Returns the maximum value of a dimension. Read-only **Double**.

? DimensionCommand Overview

DimensionCommand.Measured

Returns the measured value of a dimension. Read-only **Double**.

? DimensionCommand Overview

DimensionCommand.Min

Returns the minimum value of a dimension. Read-only **Double**.

? DimensionCommand Overview

DimensionCommand.Minus

Represents the negative tolerance of a dimension. Read/write **Double**.

? DimensionCommand Overview

DimensionCommand.OutputMode

Output mode of the dimension. Possible values include the following:

`DIMOUTPUT_STATS`

`DIMOUTPUT_REPORT`

`DIMOUTPUT_BOTH`

Read/Write **Enum_Dim_OutputType Enumeration**.

Remarks

The output mode determines where to send dimension data during execution.

DimensionCommand.OutTol

Returns the out-of-tolerance value of a dimension. Read-only **Double**.

? DimensionCommand Overview

DimensionCommand.ParallelPerpindicular

Indicates whether calculations are performed parallel or perpindicular to input for 2-D dimensions. Possible values include the following:

`DIM_PERPINDICULAR`

`DIM_PARALLEL`

Read/Write **Enum_Dim_Perp_Parallel Enumeration**.

DimensionCommand.Profile

Enumeration value indicating what type of profile should be used. Possible values include the following:

`DIM_PROF_FORM_ONLY`

`DIM_PROF_FORM_AND_LOCATION`

Read/Write **Enum_Dim_Prof_Type Enumeration**.

DimensionCommand.Plus

Returns the positive tolerance of a dimension. Read-only **Double**.

? DimensionCommand Overview

DimensionCommand.Parent

Returns the parent **Command** object. Read-only.

Remarks

The parent of a **DimensionCommand** object is the same underlying PC-DMIS object as the **DimensionCommand** object itself. Getting the parent allows you to access the generic **Command** properties and methods of a given object.

? Command Overview

? DimensionCommand Overview

DimensionCommand.RadiusType

Radius calculation type used with true position dimensions. Possible values include the following:

DIM_NO_RADIUS

DIM_ADD_RADIUS

DIM_SUB_RADIUS

Read/Write **Enum_Dim_Radius_Type Enumeration**.

DimensionCommand.TextualAnalysis

Flag indicating whether textual analysis is ON for the dimension. Read/Write **Boolean.**

DimensionCommand.TruePositionModifier

Enumeration value indicating material conditions that should be used to calculate possible bonus tolerances. Possible values include the following:

DIM_RFS_RFS

DIM_RFS_MMC

DIM_RFS_LMC

DIM_MMC_RFS

DIM_MMC_MMC

DIM_MMC_LMC

DIM_LMC_RFS

DIM_LMC_MMC

DIM_LMC_LMC

Read/Write **Enum_Dim_TP_Modifier Enumeration**.

DimensionCommand.TruePosUseAxis

Enumeration value indicating axis type to use with true position dimension. Possible values include the following:

DIM_AXIS_AVERAGE

DIM_AXIS_START_POINT

DIM_AXIS_END_POINT

Read/Write **Enum_Dim_TP_Use_Axis Enumeration**.

DimensionCommand.UnitType

Unit type in use by dimension. Possible values include the following:

INCH

MM (for millimeters)

Read/Write **UnitType Enumeration**.

Dimension Format Object Overview

The Dimension Format Automation object gives access to the properties of the PC-DMIS Dimension Format command. For additional information on dimensions, see the topic "Dimension Options" in the *PC-DMIS Reference Manual*.

? Dimension Format Members

Dimension Format Members

Properties:

DimFormat.ShowDevSymbols

BOOLEAN value representing whether deviation symbols should be shown in the dimension report text.

Read/Write **Boolean**

? Dimension Format Object Overview

DimFormat.ShowDimensionText

BOOLEAN value indicating whether the top two lines of the dimension command should appear or not.

Read/Write **Boolean**

? Dimension Format Object Overview

DimFormat.ShowDimensionTextOptions

BOOLEAN value indicating whether various dimension such as arrow multiplier, graphical analysis, and textual analysis should appear in the dimension text or not.

Read/Write **Boolean**

? Dimension Format Object Overview

DimFormat.ShowHeadings

BOOLEAN value indicating whether the dimension headings such as NOM, MAX, MIN, DEV, OUTTOL, etc. should appear in the dimension text or not.

Read/Write **Boolean**

? Dimension Format Object Overview

DimFormat.ShowStdDev

BOOLEAN value indicating whether the standard deviation value should appear or not.

Read/Write **Boolean**

? Dimension Format Object Overview

Methods:

DimFormat.GetHeadingType

Syntax:

expression.GetHeadingType (Index)

Return Value: DimFormatType Enumeration value indicating the dimension information type of the position indicated by the index parameter.

Possible values include the following:

```
PCD_NOT_USED = 0
PCD_NOM = 1
PCD_TOL = 2
PCD_MEAS = 3
PCD_MAXMIN = 4
PCD_DEV = 5
PCD_OUTTOL = 6
```

expression: Required expression that evaluates to a PC-DMIS **Dimension Format** object.

Index: Required **Long** representing which index position to retrieve.

? Dimension Format Object Overview

DimFormat.SetHeadingType

Syntax:

expression.SetHeadingType (Index, HeadingType)

Return Value: Boolean indicating success or failure in setting the heading type.

expression: Required expression that evaluates to a PC-DMIS **Dim Format** object.

Index: Required long indicating the index position that is being set.

HeadingType: Required **DimFormatType Enumeration** representing the type of value to be used at the given index position.

Possible values include the following:

```
PCD_NOT_USED = 0
PCD_NOM = 1
PCD_TOL = 2
PCD_MEAS = 3
PCD_MAXMIN = 4
PCD_DEV = 5
PCD_OUTTOL = 6
```

? Dimension Format Object Overview

Dimension Information Object Overview

The Dimension Information Automation object gives access to the properties and methods of the PC-DMIS Dimension Information command. See "DIMINFO Command" in the *PC-DMIS Reference Manual* for additional information.

? Dimension Information Members

Dimension Information Members

Properties:

DimInfo.DimensionID

STRING value representing the name of the dimension for which the dimension information object will be showing information.

Read/Write **String**

? Dimension Information Object Overview

DimInfo.ShowDimensionID

BOOLEAN value indicating whether the Dimension ID should be shown in the dimension information object.

Read/Write **Boolean**

? Dimension Information Object Overview

DimInfo.ShowFeatID

BOOLEAN value indicating whether to display the feature id of the feature belonging to the dimension used in the dimension information command.

Read/Write **Boolean**

? Dimension Information Object Overview

Methods:

DimInfo.GetFieldFormat

Syntax:

expression.GetFieldFormat (Index)

Return Value: Enum_Dinfo_Field_Types Enumeration value indicating the dimension information type of the position indicated by the index parameter.

Possible values include the following:

```
DINFO_NOT_USED = 0
DINFO_MEAS = 1
DINFO_NOM = 2
DINFO_TOL = 3
DINFO_DEV = 4
DINFO_MAXMIN = 5
DINFO_OUTTOL = 6
DINFO_MEAN = 7
DINFO_STDDEV = 8
DINFO_NUMPOINTS = 9
```

expression: Required expression that evaluates to a PC-DMIS **Dimension Information** object.

Index: Required **Long** representing which index position to retrieve.

? Dimension Information Object Overview

DimInfo.GetLocationAxis

Syntax:

expression.GetLocationAxis (Index)

Return Value: Enum_Dinfo_Loc_Axes Enumeration value indicating the dimension location axis order used at the position indicated by the index parameter. This function only works if the dimension being referenced in the command is an axis location dimension.

Possible values include the following:

```
DINFO_LOC_USE_DIM_AXES = -2
DINFO_LOC_WORST = -1
DINFO_LOC_NOT_USED = 0
DINFO_LOC_X = 1
DINFO_LOC_Y = 2
DINFO_LOC_Z = 3
DINFO_LOC_D = 4
DINFO_LOC_R = 5
DINFO_LOC_V = 6
DINFO_LOC_A = 7
DINFO_LOC_L = 8
DINFO_LOC_H = 9
DINFO_LOC_PR = 10
DINFO_LOC_PA = 11
DINFO_LOC_T = 12
DINFO_LOC_RT = 13
DINFO_LOC_S = 14
DINFO_LOC_RS = 15
DINFO_LOC_PD = 16
```

expression: Required expression that evaluates to a PC-DMIS **Dimension Information** object.

Index: Required **Long** representing which index position to retrieve.

? Dimension Information Object Overview

DimInfo.GetTruePosAxis

Syntax:

expression.GetTruePosAxis (Index)

Return Value: Enum_Dinfo_TP_Axes Enumeration value indicating the dimension true position axis order used at the position indicated by the index parameter. This command only works with dimension information commands that are referencing true position dimensions.

Possible values include the following:

```
DINFO_TP_USE_DIM_AXES = -2
DINFO_TP_WORST = -1
DINFO_TP_NOT_USED = 0
DINFO_TP_X = 1
```

```
DINFO_TP_Y = 2
DINFO_TP_Z = 3
DINFO_TP_PR = 4
DINFO_TP_PA = 5
DINFO_TP_DD = 6
DINFO_TP_LD = 7
DINFO_TP_WD = 8
DINFO_TP_DF = 9
DINFO_TP_LF = 10
DINFO_TP_WF = 11
DINFO_TP_TP = 12
```

expression: Required expression that evaluates to a PC-DMIS **Dimension Information** object.

Index: Required **Long** representing which index position to retrieve.

? Dimension Information Object Overview

DimInfo.SetFieldFormat

Syntax:

expression.SetFieldFormat (Index, FieldType)

Return Value: Boolean indicating success or failure in setting the field type.

expression: Required expression that evaluates to a PC-DMIS **Dim Information** object.

Index: Required long indicating the index position that is being set.

FieldType: Required **Enum_Dinfo_Field_Types Enumeration** representing the type of value used at the given index position.

Possible values include the following:

```
DINFO_NOT_USED = 0
DINFO_MEAS = 1
DINFO_NOM = 2
DINFO_TOL = 3
DINFO_DEV = 4
DINFO_MAXMIN = 5
DINFO_OUTTOL = 6
DINFO_MEAN = 7
DINFO_STDDEV = 8
DINFO_NUMPOINTS = 9
```

? Dimension Information Object Overview

DimInfo.SetLocationAxis

Syntax:

expression.SetFieldFormat (Index, Axis)

Return Value: Boolean indicating success or failure in setting the field type. Dimension needs to be a location dimension in order for this command to succeed.

expression: Required expression that evaluates to a PC-DMIS **Dim Information** object.

Index: Required long indicating the index position that is being set.

Axis: Required **Enum_Dinfo_Loc_Axes Enumeration** representing the type the axis used at the given index position.

Possible values include the following:

```
DINFO_LOC_USE_DIM_AXES = -2
DINFO_LOC_WORST = -1
DINFO_LOC_NOT_USED = 0
DINFO_LOC_X = 1
DINFO_LOC_Y = 2
DINFO_LOC_Z = 3
DINFO_LOC_D = 4
DINFO_LOC_R = 5
DINFO_LOC_V = 6
DINFO_LOC_A = 7
DINFO_LOC_L = 8
DINFO_LOC_H = 9
DINFO_LOC_PR = 10
DINFO_LOC_PA = 11
DINFO_LOC_T = 12
DINFO_LOC_RT = 13
DINFO_LOC_S = 14
DINFO_LOC_RS = 15
DINFO_LOC_PD = 16
```

? Dimension Information Object Overview

DimInfo.SetTruePosAxis

Syntax:

expression.SetTruePosAxis (Index, Axis)

Return Value: Boolean indicating success or failure in setting the field type. Dimension needs to be a true position dimension in order for this command to succeed.

expression: Required expression that evaluates to a PC-DMIS **Dim Information** object.

Index: Required long indicating the index position that is being set.

Axis: Required **Enum_Dinfo_TP_Axes Enumeration** representing the type the axis used at the given index position.

Possible values include the following:

```
DINFO_TP_USE_DIM_AXES = -2
DINFO_TP_WORST = -1
DINFO_TP_NOT_USED = 0
DINFO_TP_X = 1
DINFO_TP_Y = 2
DINFO_TP_Z = 3
DINFO_TP_PR = 4
DINFO_TP_PA = 5
DINFO_TP_DD = 6
```

```
DINFO_TP_LD = 7
DINFO_TP_WD = 8
DINFO_TP_DF = 9
DINFO_TP_LF = 10
DINFO_TP_WF = 11
DINFO_TP_TP = 12
```

? Dimension Information Object Overview

Display Metafile Object Overview

The Display Metafile Automation object gives access to the comment properties of the PC-DMIS Display Metafile command.

? Display Metafile Members

Display Metafile Members

Properties:

DispMetafile.Comment

STRING value representing the comment to be used as a caption for the metafile object.

Read/Write **String**

? Display Metafile Object Overview

DmisDialog Object Overview

The DmisDialog object represents a PC-DMIS modeless dialog and can be used to determine if the dialog is still visible. A DmisDialog object can be obtained from the Dialog2 method of the command automation object. This object has one property: visible.

If true, the dialog is still visible to the user. If false, the dialog either no longer exists or is no longer visible to the user.

? DmisDialog Members

DmisDialog Members

Properties:

DmisDialog.Visible

Indicates whether the dialog is still visible to the user.

Read Only: **Boolean**

? DmisDialog Overview

DmisMatrix Object Overview

The DmisMatrix object is a four by three array of doubles modeled after the transformation matrices used in PC-DMIS. The first set of three doubles represent the matrix offset. The second set of three doubles represent the X axis. The third set of three doubles represent the Y axis. The fourth set of three doubles represent the Z axis.

? DmisMatrix Members

DmisMatrix Members

Properties:

DmisMatrix.Copy

Returns a copy of the matrix.

Read Only: **DmisMatrix**

? DmisMatrix Overview

DmisMatrix.Inverse

Returns an inverse matrix of the current matrix.

Read Only: **DmisMatrix**

? DmisMatrix Overview

DmisMatrix.IsIdentity

BOOLEAN property set to true if the matrix is the identity matrix.

Read Only: **Boolean**

? DmisMatrix Overview

DmisMatrix.OffsetAxis

The first set of three doubles in the matrix representing the translation offset of the matrix.

Read/Write: **PointData**

? DmisMatrix Overview

DmisMatrix.XAxis

The second set of three doubles in the matrix representing the XAxis.

Read/Write **PointData**

? DmisMatrix Overview

DmisMatrix.YAxis

The third set of three doubles in the matrix representing the YAxis.

Read/Write **PointData**

? DmisMatrix Overview

DmisMatrix.ZAxis

The fourth set of three doubles in the matrix representing the ZAxis.

Read/Write **PointData**

? DmisMatrix Overview

Methods:

DmisMatrix.Item

Syntax:

expression.Item (Num)

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

Num: Required parameter of type **long** between 1 and 12 inclusive from which the matrix data is copied.

Return Value:
Data item of matrix of type **double**.

? DmisMatrix Overview

DmisMatrix.Multiply

Syntax:

expression.Multiply (SecondMatrix)

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

SecondMatrix: Required paramter of type **DmisMatrix** representing the second matrix.

Return Value:

Matrix that is the result of multiplying the two matrices of type **DmisMatrix**.

? DmisMatrix Overview

DmisMatrix.Normalize

Syntax:

expression.Normalize ()

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

Remarks

Normalizes the matrix.

? DmisMatrix Overview

DmisMatrix.Reset

Syntax:

expression.Reset ()

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

Remarks

Resets the matrix to the identity matrix.

? DmisMatrix Overview

DmisMatrix.RotateByAngle

Syntax:

```
expression.RotateByAngle (Angle, Workplane)
```

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

Angle: Required Double parameter representing the rotation angle (in degrees).

Workplane: Optional **Long** parameter used to define which axis to rotate about. Defaults to PCD_TOP.

Remarks

Rotates the matrix by the specified angle relative to the workplane.

? DmisMatrix Overview

DmisMatrix.RotateToPoint

Syntax:

```
expression.RotateToPoint (X, Y, Workplane)
```

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

X: Required **Double** X component used in calculating rotation angle.

Y: Required **Double** Y component used in calculation rotation angle.

Workplane: Optional **Long** parameter used to define which axis to rotate about. Defaults to PCD_TOP.

Remarks

Rotates the matrix by the calculated angle relative to the workplane.

? DmisMatrix Overview

DmisMatrix.RotateToVector

Syntax:

```
expression.RotateToVector (Vector, Workplane)
```

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

Vector: Required **Pointdata** parameter specifying the vector that the primary axis should be rotated to.

Workplane: Optional **Long** parameter used to define which axis to rotate about. Defaults to PCD_TOP.

Remarks

Rotates the primary axis (as determined by the workplane parameter) to the specified vector.

? DmisMatrix Overview

DmisMatrix.SetMatrix

Syntax:

```
expression.SetMatrix (Vector, Point, Workplane)
```

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

Vector: Required **Pointdata** parameter used with the workplane parameter to establish the orientation of the matrix.

Point: Required **Pointdata** parameter used to set the matrix offset.

Workplane: Optional **Long** parameter used to define the direction of the primary axis.

Remarks

Initializes the matrix using the vector and workplane to set the matrix orientation and the point to set the matrix offset.

? DmisMatrix Overview

DmisMatrix.TransformDataBack

Syntax:

expression.TransformDataBack (PointData, TransformationType, Workplane)

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

PointData: Required **PointData** object parameter that is modifed by multiplying the data in the point by the inverse of the matrix.

TransformationType: Optional **Long** parameter that identifies the type of transformation desired. The following options are available:

ROTATE_AND_TRANSLATE = 0

ROTATE_ONLY = 1

MAJOR_MINOR_THIRD_ROT_AND_TRANS = 2

MAJOR_MINOR_THIRD_ROTATE_ONLY = 3

The default is ROTATE_AND_TRANSLATE.

Workplane: Optional **Long** parameter used to define which axis to rotate about. Defaults to PCD_TOP. This parameter is used when the MAJOR_MINOR_THIRD_ROT_AND_TRANS parameter or the MAJOR_MINOR_THIRD_ROTATE_ONLY transformation type parameter is used.

? DmisMatrix Overview

DmisMatrix.TransformDataForward

Syntax:

expression.TransformDataForward (PointData, TransformationType, Workplane)

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

PointData: Required **PointData** object parameter that is modifed by multiplying the data in the point by the matrix.

TransformationType: Optional **Long** parameter that identifies the type of transformation desired. The following options are available:

ROTATE_AND_TRANSLATE = 0

ROTATE_ONLY = 1

MAJOR_MINOR_THIRD_ROT_AND_TRANS = 2

MAJOR_MINOR_THIRD_ROTATE_ONLY = 3

The default is ROTATE_AND_TRANSLATE.

Workplane: Optional **Long** parameter used to define which axis to rotate about. Defaults to PCD_TOP. This parameter is used when the MAJOR_MINOR_THIRD_ROT_AND_TRANS parameter or the MAJOR_MINOR_THIRD_ROTATE_ONLY transformation type parameter is used.

? DmisMatrix Overview

EditWindow Object Overview

The EditWindow object represents the edit window associated with a part program. It is always present, although sometimes it is invisible. When in command mode, the edit window lists all the commands in the part program. When in report mode, the edit window lists the part program’s current report.

? EditWindow Members

EditWindow Class Members

Properties:

EditWindow.Application

Represents the read-only PC-DMIS application. The Application object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? EditWindow Overview

EditWindow.Height

The height of the edit window in screen pixels. Read/Write **Long**.

? EditWindow Overview

EditWindow.Left

The left edge of the edit window, measured from the left edge of the Windows Desktop. Read/Write **Long**.

Remarks

The Left property is measured in screen pixels.

? EditWindow Overview

EditWindow.Parent

Returns the parent PartProgram of this object. Read-only **PartProgram**.

? PartProgram Overview

? EditWindow Overview

EditWindow.ShowAlignments

This property is TRUE if alignments are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

? EditWindow Overview

EditWindow.ShowComments

This property is TRUE if comments are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

? EditWindow Overview

EditWindow.ShowDimensions

This property is TRUE if dimensions are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

? EditWindow Overview

EditWindow.ShowFeatures

This property is TRUE if features are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

? EditWindow Overview

EditWindow.ShowHeaderFooter

This property is TRUE if headers and footers are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

? EditWindow Overview

EditWindow.ShowHits

This property is TRUE if hits are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

? EditWindow Overview

EditWindow.ShowMoves

This property is TRUE if moves are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

? EditWindow Overview

EditWindow.ShowOutTolOnly

This property is TRUE if only out-of-tolerance dimensions are being shown in the edit window, FALSE otherwise. If ShowDimensions is FALSE, this property is ignored. Read/Write **Boolean**.

? EditWindow Overview

EditWindow.ShowTips

This property is TRUE if tips are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

? EditWindow Overview

EditWindow.StatusBar

This property represents the text in the edit window's status bar. Read-Write **String**.

? EditWindow Overview

EditWindow.Top

The top edge of the edit window, measured from the top edge of the Windows Desktop. Read/Write **Long**.

Remarks

The Top property is measured in screen pixels.

? EditWindow Overview

EditWindow.Visible

This property is TRUE if the edit window is visible, FALSE otherwise. Read/write **Boolean**.

? EditWindow Overview

EditWindow.Width

The width of the edit window in screen pixels. Read/Write **Long**.

? EditWindow Overview

Methods:

EditWindow.CommandMode

Syntax

expression.CommandMode

expression: Required expression that evaluates to a PC-DMIS **EditWindow** object.

This function puts the Edit window into command mode.

? EditWindow Overview

EditWindow.Print

Syntax

expression.Print

expression: Required expression that evaluates to a PC-DMIS **EditWindow** object.

This function prints the contents of the Edit window.

? EditWindow Overview

EditWindow.ReportMode

Syntax

expression.ReportMode

expression: Required expression that evaluates to a PC-DMIS **EditWindow** object.

This function puts the Edit window into report mode.

? EditWindow Overview

EditWindow.SetPrintOptions

Syntax

expression.SetPrintOptions long Location, long Draft, long FileMode, long ExtNum

expression: Required expression that evaluates to a PC-DMIS **EditWindow** object.

Location: Destination of printed data. Options include Off, File, or Printer

Draft: When destination is printer, specifies if printer should print in draft mode or not. Options include On and Off.

FileMode: When destination is file, specifies file naming and writing parameters. Options include: Append, New File, Overwrite, and Auto. Auto mode automatically increments a numeric extension for the output file.

ExtNum: Number to be used for the file extension of the output file.

This function puts the Edit window into report mode.

? EditWindow Overview

ExternalCommand Object Overview

The external command object causes PC-DMIS to launch an external program during part program execution. This object has one property: The command property. This property consists of a string value used to execute the external command.

? ExternalCommand Members

ExternalCommand Members

Properties:

ExtCommand.Command

String value which is the command to be executed. This string should be in the same format as a string entered into Window's *Run Dialog box* (i.e. The string should include full pathname and executable name of the external command to be executed).

Read/Write **String**

? ExternalCommand Overview

FeatCommand Object Overview

Objects of type **FeatCommand** are created from more generic **Command** objects to pass information specific to the feature command back and forth.

? Command.FeatCommand

? FeatCommand Members

FeatCommand Members

Properties:

FeatCommand.AlignWorkPlane

Workplane value for constructed alignment planes and lines. Possible values include the following:

```
ALIGN_ZPLUS = 0
ALIGN_ZMINUS = 1
ALIGN_XPLUS = 2
ALIGN_XMINUS = 3
ALIGN_YPLUS = 4
ALIGN_YMINUS = 5
ALIGN_CURRENT_WORKPLANE = 6
```

Enum_Align_WorkPlane Enumeration Read/Write.

Remarks

This property applies only to PC-DMIS constructed features that have a workplane field.

? Command Overview

? FeatCommand Overview

FeatCommand.AutoCircularMove

Flag indicating whether circular moves should be used between hits. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an auto circular move field.

? Command Overview

? FeatCommand Overview

FeatCommand.AutoClearPlane

Flag indicating whether clearance planes should automatically be used with the feature. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an auto clearplane field.

? Command Overview

? FeatCommand Overview

FeatCommand.AutoMove

Auto Move Flag. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an auto move field.

? Command Overview

? FeatCommand Overview

FeatCommand.AutoMoveDistance

Distance used in calculating auto move. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an auto move distance field.

? Command Overview

? FeatCommand Overview

FeatCommand.AutoPH9

Flag indicating if selected tip should be automatically adjusted during measurement of feature. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an AutoPH9 field.

? Command Overview

? FeatCommand Overview

FeatCommand.AutoReadPos

Auto Read Position Flag. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an auto read pos field.

? Command Overview

? FeatCommand Overview

FeatCommand.BestFitMathType

Value representing the best fit math algorithm to be used in calculating the measured feature values based on the measured hits. Possible values include the following.

```
BF_MATH_LEAST_SQUARES  = 0
BF_MATH_MIN_SEPARATION = 1
BF_MATH_MAX_INSCRIBED = 2
BF_MATH_MIN_CIRCUMSCRIBED = 3
BF_MATH_FIXED_RADIUS  = 4
```

ENUM_BEST_FIT_MATH_TYPES Enumeration Read/Write.

Remarks

This property applies only to the circle and cylinder measured features and best fit constructed features.

? Command Overview

? FeatCommand Overview

FeatCommand.Bound

Flag indicating whether or not feature is bound. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a bound/unbound field.

? Command Overview

? FeatCommand Overview

FeatCommand.BoxWidth

Box width value for auto high point. **Double** Read/Write.

Remarks

This property is only applicable for the PC-DMIS auto high point command.

? Command Overview

? FeatCommand Overview

FeatCommand.BoxLength

Box length value for auto high point. **Double** Read/Write.

Remarks

This property is only applicable for the PC-DMIS auto high point command.

? Command Overview

? FeatCommand Overview

FeatCommand.CircularRadiusIn

Inside circular radius value for auto high point. **Double** Read/Write.

Remarks

This property is only applicable for the PC-DMIS auto high point command.

? Command Overview

? FeatCommand Overview

FeatCommand.CircularRadiusOut

Outside circular radius value for auto high point. **Double** Read/Write.

Remarks

This property is only applicable for the PC-DMIS auto high point command.

? Command Overview

? FeatCommand Overview

FeatCommand.CornerRadius

Corner radius value for auto square slot and auto notch objects. **Double** Read/Write.

Remarks

This property is only applicable for the PC-DMIS auto square slot and auto notch commands.

? Command Overview

? FeatCommand Overview

FeatCommand.DCCFindNomsMode

Boolean read/write value that indicates if the measurement mode for an auto feature should be done in find nominals mode or not.

Remarks

This property applies only to PC-DMIS auto features with a find nominals measurement field.

? Command Overview

? FeatCommand Overview

FeatCommand.DCCMeasureInMasterMode

Boolean read/write value that indicates if the measurement mode for an auto feature should be done in master mode or not.

Remarks

This property applies only to PC-DMIS auto features with a master mode measurement field.

? Command Overview

? FeatCommand Overview

FeatCommand.Depth

Depth value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a depth field.

? Command Overview

FeatCommand Overview

FeatCommand.Deviation

Auto sphere deviation value. **Double** Read/Write.

Remarks

This property applies only to the PC-DMIS auto sphere command.

Command Overview

FeatCommand Overview

FeatCommand.DisplayConeAngle

Flag indicating whether or not to display the angle of the cone. If this value is false, then the cone length is displayed. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS cone commands that have a display option on angle vs. length.

Command Overview

FeatCommand Overview

FeatCommand.EdgeMeasureOrder

Measure order for edge points. Possible values include the following.

```
EDGE_SURFACE_FIRST = 0
EDGE_EDGE_FIRST = 1
EDGE_BOTH =2
```

Edge_Measure_Types Enumeration Read/Write.

Remarks

This property applies only to PC-DMIS edge commands.

Command Overview

FeatCommand Overview

FeatCommand.EdgeThickness

Thickness value for edge points. **Double** Read/Write.

Remarks

This property is only applicable for PC-DMIS edge commands.

Command Overview

FeatCommand Overview

FeatCommand.EndAngle

End Angle value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an end angle field.

? Command Overview

? FeatCommand Overview

FeatCommand.EndAngle2

Second End Angle value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a second end angle field.

? Command Overview

? FeatCommand Overview

FeatCommand.FilterType

Filter object filter type. Possible values include the following:

```
FILTER_LINEAR = 0
FILTER_POLAR = 1
```

Enum_Filter_Types Enumeration Read/Write.

Remarks

This property is only applicable for the PC-DMIS filter command.

? Command Overview

? FeatCommand Overview

FeatCommand.GenericAlignMode

Generic alignment mode. Possible values include the following:

```
GENERIC_ALIGN_DEPENDENT =0
GENERIC_ALIGN_INDEPENDENT =1
```

Enum_Generic_Align Enumeration Read/Write.

Remarks

This property is only applicable for the PC-DMIS generic feature command.

? Command Overview

? FeatCommand Overview

FeatCommand.GenericDisplayMode

Generic display mode. Possible values include the following:

```
GENERIC_DISPLAY_RADIUS = 0
GENERIC_DISPLAY_DIAMETER = 1
```

Enum_Generic_Display Enumeration Read/Write.

Remarks

This property is only applicable for the PC-DMIS generic feature command.

? Command Overview

? FeatCommand Overview

FeatCommand.GenericType

Generic feature type. Possible values include the following:

```
GENERIC_POINT = 0
GENERIC_PLANE = 1
GENERIC_LINE = 2
GENERIC_CIRCLE = 3
GENERIC_SPHERE = 4
GENERIC_CYLINDER = 5
GENERIC_ROUND_SLOT = 6
GENERIC_SQUARE_SLOT = 7
GENERIC_CONE = 8
GENERIC_NONE = 9
```

Enum_Generic_Types Enumeration Read/Write.

Remarks

This property is only applicable for the PC-DMIS generic feature command.

? Command Overview

? FeatCommand Overview

FeatCommand.HighPointSearchMode

Search mode for auto high point. Possible values include the following:

```
SEARCH_MODE_BOX = 0
SEARCH_MODE_CIRCULAR = 1
```

High_Point_Search_Modes Enumeration Read/Write.

Remarks

This property is only applicable for the PC-DMIS auto high point command.

? Command Overview

? FeatCommand Overview

FeatCommand.ID

Represents the ID of the feature. Read/Write **String**.

Remarks

The IDs of the various objects in a part program should be unique.

? FeatCommand Overview

FeatCommand.Increment

Increment value for auto high point. **Double** Read/Write.

Remarks

This property is only applicable for the PC-DMIS auto high point command.

? Command Overview

? FeatCommand Overview

FeatCommand.Indent

Indent distance (used with sample hits). **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an indent field.

? Command Overview

? FeatCommand Overview

FeatCommand.Indent2

Second indent distance (used with sample hits). **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a second indent field.

? Command Overview

? FeatCommand Overview

FeatCommand.Indent3

Third indent distance (used with sample hits). **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a third indent field.

? Command Overview

? FeatCommand Overview

FeatCommand.InitHits

Number of intitial sample hits. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an init hits field.

? Command Overview

? FeatCommand Overview

FeatCommand.Inner

Boolean read/write value that indicates whether the feature is a hole (inner) or a stud (outer).

Remarks

This property applies only to PC-DMIS commands that can be either inside or outside features.

? Command Overview

? FeatCommand Overview

FeatCommand.InteriorHit

Flag used to indicate type of hit for objects that can have interior/exterior hits. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an interior/exterior hit field.

? Command Overview

? FeatCommand Overview

FeatCommand.Line3D

Boolean read/write value that indicates whether the feature is a three dimensional line or a two dimensional line. A value of false indicates a two dimensional line.

Remarks

This property applies only to PC-DMIS lines features with and 2D/3D field.

? Command Overview

? FeatCommand Overview

FeatCommand.MeasAngle

Measured angle value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have an angle field.

? Command Overview

? FeatCommand Overview

FeatCommand.MeasDiam

Measured diameter value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a diameter field.

? Command Overview

? FeatCommand Overview

FeatCommand.MeasHeight

Measured height value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a height field.

? Command Overview

? FeatCommand Overview

FeatCommand.MeasMajorAxis

Measured major axis length value (ellipse). **Double** Read only.

Remarks

This property applies only to PC-DMIS commands that have a major axis field.

? Command Overview

? FeatCommand Overview

FeatCommand.MeasMinorAxis

Measured minor axis length value (ellipse). **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a minor axis field.

? Command Overview

? FeatCommand Overview

FeatCommand.MeasLength

Measured length value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a length field.

? Command Overview

? FeatCommand Overview

FeatCommand.MeasPinDiam

Measured pin diameter value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a pin diameter field.

? Command Overview

? FeatCommand Overview

FeatCommand.MeasSmallLength

Measured shorter length value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a small length field.

? Command Overview

? FeatCommand Overview

FeatCommand.MeasureSlotWidth

Flag indicating whether the slot width should be measured. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a measure slot width flag.

? Command Overview

? FeatCommand Overview

FeatCommand.NumHits

Represents the number of inputs in the feature. Read/Write **Long**.

Remarks

If this feature is constructed, it reports the number of input features.

? FeatCommand Overview

FeatCommand.NumHitsPerRow

Represents the number of hits on each row of the feature. Read/Write **Long**.

Remarks

You can use this variable only with features that have rows (such as spheres and cylinders).

? FeatCommand Overview

FeatCommand.NumRows

Represents the number of rows in the feature. Read/Write **Long**.

Remarks

You can use this variable only with features that have rows (such as spheres and cylinders).

? FeatCommand Overview

FeatCommand.Parent

Returns the parent **Command** object. Read-only.

Remarks

The parent of a **FeatCommand** object is the same underlying PC-DMIS object as the **FeatCommand** object itself. Getting the parent allows you to access the generic **Command** properties and methods of a given object.

? Command Overview

? FeatCommand Overview

FeatCommand.PermHits

Number of permanent sample hits. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a permanent hits field.

? Command Overview

? FeatCommand Overview

FeatCommand.Polar

Flag indicating whether polar coordinates are used on the feature. Usually defaults to false. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have support for polar coordinates.

? Command Overview

? FeatCommand Overview

FeatCommand.ReferenceType

Reference type used with measured circles and measured lines. **ENUM_FEATREF_TYPES Enumeration** Read/Write.

Remarks

This property applies only to PC-DMIS measured line and measured circle commands. Possible value include the following:

```
FEATREF_FEATURE  = -3 (Use ReferenceID Property to specify feature)
FEATREF_3D = -2,  (Feature is a 3D feature, no projections)
FEATREF_CURRENT_WORKPLANE = -1,
FEATREF_ZPLUS = 0,
FEATREF_XPLUS = 1,
FEATREF_YPLUS = 2,
FEATREF_ZMINUS = 3,
FEATREF_XMINUS = 4,
FEATREF_YMINUS = 5
```

? Command Overview

? FeatCommand Overview

FeatCommand.ReferenceID

ID of the feature to be used when the "ReferenceType" property is set to FEATREF_FEATURE. This property is used with measured lines or measured circles. **String** Read/Write.

Remarks

This property applies only to measured lines and circles that have the projection reference type set to feature.

? Command Overview

? FeatCommand Overview

FeatCommand.RMeasFeature

ID of the feature to be used for relative measurement. **String** Read/Write.

Remarks

This property applies only to PC-DMIS commands that support relative measurement

? Command Overview

? FeatCommand Overview

FeatCommand.Spacer

Spacer distance (Usually used with sample hits). **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a spacer field.

? Command Overview

? FeatCommand Overview

FeatCommand.StartAngle

Start Angle value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a start angle field.

? Command Overview

? FeatCommand Overview

FeatCommand.StartAngle2

Second Start Angle value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a second start angle field.

? Command Overview

? FeatCommand Overview

FeatCommand.TheoAngle

Theoretical angle value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an angle field.

? Command Overview

? FeatCommand Overview

FeatCommand.TheoDiam

Theoretical diameter value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a diameter field.

? Command Overview

? FeatCommand Overview

FeatCommand.TheoHeight

Theoretical height value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a height field.

? Command Overview

? FeatCommand Overview

FeatCommand.TheoLength

Theoretical length value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a length field.

? Command Overview

? FeatCommand Overview

FeatCommand.TheoMajorAxis

Theoretical major axis length value (ellipse). **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a major axis field.

? Command Overview

? FeatCommand Overview

FeatCommand.TheoMinorAxis

Theoretical minor axis length value (ellipse). **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a minor axis field.

? Command Overview

? FeatCommand Overview

FeatCommand.TheoPinDiam

Theoretical pin diameter value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a pin diameter field.

? Command Overview

? FeatCommand Overview

FeatCommand.TheoSmallLength

Theoretical shorter length value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a small length field.

? Command Overview

? FeatCommand Overview

FeatCommand.Thickness

Sheet metal (material) thickness. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a thickness field.

? Command Overview

? FeatCommand Overview

FeatCommand.Tolerance

Tolerance value for auto high point. **Double** Read/Write.

Remarks

This property applies only to the PC-DMIS auto high point command.

? Command Overview

? FeatCommand Overview

FeatCommand.UsePin

Boolean read/write value indicating whether pin information should be used during measurement.

Remarks

This property applies only to PC-DMIS commands that have a use pin field.

? Command Overview

? FeatCommand Overview

Methods:

FeatCommand.AddInputFeat

Syntax

Return Value=expression.AddInputFeat(*ID*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object that represents a constructed feature.

ID: Required **String** that is the ID of the feature to add to the set of input features.

This function returns TRUE if the feature was successfully added to set of input features of *expression*, FALSE otherwise.

Remarks

This function only tries to add *ID* to *expression* if the two features exist and *ID* precedes *expression* in the command list. If *expression* is not a constructed feature, this function will fail.

? FeatCommand Overview

FeatCommand.GenerateHits

Syntax

Return Value=expression.GenerateHits

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object that represents a measured feature.

This function returns TRUE if the hits were successfully added to *expression*, FALSE otherwise.

Remarks

This function tries to add evenly spaced hits to *expression*. If *expression* is not a measured feature, this function will fail.

? FeatCommand Overview

FeatCommand.GetData

Syntax

```
Return Value=expression.GetData(PointData, DataType, TheoMeas, CoordSystem, AlignID,  Workplane)
```

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

PointData: Required PointData object into which the data is stored.

DataType: Optional **Long** that is one of the following values: `FDATA_CENTROID, FDATA_VECTOR, FDATA_DIAMETER, FDATA_STARTPOINT, FDATA_MIDPOINT, FDATA_ENDPOINT, FDATA_LENGTH, FDATA_MINOR_AXIS, FDATA_ANGLE, FDATA_SURFACE_VECTOR, FDATA_THICKNESS, FDATA_SPACER, FDATA_INDENT, FDATA_AUTO_MOVE_DISTANCE, FDATA_DEPTH, FDATA_ANGLE_VECTOR, FDATA_PUNCH_VECTOR, FDATA_PIN_VECTOR, FDATA_PIN_DIAMETER, FDATA_REPORT_VECTOR, FDATA_REPORT_SURF_VECTOR, FDATA_HEIGHT, FDATA_MEASURE_VECTOR,`

FDATA_UPDATE_VECTOR, FDATA_SNAP_CENTROID, FDATA_ANALOG_DEVIATIONS, FDATA_CORNER_RADIUS, FDATA_AB_ANGLES, FDATA_ORG_HIT_VECTOR, FDATA_ANGLE2, FDATA_WIDTH, FDATA_MAJOR_AXIS, or FDATA_SLOT_VECTOR

If no value is supplied, the default value is FDATA_CENTROID.

TheoMeas: Optional Long that is one of FDATA_THEO, FDATA_MEAS, or FDATA_TARG.

If no value is supplied, the default value is FDATA_MEAS.

CoordSystem: Optional **Long** that denotes the coordinate system in which to report. Values include FDATA_POLAR, FDATA_CAD, FDATA_PARTMM3, FDATA_MACHINE, and FDATA_PART.If no value is supplied, the default value is FDATA_PART.

AlignID: Optional **String** that denotes what alignment to use. You can pass the empty string to denote the current alignment.

If no value is supplied, the default value is an empty string which causes the current alignment to be used.

Workplane: Optional **Long**. Used for the PARTMM3 and POLAR coordinate system to denote the workplane to be used. Possible values include PCD_TOP, PCD_BOTTOM, PCD_LEFT, PCD_RIGHT, PCD_FRONT, PCD_BACK.

If no value is supplied, the default value is PCD_TOP.

This function returns TRUE if the data was successfully retrieved from *expression*, FALSE otherwise.

Remarks

Not every data type can be used with every feature type. Some data types return a single value, some data types return multiple values. Some data types return both depending on the feature. For example, a cone will return two diameters in the first and second data fields of the point object while only returning one diameter for a circle object.Use the FDATA_THEO flag if you want theoretical data, FDATA_MEAS if you want measured data.

? FeatCommand.PutData

? FeatCommand Overview

FeatCommand.GetInputFeat

Syntax

Return Value=expression.GetInputFeat(*Index*)

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

Index: Required **Long** between one and *expression*.NumHits

Return Value: If successful, this function returns the **String** ID of the input feature at the specified index.

Remarks

When successful, this returns the ID of the input feature, otherwise it returns an empty string.

? FeatCommand.NumHits

? FeatCommand Overview

FeatCommand.GetInputOffset

Syntax

Return Value=expression.GetInputOffset(*Index*)

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

Index: Required **Long** between one and *expression*.NumHits

Return Value: If successful, this function returns the **Double** offset value.

Remarks

Use this function with constructed features that have offset values from input features.

? FeatCommand.NumHits

? FeatCommand Overview

? FeatCommand.SetInputOffset

FeatCommand.GetHit

Syntax

```
Return Value=expression.GetHit(Index, DataType, TheoMeas, CoordSystem, AlignID,  Workplane)
```

Return Value: This method returns a Point Data object with the values of the hit.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

Index: The index number of the desired hit object to retrieve.

DataType: Optional **Long** that is one of the following values: `FHITDATA_CENTROID, FHITDATA_VECTOR, FHITDATA_BALLCENTER`

If no value is supplied, the default value is `FHITDATA_CENTROID`.

TheoMeas: Optional Long that is one of `FDATA_THEO, FDATA_MEAS, or FDATA_TARG`.

If no value is supplied, the default value is `FDATA_MEAS`.

CoordSystem: Optional **Long** that denotes the coordinate system in which to report. Values include `FDATA_POLAR, FDATA_CAD, FDATA_PARTMM3, FDATA_MACHINE,` and `FDATA_PART`.

If no value is supplied, the default value is FDATA_PART.

AlignID: Optional **String** that denotes what alignment to use. You can pass the empty string to denote the current alignment.

If no value is supplied, the default value is an empty string which causes the current alignment to be used.

Workplane: Optional **Long**. Used for the PARTMM3 and POLAR coordinate system to denote the workplane to be used. Possible values include `PCD_TOP, PCD_BOTTOM, PCD_LEFT, PCD_RIGHT, PCD_FRONT, PCD_BACK`.

If no value is supplied, the default value is PCD_TOP.

Remarks

Use this function to obtain hit information from individual objects. This command works with objects that the hits are supplied by the user and with objects in which the hits are generated by the object itself.

? FeatCommand.PutData

? FeatCommand Overview

FeatCommand.GetPoint

Syntax

```
Return Value=expression.GetPoint(PointType, TheoMeas, X, Y, Z)
```

Return Value: This method returns a **boolean** value indicating success or failure of the call.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

VectorType: FPOINT_TYPES **enumeration**. Possible values include the following:

`FPOINT_CENTROID`

`FPOINT_STARTPOINT`

`FPOINT_MIDPOINT`

`FPOINT_ENDPOINT`

FPOINT_BALLCENTER

FPOINT_SNAP_CENTROID

TheoMeas: Long that is one of FDATA_THEO, FDATA_MEAS, or FDATA_TARG.

X: Variable of type **double** that will hold the X data for the point.

Y: Variable of type **double** that will hold the Y data for the point.

Z: Variable of type **double** that will hold the Z data for the point.

Remarks

Use this function to retrieve point information of individual objects.

? FeatCommand.PutPoint

? FeatCommand Overview

FeatCommand.GetSurfaceVectors

Syntax

```
Return Value=expression.GetSurfaceVectors(TheoMeas, I1, J1, K1, I2, J2, K2)
```

Return Value: This method returns a **boolean** value indicating success or failure of the call.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

TheoMeas: Long that is one of FDATA_THEO or FDATA_MEAS

I1: Variable of type **double** that will hold the I component of the first vector.

J1: Variable of type **double** that will hold the J component of the first vector.

K1: Variable of type **double** that will hold the K component of the first vector.

I2: Variable of type **double** that will hold the I component of the second vector.

J2: Variable of type **double** that will hold the J component of the second vector.

K2: Variable of type **double** that will hold the K component of the second vector.

Remarks

Use this function to get the surface vectors of an angle hit function.

? FeatCommand.PutSurfaceVectors

? FeatCommand Overview

FeatCommand.GetVector

Syntax

```
Return Value=expression.GetVector(VectorType, TheoMeas, I, J, K)
```

Return Value: This method returns a **boolean** value indicating success or failure of the call.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

VectorType: FVECTOR_TYPES **enumeration**. Possible values include the following:

FVECTOR_VECTOR,

FVECTOR_SURFACE_VECTOR

FVECTOR_ANGLE_VECTOR

FVECTOR_PUNCH_VECTOR

FVECTOR_PIN_VECTOR

FVECTOR_REPORT_VECTOR

```
FVECTOR_REPORT_SURF_VECTOR
FVECTOR_MEASURE_VECTOR
FVECTOR_UPDATE_VECTOR
FVECTOR_ORG_HIT_VECTOR
FVECTOR_CORNER_VECTOR2
FVECTOR_CORNER_VECTOR3
FVECTOR_SLOT_VECTOR
```

TheoMeas: Long that is one of FDATA_THEO, FDATA_MEAS, or FDATA_TARG.

I: Variable of type **double** that will hold the I component of the vector.

J: Variable of type **double** that will hold the J component of the vector.

K: Variable of type **double** that will hold the K component of the vector.

Remarks

Use this function to retrieve vector components of individual objects.

? FeatCommand.PutVector

? FeatCommand Overview

FeatCommand.PutData

Syntax

```
Return Value=expression.PutData(Data, DataType, TheoMeas, CoordSystem, AlignID, Workplane)
```

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

Data: Required PointData object into from which the data is taken to set values in the corresponding object.

DataType: Optional **Long** that is one of the following values:

```
FDATA_CENTROID, FDATA_VECTOR, FDATA_DIAMETER, FDATA_STARTPOINT, FDATA_MIDPOINT, FDATA_ENDPOINT,
    FDATA_LENGTH, FDATA_MINOR_AXIS, FDATA_ANGLE, FDATA_SURFACE_VECTOR, FDATA_THICKNESS, FDATA_SPACER,
    FDATA_INDENT, FDATA_AUTO_MOVE_DISTANCE, FDATA_DEPTH, FDATA_ANGLE_VECTOR, FDATA_PUNCH_VECTOR,
    FDATA_PIN_VECTOR, FDATA_PIN_DIAMETER, FDATA_REPORT_VECTOR, FDATA_REPORT_SURF_VECTOR, FDATA_HEIGHT,
    FDATA_MEASURE_VECTOR, FDATA_UPDATE_VECTOR, FDATA_SNAP_CENTROID, FDATA_ANALOG_DEVIATIONS,
    FDATA_CORNER_RADIUS, FDATA_AB_ANGLES, FDATA_ORG_HIT_VECTOR, FDATA_ANGLE2, FDATA_WIDTH,
    FDATA_MAJOR_AXIS, or FDATA_SLOT_VECTOR
```

If no value is supplied, the default value is FDATA_CENTROID.

TheoMeas: Optional Long that is one of FDATA_THEO, FDATA_MEAS, or FDATA_TARG.

If no value is supplied, the default value is FDATA_MEAS.

CoordSystem: Optional **Long** that denotes the coordinate system in which to report. Values include FDATA_POLAR, FDATA_CAD, FDATA_PARTMM3, FDATA_MACHINE, and FDATA_PART.If no value is supplied, the default value is FDATA_PART.

AlignID: Optional **String** that denotes what alignment to use. You can pass the empty string to denote the current alignment.

If no value is supplied, the default value is an empty string which causes the current alignment to be used.

Workplane: Optional **Long**. Used for the PARTMM3 and POLAR coordinate system to denote the workplane to be used. Possible values include `PCD_TOP, PCD_BOTTOM, PCD_LEFT, PCD_RIGHT, PCD_FRONT, PCD_BACK`.

If no value is supplied, the default value is PCD_TOP.

This function returns TRUE if the data was successfully retrieved from *expression*, FALSE otherwise.

Remarks

Not every data type can be used with every feature type. Some data types take a single value, some data types take multiple values. Some data types take one or more depending on the feature. For example, a cone can take two diameters in the first and second data fields of the point object while the circle object only takes one diamter.

Use the FDATA_THEO flag if you want theoretical data, FDATA_MEAS if you want measured data.

? FeatCommand.GetData

? FeatCommand Overview

FeatCommand.PutPoint

Syntax

```
Return Value=expression.PutPoint(PointType, TheoMeas, X, Y, Z)
```

Return Value: This method returns a **boolean** value indicating success or failure of the call.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

VectorType: FPOINT_TYPES **enumeration**. Possible values include the following:

```
FPOINT_CENTROID
FPOINT_STARTPOINT
FPOINT_MIDPOINT
FPOINT_ENDPOINT
FPOINT_BALLCENTER
FPOINT_SNAP_CENTROID
```

TheoMeas: Long that is one of FDATA_THEO, FDATA_MEAS, or FDATA_TARG.

X: **Double** representing X value of the point.

Y: **Double** representing Y value of the point.

Z: **Double** representing Z value of the point.

Remarks

Use this function to set point information for individual objects.

? FeatCommand.GetPoint

? FeatCommand Overview

FeatCommand.PutSurfaceVectors

Syntax

```
Return Value=expression.PutSurfaceVectors(TheoMeas, I1, J1, K1, I2, J2, K2)
```

Return Value: This method returns a **boolean** value indicating success or failure of the call.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

TheoMeas: Long that is one of FDATA_THEO or FDATA_MEAS

I1: **Double** representing the I component of the first vector.

J1: **Double** representing the J component of the first vector.

*K*1: **Double** representing the K component of the first vector.

I2: **Double** representing the I component of the second vector.

J2: **Double** representing the J component of the second vector.

K2: **Double** representing the K component of the second vector.

Remarks

Use this function to set the surface vectors for an angle hit object.

? FeatCommand.GetSurfaceVectors

? FeatCommand Overview

FeatCommand.PutVector

Syntax

```
Return Value=expression.PutVector(VectorType, TheoMeas, I, J, K)
```

Return Value: This method returns a **boolean** value indicating success or failure of the call.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

VectorType: FVECTOR_TYPES **enumeration**. Possible values include the following:

```
FVECTOR_VECTOR
FVECTOR_SURFACE_VECTOR
FVECTOR_ANGLE_VECTOR
FVECTOR_PUNCH_VECTOR
FVECTOR_PIN_VECTOR
FVECTOR_REPORT_VECTOR
FVECTOR_REPORT_SURF_VECTOR
FVECTOR_MEASURE_VECTOR
FVECTOR_UPDATE_VECTOR
FVECTOR_ORG_HIT_VECTOR
FVECTOR_CORNER_VECTOR2
FVECTOR_CORNER_VECTOR3
FVECTOR_SLOT_VECTOR
```

TheoMeas: Long that is one of FDATA_THEO, FDATA_MEAS, or FDATA_TARG.

I: **Double** indicating the I component of the vector.

J: **Double** indicating the J component of the vector.

K: **Double** indicating the K component of the vector.

Remarks

Use this function to set vector components of individual objects.

? FeatCommand.GetVector

? FeatCommand Overview

FeatCommand.RemoveInputFeat

Syntax

Return Value=expression.RemoveInputFeat(*Index*)

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

Index: Required **Long** between one and *expression*.NumHits

Return Value: This function returns TRUE if *expression* is a constructed feature and *Index* is the index of a input feature, FALSE otherwise.

Remarks

When successful, this function removes the feature at the specified index position.

? FeatCommand.NumHits

? FeatCommand Overview

FeatCommand.SetInputFeat

Syntax

Return Value=expression.SetInputFeat(*ID, Index*)

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

ID: Required **String** that is the ID of a feature.

Index: Required **Long** between one and *expression*.NumHits

Return Value: This function returns TRUE if *expression* is a constructed feature and *ID* is the ID of a valid input feature, and *Index* is the index of a input feature, FALSE otherwise.

Remarks

When successful, this function replaces the input feature at position *Index* in *expression*'s list of input features with *ID*.

? FeatCommand.NumHits

? FeatCommand Overview

FeatCommand.SetInputOffset

Syntax

Return Value=expression.SetInputOffset(*Index, Offset*)

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

Index: Required **Long** between one and *expression*.NumHits

Offset: Required **Double** which specifies the offset value

Return Value: If successful, this function returns the **Boolean** set to true.

Remarks

Use this function with constructed features to set the offset values for input features.

? FeatCommand.NumHits

? FeatCommand Overview

? FeatCommand.GetInputOffset

FeatData Object Overview

The FeatData object is similar to a type define as follows:

Type FeatData

X as Double
Y as Double
Z as Double
I as Double
J as Double
K as Double

DIAM as Double
LENGTH as Double
ANGLE as Double
SmallDiam as Double
StartAngle as Double
EndAngle as Double
StartAngle2 as Double
EndAngle2 as Double
F as Double
TP as Double
P1 as Double
P2 as Double
ID as String

End Type

It is be used to pass feature data in automation functions that accept this type

? FeatData Object Members

FeatData Members

Properties

FeatData.X

Represents the X member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.Y

Represents the Y member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.Z

Represents the Z member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.I

Represents the I member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.J

Represents the J member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.K

Represents the K member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.DIAM

Represents the DIAM member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.LENGTH

Represents the LENGTH member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.ANGLE

Represents the ANGLE member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.SmallDiam

Represents the SmallDiam member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.StartAngle

? FeatData Object Overview

FeatData.EndAngle

Represents the EndAngle member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.StartAngle2

Represents the StartAngle2 member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.EndAngle2

Represents the EndAngle2 member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.F

Represents the F member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.TP

Represents the TP member of this object. Read/write **Double**.

? FeatData Object Overview

FeatData.P1

Represents the P1 member of this object. Read/write **Double**.

Remarks

The P1 member is never set or used by PC-DMIS. It is available for the programmer to use as he wishes.

? FeatData Object Overview

FeatData.P2

Represents the P2 member of this object. Read/write **Double**.

Remarks

The P2 member is never set or used by PC-DMIS. It is available for the programmer to use as he wishes.

? FeatData Object Overview

FeatData.ID

Represents the ID member of this object. Read/write **String**.

Remarks

The ID member is the default property.

The ID member is the default

? FeatData Object Overview

File IO Object Overview

The File IO object is used to access the PC-DMIS File I/O object. Properties provide access to the file mode: open, close, readline, etc.; the expression to write or read, the filename, etc. For additional information, see "File I/O" in, the "Utilities" section of the *PC-DMIS Reference Manual*.

? File IO Members

File IO Members

Properties:

FileIO.BufferSize

LONG value representing the buffer size used with the Read Block File I/O command.

Read/Write **Long**

? File IO Overview

FileIO.Expression

STRING value representing the text to be used in reading from or writing to the opened file.

Read/Write **String**

? File IO Overview

FileIO.FailIfExists

BOOLEAN value indicating whether a file copy operation should fail or not if the destination file already exists.

Read/Write **Boolean**

? File IO Overview

FileIO.FileIOType

Value of ENUM_FILE_IO_TYPES enumeration type which specifies the type of File I/O operation the object will perform. Possible values include the following:

```
PCD_FILE_OPEN = 0
PCD_FILE_CLOSE = 1
PCD_FILE_WRITELINE = 2
PCD_FILE_READLINE = 3
PCD_FILE_WRITECHARACTER = 4
PCD_FILE_READCHARACTER = 5
PCD_FILE_WRITEBLOCK = 6
PCD_FILE_READBLOCK = 7
PCD_FILE_REWIND = 8
PCD_FILE_SAVEPOSITION = 9
PCD_FILE_RECALLPOSITION = 10
PCD_FILE_COPY = 11
PCD_FILE_MOVE = 12
PCD_FILE_DELETE = 13
PCD_FILE_EXISTS = 14
PCD_FILE_DIALOG = 15
```

Read/Write **Enum_File_IO_Types enumeration**

? File IO Overview

FileIO.FileName1

STRING value representing the file name to be used in the File I/O operation. This parameter is used with the File Open, File Copy, File Move, File Delete, and File Exists File I/O types.

Read/Write **String**

? File IO Overview

FileIO.FileName2

STRING value representing the second filename to be used in the File I/O operation. This parameter is used as the destination file in the File Copy and File Move File I/O commands.

Read/Write **String**

? File IO Overview

FileIO.FileOpenType

Value of ENUM_FILE_OPEN_TYPES enumeration type which specifies the file open mode used in opening a file. Possible values include the following:

```
PCD_FILE_WRITE = 1
PCD_FILE_READ = 2
PCD_FILE_APPEND = 3
```

Read/Write **Enum_File_Open_Types enumeration**

? File IO Overview

FileIO.FilePointerID

STRING value representing the file pointer Id to be used in the File I/O operation. The file pointer ID is established and linked to a specific file in the File Open command.

Read/Write **String**

? File IO Overview

FileIO.VariableID

STRING value representing the name of the variable to be used to hold the results of the File I/O operation of the File I/O command.

Read/Write **String**

? File IO Overview

FlowControlCommand Object Overview

Objects of type **FlowControlCommand** are created from more generic **Command** objects to pass information specific to the flow control command back and forth.

? Command.FlowControlCommand

? FlowControlCommand Members

FlowControlCommand Members

Properties:

FlowControlCommand.AngleOffset

Represents the angular offset of a LOOP_START object. Read/write **Double**.

Remarks

This property only affects objects of type LOOP_START. For other objects, setting the property has no effect, and getting it always returns zero.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.GetEndNum

Represents the end value of a LOOP_START object. Read/write **Long**.

Remarks

This property only affects objects of type LOOP_START. For other objects, setting the property has no effect, and getting it always returns zero.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.ErrorMode

Represents the error mode of a ONERROR object. Read/write **Long**.

Remarks

This property only affects objects of type ONERROR. For other objects, setting the property has no effect, and getting it always returns zero.

The valid values for ErrorMode: 0 for off, 1 for jump to label, and 2 for set a variable.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.ErrorType

Represents the error mode of a ONERROR object. Read/write **Long**.

Remarks

This property only affects objects of type ONERROR. For other objects, setting the property has no effect, and getting it always returns zero.

The valid values for ErrorMode: 0 for off, 1 for jump to label, and 2 for set a variable.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.Expression

Represents the test expression of an IF_COMMAND object. Read/write **String**.

Remarks

This property only affects objects of type IF_COMMAND. For other objects, setting the property has no effect, and getting it always returns the empty string.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.FileName

Represents the file name of an external subroutine in a CALL_SUBROUTINE object. Read/write **String**.

Remarks

This property only affects objects of type CALL_SUBROUTINE. For other objects, setting the property has no effect, and getting it always returns the empty string.

This property only returns the name of the file, not its full path. The path is determined by the settings in PCDMIS's Search Directory dialog.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.ID

Represents the id of a CALL_SUBROUTINE object. Read/write **String**.

Remarks

This property only affects objects of type CALL_SUBROUTINE. For other objects, setting the property has no effect, and getting it always returns the empty string.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.Label

Represents the label associated with an object. Read/write **String**.

Remarks

This property only affects objects of type GOTO, IF_COMMAND, ONERROR, and LABEL. For other objects, setting the property has no effect, and getting it always returns the empty string.

For objects of type LABEL, this property is the id of the object. For the other valid types, this property is the label to which execution is redirected when the appropriate conditions are met. For GOTO, redirection always occurs. For IF_COMMAND, the redirection occurs only when the expression is TRUE. For ONERROR, the redirection happens when the error condition is met.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.NumArguments

Returns the number of arguments in a START_SUBROUTINE or CALL_SUBROUTINE object. Read-only **Long**.

Remarks

This property only affects objects of type START_SUBROUTINE and CALL_SUBROUTINE. For other objects it always returns zero.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.SkipCount

Returns the number of skipped numbers in a LOOP_START object. Read-only **Long**.

Remarks

This property only affects objects of type LOOP_START. For other objects it always returns zero.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.StartNum

Represents the start number of a LOOP_START object. Read/write Long.

Remarks

This property only affects objects of type LOOP_START. For other objects, setting the property has no effect, and getting it always returns zero.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.SubName

Represents the subroutine name of a START_SUBROUTINE and CALL_SUBROUTINE object. Read/write **String**.

Remarks

This property only affects objects of type START_SUBROUTINE and CALL_SUBROUTINE. For other objects, setting the property has no effect, and getting it always returns the empty string.

For the START_SUBROUTINE object, it is the name of the subroutine. For the CALL_SUBROUTINE, it is the name of the called subroutine.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.XAxisOffset

Represents the X-axis offset of a LOOP_START object. Read/write Long.

Remarks

This property only affects objects of type LOOP_START. For other objects, setting the property has no effect, and getting it always returns zero.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.YAxisOffset

Represents the Y-axis offset of a LOOP_START object. Read/write Long.

Remarks

This property only affects objects of type LOOP_START. For other objects, setting the property has no effect, and getting it always returns zero.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.ZAxisOffset

Represents the Z-axis offset of a LOOP_START object. Read/write Long.

Remarks

This property only affects objects of type LOOP_START. For other objects, setting the property has no effect, and getting it always returns zero.

? Command.Type Property

? FlowControlCommand Overview

Methods:

FlowControlCommand.AddArgument

Syntax

Return Value=expression.AddArgument(*Position*, *Name*, *Description*, *DefaultValue*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to **FlowControlCommand** object.

Position: Required **Long** that indicates the index of the argument to add in the list of arguments.

Name: Required **String** that indicates the name of the argument to be added.

Description: Required String that is the description of the argument to be added.

DefaultValue: Required String that indicates the default value of the argument to be added.

The AddArgument adds or replaces an argument in objects of type CALL_SUBROUTINE and START_SUBROUTINE. When used with objects of other types, it has no effect.

This function returns TRUE if the argument was added successfully, FALSE otherwise.

When used with objects of type CALL_SUBROUTINE, the *Name* and *Description* fields are ignored, and the *DefaultValue* field is used to set the value.

If *Position* is equal to 1 + *expression*.NumArguments, an argument is added to the tail of the list of arguments . If *Position* is between 1 and *expression*.NumArguments, the current argument is replaced. To completely remove an argument, use DimensionCommand.RemoveArgument.

? Command.Type Property

? FlowControlCommand Overview

? FlowControlCommand.NumArguments (Property)

? FlowControlCommand.RemoveArgument (Method)

FlowControlCommand.AddSkipNum

Syntax

Return Value=expression.AddSkipNum(*Number*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to **FlowControlCommand** object.

Number: Required **Long** that indicates the number to skip.

The AddSkipNum function adds a number to be skipped to an object of type LOOP_START. For objects of other types, it does nothing.

This function returns TRUE if *Number* was successfully added to the LOOP_START object's skip list, FALSE otherwise.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.GetArgumentDescription

Syntax

Return Value=expression.GetArgumentDescription(*Position*)

Return Value: This function returns a string value.

expression: Required expression that evaluates to **FlowControlCommand** object.

Position: Required **Long** that indicates the number of the argument from which to obtain the description..

The GetArgumentDescription function returns the description of an argument to an object of type START_SUBROUTINE. For objects of other types, it returns the empty string.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.GetArgumentExpression

Syntax

Return Value=expr.GetArgumentExpression(*Expression*)

Return Value: This function returns a string value.

expr: Required expression that evaluates to **FlowControlCommand** object.

Expression: Required **Long** that indicates the number of the argument from which to obtain the value.

The GetArgumentDescription function returns the value or default value of an argument to an object of type CALL_SUBROUTINE or START_SUBROUTINE, respectively. For objects of other types, it returns the empty string.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.GetArgumentName

Syntax

Return Value=expression.GetArgumentName(*Position*)

Return Value: This function returns a string value.

expression: Required expression that evaluates to **FlowControlCommand** object.

Number: Required **Long** that indicates the number of the argument from which to obtain the name..

The GetArgumentName function returns the Name of an argument to an object of type START_SUBROUTINE. For objects of other types, it returns the empty string.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.GetSkipNum

Syntax

Return Value=expression.GetSkipNum(*Index*)

Return Value: This function returns an integer. The integer is the nth skip number where n is indicated by the value of index.

expression: Required expression that evaluates to **FlowControlCommand** object.

Index: Required **Long** that indicates which skip number of the set of skip numbers to retrieve.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.IsExpressionValid

Syntax

Return Value=expr.IsExpressionValid(*Expression*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expr: Required expression that evaluates to **FlowControlCommand** object.

Expression: Required **String** that is the expression to evaluate for validity.

This function returns TRUE if the expression is valid, and FALSE otherwise.

? FlowControlCommand Overview

FlowControlCommand.IsValidLeftHandValue

Syntax

Return Value=expr.IsValidLeftHandValue(*Expression*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expr: Required expression that evaluates to **FlowControlCommand** object.

Expression: Required **String** that is the expression to evaluate for validity.

This function returns TRUE if the expression can be used as a valid left hand value (i.e. can be used on the left-hand side of an assigment statement), and FALSE otherwise.

? FlowControlCommand Overview

FlowControlCommand.IsValidSubroutineArgumentName

Syntax

Return Value=e*xpr*.IsValidSuborutineArgumentName(*Expression*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expr: Required expression that evaluates to **FlowControlCommand** object.

Expression: Required **String** that is the argument name to evaluate for validity.

This function returns TRUE if the expression can be used as a valid suroutine argument name, and FALSE otherwise.

? FlowControlCommand Overview

FlowControlCommand.RemoveArgument

Syntax

Return Value=expression.RemoveArgument(*Position*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to **FlowControlCommand** object.

Position: Required Long that indicates which argument to remove.

This function removes an argument from an object of type CALL_SUBROUTINE or START_SUBROUTINE. It returns TRUE if an argument is removed successfully, FALSE otherwise.

This function has an effect only on objects of type CALL_SUBROUTINE and START_SUBROUTINE. It has no effect on objects of other types. If used on other types it returns FALSE even if nothing is being done.

The *Position* argument should be between one and *expression*.NumArguments.

? Command.Type Property

? FlowControlCommand Overview

? FlowControlCommand.NumArguments property

FlowControlCommand.RemoveSkipNum

Syntax

expression.RemoveSkipNum(*Index*)

expression: Required expression that evaluates to **FlowControlCommand** object.

Index: Required Long that indicates which argument to remove.

This function removes one of the skip numbers for the Loop Start object from the list of skip numbers. The number removed is determined by the index parameter.

The *Index* argument should be between one and *expression*.SkipCount.

? FlowControlCommand.SkipCount property

? FlowControlCommand Overview

FlowControlCommand.SetArgumentDescription

Syntax

Return Value=expression.SetArgumentDescription(*Position, Description*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to **FlowControlCommand** object.

Number: Required **Long** that indicates the number of the argument description to set.

Description: Required **String** that is the text of the description to set.

This function sets the description of an argument of an object of type START_SUBROUTINE. It does nothing and returns FALSE if the object is not of this type.

The function returns TRUE if the description was set successfully, FALSE otherwise.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.SetArgumentExpression

Syntax

Return Value=expr.GetArgumentExpression(*Position, Expression*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expr: Required expression that evaluates to **FlowControlCommand** object.

Position: Required **Long** that indicates the number of the argument value to set.

Expression: Required **String** that indicates the argument value to set.

This function sets the value or default value of an argument of an object of type CALL_SUBROUTINE or START_SUBROUTINE, respectively. It does nothing and returns FALSE if the object is not one of these types.

The function returns TRUE if the value was set successfully, FALSE otherwise.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.SetArgumentName

Syntax

Return Value=expr.GetArgumentExpression(*Position, Expression*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expr: Required expression that evaluates to **FlowControlCommand** object.

Position: Required **Long** that indicates the number of the argument value to set.

Name: Required **String** that indicates the argument name to set.

This function sets the name of an argument of an object of type START_SUBROUTINE. It does nothing and returns FALSE if the object is not of this type.

The function returns TRUE if the value was set successfully, FALSE otherwise.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.SetLeftSideOfAssignment

Syntax

expr.SetLeftSideOfAssignment*Expression*

expr: Required expression that evaluates to **FlowControlCommand** object.

Expression: Required **String** that indicates the expression to be used for the left side of the assigment.

The function sets the left-hand side of the Assign statement to the expression passed in. Use the function IsValidLeftHandValue to determine validity of expression for a left-hand side before using this function.

? Command.Type Property

? FlowControlCommand Overview

FlowControlCommand.SetRightSideOfAssignment

Syntax

expr.SetRightSideOfAssignment*Expression*

expr: Required expression that evaluates to **FlowControlCommand** object.

Expression: Required **String** that indicates the expression to be used for the right side of the assigment.

The function sets the right-hand side of the Assign statement to the expression passed in. Use the function IsExpressionValid to determine validity of expression before using this function.

? Command.Type Property

? FlowControlCommand Overview

Leitz Motion Object Overview

The leitz motion automation command object changes motion settings for the PC-DMIS leitz motion command object. This section does not define the meaning of the different properties. More information on the properties can be found under "Optional Probe" in "System Options" of the *PC-DMIS Reference Manual*.

? Leitz Motion Members

Leitz Motion Members

Properties:

LeitzMot.LowForce

Double value used to set or get the low force setting for the probe.

Read/Write **Double**

? Leitz Motion Object Overview

LeitzMot.MaxForce

Double value used to set or get the max force setting for the probe.

Read/Write **Double**

? Leitz Motion Command Object Overview

LeitzMot.PositionalAccuracy

Double value used to set or get the positional accuracy setting.

Read/Write **Double**

? Leitz Motion Command Object Overview

LeitzMot.ProbeAccuracy

Double value used to set or get the probe accuracy setting.

Read/Write **Double**

? Leitz Motion Command Object Overview

LeitzMot.ReturnData

Double value used to set or get the return data setting.

Read/Write **Double**

? Leitz Motion Command Object Overview

LeitzMot.ReturnSpeed

Double value used to set or get the return speed.

Read/Write **Double**

? Leitz Motion Command Object Overview

LeitzMot.ScanPointDensity

Double value used to set or get the scan point density.

Read/Write **Double**

? Leitz Motion Command Object Overview

LeitzMot.TriggerForce

Double value used to set or get the trigger force setting for the probe.

Read/Write **Double**

? Leitz Motion Command Object Overview

LeitzMot.UpperForce

Double value used to set or get the upper force setting for the probe.

Read/Write **Double**

? Leitz Motion Command Object Overview

Load Machine Object Overview

The Load Machine object gives access to the machine name property of the PC-DMIS Load Machine command.

? Load Machine Members

Load Machine Members

Properties:

LoadProbes.MachineName

STRING value representing the name of the machine to be loaded.

Read/Write **String**

? Load Machine Object Overview

Load Probes Object Overview

The Load Probes object gives access to the filename property of the PC-DMIS Load Probes command.

? Load Probes Members

Load Probes Members

Properties:

LoadProbes.Filename

STRING value representing the name of the probes file to be loaded.

Read/Write **String**

? Load Probes Object Overview

Machine Object Overview

The **Machine** object represent a CMM, or a virtual off-line “machine”. The **Machine** objects are contained in the **Machines** collection.

The **Machine** object is primarily an event source.

Events differ from methods and properties in that PC-DMIS is the source of the action, instead of the destination. To take advantage of events, the automation controller application must support events. Visual Basic supports events. Handling events involves declaring an object of type Machine and then adding handling functions for the different events.

? Machine Members

Machine Object Members

Properties:

Machine.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? Machines Overview

? Machine Overview

Machine.Name

Returns the name of the Machine object. Read-only **String**.

? Machines Overview

? Machine Overview

Machine.Parent

Returns the read-only **Machines** collection object to which the machine belongs.

? Machines Overview

? Machine Overview

Events:

- `LearnHit (Double X, Double Y, Double Z, Double I, Double J, Double K)`

  `This function will be called in your application when a hit is taken in PC-DMIS in learn mode. The values of X, Y, Z and I, J, K are the location of the hit and the vector of the hit in machine coordinates.`

- `ExecuteHit (Double X, Double Y, Double Z, Double I, Double J, Double K)`

  `This function will be called in your application when a hit is taken in PC-DMIS in execute mode. The values of X, Y, Z and I, J, K are the location of the hit and the vector of the hit in machine coordinates.`

- `ErrorMsg(String ErrorText, Long ErrorType)`

  `This function is called when an error occurs on the CMM. The ErrorText variabel contains the error message, and the ErrorType variable contains the type of error. (missed hit, unexpected hit)`

- `Command(Long code)`

  `This function is called when a command button is pressed on the CMM controller. The code can be used to determine which button was pressed.`

Machines Object Overview

The Machines object is the collection of all Machine objects currently available in PC-DMIS. Each **Machine** object is bound to exactly one **PartProgram** object, and *vice versa*. Use **Machines**(*index*) where *index* is the index number or on-line machine's name to return a single **Machine** object.

Remarks

There may be multiple machines named "OFFLINE", one for each open off-line part program. To distinguish between them, use the index number, or use the machine's Parent member.

? Machines Object Members

Machines Object Members

Properties:

Machines.Application

Represents the read-only PC-DMIS application. The Application object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? Machines Overview

Machines.Count

Represents the number of Machine objects currently active in PC-DMIS. Read-only **Integer**.

? Machines Overview

Machines.Parent

Represents the read-only PC-DMIS application. The Application object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? Machines Overview

Methods:

Machines.Item

Syntax 1

Return Value=expression.Item(*NameOrNum*)

Syntax 2

expression(*NameOrNum*)

Return Value=The Item function returns a **Machine** object.

expression: Required expression that evaluates to a **Machines** object identified by the *NameOrNum* parameter.

NameOrNum: Required **Variant** that indicates which **Machine** object to return. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number of the **Machine** object in the **Machines** collection. If it is a **String**, it is the ID of the **Machine** object.

Remarks

There may be several machines named “OFFLINE”. To avoid possible confusion with off-line machines, use the index number with these machines.

Since the Item method is the default, the function name can be omitted as in Syntax 2.

? Machine Overview

? Machines Overview

ModalCommand Object Overview

Objects of type **AlignCommand** are created from more generic **Command** objects to pass information specific to the modal command back and forth.

? Command.ModalCommand

? ModalCommand Members

ModalCommand Members

Properties:

ModalCommand.ClearPlane

Represents the clearance plane of a CLEARANCE_PLANES type object. Read/Write **Long**.

Remarks

This property is only useful for objects of type CLEARANCE_PLANES. For objects of other types, setting this property does nothing and getting it always returns PCD_ZPLUS.

Allowable values for this property are PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS, PCD_ZPLUS, and PCD_ZMINUS.

? Command.Type Property

? ModalCommand Object Overview

ModalCommand.Digits

Represents the number of digits of a DISPLAYPRECISION type object. Read/write **Long**.

Remarks

This property is only useful for objects of type DISPLAYPRECISION. For objects of other types, setting this property does nothing and getting it always returns zero.

? Command.Type Property

? ModalCommand Object Overview

ModalCommand.Distance

Represents the distance to move for this object. Read/write **Double**.

Remarks

This property is only useful for objects of type PREHIT, CLAMP, RETRACT, CHECK, and CLEARANCE_PLANES. For objects of other types, setting this property does nothing and getting it always returns zero.

For objects of type PREHIT, CLAMP, RETRACT, and CHECK, the *Distance* property is the distance to move during that operation. For CLEARANCE_PLANES objects, it is the distance from the axes plane to move. For example, if the clearance plane is LEFT, and the *Distance* is 2.0, the clearance plane will move to the X=2.0 plane.

? Command.Type Property

? ModalCommand.Distance2

? ModalCommand.PassPlane

? ModalCommand Object Overview

ModalCommand.Distance2

Represents the pass-through distance to move for this CLEARANCE_PLANES object. Read/write **Double**.

Remarks

This property is only useful for objects of type CLEARANCE_PLANES. For objects of other types, setting this property does nothing and getting it always returns zero.

? Command.Type Property

? ModalCommand.Distance

? ModalCommand.PassPlane

? ModalCommand Object Overview

ModalCommand.Mode

Represents the mode of this object. Read/write **Long**.

Remarks

This property is only useful for objects of type MAN_DCC_MODE and RMEAS_MODE. For objects of other types, setting this property does nothing and getting it always returns zero.

For objects of type MAN_DCC_MODE, the mode can take values 0 for DCC mode and 1 for manual mode. For objects of type RMEAS_MODE, the mode can take values 0 for NORMAL, and 1 for ABSOLUTE.

? Command.Type Property

? ModalCommand Object Overview

ModalCommand.Name

Returns the name of this GET_PROBE_DATA object. Read-only **String**.

Remarks

This property is only useful for objects of type GET_PROBE_DATA (LoadProbe). For objects of other types, it always returns the empty string.

? Command.Type Property

? ModalCommand Object Overview

ModalCommand.On

Represents the on/off state of this object. Read/write **Boolean**.

Remarks

This property is only useful for objects of types PROBE_COMPENSATION, POLARVECTORCOMP, GAP_ONLY, RETROLINEAR_ONLY, FLY_MODE, and COLUMN132. For objects of other types, setting this property does nothing and getting it always returns FALSE.

? Command.Type Property

? ModalCommand Object Overview

ModalCommand.Parent

Returns the parent **Command** object. Read-only.

Remarks

The parent of a **ModalCommand** object is the same underlying PC-DMIS object as the **ModalCommand** object itself. Getting the parent allows you to access the generic **Command** properties and methods of a given object.

? Command.Type Property

? ModalCommand Object Overview

ModalCommand.PassPlane

Represents the pass-through plane to move for this CLEARANCE_PLANES object. Read/write **Long**.

Remarks

This property is only useful for objects of type CLEARANCE_PLANES. For objects of other types, setting this property does nothing and getting it always returns PCD_ZPLUS.

Allowable values for this property are PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS, PCD_ZPLUS, and PCD_ZMINUS.

? Command.Type Property

? ModalCommand.Distance

? ModalCommand.Distance2

? ModalCommand Object Overview

ModalCommand.Speed

Represents the speed for this object. Read/write **Double**.

Remarks

This property is only useful for objects of type MOVE_SPEED, TOUCH_SPEED, and SCAN_SPEED. For objects of other types, setting this property does nothing and getting it always returns zero.

? Command.Type Property

? ModalCommand Object Overview

ModalCommand.WorkPlane

Represents the workplane to move for this SET_WORKPLANE object. Read/write **Long**.

Remarks

This property is only useful for objects of type SET_WORKPLANE. For objects of other types, setting this property does nothing and getting it always returns PCD_ZPLUS.

Allowable values for this property are PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS, PCD_ZPLUS, and PCD_ZMINUS.

? Command.Type Property

? ModalCommand Object Overview

MoveCommand Object Overview

Objects of type **MoveCommand** are created from more generic **Command** objects to pass information specific to the move command back and forth

? Command.MoveCommand

? MoveCommand Members

MoveCommand Members

Properties:

MoveCommand.Angle

Represents the rotation angle of this MOVE_ROTAB object. Read/Write **Double**.

Remarks

This property is only useful for objects of type MOVE_ROTAB. For objects of other types, setting this property does nothing and getting it always returns zero.

? Command.Type Property

? MoveCommand Overview

MoveCommand.Direction

Represents the rotation direction of this MOVE_ROTAB object. Read/Write **Double**.

Remarks

This property is only useful for objects of type MOVE_ROTAB. For objects of other types, setting this property does nothing and getting it always returns zero.

For objects of type MOVE_ROTAB, the allowable values of this property are PCD_CLOCKWISE, PCD_COUNTERCLOCKWISE, and PCD_SHORTEST.

? Command.Type Property

? MoveCommand Overview

MoveCommand.NewTip

Represents the new tip position of this MOVE_PH9_OFFSET object. Read/Write **String**.

Remarks

This property is only useful for objects of type MOVE_PH9_OFFSET. For objects of other types, setting this property does nothing and getting it always returns the empty string.

For objects of type MOVE_PH9_OFFSET, this property should have the value of the *ID* of any tip in this part program.

? Tip.ID Property

? Command.Type Property

? MoveCommand Overview

MoveCommand.OldTip

Represents the new tip position of this MOVE_PH9_OFFSET object. Read/Write **String**.

Remarks

This property is only useful for objects of type MOVE_PH9_OFFSET. For objects of other types, setting this property does nothing and getting it always returns the empty string.

For objects of type MOVE_PH9_OFFSET, this property should have the value of the *ID* of any tip in this part program.

? Tip.ID Property

? Command.Type Property

? MoveCommand Overview

MoveCommand.Parent

Returns the parent **Command** object. Read-only.

Remarks

The parent of a **MoveCommand** object is the same underlying PC-DMIS object as the **MoveCommand** object itself. Getting the parent allows you to access the generic **Command** properties and methods of a given object.

? Command Overview

? MoveCommand Overview

MoveCommand.XYZ

A PointData object that represents the location to which to move, or in the case of MOVE_INCREMENT, the location offset. Read/Write.

Remarks

This property is only useful for objects of type MOVE_POINT, MOVE_INCREMENT, and MOVE_CIRCULAR. For objects of other types, setting this property does nothing and getting it always returns **Nothing**.

? Command.Type Property

? MoveCommand Overview

Opt Motion Object Overview

The opt motion automation command object is used to change motion settings for the PC-DMIS probe motion command object. This section does not define the meaning of the different properties. Additional information on the properties can be found in "System Options" of the *PC-DMIS Reference Manual*, under the title "Optional Motion".

? Opt Motion Members

Opt Motion Members

Properties:

OptMotion.MaxTAcceleration

Double value used to set or get the maximum acceleration in T setting.

Read/Write **Double**

? Opt Motion Object Overview

OptMotion.MaxTSpeed

Double value used to set or get the maximum speed in T setting.

Read/Write **Double**

? Opt Motion Command Object Overview

OptMotion.MaxXAcceleration

Double value used to set or get the maximum acceleration in X setting.

Read/Write **Double**

? Opt Motion Command Object Overview

OptMotion.MaxYAcceleration

Double value used to set or get the maximum acceleration in Y setting.

Read/Write **Double**

? Opt Motion Command Object Overview

OptMotion.MaxZAcceleration

Double value used to set or get the maximum acceleration in Z setting.

Read/Write **Double**

? Opt Motion Command Object Overview

OptMotion.MovePositionalAccuracy

Double value used to set or get the move positional accuracy setting.

Read/Write **Double**

? Opt Motion Command Object Overview

PartProgram Object Overview

The **PartProgram** object represents a part program currently available in PC-DMIS. The **PartProgram** object is the main object used to manipulate part programs.

? PartProgram Members

PartProgram Members

Properties:

PartProgram.ActiveMachine

Returns the **Machine** object associated with this part program. Read-only.

? Machine Overview

? PartProgram Overview

PartProgram.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? PartProgram Overview

PartProgram.Commands

Returns the **Commands** collection object of this part program. Read-only.

? Command Overview

? PartProgram Overview

PartProgram.EditWindow

Returns the **Editwindow** object associated with this part program. Read-only.

? EditWindow Overview

? PartProgram Overview

PartProgram.FullName

Returns the part program's full file path and name. Read-only **String**. If the file name of the part program is C:\PCDMISW\PARTS\1.PRG, the FullName returns "C:\PCDMISW\PARTS\1.PRG".

? PartProgram Overview

PartProgram.Name

Returns the part program's file name. Read/Write **String**. If the file name of the part program is C:\PCDMISW\PARTS\1.PRG, the FullName returns "1.PRG".

? PartProgram Overview

PartProgram.OldBasic

Returns this part program's **OldBasic** object. Read-only.

The OldBasic object contains all of the methods from the old basic command set used in previous versions of PC-DMIS.

? OldBasic PC-DMIS Functions

? PartProgram Overview

PartProgram.Parent

Returns the **PartPrograms** collection object to which this part program belongs. Read-only.

? PartPrograms Overview

? PartProgram Overview

PartProgram.PartName

Represents the part name of the part program. Read/Write **String**.

Remarks

The part name is not the same as the file name. You can view and set the part name in the Properties of the file containing the part program, as well as at the top of the edit window within PC-DMIS.

? PartProgram Overview

PartProgram.Path

Returns the part program's file path. Read/Write **String**. If the file name of the part program is C:\PCDMISW\PARTS\1.PRG, the FullName returns "C:\PCDMISW\PARTS\".

? PartProgram Overview

PartProgram.Probes

The **Probes** property returns this part program's **Probes** collection object. Read-only.

? Probes Overview

? PartProgram Overview

PartProgram.RevisionNumber

Represents the part program's revision number. Read/Write **String**.

Remarks

You can view and set the revision number in the Properties of the file containing the part program, as well as at the top of the edit window within PC-DMIS.

? PartProgram Overview

PartProgram.SerialNumber

Represents the part program's serial number. Read/Write **String**.

Remarks

You can view and set the serial number in the Properties of the file containing the part program, as well as at the top of the edit window within PC-DMIS.

? PartProgram Overview

PartProgram.Tools

The Tools property returns this part program's **Tools** collection object. Read-only.

? Tools Overview

? PartProgram Overview

PartProgram.Visible

Represents the part program's visibility status. Read/Write **Boolean**.

? PartProgram Overview

Methods:

PartProgram.Close

Syntax

expression.Close

expression: Required expression that evaluates to a **PartProgram** object.

This subroutine saves, closes, and deactivates the part program.

? PartProgram Overview

PartProgram.Export

Syntax

Return Value=expression.Export(*FileName*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a **PartProgram** object.

FileName: Required **String** that denotes the file name to which to export.

Remarks

This function exports CAD or part data from the part program to the indicated file. The export format is determined by the file name extension of *FileName*.

? PartProgram Overview

PartProgram.Import

Syntax

Return Value=expression.Import(*FileName*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a **PartProgram** object.

FileName: Required **String** that denotes the file name from which to import.

Remarks

This function imports CAD or part data from the indicated file to the part program. The file format is determined by the file name extension of *FileName*.

? PartProgram Overview

PartProgram.MessageBox

Syntax

Return Value=expression.MessageBox(*Message,Title,Type*)

Return Value: Integer value of the button chosen by the user.

expression: Required expression that evaluates to a **PartProgram** object.

Message: Required **String** that is the message of the message box

Title: Optional **String** that is the title of the message box. If ommitted, the title will be the name and version of PC-DMIS.

Type: Optional **Long** used to indicate the button types to be used in the message box. Examples include, “OK”, “Cancel”, “Retry”, “Yes”, “No” etc. If ommitted, the default is “OK”.

Remarks

This function uses the PC-DMIS message box function. It includes all functionality including cancelling of execution tied to the Cancel button.

? PartProgram Overview

PartProgram.Quit

Syntax

Return Value=expression.Quit

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails. TRUE if the part was quit successfully, FALSE otherwise.

expression: Required expression that evaluates to a **PartProgram** object.

This subroutine closes, and deactivates the part program without saving

Return Value

? PartProgram Overview

PartProgram.Save

Syntax

Return Value=expression.Save

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails. TRUE if the part was saved successfully, FALSE otherwise.

expression: Required expression that evaluates to a **PartProgram** object.

This subroutine saves the part program. If the part program has not been saved before, it opens a *Save As Dialog box* which requires that you name the file.

? PartProgram Overview

PartProgram.SaveAs

Syntax

Return Value=expression.SaveAs(*name*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails. TRUE if the part was saved successfully, FALSE otherwise.

expression: Required expression that evaluates to a **PartProgram** object.

name: Optional expression that evaluates to a **String**. The file name to which to save.

This subroutine saves the part program. If the *name* parameter is missing or empty, PC-DMIS asks for a file name using a Save As dialog.

? PartProgram Overview

PartPrograms Object Overview

The **PartPrograms** object contains all the open part programs in PC-DMIS.

Using the PartPrograms Collection

Use Add to create a fresh new part program and add it to the **PartPrograms** collection.

Use PartPrograms(*index*) where *index* is the part name or index number to access an individual part program.

? PartPrograms Members

PartPrograms Object Members

Properties:

PartPrograms.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the **ActivePartProgram** property returns a **PartProgram** object.

? Application Overview

? PartProgram Members

? PartPrograms Overview

PartPrograms.Count

Returns the number of part programs active in PC-DMIS. Read-only **Long**.

? PartPrograms Overview

? PartProgram Members

PartPrograms.Parent

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the Active**PartProgram** property returns a **PartProgram** object.

? Application Overview

? PartProgram Members

? PartPrograms Overview

Methods:

PartPrograms.Add

Syntax

Return Value=expression.Add(*FileName*, *Units*)

Return Value: This function returns the added **PartProgram** object

expression: Required. An expression that returns a **PartPrograms** object.

FileName: Required **String**. The file name in which to store the new **PartProgram**.

Units: Required **Long**. Set units to 1 for inches, anythings else for millimeters.

Remarks

The Add function creates a new part program and activates it in PC-DMIS. If a part program named *FileName* exists, it is loaded and the *Units* parameter is ignored.

? PartProgram Overview

? PartProgram Members

? PartPrograms Overview

PartPrograms.CloseAll

Syntax

expression.CloseAll

expression: Required. An expression that returns a **PartPrograms** object.

Remarks

Closes and deactivates all active part programs in PC-DMIS.

? PartProgram Overview

? PartProgram Members

? PartPrograms Overview

PartPrograms.Item

Syntax 1

Return Value=expression.Item(*NameOrNum*)

Syntax 2

expression(*NameOrNum*)

Return Value=The Item function returns a PartProgram object.

expression: Required expression that evaluates to a **PartPrograms** object.

NameOrNum: Required **Variant** that indicates which **PartProgram** object to return. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number of the **PartProgram** object in the **PartPrograms** collection. If it is a **String**, it is the ID of the **PartProgram** object.

Remarks

Since the Item method is the default, the function name can be omitted as in Syntax 2.

Return Value

The **PartProgram** Object identified by the *NameOrNum* parameter.

? PartProgram Overview

? PartProgram Members

? PartPrograms Overview

PartPrograms.Open

Syntax

*Return Value=expression.*Open(*FileName*)

Return Value: This function returns the opened **PartProgram** object. If the file does not exist, the function returns **Nothing.**

expression: Required. An expression that returns a **PartPrograms** object.

FileName: Required **String**. The file name of the **PartProgram** to open.

Remarks

The Open Function activates the part program stored in the file *FileName*. If the file does not exist, nothing happens.

? PartProgram Overview

? PartProgram Members

? PartPrograms Overview

PartPrograms.Remove

Syntax

Return Value=expression.Remove(*NameOrNum*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails. If the function was able to close a part program, it returns TRUE, otherwise FALSE.

expression: Required expression that evaluates to a **PartPrograms** object.

NameOrNum: Required **Variant** that indicates which **PartProgram** object to return. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number of the **PartProgram** object in the **PartPrograms** collection. If it is a **String**, it is the ID of the **PartProgram** object.

Remarks

The Remove function saves, closes, and deactivates the indicated part program. That part program is no longer active in PC-DMIS.

? PartProgram Overview

? PartPrograms Overview

? PartProgram Members

PointData Object Overview

The PointData object is similar to a type define as follows

Type PointData

X as Double
    Y as Double
    Z as Double

End Type

It is be used to pass points and vectors in automation functions that accept this type

? PointData Object Members

PointData Members

Properties

PointData.X

Represents the X member of this object. Read/write **Double**.

? PointData Object Overview

PointData.Y

Represents the Y member of this object. Read/write **Double**.

? PointData Object Overview

PointData.Z

Represents the Z member of this object. Read/write **Double**.

? PointData Object Overview

PointData.I

Represents the X member of this object. Read/write **Double**.

Remarks

This property is exactly the same as the X property, but was included for semantic reasons when working with vectors.

? PointData Object Overview

PointData.J

Represents the X member of this object. Read/write **Double**.

Remarks

This property is exactly the same as the Y property, but was included for semantic reasons when working with vectors.

? PointData Object Overview

PointData.K

Represents the Z member of this object. Read/write **Double**.

Remarks

This property is exactly the same as the Z property, but was included for semantic reasons when working with vectors.

? PointData Object Overview

Probe Object Overview

The Probe object provides information about a given probe description file. It also allows you to manipulate the Probe dialog in PC-DMIS.

? Probe Object Members

Probe Members

Properties:

Probe.ActiveComponent

Represents the highlighted probe component in PC-DMIS's Probe dialog. Read/write **Long**.

Example:
The following VB code illustrates how to create a probe containing a PH9, a TP2, and a 5 mm tip, from scratch in the active part program

```
set app = GetObject("Pcdlrn.Application")
set part = app.GetActiveProgram
set probe = part.Probes.Add("NewProbe")
```

```
probe.ActiveComponent=0
for I = 0 to probe.ConnectionCount – 1
  if (probe.ConnectionDescription(I) = “PROBEPH9”)
    probe.ActiveConnection = I
  end if
next I
probe.ActiveComponent = ComponentCount – 1
for I = 0 to probe.ConnectionCount – 1
  if (probe.ConnectionDescription(I) = “PROBETP2”)
    probe.ActiveConnection = I
  end if
next I
probe.ActiveComponent = ComponentCount – 1
for I = 0 to probe.ConnectionCount – 1
  if (probe.ConnectionDescription(I) = “TIP5BY50MM”)
    probe.ActiveConnection = I
  end if
next I
```

? Probe Overview

Probe.ActiveConnection

Represents the highlighted probe connection in PC-DMIS’s Probe dialog’s connection drop-down list. Read/write **Long**.

? Probe Overview

Probe.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? Probe Overview

Probe.ComponentCount

Returns the number of components in the component list box. There is always at least one, even when it appears that there are no entries. In that case, the one entry is invisible, but it can still be made active.

? Probe Overview

Probe.ConnectionCount

Returns the number of connections in the connection drop-down list. The contents of this list depend on which component is active.

? Probe.ActiveComponent

? Probe Overview

Probe.FullName

Returns the full name of the file containing this probe’s information. Read-only **String**. If the fully qualified path name is C:\PCDMISW\PROBE\SP600.PRB, *FullName* returns “C:\PCDMISW\PROBE\SP600.PRB”.

? Probe Overview

Probe.Name

Returns the name of the file containing this probe's information. Read-only **String**. If the fully qualified path name is C:\PCDMISW\PROBE\SP600.PRB, *FullName* returns "SP600.PRB".

? Probe Overview

Probe.Parent

Returns the **Probes** collection object to which this object belongs.

? Probes Overview

? Probe Overview

Probe.Path

Returns the path to the file containing this probe's information. Read-only **String**. If the fully qualified path name is C:\PCDMISW\PROBE\SP600.PRB, *Path* returns "C:\PCDMISW\PROBE\".

? Probe Overview

Probe.Tips

Returns the **Tips** object associated with this **Probe** object.

? Tips Overview

? Probe Overview

Methods:

Probe.ClearAllTips

Syntax

expression.ClearAllTips

expression: Required expression that evaluates to a PC-DMIS **Probe** object.

Clears all tips selected for qualification. Use the "Probe.SelectAllTips" funtion on page 291 to select all tips. Use the "Tip.Selected" property of the tip object on page 310 to select or deselect individual tips for probe qualification.

Probe.ComponentDescription

Syntax

Return Value=expression.ComponentDescription(*Item*)

Return Value: This function returns a string which is the nth component description of the component list box as determined by the item parameter.

expression: Required expression that evaluates to a PC-DMIS **Probe** object.

Item: Required **Long**. The zero-based index of the string from the list box to return. This must be between 0 and *expression*.ComponentCount – 1.

? Probe Overview

Probe.ConnectionDescription

Syntax

Return Value=expression.ComponentDescription(*Item*)

Return Value: This function returns the *Item* number string in the connection drop down list..

expression: Required expression that evaluates to a PC-DMIS **Probe** object.

Item: Required **Long**. The zero-based index of the string from the drop down list to return. This must be between 0 and *expression*.ConnectionCount – 1.

? Probe Overview

Probe.Dialog

Syntax

Return Value=expression.Dialog

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **Probe** object.

Opens the PC-DMIS Probe Utilities dialog for *expression*.

Probe.Qualify

Syntax

expression.Qualify

expression: Required expression that evaluates to a PC-DMIS **Probe** object.

Qualifies all of *expression*'s tips.

? Probe Overview

Probe.SelectAllTips

Syntax

expression.SelectAllTips

expression: Required expression that evaluates to a PC-DMIS **Probe** object.

Selects all tips in tip list for qualification. Use the "Probe.ClearAllTips" function on page 290 to clear all selected tips. Use the "Tip.Selected" property of the tip object on page 310 to select or deselect individual tips for probe qualification.

Probes Object Overview

The Probes object is the collection of all Probe objects currently available to a part program. Use Probes (*index*) where *index* is the index number or name of the requested probe file.

? Probes Members

Probes Object Members

Properties:

Probes.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? Probes Overview

Probes.Count

Represents the number of Machine objects currently active in PC-DMIS. Read-only **Integer**.

? Probes Overview

Probes.Parent

Returns the parent PartProgram of this object. Read-only **PartProgram**.

? PartProgram Overview

? Probes Overview

Methods:

Probes.Add

Syntax 1

Return Value=expression.Add(*FileName*)

The Add function sets the probe name to *FileName*. This allows the user to start creating a new probe.

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a **Probes** object.

FileName: Required **String** that indicates the name of the new probe.

? Probes Overview

Probes.Item

Syntax 1

Return Value=expression.Item(*NameOrNum*)

Syntax 2

expression(*NameOrNum*)

Return Value=The Item function returns a **Probe** object.

expression: Required expression that evaluates to a **Probes** object.

NameOrNum: Required **Variant** that indicates which **Probe** object to return. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number of the **Probe** object in the **Probes** collection. If it is a **String**, it is the name of the **Probe** object.

Remarks

Since the Item method is the default, the function name can be omitted as in Syntax 2.

? Probe Overview

? Probes Overview

ScanCommand Object Overview

Objects of type **ScanCommand** are created from more generic **Command** objects to pass information specific to the scan command back and forth. At present only DCC and Manual scans are user accessible.

? Command.BasicScanCommand

? ScanCommand Members

ScanCommand Members

Properties

Scan.BoundaryCondition

Represents the boundary condition type. Read/write of enumeration BSBOUNDCOND_ENUM. All Properties and Methods related to the Boundary Conditions apply only to DCC scans.

The following are the allowable values:

BSBOUNDCOND_SPHENTRY: Represents a Spherical Boundary Condition. This Boundary condition requires the following parameters to be set by you using Automation Properties and/or Automation Methods:

```
BoundaryConditionCenter
BoundaryConditionEndApproach
Diameter
Number of Crossings
```

BSBOUNDCOND_PLANECROSS: Represents a Planar Boundary Condition. This Boundary condition requires the following parameters to be set by you using Automation Properties and/or Automation Methods:

```
BoundaryConditionCenter
BoundaryConditionEndApproach
BoundaryConditionPlaneV
Number of Crossings
```

BSBOUNDCOND_CYLINDER: Represents a Cylindrical Boundary Condition. This Boundary condition requires the following parameters to be set by you using Automation Properties and/or Automation Methods:

```
BoundaryConditionCenter
BoundaryConditionEndApproach
BoundaryConditionAxisV
Diameter
Number of Crossings
```

BSBOUNDCOND_CONE: Represents a Conical Boundary Condition. This Boundary condition requires the following parameters to be set you user using Automation Properties and/or Automation Methods:

```
BoundaryConditionCenter
BoundaryConditionEndApproach
BoundaryConditionAxisV
```

```
HalfAngle
Number of Crossings
```

The SetBoundaryConditionParams Method should be used to set the HalfAngle, number of Crossings and Diameter values.

? BasicScanCommand Overview

Scan.BoundaryConditionAxisV

This property represents the boundary condition axis vector. Read/write **PointData** object. This vector is used as the axis of the Cylindrical and Conical BoundaryConditions.

? BasicScanCommand Overview

Scan.BoundaryConditionCenter

This property represents the boundary condition center. Read/write **PointData** object.

This point is used by all Boundary Conditions and is the location of the Boundary Condition.

? ScanCommand Overview

Scan.BoundaryConditionEndApproach

This property represents the boundary condition end approach vector. Read/write **PointData** object.

This vector is used by all Boundary Conditions and is the Approach Vector of the Probe as it crosses the Boundary condition.

? ScanCommand Overview

Scan.BoundaryConditionPlaneV

This property represents the boundary condition plane vector. Read/write **PointData** object.

This vector is the normal vector of the plane used by the Plane and OldStyle Boundary Conditions.

Boundary Condition	**Properties Required**
Plane	BoundaryConditionCenter BoundaryConditionEndApproach BoundaryConditionPlaneV
Cone	BoundaryConditionCenter BoundaryConditionEndApproach BoundaryConditionAxisV
Cylinder	BoundaryConditionCenter BoundaryConditionEndApproach BoundaryConditionAxisV
Sphere	BoundaryConditionCenter BoundaryConditionEndApproach

? ScanCommand Overview

Scan.Filter

This property represents the filter type. Read/write of enumeration BSF_ENUM.

The following are the allowable values:

BSF_DISTANCE: PC-DMIS determines each hit based on the set increment and the last two measured hits. The approach of the probe is perpendicular to the line between the last two measured hits. The probe will stay on the cut plane. PC-DMIS will start at the first boundary point and continue taking hits at the set increment, stopping when it satisfies the Boundary Condition. In the case of a continous scan, PC-DMIS would filter the data from the CMM and keep only the hits that are apart by at least the increment. Both DCC and Manual scans can use this filter.

BSF_BODYAXISDISTANCE: PC-DMIS will take hits at the set increment along the current part's coordinate system. The approach of the probe is perpendicular to the indicated axis. The probe will stay on the cut plane. The approach vector will be normal to the selected axis and on the cut plane. This technique uses the same approach for taking each hit (unlike the previous technique which adjusts the approach to be perpendicular to the line between the previous two hits). The above behaviour applies to DCC scans.

When this filter is applied to Manual scans, the following behaviour happens:

This Filter property allows you to scan a part by specifying a cut plane on a certain part axis and dragging the probe across the cut plane. As you scan the part, you should scan so that the probe crisscrosses the defined Cut Plane as many times as desired. PC-DMIS then follows this procedure:

1) PC-DMIS gets data from the controller and finds the two data hits that are closest to the Cut Plane on either side as you crisscross.

2) PC-DMIS then forms a line between the two hits which will pierce the Cut Plane.

3) The pierced point then becomes a hit on the Cut Plane.

This operation happens every time you cross the Cut Plane and you will finally have many hits that are on the Cut Plane.

BSF_VARIABLEDISTANCE: This technique allows you to set specific maximum and minimum angle and increment values that will be used in determining where PC-DMIS will take a hit. The probe's approach is perpendicular to the line between the last two measured hits. You should provide the maximum and minimum values that will be used to determine the increments between hits. You also must enter the desired values for the maximum and minimum angles. PC-DMIS will take three hits using the minimum increment. It will then measure the angle between hit's 1-2 and 2-3.

- If the measured angle is between the maximum and minimum values defined, PC-DMIS will continue to take hits at the current increment.
- If the angle is greater than the maximum value, PC-DMIS will erase the last hit and measure it again using one quarter of the current increment value.
- If the angle is less than the minimum increment, PC-DMIS will take the hit at the minimum increment value.

PC-DMIS will again measure the angle between the newest hit and the two previous hits. It will continue to erase the last hit and drop the increment value to one quarter of the increment until the measured angle is within the range defined, or the minimum value of the increment is reached.

If the measured angle is less than the minimum angle, PC-DMIS will double the increment for the next hit. (If this is greater than the maximum increment value it will take the hit at the maximum increment.) PC-DMIS will again measure the angle between the newest hit and the two previous hits. It will continue to double the increment value until the measured angle is within the range defined, or the maximum increment is reached. The above behaviour applies to DCC scans.

When this filter is applied to Manual scans, the following behaviour occurs:

The filter defines the distance between hits based on the part. PC-DMIS allows you to specify the speed at which it will read hits and the drop point distance. After the scanning process is complete, PC-DMIS will calculate the total number of hits that were measured and the total number that were kept after completing the drop point distance calculations. The reduced data is then converted to hits.

BSF_TIME_DELTA: The Time Delta method of scanning allows you to reduce the scan data by setting a time increment. PC-DMIS will start from the first hit and reduce the scan by deleting hits that are read in faster than the time delta specified.

? ScanCommand Overview

Scan.HitType

Represents the type of hit to use. Read/write of enumeration BSCANHIT_ENUM.

The following are the allowable values:

BSCANHIT_VECTOR – use vector hits for this scan

BSCANHIT_SURFACE – use surface hits for this scan

BSCANHIT_EDGE – use edge hits for this scan.

BSCANHIT_BASIC – use basic hits for this scan. Only Manual scans use this hit type. Currently there are no Manual Scans.

Remarks

Not every hit type can be used with every method and filter combination.

Method	Vector Hit	Surface Hit	Basic Hit	Edge Hit
Open	Y	Y	-	Y
Close	Y	Y	-	Y
Patch	Y	Y	-	Y
HardProb	-	-	-	Y
TTP	-	-	-	Y

? ScanCommand Overview

Scan.Method

This property represents the method type for this scan. Read/write of enumeration BSMETHOD_ENUM.

The following are the allowable values:

BSCANMETH_OPEN: This method will scan the surface along a line. This procedure uses the starting and ending point for the line and also includes a direction point. The probe will always remain within the cut plane while doing the scan. This is valid oly for DCC scans.

BSCANMETH_CLOSE: This method will scan the surface along a line. This procedure uses the starting and ending point for the line and also includes a direction point. The probe will always remain within the cut plane while doing the scan.The scan will start and finish at the same Point. This is valid oly for DCC scans.

BSCANMETH_PATCH: This method will scan the surface in multiple rows depending on the Boundary Points. This is valid oly for DCC scans.

BSCANMETH_MANUAL_TTP: This is valid only for Manual scans and will allow you to use a Touch Trigger Probe to take Manual hits.

BSCANMETH_MANUAL_FIXED_PROBE: This is valid only for Manual scans and will allow you to use a Hard Probe to take Manual hits.

? ScanCommand Overview

Scan.MethodCutPlane

This property represents the method's cut plane vector. Read/write **PointData** object.

? ScanCommand Overview

Scan.MethodEnd

This property represents the scan's end point. Read/write **PointData** object.

? ScanCommand Overview

Scan.MethodEndTouch

This property represents the method's end touch vector. Read/write **PointData** object.

? ScanCommand Overview

Scan.MethodInitDir

This property represents the method's initial direction vector. Read/write **PointData** object.

? ScanCommand Overview

Scan.MethodInitTopSurf

This property represents the initial Surface Vector for the Edge method. Read/write **PointData** object.

? ScanCommand Overview

Scan.MethodInitTouch

This represents the method's initial touch vector. Read/write **PointData** object.

? ScanCommand Overview

Scan.MethodStart

This property represents the scan's start point. Read/write **PointData** object.

Method	Method Start	Method End	Method Cutplane	Method InitDir	Method InitTouch	Method InitTopSurf	Method EndTouch
Open	Y	Y	Y	Y	Y	-	Y
Close	Y	Y	Y	Y	Y	-	-
Patch	-	-	Y	Y	Y	-	Y
TTP	-	-	Y	Y	Y	-	-
HardProbe	Y	Y	Y	Y	Y	-	-

? ScanCommand Overview

Scan.NominalMode

This property represents how to determine the nominals for this scan. Read/write of enumeration BSCANNMODE_ENUM.

The following are the allowable values:

BSCANNMODE_FINDCADNOMINAL: This mode would find the Nominal data from CAD after scanning. This mode is useful only when CAD surface data is available.

SCANNMODE_MASTERDATA: This mode keeps the data scanned the first time as Master data.

? ScanCommand Overview

Scan.OperationMode

This property represents mode of operation of the scan. Read/write of enumeration BSOPMODE_ENUM.

The following are the allowable values:

BSCANOPMODE_REGULARLEARN: When this mode is used, PC-DMIS will execute the scan as though it is learning it. All learned measured data will replace the new measured data. The nominal will be re-calculated depending on the Nominals mode.

BSCANOPMODE_DEFINEPATHFROMHITS: This mode is available only when using analog probe heads that can do continuous contact scanning. When this option is selected, PC-DMIS allows the controller to 'define' a scan. PC-DMIS gathers all hit locations from the editor and passes them onto the controller for scanning. The controller will then adjust the path allowing the probe to pass through all the points. The data is then reduced according to the increment provided and the new data will replace any old measured data This value cannot be used currently through Automation because there is no Method provided to define a path.

BSCANOPMODE_NORMALEXECUTION: If a DCC scan is executed, PC-DMIS will take hits at each of the learned locations in Stitch scanning mode, storing the newly measured data.

Method	Regular Learn	Defined Path	Normal
Open	Y	-	Y
Close	Y	-	Y
Patch	Y	-	Y
TTP	Y	-	Y
HardProbe	Y	-	Y

? ScanCommand Overview

Methods:

Scan.GetBoundaryConditionParams

Syntax

Return Value=expression. GetBoundaryConditionParams (*nCrossings, dRadius, dHalfAngle*)

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

nCrossing: Required **Long** variable that gets the number of crossings for this boundary condition. The scan would stop after the probe crosses (breaks) the Boundary Condition like a Sphere, Cylinder, Cone, or a Plane the given number of times.

dRadius: Required **Double** variable that gets the radius of the boundary condition. This is used by the Spherical and Cylindrical Boundary Conditions.

dHalfAngle: Required **Double** variable that gets the half-angle of the cone-type boundary condition, or gets zero if the boundary condition is not of cone type.

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

Remarks

Boundary Condition	GetBoundaryConditionParams (*nCrossings, dRadius, dHalfAngle*)
Plane	*Ncrossings*
Cone	*NCrossings, ,dHalfAngle*
Cylinder	*NCrossings, dRadius*
Sphere	*NCrossings, dRadius*

? ScanCommand Overview

Scan.GetFilterParams

Syntax

Return Value=expression. GetFilterParams (*dCutAxisLocation, nAxis, dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

dCutAxisLocation: Used for Manual scans with Filter property set to BSF_BODYAXISDISTANCE.

nAxis: Required **Long** variable that gets the cut axis. Returns non-zero only for axis filters. For axis filters, 0 means the X axis, 1 means the Y-axis, and 2 means the Z-axis.

dMaxIncrement: Required **Double** variable that gets the maximum increment. For fixed-length filters, this is simply the fixed increment. This is the Time delta valus in case the filter is BSF_TIME_DELTA or BSF_VARIABLEDISTANCE for Manual scans.

dMinIncrement: Required **Double** variable that gets the minimum increment for Variable Distance Filters. This is the Drop Point distance when a Manul scan is being used with the filter set to BSF_VARIABLEDISTANCE.

dMaxAngle: Required **Double** variable that gets the maximum angle used in Variable Distance Filters.

dMinAngle: Required **Double** variable that gets the minimum angle used in Variable Distance Filters.

Remarks

Filter	**GetFilterParams (*dCutAxisLocation, nAxis, dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*)**
Distance	*,,dMaxIncrement*
BodyAxisDistance (DCC)	*,nAxis, dMaxIncrement*
BodyAxisDistance (Manual)	*NCutLocation,nAxis*
Time	*,,dMaxIncrement*
VariableDistance	*,,dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*

? ScanCommand Overview

Scan.GetHitParams

Syntax

Return Value=expression. GetHitParams (*nInitSamples, nPermSamples, dSpacer, dIndent, dDepth*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

nInitSamples: Required **Long** variable that gets the number of initial sample hits for the hits in this scan. It always returns zero for basic hits and vector hits.

nPermSamples: Required **Long** variable that gets the number of permanent sample hits for the hits in this scan. It always returns zero for basic hits and vector hits.

dSpacer: Required **Double** variable that gets the spacing of the sample hits from the hit center. It always returns zero for basic hits and vector hits.

dIndent: Required **Double** variable that gets the indent of the sample hits from the hit center. It always returns zero for basic hits, vector hits, and surface.

dDepth: Required **Double** variable that gets the depth of the sample hits from the hit center. It always returns zero for basic hits, vector hits, and surface.

? ScanCommand Overview

Scan.GetMethodPointData

Syntax

Return Value=expression. GetMethodPointData (*MethodStart, MethodEnd, MethodInitTouch, MethodEndTouch, MethodInitDir, MethodCutPlane*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

MethodStart: Required **PointData** object that gets the MethodStart property.

MethodEnd: Required **PointData** object that gets the MethodEnd property.

MethodInitTouch: Required **PointData** object that gets the MethodInitTouch property.

MethodEndTouch: Required **PointData** object that gets the MethodEndTouch property.

MethodInitDir: Required **PointData** object that gets the MethodInitDir property.

MethodCutPlane: Required **PointData** object that gets the MethodCutPlane property.

Remarks

If scan is a **ScanCommand** object, and MS, ME, MIT, MET, MID,and MCP are all **Dim**ensioned as **Object**, the following are equivalent:

scan.GetMethodParams MS,ME,MIT,MET,MID,MCP

```
set MS = scan.MethodStart
    set ME = scan.MethodEnd
    set MIT = scan.MethodInitTouch
    set MET = scan.MethodEndTouch
    set MID = scan.MethodInitDir
    set MCP = scan.MethodCutPlane
```

This method is provided as a shortcut to getting these commonly used properties all at once.

? Scan.MethodStart

? Scan.MethodEnd

? Scan.MethodInitTouch

? Scan.MethodEndTouch

? Scan.MethodInitDir

? Scan.MethodCutPlane

? ScanCommand Overview

Scan.GetNomsParams

Syntax

Return Value=*expression*. GetNomsParams (*dFindNomsTolerance*, *dSurfaceThickness*, *dEdgeThickness*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

dFindNomsTolerance: Required **Double** variable that gets the Find Noms tolerance and is used only when the **NominalMode** property is BSCANNMODE_FINDCADNOMINAL.

dSurfaceThickness: Required **Double** variable that gets the surface thickness and is used only when the **NominalMode** property is BSCANNMODE_FINDCADNOMINAL.

dEdgeThickness: Required **Double** variable that gets the edge thickness and is used only when the **NominalMode** property is BSCANNMODE_FINDCADNOMINAL and when the **Method** property is BSCANMETH_EDGE.

? ScanCommand Overview

Scan.GetParams

Syntax

Return Value=expression. GetParams (*Method, Filter, OperationMode, HitType, NominalMode, BoundaryCondition*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

Method: Required **Long** variable that gets the Method property.

Filter: Required **Long** variable that gets the Filter property.

OperationMode: Required **Long** variable that gets the OperationMode property.

HitType: Required **Long** variable that gets the HitType property.

NominalMode: Required **Long** variable that gets the NominalMode property.

BoundaryCondition: Required **Long** variable that gets the BoundaryCondition property.

Remarks

If scan is a **ScanCommand** object, and M, F, O, H, N,and B are all dimensioned as **Object**, the following are equivalent:

scan.GetParams M, F, O, H, N, B

```
M = scan.Method
    F = scan.Filter
    O = scan.OperationMode
    H = scan.HitType
    N = scan.NominalMode
    B = scan.BoundaryCondition
```

This method is provided as a shortcut to getting these commonly used properties all at once.

? Scan.Method Property

? Scan.Filter Property

? Scan.OperationMode Property

? Scan.HitType Property

? Scan.NominalMode Property

? Scan.BoundaryCondition Property

? ScanCommand Overview

Scan.SetBoundaryConditionParams

Syntax

Return Value=expression.SetBoundaryConditionParams (*nCrossings, dRadius, dHalfAngle*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

nCrossing: Required **Long** that sets the number of crossings for this boundary condition.

dRadius: Required **Double** that sets the radius of the boundary condition.

dHalfAngle: Required **Double** that sets the half-angle of the cone-type boundary condition, or is ignored if the boundary condition is not of cone type.

Remarks

Boundary Condition	**SetBoundaryConditionParams (*nCrossings, dRadius, dHalfAngle*)**
Plane	*Ncrossings*
Cone	*NCrossings,, dHalfAngle*
Cylinder	*NCrossings, dRadius*
Sphere	*NCrossings, dRadius*

? ScanCommand Overview

Scan.SetFilterParams

Syntax

Return Value=expression.SetFilterParams (*dCutAxisLocation, nAxis, dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

dCutAxisLocation: Used for Manual scans with Filter property set to BSF_BODYAXISDISTANCE.

nAxis: **Long** variable that gets the cut axis. Returns non-zero only for axis filters. For axis filters, 0 means the X axis, 1 means the Y-axis, and 2 means the Z-axis.

dMaxIncrement: **Double** variable that gets the maximum increment. For fixed-length filters, this is simply the fixed increment. This is the Time delta valus in case the filter is BSF_TIME_DELTA or BSF_VARIABLEDISTANCE for Manual scans.

dMinIncrement: **Double** variable that gets the minimum increment for Variable Distance Filters. This is the Drop Point distance when a Manul scan is being used with the filter set to BSF_VARIABLEDISTANCE.

dMaxAngle: **Double** variable that gets the maximum angle used in Variable Distance Filters.

dMinAngle: **Double** variable that gets the minimum angle used in Variable Distance Filters.

Remarks

Filter	**SetFilterParams (*dCutAxisLocation, nAxis, dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*)**
Distance	*,,dMaxIncrement*
BodyAxisDistance	*,nAxis, dMaxIncrement*
VariableDistance	*,,dMaxIncrement, dMinIncrement, dMaxAngle, dMinAngle*

? ScanCommand Overview

Scan.SetHitParams

Syntax

Return Value=expression.SetHitParams (*nInitSamples*, *nPermSamples*, *dSpacer*, *dIndent*, *dDepth*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

nInitSamples: Required **Long** that sets the number of initial sample hits for the hits in this scan. It is ignored for basic hits and vector hits.

nPermSamples: Required **Long** that sets the number of permanent sample hits for the hits in this scan. It is ignored for basic hits and vector hits.

dSpacer: Required **Double** that sets the spacing of the sample hits from the hit center. It is ignored for basic hits and vector hits.

dIndent: Required **Double** that sets the indent of the sample hits from the hit center. It is ignored for basic hits, vector hits, and surface.

dDepth: Required **Double** that sets the depth of the sample hits from the hit center. It is ignored for basic hits, vector hits, and surface.

? ScanCommand Overview

Scan.SetMethodPointData

Syntax

Return Value=expression.SetMethodPointData *(MethodStart, MethodEnd, MethodInitTouch, MethodEndTouch, MethodInitDir, MethodCutPlane)*

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

MethodStart: Required **PointData** object that sets the MethodStart property.

MethodEnd: Required **PointData** object that sets the MethodEnd property.

MethodInitTouch: Required **PointData** object that sets the MethodInitTouch property.

MethodEndTouch: Required **PointData** object that sets the MethodEndTouch property.

MethodInitDir: Required **PointData** object that sets the MethodInitDir property.

MethodCutPlane: Required **PointData** object that sets the MethodCutPlane property.

Remarks

If scan is a **ScanCommand** object, and MS, ME, MIT, MET, MID,and MCP are all dimensioned as **Object**, the following are equivalent:

scan.SetMethodParams MS,ME,MIT,MET,MID,MCP

```
set scan.MethodStart = MS
    set scan.MethodEnd = ME
    set scan.MethodInitTouch = MIT
    set scan.MethodEndTouch = MET
    set scan.MethodInitDir = MID
    set scan.MethodCutPlane = MCP
```

This method is provided as a shortcut to setting these commonly used properties all at once.

? Scan.MethodStart

? Scan.MethodEnd

? Scan.MethodInitTouch

? Scan.MethodEndTouch

? Scan.MethodInitDir

? Scan.MethodCutPlane

? Scan.Command Overview

Scan.SetNomsParams

Syntax

Return Value=expression.SetNomsParams (*dFindNomsTolerance*, *dSurfaceThickness*, *dEdgeThickness*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

dFindNomsTolerance: Required **Double** that sets the Find Noms tolerance.

dSurfaceThickness: Required **Double** that sets the surface thickness.

dEdgeThickness: Required **Double** that sets the edge thickness.

Remarks

? ScanCommand Overview

Scan.SetParams

Syntax

Return Value=expression.SetParams (*Method*, *Filter*, *OperationMode*, *HitType*, *NominalMode*, *BoundaryCondition*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

Method: Required **Long** that sets the Method property.

Filter: Required **Long** that sets the Filter property.

OperationMode: Required **Long** that sets the OperationMode property.

HitType: Required **Long** that sets the HitType property.

NominalMode: Required **Long** that sets the NominalMode property.

BoundaryCondition: Required **Long** that sets the BoundaryCondition property.

Remarks

If scan is a **ScanCommand** object, and M, F, O, H, N,and B are all dimensioned as **Object**, the following are equivalent:

scan.SetParams M, F, O, H, N, B

```
scan.Method = M
    scan.Filter = F
    scan.OperationMode = O
    scan.HitType = H
    scan.NominalMode = N
    scan.BoundaryCondition = B
```

This method is provided as a shortcut to setting these commonly used properties all at once.

? ScanCommand Overview

Scan.CreateBasicScan

Syntax

Return Value=expression. CreateBasicScan()

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a PC-DMIS **ScanCommand** object.

This method has to be called after calling other Properties/Methods. This method creates the necessary BasicScans needed by DCC and Manual scans and inserts them into the Part Program.

? ScanCommand Overview

Statistics Object Overview

The Statistics Automation object gives access to the properties and data members of the PC-DMIS Statistics command.

? Statistics Members

Statistics Members

Properties:

Statistics.CalcMode

LONG value representing whether calculation mode inside of DataPage is turned off or on.

Read/Write **Long**

? Statistics Object Overview

Statistics.MemoryPages

LONG value representing number of memory pages to be used by DataPage.

Read/Write **Long**

? Statistics Object Overview

Statistics.NameType

ENUM_STAT_NAME_TYPES enumeration value indicating whether the feature name or the dimension name should be sent to DataPage. If set to PCD_STAT_FEAT_NAME (1), the feature name is used. If set to PCD_STAT_DIM_NAME (0), the dimension name is used.

Read/Write **ENUM_STAT_NAME_TYPES enumeration**

? Statistics Object Overview

Statistics.ReadLock

LONG value representing the number of seconds in timeout period that DataPage uses when trying to read the port lock.

Read/Write **Long**

? Statistics Object Overview

Statistics.StatMode

ENUM_PCD_STAT_TYPES enumeration value representing the mode or function of the statistics command. Possible values include the following:

```
PCD_STATS_OFF = 0
PCD_STATS_ON = 1
PCD_STATS_TRANSFER = 2
PCD_STATS_UPDATE = 3
```

Read/Write **ENUM_PCD_STAT_TYPES enumeration**

? Statistics Object Overview

Statistics.WriteLock

LONG value representing number of seconds in timeout period that DataPage uses when trying to write to the port lock.

Read/Write **Long**

? Statistics Object Overview

Methods:

Statistics.AddStatsDir

Syntax:

expression.AddStatsDir (Dir)

Return Value: *Boolean* value indicating success or failure of call to method.

expression: Required expression that evaluates to a PC-DMIS **Statistics** object.

Dir: Required **String** representing the name of the directory to be added to the list of statistics directories.

? Statistics Object Overview

Statistics.GetStatsDir

Syntax:

expression.GetStatsDir (Index)

Return Value: *String* representing the name of the directory at the specified index value. If index value is greater than the number of directories in the list, the string will be empty.

expression: Required expression that evaluates to a PC-DMIS **Statistics** object.

Index: Required **Long** representing the index of the directory name to be retrieved.

? Statistics Object Overview

Statistics.RemoveStatsDir

Syntax:

expression.RemoveStatsDir (Index)

Return Value: *Boolean* value indicating success or failure of call to remove directory from the list of directories. If index is greater than the number of directories in the list, the call will fail.

expression: Required expression that evaluates to a PC-DMIS **Statistics** object.

Index: Required **Long** representing the line of text to be removed.

? Statistics Object Overview

Statistics.SetStatsDir

Syntax:

expression.SetStatsDir (Index, Dir)

Return Value: Boolean value indicating success or failure of call to set name of the directory specified by Index. If the index value is greater than the number of directories, the call will fail.

expression: Required expression that evaluates to a PC-DMIS **Statistics** object.

Index: Required **Long** representing the directory name to change.

Dir: Required **String** which is the new name of the directory.

? Statistics Object Overview

Temperature Compensation Object Overview

The Temperature Compensation Automation object gives access to the properties of the PC-DMIS Temperature Compensation command. For additional information about Temperature Compensation, see "Temperature Compensation Setup" in the "System Options" section of the *PC-DMIS Reference Manual*.

? Temperature Compensation Members

Temperature Compensation Members

Properties:

TempComp.HighThreshold

DOUBLE value representing the high temperature threshold.

Read/Write **Double**

? Temperature Compensation Object Overview

TempComp.LowThreshold

DOUBLE value representing the low temperature threshold.

Read/Write **Double**

? Temperature Compensation Object Overview

TempComp.MaterialCoefficient

DOUBLE value indicating the material coefficient.

Read/Write **Double**

? Temperature Compensation Object Overview

TempComp.RefTemp

DOUBLE value representing the reference temperature.

Read/Write **Double**

? Temperature Compensation Object Overview

TempComp.Sensors

STRING value representing the list of sensors—by number—to be used for temperature compensation. The format of the list is a series of consecutive sensor numbers. The series are specified using the hyphen between the first number and the last number of the series. Each non-consecutive sensor or group of sensors is separated by the comma (or the typical separator for the given locale).

Read/Write **String**

Example: The sensors 2, 4, 5, 6, 8, 10, 11, 12, 13 would be represented as “2,4-6,8,10-13”.

? Temperature Compensation Object Overview

Methods:

TempComp.GetOrigin

Syntax:

expression.GetOrigin (X, Y, Z)

expression: Required expression that evaluates to a PC-DMIS **TempComp** object.

X: Required **Long** variable that receives the X value of the temperature compensation origin.

Y: Required **Long** variable that receives the Y value of the temperature compensation origin.

Z: Required **Long** variable that receives the Z value of the temperature compensation origin.

? Temperature Compensation Object Overview

TempComp.SetOrigin

Syntax:

expression.SetOrigin (X, Y, Z)

expression: Required expression that evaluates to a PC-DMIS **TempComp** object.

X: Required **Long** that sets the X value of the temperature compensation origin.

Y: Required **Long** that sets the Y value of the temperature compensation origin.

Z: Required **Long** that sets the Z value of the temperature compensation origin.

? Temperature Compensation Object Overview

Tip Object Overview

The Tip object describes a single tip of a probe. All of its properties are read-only.

? Tip Members

Tip Members

Properties:

Tip.A

Returns the A angle of the tip. Read-only **Double**.

? Tip Overview

Tip.B

Returns the B angle of the tip. Read-only **Double**.

? Tip Overview

Tip.Date

Returns the PC-DMIS representation of the most recent calibration date of the tip. Read-only **String**.

? Tip Overview

Tip.Diam

Returns the diameter of the tip. Read-only **Double**.

? Tip Overview

Tip.ID

Returns the ID string of the tip. Read-only **String**.

? Tip Overview

Tip.IJK

A **PointData** object that returns the vector along which the tip lies. Read-only.

Remarks

If there is a rotary table, the table rotation is taken into account.

? Tip Overview

Tip.MeasDiam

Returns the measured diameter of the tip. Read-only **Double**.

? Tip Overview

Tip.MeasThickness

Returns the measured thickness of the tip. Read-only **Double**.

? Tip Overview

Tip.MeasXYZ

Returns the measured location of the tip as a **PointData**. Read-only.

? PointData Overview

? Tip Overview

Tip.Parent

Returns the **Tips** collection object that contains this tip. Read-only.

? Tips Overview

? Tip Overview

Tip.Selected

Determines whether tip is selected for qualification. Read/Write **Boolean**

Remarks:

Use the "Probe.SelectAllTips" method of the probe object on page 291 to select all tips at once and the "Probe.ClearAllTips" method of the probe object on page 290 to clear all tips at once.

? Tip Overview

Tip.Thickness

Returns the nominal thickness of the tip. Read-only **Double**.

? Tip Overview

Tip.Time

Returns the PC-DMIS representation of the most recent calibration time of the tip. Read-only **String**.

? Tip Overview

Tip.TipNum

Returns the tip number in the list of tips. Read-only **Long**.

Remarks

This is PC-DMIS's internal representation of tip number. It may be different from the number passed to Tips.Item to retrieve the tip.

? Tip Overview

Tip.Type

Returns the type of the tip. Read-only **Long**.

Remarks

The following tip types are defined. They can be combined via bitwise operations.

```
TIPBALL // Default
    TIPDISK
    TIPSHANK
    TIPOPTIC
    TIPANALOG
    TIPANALOGBALL  = TIPANALOG + BALL
    TIPANALOGDISK  = TIPANALOG + DISK
    TIPANALOGSHANK  = TIPANALOG + SHANK
    TIPANALOGOPTIC  = TIPANALOG + OPTIC
    TIPFIXED
    TIPFIXEDBALL = TIPFIXED + BALL
    TIPFIXEDDISK = TIPFIXED + DISK
    TIPFIXEDSHANK = TIPFIXED + SHANK
    TIPFIXEDOPTIC = TIPFIXED + OPTIC
    TIPSP600 // renishaw sp600 analog probe
```

```
TIPWBOPTIC // wolf and beck laser probe
TIPINFINITARM // renishaw infinite index arm
TIPSLAVE // tip belongs to slave arm
```

? Tip Overview

Tip.WristOffset

Returns the wrist offset of the tip. Read-only **PointData**.

? PointData Overview

? Tip Overview

Tip.WristTipIJK

Returns the wrist tip vector of the tip. Read-only **PointData**.

? PointData Overview

? Tip Overview

Tip.XYZ

Returns the location of the tip. Read-only **PointData**.

? PointData Overview

? Tip Overview

Tips Object Overview

The **Tips** object is the collection of all **Tip** objects for a **Probe** object. The **Probe** object that the **Tips** stores **Tip** objects for is contained in the Parent property.

? Tips Object Members

Tips Members

Properties:

Tips.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? Tips Overview

Tips.Count

Represents the number of Tip objects in the parent Probe object. Read-only **Integer**.

? Tips Overview

Tips.Parent

Returns the parent **Probe** object. Read-only.

? Probe Overview

? Tips Overview

Methods:

Tips.Add

Syntax

expression.Add *a*, *b*

expression: Required expression that evaluates to a PC-DMIS **Tips** object.

a: Required Double that is the A parameter of the new tip.

b: Required Double that is the B parameter of the new tip.

This function adds a new tip position to this collection. The new tip is unqualified.

? Tips Overview

Tips.Item

Syntax 1

Return Value=expression.Item(*NameOrNum*)

Syntax 2

expression(*NameOrNum*)

Return Value: The Item function returns a **Tip** object.

expression: Required expression that evaluates to a **Tips** object.

NameOrNum: Required **Variant** that indicates which **Tip** object to return. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number of the **Tip** object in the **Tips** collection. If it is a **String**, it is the ID of the **Tip** object.

Remarks

Since the Item method is the default, the function name can be omitted as in Syntax 2.

? Tips Overview

Tips.Remove

Syntax

expression.Remove*Num*

expression: Required expression that evaluates to a **Tips** object.

Num: Required **Long** that indicates which **Tip** object to remove.

This function removes the indicated **Tip** object from this collection.

? Tips Overview

Tool Object Overview

The **Tool** object represents a single probe calibration tool. Use **Tools**(*index*) where *index* is the index number or tool name to return a single **Tool** object.

? Tool Object Members

Tool Members

Properties:

Tool.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? Tool Overview

Tool.Diam

Returns the diameter of the tool. Read-only **Double**.

? Tool Overview

Tool.ID

Returns the ID of the tool. Read-only **String**.

? Tool Overview

Tool.Parent

Returns the parent **Tools** object. Read-only.

? Tools Overview

? Tool Overview

Tool.ShankIJK

Returns the shank vector of the tool as a **PointData**. Read-only.

? PointData Overview

? Tool Overview

Tool.ToolType

Returns the type of the tool. Read-only **Long**.

Remarks

There is only one type of tool currently available, TOOLSPHERE.

? Tool Overview

Tool.Width

Returns the width of the tool. Read-only **Double**.

? Tool Overview

Tool.XYZ

Returns the location of the tool. Read-only **PointData**.

? PointData Overview

? Tool Overview

Tools Object Overview

The **Tools** collection object contains the tools available to the parent **PartProgram** object. Use **Tools**(*index*) where *index* is the index number or tool name to return a single **Tool** object.

? Tools Object Members

Tools Members

Properties:

Tools.Application

Represents the read-only PC-DMIS application. The **Application** object includes properties and methods that return top-level objects. For example, the ActivePartProgram property returns a **PartProgram** object.

? Application Overview

? Tools Overview

Tools.Count

Represents the number of Tool objects in the parent **PartProgram** object. Read-only **Integer**.

? Tools Overview

Tools.Parent

Returns the parent **PartProgram** object. Read-only.

? PartProgram Overview

? Tools Overview

Methods:

Tools.Add

Syntax

Return Value=expression.Add(*ID*)

Return Value: Returns a **Tool** object.

expression: Required expression that evaluates to a PC-DMIS **Tips** object.

ID: Required **String** that is the name of the new tool.

This function adds a new tool to this collection. The new tool is unqualified.

? Tool Overview

? Tools Overview

Tools.Item

Syntax 1

Return Value=expression.Item(*NameOrNum*)

Syntax 2

expression(*NameOrNum*)

Return Value: The Item function returns a **Tool** object.

expression: Required expression that evaluates to a **Tools** object.

NameOrNum: Required **Variant** that indicates which **Tool** object to return. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number of the **Tool** object in the **Tools** collection. If it is a **String**, it is the ID of the **Tool** object.

Remarks

Since the Item method is the default, the function name can be omitted as in Syntax 2.

? Tool Overview

? Tools Overview

Tools.Remove

Syntax

Return Value=expression.Remove(*ID*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails.

expression: Required expression that evaluates to a **Tools** object.

ID: Required **String** that indicates which **Tool** object to remove.

This function removes the indicated **Tool** object from this collection.

? Tool Overview

? Tools Overview

Tracefield Object Overview

The Tracefield Automation object gives access to the name and value properties of the PC-DMIS Tracefield command. For additional information on this command see "Trace Field" in the "Utilities" section of the *PC-DMIS Reference Manual*.

? Tracefield Members

Tracefield Members

Properties:

Tracefield.Name

STRING value representing the name of the tracefield.

Read/Write **String**

? Tracefield Object Overview

Tracefield.Value

STRING value representing the value for the tracefield.

Read/Write **String**

? Tracefield Object Overview

Old PC-DMIS Basic Functions

Introduction

These PC-DMIS OldBasic functions were made available in previous version of PC-DMIS basic and are provided here, listed in alphabetical order, for backwards compatibility.

Fuctions A

AddBoundaryPoint

AddBoundaryPoint x:=(Double), y:=(Double), z:=(Double)

This function is used to add the initial point, end point, and other boundary points in the case of patch scans. It should be called for each boundary point to be added. It should not be called more than num_bnd_pnts times (as specified in the call to StartScan).

x,y,z: Coordinates of the boundary point.

AddFeature

AddFeature ID:=(String), off1:=(Double), off2:=(Double), off3:=(Double)

ID: ID string of the feature to add.

off1: X offset for an offset point. Single offset for this feature for an offset plane or line.

off2: Y offset for an offset point.

off3: Z offset for an offset point.

Note: This function is used for constructed features only. The parameters off1, off2, and off3 are only used in the case of offset points, planes or lines.

AddLevelFeat

AddLevelFeat ID:=(String)

ID: Name of level feature to be added

This function is used in conjunction with the iterate alignment command

AddOriginFeat

AddOriginFeat ID:=(String)

ID: Name of origin feature to be added

This function is used in conjunction with the iterate alignment command

AddRotateFeat

AddRotateFeat ID:=(String)

ID: Name of rotation feature to be added

This function is used in conjunction with the iterate alignment command

ArcSin

ArcSin x:=(Double)

Returns the arc sine of x in degrees.

ArcCos

ArcCos x:=(Double)

Return the arc cosine of x in degrees.

Functions B

BestFit2D

BestFit2D num_inputs:= (Integer), workplane:= (Integer)

num_inputs: The number of features to use to create the best fit alignment. There must be a corresponding number of calls to Feature before the call to EndAlign.

workplane: The workplane of the 2D alignment. Must be PCD_TOP, PCD_BOTTOM, PCD_FRONT, PCD_BACK, PCD_LEFT, or PCD_RIGHT.

BestFit3D

BestFit3D num_inputs:= (Integer)

num_inputs: The number of features to use to create the best fit alignment. There must be a corresponding number of calls to Feature before the call to EndAlign.

Functions C

Calibrate

Calibrate sphere:=(String), tool:=(String)[, moved:=(Integer)]

sphere: Id of measured sphere used in calibration.

tool: Id of tool object used in calibration.

moved: Toggle indicating whether first hit should be taken manually or not. Can be either PCD_NO or PCD_YES.

CatchMotionError

CatchMotionError tog:=(Integer), catch_error:=(Integer)

tog: PCD_CATCH_IN_INTEGER: All subsequent motion errors will cause the integer passed by reference as the catch_error parameter to be set to a non-zero value.

PCD_TRIGGER_ERROR: All subsequent motion errors will generate runtime error 1001. These error may be caught using the On Error statement.

PCD_OFF: Turns off error catching. The basic script will no longer be notified when motion errors occur.

catch_error: A reference to the integer that will be set to a non-zero value if a CMM error occurs. When error catching is turned on, this integer is automatically initialized to zero. Only used when tog is set to PCD_CATCH_IN_INTEGER.

Check

Check distance:= (Double)

distance: The new check distance.

ClearPlane

ClearPlane plane1:= (Integer), val1:= (Double), plane2:= (Integer), val2:= (Double)

plane1: Clearance plane. Must be one of the following values:

PCD_TOP, PCD_BOTTOM, PCD_LEFT, PCD_RIGHT, PCD_FRONT, PCD_BACK

val1: The height of the workplane.

plane2: Pass through plane. Must be one of the values listed in the description of plane1.

val2: The height of the pass through plane.

Column132

Column132 tog:=(Integer)

Turns on or off 132 column mode.

tog: PCD_ON or PCD_OFF

Comment

Comment ctype:=(Integer), comment:=(String)

ctype: PCD_REPORT, PCD_OPERATOR, or PCD_INPUT.

comment: The comment string.

CreatID

CreateID ID:=(String), ftype:=(Integer)

ID: Reference to a string to hold the newly created ID.

ftype: MEAS_POINT, MEAS_CIRCLE, MEAS_SPHERE, MEAS_LINE, MEAS_CONE, MEAS_CYLINDER, MEAS_PLANE, MEAS_SET, READ_POINT, CONST_ORIG_POINT, CONST_OFF_POINT, CONST_PROJ_POINT, CONST_MID_POINT, CONST_DROP_POINT, CONST_PIERCE_POINT,

CONST_INT_POINT, CONST_CAST_POINT, CONST_CORNER_POINT, CONST_BFRE_CIRCLE, CONST_BF_CIRCLE, CONST_PROJ_CIRCLE, CONST_REV_CIRCLE, CONST_CONE_CIRCLE, CONST_CAST_CIRCLE, CONST_INT_CIRCLE, CONST_BFRE_SPHERE, CONST_BF_SPHERE, CONST_PROJ_SPHERE, CONST_REV_SPHERE, CONST_CAST_SPHERE,CONST_BFRE_LINE, CONST_BF_LINE, CONST_PROJ_LINE, CONST_REV_LINE, CONST_MID_LINE, CONST_CAST_LINE, CONST_INT_LINE, CONST_OFF_LINE, CONST_ALN_LINE, CONST_PRTO_LINE, CONST_PLTO_LINE, CONST_BFRE_CONE, CONST_BF_CONE,CONST_PROJ_CONE, CONST_REV_CONE,CONST_CAST_CONE,CONST_BFRE_CYLINDER, CONST_BF_CYLINDER, CONST_PROJ_CYLINDER,CONST_REV_CYLINDER, CONST_CAST_CYLINDER, CONST_BFRE_PLANE, CONST_BF_PLANE, CONST_REV_PLANE, CONST_MID_PLANE, CONST_CAST_PLANE, CONST_OFF_PLANE, CONST_ALN_PLANE, CONST_PRTO_PLANE, CONST_PLTO_PLANE,CONST_HIPNT_PLANE, CONST_SET, AUTO_VECTOR_HIT, AUTO_SURFACE_HIT, AUTO_EDGE_HIT, AUTO_ANGLE_HIT, AUTO_CORNER_HIT, AUTO_CIRCLE, AUTO_SPHERE, AUTO_CYLINDER, AUTO_ROUND_SLOT, AUTO_SQUARE_SLOT, AUTO_ELLIPSE, PCD_CURVE, DIM_LOCATION, DIM_STRAIGHTNESS, DIM_ROUNDNESS, DIM_FLATNESS, DIM_PERPENDICULARITY, DIM_PARALLELISM, DIM_PROFILE, DIM_3D_DISTANCE, DIM_2D_DISTANCE, DIM_3D_ANGLE, DIM_2D_ANGLE, DIM_RUNOUT, DIM_CONCENTRICITY, DIM_ANGULARITY, DIM_KEYIN, DIM_TRUE_POSITION, PCD_ALIGNMENT

Functions D

DefaultAxes

DefaultAxes

This command is used only for location and true position dimensions. If present, the default dimension axes are created. Calls to SetNoms with other axes passed as the dtype parameter will have no effect if this command is used.

DefaultHits

DefaultHits

This command is used within a Startfeature…EndFeature block and is used to cause the hits specified in the hits parameter of the StartFeature command to be automatically generated.

DimFormat

DimFormat flags:=(Integer), heading1:=(Integer), heading2:=(Integer), heading3:=(Integer), heading4:=(Integer), heading5:=(Integer), heading6:=(Integer)

flags: PCD_HEADINGS, PCD_SYMBOLS. (Optional)

heading1: PCD_DEV, PCD_MAXMIN, PCD_MEAS, PCD_NOM, PCD_OUTTOL, PCD_TOL. (Optional)

heading2: PCD_DEV, PCD_MAXMIN, PCD_MEAS, PCD_NOM, PCD_OUTTOL, PCD_TOL. (Optional)

heading3: PCD_DEV, PCD_MAXMIN, PCD_MEAS, PCD_NOM, PCD_OUTTOL, PCD_TOL. (Optional)

heading4: PCD_DEV, PCD_MAXMIN, PCD_MEAS, PCD_NOM, PCD_OUTTOL, PCD_TOL. (Optional)

heading5: PCD_DEV, PCD_MAXMIN, PCD_MEAS, PCD_NOM, PCD_OUTTOL, PCD_TOL. (Optional)

heading6: PCD_DEV, PCD_MAXMIN, PCD_MEAS, PCD_NOM, PCD_OUTTOL, PCD_TOL. (Optional)

Functions E

EndAlign

EndAlign

This function must be called to end an alignment block.

EndDim

EndDim

EndDim takes no parameters, but must be called to finish off the dimension block.

EndFeature

EndFeature

This function ends a measured, constructed, or auto feature block. It must always be present as the last function call in a feature block.

EndGetFeatPoint

EndGetFeatPoint

Use this command to release the memory allocated for use by the StartGetFeatPoint and GetFeatPoint commands.

EndScan

EndScan

Call this when all of the other scan functions needed have been called.

The scan object is inserted in the command list with a call to this function.

EquateAlign

EquateAlign align1:=(String), align2:=(String)

Creates Equate alignment object

Align1: Name of alignment 1

Align2: Name of alignment 2

Functions F

Feature

Feature ID:=(String), pnt_tol:=(Double)

ID: ID string of the feature to add as an input for a best fit or iterative alignment.

pnt_tol: The point tolerance of the feature. Only used with best fit alignments.

This function must only be called after a call to BestFit2D, BestFit3D, or Iterate

Flatness

SHORT Flatness ID:=(String), out_zone:=(Double)

Return value: Non-zero if successfull. Zero if the object with the given ID string cannot be found.

ID: The string ID of the object to query.

out_zone: A reference to a double to hold the output zone.

Note: This function was added for the tutor translator, and should be used with caution.

Functions G

GapOnly

GapOnly tog:=(Integer)

tog: PCD_ON, PCD_OFF

GetDimData

GetDimData ID:= (String), buffer:= (DimData), dtype:= (Integer)

ID: The ID string of the dimension to access.

buffer: A record variable of type DimData in which to put the retrieved values. See below for a description of the DimData structure.

dtype: The type of data to retrieve for location or true position dimensions. Not needed for any other dimension type.

For location: PCD_X, PCD_Y, PCD_Z, PCD_D, PCD_R, PCD_A, PCD_T, PCD_PA, PCD_PR, PCD_V, PCD_L

For true position: PCD_X, PCD_Y, PCD_Z, PCD_DD, PCD_DF, PCD_PA, PCD_PR, PCD_TP

The definition of the DimData record type is as follows:

Type DimData

Nom As Double
Plus As Double
Minus As Double
Meas As Double
Max As Double
Min As Double
Dev As Double
Out As Double
Dev_Angle As Double
Bonus As Double

End Type

Note: The GetDimData function may not be called mid block.

Note: The GetDimData function should only be called on dimensions. It is up to the user to make sure that the ID string passed in does not belong to a feature or an alignment. For retrieving data from features, use GetFeatData.

GetDimOutTol

GetDimOutTol

Returns the number of features that are out of tolerance at the time that this command is executed

GetFeatData

GetFeatData ID:= (String), buffer:= (FeatData), dtype:= (Integer), xyz:=(Integer), ijk:= (Integer)

ID: The ID string of the feature to access.

buffer: A record variable of type FeatData in which to put the retrieved values. See below for a description of the FeatData structure.

dtype: The type of data to retrieve. Must be either PCD_MEAS or PCD_THEO.

xyz: Type of data to put in xyz. Allowed values are: PCD_CENTROID, PCD_BALLCENTER, PCD_STARTPOINT, PCD_ENDPOINT, PCD_MIDPOINT

ijk: Type of data to put in ijk. Allowed values are: PCD_VECTOR, PCD_SLOTVECTOR, PCD_SURFACEVECTOR, PCD_STARTPOINT, PCD_ENDPOINT, PCD_MIDPOINT

The definition of the FeatData record type is as follows:

```
Type FeatData

X As Double
Y As Double
Z As Double
I As Double
J As Double
K As Double
Diam As Double
Length As Double
Angle  As Double
Small_Diam As Double
Start_Angle As Double
End_Angle As Double
Start_Angle2 As Double
End_Angle2 As Double
F As Double
TP As Double
P1 As Double
P2 As Double
ID As String

End Type
```

Note: The GetFeatData function may not be called mid block.

Note: The GetFeatData function should only be called on measured, constructed, and auto features. It is up to the user to make sure that the ID string passed in does not belong to a dimension or an alignment. For retrieving data from dimensions, use GetDimData.

GetFeatID

Integer GetFeatID index:=(Integer), ID:=(String), type:=(Integer)

Index: The count backwards that should be used to find the next item with an id.

ID: This string is filled in with the id of the nth object back from the current point when n is specified by index

Type: type of object to be considered. PCD_FEATURE, PCD_ALIGNMENT, PCD_DIMENSION

GetFeatPoint

Integer GetFeatPoint buffer:= (PointData), index:= (Integer)

This function is called after a call to StartGetFeatPoint to retrieve the actual points.

Return value: The number of points available from the object.

buffer: A record variable of type PointData in which to put the retrieved point.

index: The 1 based index of the point to retrieve.

The definition of the PointData record type is as follows:

Type PointData

X As Double
Y As Double
Z As Double

End Type

GetFeature

Integer GetFeature ID:=(String)

Return value: The feature type of the object, or 0 if unsuccessful. Possible feature types are the following: PCD_F_POINT, PCD_F_CIRCLE, PCD_F_SPHERE, PCD_F_LINE, PCD_F_CONE, PCD_F_CYLINDER, PCD_F_PLANE, PCD_F_CURVE, PCD_F_SLOT, PCD_F_SET, PCD_F_ELLIPSE, PCD_F_SURFACE

ID: The string ID of the object to query.

Note: This function was added for the tutor translator, and should be used with caution.

GetPH9Status

SHORT GetPH9Status

Return value: Returns 1 if the probe has a PH9 and 0 if no PH9 is available.

GetProbeOffsets

GetProbeOffsets buffer:= (PointData)

buffer: A record of type pointdata that receives the values of the current xyz offset from the probe base.

GetProbeRadius

Double GetProbeRadius

Returns the current probe radius

GetProgramOption

Integer GetProgramOption opt:=(Integer)

Return value: returns 1 if the option is on and 0 if the option is off

Opt: The option's status that is being checked. PCD_AUTOTIPSELECT, PCD_AUTOPREHIT, PCD_ISONLINE, PCD_AUTOPROJREFPLANE, PCD_ISARMTYPECMM, PCD_HASINDEXPH9, PCD_HASINDEXROTTABLE, PCD_DISPSPEEDS, PCD_HASMANPH9, PCD_HASPHS, PCD_HASMANROTTABLE, PCD_HASROTTABLE, PCD_HASPH9, PCD_ENDKEY, PCD_EXTSHEETMETAL, PCD_FLYMODE, PCD_TABLEAVOIDANCE, PCD_USEDIMCOLORS

GetProgramValue

Double GetProgramValue opt:=(Integer)

Return value: returns the current value of the given option

Opt: The option's value that is being retrieved. PCD_ROTTABLEANGLE, PCD_PROBERADIUS, PCD_DIMPLACES, PCD_FLYRADIUS, PCD_AUTOTRIGDISTANCE, PCD_TABLETOL, PCD_MANRETRACT, PCD_MEASSCALE, PCD_PH9WARNDELTA, PCD_VALISYSERRTIMEOUT

GetTopMachineSpeed

DOUBLE GetTopMachineSpeed

Return value: Returns the top machine speed of the CMM.

GetType

SHORT GetType ID:=(String)

Return value: The type of the object, or 0 if unsuccessful. Possible types are any of the types passed to StartFeature or StartDim.

ID: The string ID of the object to query.

Note: This function was added for the tutor translator, and should be used with caution.

GetUnits

SHORT GetUnits

Return value: The units of the current part program. A value of 1 is returned when units are in inches and 0 when units are in millimeters.

Functions H

Hit

Hit x:=(Double), y:=(Double), z:=(Double), i:=(Double), j:=(Double), k:=(Double)

x,y,z, i,j,k: Theoretical x,y,z and approach vector of hit.

Note: This function is used for measured features only. It may be omitted on measured circles, cones, cylinders, spheres and points as these features generate default hits. However, if circular moves are required between each hit, the hit

function should be provided as a place holder. The parameters may be eliminated, in which case the default hit x, y, z and i, j, k are used.

Functions I

IgnoreMotionError

IgnoreMotionError tog:=(Integer)

tog: TRUE or FALSE. TRUE indicates that we wish to begin ignoring CMM motion **errors**. FALSE means we wish to stop ignoring CMM motion errors.

Iterate

Iterate num_inputs:= (Integer), pnt_tol:= (Double), flags:= (Integer)

num_inputs: The number of features to use to create the iterative alignment. Must be no more than six. There must be a corresponding number of calls to Feature before the call to EndAlign.

pnt_tol: The point tolerance.

flags: Any Ored combination of the following: PCD_BODY_AX, PCD_AV_ERROR, PCD_MEAS_ALL, PCD MEAS ALL ALWAYS.

Functions L

Level

Level axis:= (Integer), feat:= (String)

axis: Axis to level. PCD_ZPLUS, PCD_ZMINUS, PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS

feat: ID string of the feature to level to.

LoadProbe

LoadProbe probe:= (String)

probe: The probe to load.

Functions M

MaxMineAve

SHORT MaxMinAve ID:=(String), in_vector:=(PointData), out_max:=(Double), out_min:=(Double), out_ave:=(Double)

Return value: Non-zero if successfull. Zero if the object with the given ID string cannot be found.

ID: The string ID of the object to query.

in_vector: Input vector.

out_max: A reference to a double to hold the output maximum.

out_min: A reference to a double to hold the output minimum.

out_ave: A reference to a double to hold the output average.

Note: This function was added for the tutor translator, and should be used with caution.

Mode

Mode mode:= (Integer)

mode: PCD_DCC, PCD_MANUAL

Move

Move tog:= (Integer), x:= (Double), y:= (Double), z:= (Double), direction:=(Integer)

tog: PCD_CLEARPLANE, PCD_INCREMENT, PCD_CIRCULAR, PCD_POINT, PCD_ROTAB

x,y,z: Point or increment x,y,z if tog is PCD_INCREMENT or PCD_POINT.

x is angle if tog is PCD_ROTAB.

direction: PCD_CLOCKWISE, PCD_COUNTERCLOCKWISE, PCD_SHORTEST. Used only for PCD_ROTAB.

MoveSpeed

Movespeed percent:= (Double)

percent: Move speed of the probe as a percentage of the maximum probe speed.

Functions O

OpenCommConnection

Integer OpenCommConnection port:=(Integer), baud:=(Integer), parity:=(Integer), data:=(Integer), stop:=(Integer), flow:=(Integer)

Opens a connection to the specified comm port.

RETURN VALUE: 0 if successfull, -1 on error.

port: The comm port to open. Required.

baud: The baud rate at which to communicate with the port. Must be one of the following values: PCD_BAUD_110, PCD_BAUD_300, PCD_BAUD_600, PCD_BAUD_1200, PCD_BAUD_2400, PCD_BAUD_4800, PCD_BAUD_9600, PCD_BAUD_14400, PCD_BAUD_19200, PCD_BAUD_38400, PCD_BAUD_56000, PCD_BAUD_128000, PCD_BAUD_256000. Optional. Default is PCD_BAUD_9600.

parity: PCD_NOPARITY, PCD_EVENPARITY, PCD_ODDPARITY, PCD_MARKPARITY, PCD_SPACEPARITY. Optional. Default is PCD_NOPARITY.

data: Data bits. PCD_DATA8 or PCD_DATA7. Optional. Default is PCD_DATA8.

stop: Stop bits. PCD_ONESTOPBIT, PCD_ONE5STOPBITS, PCD_TWOSTOPBITS. Optional. Default is PCD_ONESTOPBIT.

flow: Flow control. PCD_DTRDSR, PCD_RTSCTS, PCD_XONXOFF. Optional. Default is PCD_RTSCTS.

Functions P

Prehit

Prehit distance:= (Double)

distance: New prehit distance.

ProbeComp

ProbeComp tog:= (Integer)

tog: PCD_ON, PCD_OFF. Turns probe compensation on or off.

PutFeatData

PutFeatData ID:= (String), buffer:= (FeatData), dtype:= (Integer),

xyz:= (Integer), ijk:= (Integer)

Parameters, allowed values, and limitations are identical to those of GetFeatData. The data currently in buffer is stored in the feature identified by the ID string.

Functions R

ReadCommBlock

Integer ReadCommBlock port:=(Integer), buffer:=(String), count:=(Integer)

Reads characters from the comm port specified.

RETURN VALUE: 0 if successfull, -1 on error.

port: The comm port from which to read. Required.

buffer: The string in which to put the read characters. Required.

count: The maximum number of characters to read from the port. Required.

RecallIn

RecallIn recallID:= (String)

recallIn: String ID of internal alignment to recall.

Note: This function does not need to be called within an alignment block.

RecallEx

RecallEx recallID:= (String)

recallID: String ID of external alignment to recall.

Note: This function does not need to be called within an alignment block.

Retract

Retract distance:= (Double)

distance: New retract distance.

RetroOnly

RetroOnly tog:=(Integer)

tog: PCD_ON, PCD_OFF

Rotate

Rotate axis1:= (Integer), feat:= (String), axis2:= (Integer)

axis1: Axis to rotate. PCD_ZPLUS, PCD_ZMINUS, PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS

feat: ID string of the feature to rotate to.

axis2: Axis to rotate about. PCD_ZPLUS, PCD_ZMINUS, PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS

RotateCircle

RotateCircle feat1:= (String), feat2:= (String), axis1:= (Integer), axis2:= (Integer)

feat1: ID string of circle.

feat2: ID string of second circle.

axis1: Axis to rotate. PCD_ZPLUS, PCD_ZMINUS, PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS

axis2: Axis to rotate about. PCD_ZPLUS, PCD_ZMINUS, PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS

RotateOffset

RotateOffset offset:= (Double), axis:= (Integer)

offset: Offset value.

axis: Axis to rotate about. PCD_ZPLUS, PCD_ZMINUS, PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS

Roundness

SHORT Roundness ID:=(String), out_zone:=(Double)

Return value: Non-zero if successfull. Zero if the object with the given ID string cannot be found.

ID: The string ID of the object to query.

out_zone: A reference to a double to hold the output zone.

Note: This function was added for the tutor translator, and should be used with caution.

Runout

SHORT Runout ID:=(String), in_datumxyz:=(PointData), in_datumijk:=(PointData), out_zone:=(Double)

Return value: Non-zero if successfull. Zero if the object with the given ID string cannot be found.

ID: The string ID of the object to query.

in_datumxyz: Input xyz.

in_datumijk: input ijk.

out_zone: A reference to a double to hold the output zone.

Note: This function was added for the tutor translator, and should be used with caution.

Functions S

SaveAlign

SaveAlign alignID:=(String), fname:=(String)

alignID: ID string of the alignment to save.

fname: File in which to save the alignment.

SetAutoParams

SetAutoParams init_hits:=(Integer), perm_hits:=(Integer), depth:=(Double), height:=(Double), wdth:=(Double), radius:=(Double), spacer:=(Double), indent:=(Double), thickness:=(Double), major:=(Double), minor:=(Double)

init_hits: sample hits for initial execution

perm_hits: sample hits for subsequent executions

depth: sheet metal measuring depth

height: height of stud for a sheet metal circle, sheet metal cylinder or sheet metal ellipse; or the long length of a slot

width: short width of a slot

radius: corner radius of a square slot

spacer: distance from the nominal feature or nominal feature edge where sample hits are taken.

indent: like spacer but in a different direction. Used in edge points, corner points, and angle points

thickness: thickness of the sheetmetal

major: major axis of ellipse

minor: minor axis of ellipse

Note: This function is used for auto features only.

SetAutoVector

SetAutoVector index:=(Integer), i:=(Double), j:=(Double), k:=(Double)

index: Which vector to set. Can be any of the following: PCD_VECTOR1, PCD_VECTOR2, PCD_VECTOR3, PCD_PUNCH_VECTOR, PCD_PIN_VECTOR, PCD_ANGLE_VECTOR, PCD_REPORT_VECTOR, PCD_EDGE_REPORT_VECTOR, PCD_SURF_REPORT_VECTOR, PCD_MEASURE_VECTOR, PCD_UPDATE_VECTOR, PCD_VECTOR1 is normally not needed as the first ijk values are set with a call to SetTheos.

i,j,k: The parameters of the vector.

Note: This function is used for auto features only.

SetNoms

SetNoms nom:=(Double), plus_tol:=(Double), minus_tol:=(Double), dtype:=(Integer), multiplier:=(Double)

nom: Double value indicating nominal. May be omitted when no nominal is needed.

plus_tol: Double value indicating plus tolerance.

minus_tol: Double value indicating minus tolerance. May be omitted when no minus tolerance is needed.

dtype: For Location only: PCD_X, PCD_Y, PCD_Z, PCD_D, PCD_R, PCD_A, PCD_T, PCD_PA, PCD_PR, PCD_V, PCD_L, PCD_PX, PCD_PY, PCD_PZ, PCD_PD, PCD_PT

For True Position only: PCD_X, PCD_Y, PCD_Z, PCD_DD, PCD_DF, PCD_PA, PCD_PR, PCD_TP

IMPORTANT: This parameter should be omitted for all other dimension types.

multiplier: Arrow multiplier for dimension. Optional. Defaults to 1.0.

When the DefaultAxes command is not used for dimensions of type location and true position, an axis corresponding to the dtype parameter is added for every call to SetNoms.

SetPrintOptions

SetPrintOptions location:=(Integer), draft:=(Integer), filemode:=(Integer), nextnum:=(Integer)

Location: location of output. Can be PCD_OFF, PCD_PRINTER, or PCD_FILE

Draft: mode of output to printer. PCD_ON or PCD_OFF

Filemode: naming mode for output file. PCD_APPEND, PCD_NEWFILE, PCD_OVERWRITE, PCD_AUTO

NextNum: used with PCD_AUTO mode naming scheme for output file

SetProgramOption

SetProgramOption opt:=(Integer), tog:=(Integer)

Opt: Program option to set: PCD_AUTOTIPSELECT, PCD_AUTOPREHIT, PCD_AUTOPROJREFPLANE, PCD_DISPSPEEDS, PCD_ENDKEY, PCD_EXTSHEETMETAL, PCD_FLYMODE, PCD_TABLEAVOIDANCE, PCD_USEDIMCOLORS

Tog: Specifies whether option should be turned on or off. PCD_ON or PCD_OFF

SetProgramValue

SetProgramValue opt:=(Integer), val:=(Double)

Opt: Program value to set: PCD_PROBERADIUS, PCD_DIMPLACES, PCD_FLYRADIUS, PCD_AUTOTRIGDISTANCE, PCD_TABLETOL, PCD_MANRETRACT, PCD_MEASSCALE, PCD_PH9WARNDELTA, PCD_VALISYSERRTIMEOUT

Val: New value for program value being set.

SetReportOptions

SetReportOptions opt:=(Integer)

Opt: Any of the combined flags can be used to turn on or off the reporting object types: PCD_FEATURES, PCD_ALIGNMENTS, PCD_MOVES, PCD_COMMENTS, PCD_DIMENSIONS, PCD_HITS, PCD_OUTTOL_ONLY

SetRmeasMode

SetRmeasMode mode:=(Integer)

Mode: The mode to be used for auto features using the RMEAS functionality. PCD_RELATIVE or PCD_ABSOLUTE

SetSlaveMode

SetSlaveMode tog:=(Integer)

Tog: Turns slave mode off or on for all subsequent created commands. PCD_ON or PCD_OFF

SetScanHitParams

SetScanHitParams htype:=(Integer), init_hits:=(Integer), perm_hits:=(Integer), spacer:=(Double), depth:=(Double), indent:=(Double), flags:=(Integer)

Note: This function is only used for DCC scans and should not be called for manual scans.

htype: Type of hits to use. PCD_VECTORHIT, PCD_SURFACEHIT, PCD_EDGEHIT, PCD_ANGLEHIT.

init_hits: Number of init sample hits to use. Optional.

perm_hits: Number of permanent hits. Optional.

spacer: Spacer value. Optional.

depth: Depth value. Optional.

indent: Indent value. Optional.

flags: For now, just PCD_EXTERIOR or PCD_INTERIOR. Default is PCD_EXTERIOR. Optional.

SetScanHitVectors

SetScanHitVectors vector:=(Integer), i:=(Double), j:=(Double), k:=(Double)

Note: This function is only used for DCC scans.

vector: Hit vector to set. PCD_TOP_SURFACE, PCD_SIDE_SURFACE, PCD_BOUNDARY_PLANE.

i,j,k: Values to set.

SetScanParams

SetScanParams incr:=(Double), axis:=(Integer), max_incr:=(Double), min_incr:=(Double), max_angle:=(Double), in_angle:=(Double), delta:=(Double), distance:=(Double), incr2:=(Double), axis2:=(Integer), surf_thickness:=(Double)

incr: Increment value for LINE, BODY, and CUTAXIS scan techniques. Optional.

axis: Axis for BODY and CUTAXIS scan techniques. PCD_XAXIS, PCD_YAXIS, PCD_ZAXIS. Optional.

max_incr, min_incr, max_angle, min_angle: For VARIABLE scan techniques. Optional.

delta: Distance delta for FIXED_DELTA scans, time delta for VARIABLE_DELTA and TIME_DELTA scans. Optional.

distance: Drop point distance for VARIABLE_DELTA scan, distance for CUTAXIS scan. Optional.

incr2: Increment value in second direction for a patch scan. Optional.

axis2: Second axis value for a patch scan (BODY scan technique only). Optional.

surf_thickness: Surface thickness used to offset centroid calculation if necessary. Optional.

SetScanVectors

SetScanVectors vector:=(Integer), i:=(Double), j:=(Double), k:=(Double)

vector: Vector to set. PCD_CUTVECTOR, PCD_INITTOUCH, PCD_INITDIR, PCD_ROWEND_APPROACH.

i,j,k: Values to set.

SetTheos

SetTheos x:=(Double), y:=(Double), z:=(Double), i:=(Double), j:=(Double), k:=(Double), diam:=(Double), length:=(Double), angle:=(Double), small_diam:=(Double), start_angle:=(Double), end_angle:=(Double), start_angle2:=(Double), end_angle2:=(Double)

Note: A call to SetTheos is mandatory for all measured features.

x,y,z, i,j,k: On a bound line, (i,j,k) is the ending point.

diam: Diameter of a circle, cylinder, or sphere. Big diameter of a cone.

length: Length of a cylinder.

angle: Angle of a cone.

small_diam: Small diameter of a cone.

start_angle, **end_angle**: Starting and ending angles for circles, cylinders, and spheres.

start_angle2, **end_angle2**: Second starting and ending angles for spheres.

ShowXYZWindow

ShowXYZWindow show:=(Integer)

Show: Show or hides the probe position window. PCD_ON or PCD_OFF

Sleep

Sleep seconds:=(Single)

Pauses execution for the specified number of seconds after the previous feature has finished executing.

Note: Sleep calls the Wait function to ensure that the sleeping does not begin before all previous features have been executed.

seconds: The number of seconds to pause. Any precision beyond milliseconds is ignored.

StartAlign

StartAlign ID:= (String), recallID:= (String)

ID: ID string of the alignment to create.

recallID: ID string of the alignment to recall.

StartDim

StartDim dtype:=(Integer), ID:=(String), feat1:=(String), feat2:=(String), feat3:=(String), axis:=(Integer), length:=(Double), angle:=(Double), flags:=(Integer)

dtype: DIM_LOCATION, DIM_STRAIGHTNESS, DIM_ROUNDNESS, DIM_FLATNESS, DIM_PERPENDICULARITY, DIM_PARALLELISM, DIM_PROFILE, DIM_3D_DISTANCE, DIM_2D_DISTANCE, DIM_3D_ANGLE, DIM_2D_ANGLE, DIM_RUNOUT, DIM_CONCENTRICITY, DIM_ANGULARITY, DIM_KEYIN, DIM_TRUE_POSITION

ID: ID string of the dimension to create

feat1: ID string of the Of Feature or From Feature

feat2: ID string of the To Feature

feat3: ID string of the third feature, if any

axis: PCD_XAXIS, PCD_YAXIS, PCD_ZAXIS. Only needed for dimensions using an axis or workplane.

length: Extended length for angularity, profile, perpendicularity, or parallelism.

angle: Angle for angularity.

flags: PCD_ADD_RADIUS, PCD_SUB_RADIUS, PCD_NO_RADIUS, PCD_PAR_TO, PCD_PERP_TO. Some of these values may be Ored together.

Example: PCD_ADD_RADIUS Or PCD_PAR_TO) True Position dimensions can take one of the following flags as well:

PCD_RFS_RFS, PCD_RFS_MMC, PCD_RFS_LMC, PCD_MMC_RFS, PCD_MMC_MMC, PCD_MMC_LMC, PCD_LMC_RFS, PCD_LMC_MMC, PCD_LMC_LMC.

The datum computation type comes first. For example, PCD_RFS_LMC specifies RFS for the datum and LMC for the feature.

StartFeature

StartFeature ftype:=(Integer), ID:=(string), hits:=(Integer), inputs:=(Integer), flags:=(Long)

ftype: MEAS_POINT, MEAS_CIRCLE, MEAS_SPHERE, MEAS_LINE, MEAS_CONE, MEAS_CYLINDER, MEAS_PLANE, MEAS_SET, READ_POINT,CONST_ORIG_POINT, CONST_OFF_POINT, CONST_PROJ_POINT, CONST_MID_POINT, CONST_DROP_POINT, CONST_PIERCE_POINT, CONST_INT_POINT, CONST_CAST_POINT, CONST_CORNER_POINT, CONST_BFRE_CIRCLE, CONST_BF_CIRCLE, CONST_PROJ_CIRCLE, CONST_REV_CIRCLE, CONST_CONE_CIRCLE, CONST_CAST_CIRCLE, CONST_INT_CIRCLE, CONST_BFRE_SPHERE, CONST_BF_SPHERE, CONST_PROJ_SPHERE, CONST_REV_SPHERE, CONST_CAST_SPHERE, CONST_BFRE_LINE, CONST_BF_LINE, CONST_PROJ_LINE, CONST_REV_LINE, CONST_MID_LINE, CONST_CAST_LINE, CONST_INT_LINE, CONST_OFF_LINE, CONST_ALN_LINE, CONST_PRTO_LINE, CONST_PLTO_LINE, CONST_BFRE_CONE, CONST_BF_CONE, CONST_PROJ_CONE, CONST_REV_CONE, CONST_CAST_CONE, CONST_BFRE_CYLINDER, CONST_BF_CYLINDER, CONST_PROJ_CYLINDER, CONST_REV_CYLINDER, CONST_CAST_CYLINDER,

CONST_BFRE_PLANE, CONST_BF_PLANE, CONST_REV_PLANE, CONST_MID_PLANE, CONST_CAST_PLANE, CONST_OFF_PLANE, CONST_ALN_PLANE, CONST_PRTO_PLANE, CONST_PLTO_PLANE,CONST_HIPNT_PLANE, CONST_SET, AUTO_VECTOR_HIT, AUTO_SURFACE_HIT, AUTO_EDGE_HIT, AUTO_ANGLE_HIT, AUTO_CORNER_HIT, AUTO_CIRCLE, AUTO_SPHERE, AUTO_CYLINDER, AUTO_ROUND_SLOT, AUTO_SQUARE_SLOT, AUTO_ELLIPSE, PCD_CURVE

ID: ID string of the feature

hits: Measured and auto features only. The number of hits to take to measure the feature.

inputs: Constructed features only. The number of features that will be used in the construction. There must be a corresponding number of calls to AddFeature before the EndFeature statement.

flags: Any of the following flags Ored together:

PCD_POLR: Values are reported in cylindrical coordinates. Should not be ored with PCD_RECT.

PCD_RECT: Values are in rectangular coordinates. Should not be ored with PCD_POLR. Default.

PCD_BND: Bound line. Should not be ored with PCD_UNBND.

PCD_UNBND: Unbound line. Should not be ored with PCD_BND. Default.

PCD_IN: Inside circle, sphere, cone, or cylinder. Should not be ored with PCD_OUT.

PCD_OUT: Outside circle, sphere, cone, or cylinder. Should not be ored with PCD_IN. Default.

PCD_LENGTH: Cone reports its length as opposed to angle. Do not or with PCD_ANGLE. Default.

PCD_ANGLE: Cone reports its angle as opposed to length. Do not or with PCD_LENGTH.

PCD_EXTERIOR: Exterior angle hit. Only used for auto angle hits. Do not or with PCD_INTERIOR. Default.

PCD_INTERIOR: Interior angle hit. Only used for auto angle hits. Do not or with PCD_EXTERIOR.

PCD_LINE_3D: 3D line. Used only for best fit lines. Default is a 2D line.

PCD_RECALC_NOMS: Indicates that the theoretical values should be recalculated based on the theoretical hit values.

workplane axis: A workplane/axis flag is only used with alignment lines and planes. Possible flag values are the following: PCD_FRONT, PCD_BACK, PCD_LEFT, PCD_RIGHT, PCD_TOP, PCD_BOTTOM, PCD_ZPLUS, PCD_ZMINUS, PCD_XPLUS, PCD_XMINUS, PCD_YPLUS, PCD_YMINUS, PCD_ZAXIS, PCD_XAXIS, PCD_YAXIS.

PCD_MEASURE_SURFACE: Sets measure order. For auto edge points only. Default.

PCD_MEASURE_EDGE: Sets measure order. For auto edge points only.

PCD_MEASURE_BOTH: Sets measure order. For auto edge points only.

PCD_HEM: For auto edge points only. Should not be ored with PCD_TRIM.

PCD_TRIM: For auto edge points only. Should not be ored with PCD_HEM. Default.

PCD_PIN: For auto circles, cylinders, ellipses, and slots. Do not or with PCD_NORM.

PCD_NORM: For auto circles, cylinders, ellipses, and slots. Do not or with PCD_PIN. Default.

PCD_READPOS: Turn read position on. For auto circles, cylinders, ellipses, and slots. Defaults to off.

PCD_AUTOMOVE: Causes move points to be automatically generated for auto features.

PCD_FINDHOLE: For Auto Circles. Automatic finding of holes.

PCD_MEASURE_WIDTH: Flag for Auto Square Slots

StartGetFeatPoint

Integer StartGetFeatPoint ID:= (String), dtype:= (Integer), xyz:= (Integer)

This function is used to retrieve the hit or input data from constructed, measured, and auto features, as well as the hit data for scans. To retrieve the actual points, subsequent calls to GetFeatPoint must be made. When all of the needed point values have been retrieved, a call to EndGetFeatPoint must be made to free the memory allocated for the points.

Return value: The number of points retrieved from the object.

ID: The ID string of the feature to access.

dtype: The type of data to retrieve. Must be either PCD_MEAS or PCD_THEO.

xyz: Type of data to put in xyz. Allowed values are: PCD_BALLCENTER, PCD_CENTROID, PCD_VECTOR

Note: The StartGetFeatPoint function may not be called mid block.

StartScan

StartScan ID:=(String), mode:=(Integer), stype:=(Integer), dir1:=(Integer), dir2:=(Integer), technique:=(Integer), num_bnd_pnts:=(Integer), flags:=(Integer)

ID: ID string of the scan.

mode: Mode of the scan. Must be PCD_DCC or PCD_MANUAL.

stype: Type of scan. For DCC scans, stype must be PCD_LINEAR_OPEN, PCD_LINEAR_CLOSED, PCD_SECTION, PCD_PERIMETER, or PCD_PATCH. For manual scans, stype must be PCD_MANUALTTP or PCD_HPROBE.

dir1: Only used for DCC scans. PCD_LINE, PCD_BODY, PCD_VARIABLE. Optional.

dir2: Only used for DCC patch scans. PCD_LINE, PCD_BODY. Optional.

technique: Only used for manual scans. PCD_FIXED_DELTA, PCD_VARIABLE_DELTA, PCD_TIME_DELTA, PCD_CUTAXIS. Optional.

num_bnd_pnts: Number of points defining the boundary for the scan. Only used for DCC patch scans. Optional.

flags: Special scan flags. PCD_SINGLEPOINT, PCD_MASTERMODE, PCD_RELEARNMODE, PCD_AUTOCLEARPLANE, PCD_HITNOTDISPLAYED. Any of these values may be Ored together. Optional.

Straitness

SHORT Straitness ID:=(String), Put_zone:=(Double)

Return value: Non-zero if successfull. Zero if the object with the given ID string cannot be found.

ID: The string ID of the object to query.

out_zone: A reference to a double to hold the output zone.

Note: This function was added for the tutor translator, and should be used with caution.

Stats

Stats tog:=(Integer), dbase_dir:=(String), read_lock:=(Integer), write_lock:=(Integer), mem_page:=(Integer), flags:=(Integer)

tog: Indicates whether stats is on or off. PCD_ON or PCD_OFF.

dbase_dir: Database directory. Optional.

read_lock: Optional.

write_lock: Optional.

mem_page: Optional.

flags: PCD_USE_FEAT_NAME, PCD_USE_DIM_NAME, PCD_DO_CONTROL_CALCS. Optional.

Functions T

Tip

Tip tip:= (String)

tip: The tip to load.

Touchspeed

Touchspeed percent:= (Double)

percent: Touchspeed of the probe as a percentage of the maximum probe speed.

Trace

Trace field:=(String)

field: Name of the field to trace.

Translate

Translate axis:= (Integer), feat:= (String)

axis: Axis to translate. PCD_ZAXIS, PCD_XAXIS, PCD_YAXIS

feat: ID string of feature to translate to.

TranslateOffset

TranslateOffset offset:= (Double), axis:= (Integer)

offset: Value of offset.

axis: PCD_ZAXIS, PCD_XAXIS, PCD_YAXIS

Functions W

Wait

Wait

Waits until all preceding commands have been executed. The basic script creates commands and places them on the execute list more rapidly than the commands are executed. In a script it is often useful to pop up a dialog box for input after a certain series of commands has been executed. The script commands may complete long before the actual commands have been executed. The Wait command is useful to prevent the dialog box from popping up prematurely.

Workplane

Integer Workplane plane:= (Integer)

Return value: The previous workplane.

plane: PCD_TOP, PCD_BOTTOM, PCD_FRONT, PCD_BACK, PCD_LEFT, PCD_RIGHT.

Optional. If not provided, the current workplane is returned but no new workplane is set.

WriteCommBlock

Integer WriteCommBlock port:=(Integer), buffer:=(String), count:=(Integer)

Writes characters to the specified comm port.

RETURN VALUE: 0 if successfull, -1 on error.

port: The comm port to write to. Required.

buffer: The string to write to the port. Required.

count: The number of characters to write to the port. Optional. Defaults to the length of the buffer string.

Integer CloseCommConnection port:=(Integer)

Closes the connection to the specified comm port.

RETURN VALUE: 0 if successfull, -1 on error.

port: The comm port to close. Required.

Glossary of Terms

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Y