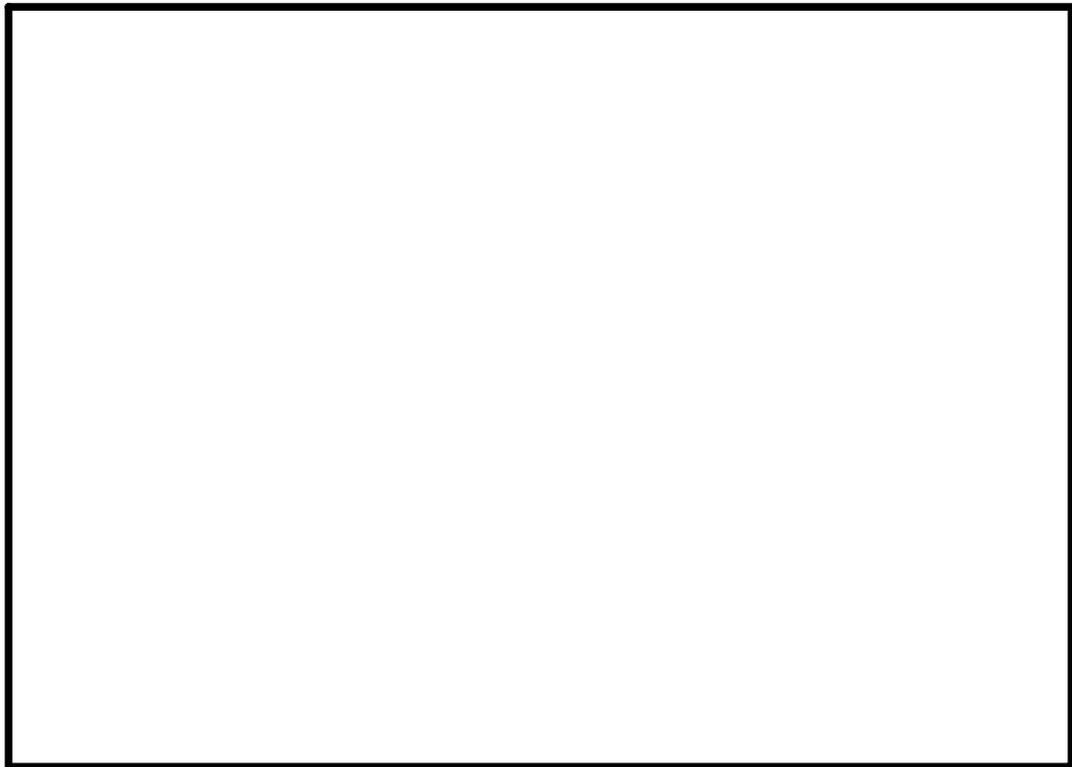

PC-DMIS 4.0

BASIC Language Reference Manual



By Wilcox Associates, Inc.

Copyright © 1999-2001, 2002-2005, 2006 Hexagon Metrology and Wilcox Associates Incorporated. All rights reserved.

PC-DMIS, Direct CAD, Tutor for Windows, Remote Panel Application, DataPage, and Micro Measure IV are either registered trademarks or trademarks of Hexagon Metrology and Wilcox Associates, Incorporated.

SPC-Light is a trademark of Lighthouse.

HyperView is a trademark of Dundas Software Limited and HyperCube Incorporated.

Orbix 3 is a trademark of IONA Technologies.

I-DEAS and Unigraphics are either trademarks or registered trademarks of EDS.

Pro/ENGINEER is a registered trademark of PTC.

CATIA is either a trademark or registered trademark of Dassault Systemes and IBM Corporation.

ACIS is either a trademark or registered trademark of Spatial and Dassault Systemes.

3DxWare is either a trademark or registered trademark of 3Dconnexion.

lp_solve is a free software package licensed and used under the GNU LGPL.

PC-DMIS for Windows version 4.0 and beyond uses a free, open source package called lp_solve (or lpsolve) that is distributed under the GNU lesser general public license (LGPL).

lpsolve citation data

Description : Open source (Mixed-Integer) Linear Programming system
Language : Multi-platform, pure ANSI C / POSIX source code, Lex/Yacc based parsing
Official name : lp_solve (alternatively lpsolve)
Release data : Version 5.1.0.0 dated 1 May 2004
Co-developers : Michel Berkelaar, Kjell Eikland, Peter Notebaert
Licence terms : GNU LGPL (Lesser General Public Licence)
Citation policy : General references as per LGPL
Module specific references as specified therein
You can get this package from:
http://groups.yahoo.com/group/lp_solve/

Table of Contents

PC-DMIS BASIC LANGUAGE REFERENCE.....	1
INTRODUCTION	1
WHAT IS CYPRESS ENABLE?	1
ORGANIZATION OF THE MANUAL	1
BASIC SCRIPT EDITOR.....	3
INTRODUCTION	3
FILE MENU.....	3
EDIT MENU	4
VIEW MENU	7
RUN MENU.....	7
HELP MENU	7
BASIC SCRIPT TOOLBAR.....	8
CYPRESS ENABLE SCRIPTING LANGUAGE ELEMENTS.....	11
INTRODUCTION	11
<i>Numbers.....</i>	<i>12</i>
<i>Variable and Constant Names.....</i>	<i>12</i>
<i>Variable Types</i>	<i>12</i>
<i>Other Data Types</i>	<i>13</i>
<i>Control Structures.....</i>	<i>14</i>
<i>Subroutines and Functions.....</i>	<i>17</i>
<i>Calling Procedures in DLLs</i>	<i>21</i>
<i>File Input/Output.....</i>	<i>22</i>
<i>Arrays.....</i>	<i>24</i>

<i>User Defined Types</i>	27
<i>Dialog Support</i>	28
<i>Statements and Functions Used in Dialog Functions</i>	38
<i>OLE Automation</i>	43
<i>Accessing an Object</i>	44
<i>What is an OLE Object?</i>	44
<i>OLE Fundamentals</i>	45
<i>OLE Automation and Microsoft Word Example:</i>	46
<i>Making Applications Work Together</i>	47
<i>The Registration database</i>	47
SCRIPTING LANGUAGE OVERVIEW	49
INTRODUCTION	49
<i>Quick Reference of Functions and Statements Available</i>	49
LANGUAGE REFERENCE A – Z	55
INTRODUCTION	55
<i>Abs Function</i>	55
<i>AppActivate Statement</i>	56
<i>Asc Function</i>	56
<i>Atn Function</i>	57
<i>Beep Statement</i>	58
<i>Call Statement</i>	59
<i>CBool Function</i>	60
<i>CDate Function</i>	60
<i>CDBl Function</i>	61
<i>ChDir Statement</i>	62

<i>ChDrive Statement</i>	63
<i>CheckBox</i>	63
<i>Choose Function</i>	64
<i>Chr Function</i>	65
<i>CInt Function</i>	65
<i>CLng Function</i>	66
<i>Close Statement</i>	67
<i>Const Statement</i>	68
<i>Cos Function</i>	69
<i>CreateObject Function</i>	70
<i>CSng Function</i>	71
<i>CStr Function</i>	72
<i>CurDir Function</i>	72
<i>CVar Function</i>	73
<i>Date Function</i>	74
<i>DateSerial Function</i>	75
<i>DateValue Function</i>	76
<i>Day Function</i>	76
<i>Declare Statement</i>	77
<i>Dialog, Dialog Function</i>	79
<i>Dim Statement</i>	81
<i>Dir Function</i>	81
<i>DlgEnable Statement</i>	82
<i>DlgText Statement</i>	84
<i>DlgVisible Statement</i>	84

<i>Do...Loop Statement</i>	85
<i>End Statement</i>	86
<i>EOF Function</i>	87
<i>Erase Statement</i>	88
<i>Exit Statement</i>	88
<i>Exp</i>	89
<i>FileCopy Function</i>	90
<i>FileLen Function</i>	90
<i>Fix Function</i>	91
<i>For each ... Next Statement</i>	91
<i>For...Next Statement</i>	92
<i>Format Function</i>	93
<i>FreeFile Function</i>	102
<i>Function Statement</i>	103
<i>Get Statement</i>	104
<i>Get Object Function</i>	105
<i>Global Statement</i>	105
<i>GoTo Statement</i>	106
<i>Hex</i>	107
<i>Hour Function</i>	108
<i>HTMLDialog</i>	109
<i>If...Then...Else Statement</i>	109
<i>Input # Statement</i>	111
<i>Input Function</i>	112
<i>InputBox Function</i>	112

<i>InStr</i>	113
<i>Int Function</i>	114
<i>IsArray Function</i>	114
<i>IsDate</i>	115
<i>IsEmpty</i>	115
<i>IsNull</i>	116
<i>IsNumeric</i>	116
<i>IsObject Function</i>	117
<i>Kill Statement</i>	118
<i>LBound Function</i>	119
<i>LCase, Function</i>	119
<i>Left</i>	120
<i>Len</i>	121
<i>Let Statement</i>	121
<i>Line Input # Statement</i>	122
<i>LOF</i>	123
<i>Log</i>	123
<i>Mid Function</i>	124
<i>Minute Function</i>	125
<i>MkDir</i>	126
<i>Month Function</i>	127
<i>MsgBox Function MsgBox Statement</i>	128
<i>Name Statement</i>	130
<i>Now Function</i>	130
<i>Oct Function</i>	130

<i>OKButton</i>	131
<i>On Error</i>	132
<i>Open Statement</i>	136
<i>Option Base Statement</i>	138
<i>Option Explicit Statement</i>	138
<i>Print Method</i>	139
<i>Print # Statement</i>	139
<i>Randomize Statement</i>	142
<i>ReDim Statement</i>	142
<i>Rem Statement</i>	143
<i>Right Function</i>	143
<i>Rmdir Statement</i>	144
<i>Rnd Function</i>	145
<i>Second Function</i>	146
<i>Seek Function</i>	147
<i>Seek Statement</i>	148
<i>Select Case Statement</i>	149
<i>SendKeys Function</i>	150
<i>Set Statement</i>	151
<i>Shell Function</i>	152
<i>Sin Function</i>	153
<i>Space Function</i>	154
<i>Sqr Function</i>	154
<i>Static Statement</i>	155
<i>Stop Statement</i>	156

<i>Str Function</i>	157
<i>StrComp Function</i>	157
<i>String Function</i>	158
<i>Sub Statement</i>	159
<i>Tan Function</i>	160
<i>Text Statement</i>	160
<i>TextBox Statement</i>	161
<i>Time Function</i>	162
<i>Timer Event</i>	162
<i>TimeSerial - Function</i>	163
<i>TimeValue - Function</i>	163
<i>Trim, LTrim, RTrim Functions</i>	164
<i>Type Statement</i>	165
<i>UBound Function</i>	167
<i>UCase Function</i>	168
<i>Val</i>	168
<i>VarType</i>	169
<i>Weekday Function</i>	169
<i>While...Wend Statement</i>	170
<i>With Statement</i>	171
<i>Write # - Statement</i>	172
<i>Year Function</i>	173
AUTOMATION	175
INTRODUCTION	175
<i>Accessing an Object's Properties, Methods and Events</i>	177

<i>Active Tip Object Overview</i>	182
<i>AlignCommand Object Overview</i>	184
<i>Application Object Overview</i>	190
<i>Application Object Events Object Overview</i>	202
<i>Application Settings Object Overview</i>	207
<i>Array Index Object Overview</i>	208
<i>Attach Object Overview</i>	210
<i>Autotrigger Object Overview</i>	210
<i>BasicScanCommand Object Overview</i>	211
<i>CadModel Object Overview:</i>	231
<i>CadWindow Object Overview:</i>	233
<i>CadWindows Object Overview</i>	235
<i>Calibration Object Overview</i>	236
<i>Command Object Overview</i>	236
<i>Commands Object Overview</i>	263
<i>Comment Object Overview</i>	267
<i>ControlPoint Object Overview</i>	269
<i>DataType Object Overview</i>	270
<i>DataTypes Object Overview</i>	271
<i>DimData Object Overview</i>	272
<i>DimensionCommand Object Overview</i>	274
<i>Dimension Format Object Overview</i>	280
<i>Dimension Information Object Overview</i>	282
<i>Display Metafile Object Overview</i>	288
<i>DmisDialog Object Overview</i>	288

<i>DmisMatrix Object Overview</i>	289
<i>EditWindow Object Overview</i>	294
<i>ExecutedCommands Object Overview</i>	299
<i>ExternalCommand Object Overview</i>	302
<i>FeatCommand Object Overview</i>	302
<i>FeatData Object Overview</i>	333
<i>File IO Object Overview</i>	336
<i>FlowControlCommand Object Overview</i>	338
<i>FPanel Object Overview</i>	348
<i>Leapfrog Object Overview</i>	349
<i>Leitz Motion Object Overview</i>	350
<i>Load Machine Object Overview</i>	351
<i>Load Probes Object Overview</i>	351
<i>Machine Object Overview</i>	352
<i>Machines Object Overview</i>	353
<i>MasterSlaveDlg Object Overview</i>	354
<i>ModalCommand Object Overview</i>	356
<i>MoveCommand Object Overview</i>	359
<i>Opt Motion Object Overview</i>	360
<i>OptProbe Object Overview</i>	361
<i>PartProgram Object Overview</i>	363
<i>PartProgram Settings Object Overview</i>	375
<i>PartPrograms Object Overview</i>	376
<i>PointData Object Overview</i>	379
<i>Probe Object Overview</i>	380

<i>Probes Object Overview</i>	384
<i>QualificationSettings Object Overview</i>	385
<i>ReportData Object Overview</i>	389
<i>ReportControls Object Overview</i>	392
<i>ReportTemplates Object Overview</i>	395
<i>ReportTemplate Object Overview</i>	396
<i>ReportWindow Object Overview</i>	398
<i>ScanCommand Object Overview</i>	399
<i>Section Object Overview</i>	415
<i>Sections Object Overview</i>	415
<i>Statistics Object Overview</i>	417
<i>Temperature Compensation Object Overview</i>	420
<i>Tip Object Overview</i>	421
<i>Tips Object Overview</i>	424
<i>Tool Object Overview</i>	425
<i>Tools Object Overview</i>	426
<i>Tracefield Object Overview</i>	428
<i>Variable Object Overview</i>	428
OLD PC-DMIS BASIC FUNCTIONS	430
INTRODUCTION	430
<i>Functions A</i>	430
<i>Functions B</i>	432
<i>Functions C</i>	432
<i>Functions D</i>	434
<i>Functions E</i>	435

<i>Functions F</i>	436
<i>Functions G</i>	436
<i>Functions H</i>	441
<i>Functions I</i>	441
<i>Functions L</i>	442
<i>Functions M</i>	442
<i>Functions O</i>	443
<i>Functions P</i>	443
<i>Functions R</i>	444
<i>Functions S</i>	446
<i>Functions T</i>	455
<i>Functions W</i>	456
INDEX	459

PC-DMIS Basic Language Reference

Introduction

While PC-DMIS for Windows contains a myriad of valuable options and features to aid you in your part measurements, there may be times when you want greater customizability to meet specific needs. Maybe you want the capability to globally change a particular value inside the Edit window, or maybe you want to export statistical data to a specific application. Rather than waiting for an item on your wish list to be coded into a future version of PC-DMIS, PC-DMIS allows you to create your own BASIC scripts and run them inside PC-DMIS. In fact, using PC-DMIS's comprehensive list of automation commands, properties and methods, you can run PC-DMIS entirely from a custom-built third party application.

What is Cypress Enable?

PC-DMIS comes with a BASIC Script Editor that uses the Cypress Enable Scripting Language, a powerful subset of the BASIC language. If you have a working knowledge of BASIC, the tools given in this manual will be invaluable to creating your own mini-applications that work in conjunction with PC-DMIS.

Organization of the Manual

This manual contains the following chapters:

- **Basic Script Editor** – this discusses how to open and use the Basic Script Editor from within PC-DMIS to create and compile your BASIC scripts.
- **Cypress Enable Scripting Language Elements** – this discusses the language elements used to create BASIC scripts using Cypress Enable.
- **Scripting Language Overview** – this contains quick reference charts on functions, statements, data types, operators, reserved words, and precedence.
- **Language Reference A-Z** – this contains a full fledged language reference of BASIC code that you can use.
- **Automation** – this contains all the properties and methods associated with every PC-DMIS object that you can use via your BASIC scripts.
- **Old PC-DMIS Basic functions** – this contains an alphabetical reference of the older BASIC functions used in previous versions of PC-DMIS. They are provided here for backward compatibility.

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's **Standard** 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

Tip: You do not have to use the BASIC Script Editor to create your scripts. Many users prefer to use the Visual BASIC interface that comes with Microsoft Office applications such as Visual BASIC for Microsoft Excel or Microsoft Word. These applications contain advanced debugging tools and visual aids to help you create your scripts. See "Using the Object Browser in Other Editors" in the "Automation" chapter.

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 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.

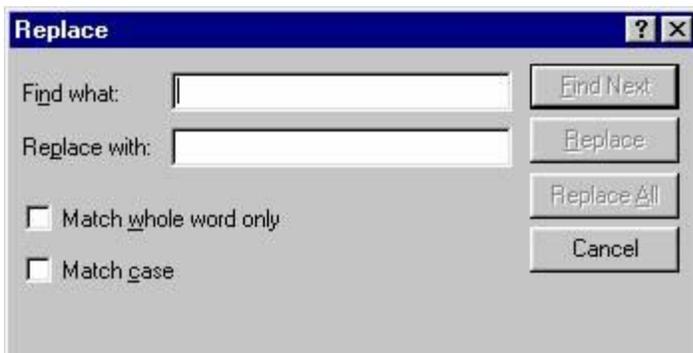
- If you choose the **Match whole word only** check box the dialog will display only those words that match the entire word.
- If you choose the **Match Case** check box, then the dialog box will display only those terms that match the case (Uppercase or Lowercase) that you used in the **Find what**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

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 box 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** box.

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** box.

Replace All

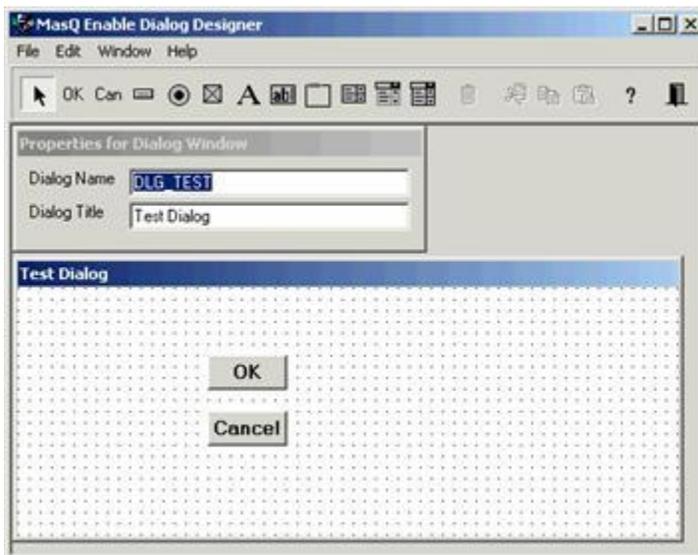
The **Replace All** button allows you to replace all instances in the Basic Script Editor that meet the search qualifications with what is in the **Replace with** box.

Cancel

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

Dialog Editor

The **Dialog Editor** menu option launches the **MasQ Enable Dialog Designer**:



MasQ Enable Dialog Designer

This application allows you to design dialog boxes for use with the Basic Script Editor. While it doesn't have the full power of a Visual Basic form designer, it provides you with a quick way to generate and place dialog box code into the Basic Script Editor.

For additional information on the Cypress Enable dialog box code, see the "Dialog Support" topic.

To Create a New Dialog Box:

1. Select **Edit | Dialog Editor**. The **MasQ Enable Dialog Designer** appears.
2. Use the dialog designer toolbar to select and place controls into the Design Window.
3. Change caption properties as needed by using the **Properties** option from the **Window** menu.
4. Align controls on the Design Window by using options in the **Edit** menu.

5. When you have finished designing your dialog box, select **File | Put Dialog on Clipboard**. This sends the code for the dialog box to the Windows Clipboard.
6. Select **File | Close Dialog Designer**.
7. Access the **Basic Script Editor** inside PC-DMIS.
8. Press CTRL + V to paste the code from the clipboard into the Basic Script Editor. The dialog box code begins with *Begin Dialog* and ends with *End Dialog*.
9. Modify the code as needed.

To Modify an Existing Dialog Box:

This procedure assumes you have already pasted some sort of dialog box code from the clipboard into the Basic Script Editor, as described in the "To Create a New Dialog Box:" topic.

1. Access the **Basic Script Editor** inside PC-DMIS.
2. Select the dialog box code and press CTRL + C to copy it to the Clipboard. The dialog box code begins with *Begin Dialog* and ends with *End Dialog*.
3. Select **Edit | Dialog Editor**. The **MasQ Enable Dialog Designer** appears.
4. From the dialog designer's menu bar, select **File | Load Dialog from Clipboard**.
5. Use the dialog designer to further modify your dialog box.
6. When you have finished modifying your dialog box, select **File | Put Dialog on Clipboard**. This sends the code for the dialog box to the Windows Clipboard.
7. Select **File | Close Dialog Designer**.
8. Access the **Basic Script Editor** inside PC-DMIS.
9. Press CTRL + V to paste the code from the clipboard into the Basic Script Editor.
10. Modify the code as needed.

Convert OldBasic Script

The **Convert OldBasic Script** menu item, allows you to convert scripts from older versions (versions 2.3 and previous) into the latest format. To use this option, first load the old script into the Basic Script Editor and then select the **Edit | Convert OldBasic Script** menu item.

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 script or **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 (Pcdbasic.hlp or Pcdbasic.chm) created for the add on Basic Module.

Basic Script Toolbar



The **Standard** toolbar for basic scripts 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



This 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.

Compile



This icon compiles (makes the script understandable and ready to run on the computer system) the current BASIC script. You must compile a script before running it.

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

Introduction

In this chapter, 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.

For example:

```
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 the "Language Reference A – Z" chapter 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 described in the "IsNumeric" topic in the "Language Reference A – Z" chapter.

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 (see "IsEmpty").

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.

For example:

```

Sub Main

    Dim X As Integer

    Dim Y As Double

    Dim YourName$, YourAge%      ' multiple declaration on one line Dim v

End Sub

```

Notice that the variables `YourName` and `YourAge` use a symbol to declare the variable type instead of the term **As**.

See "Dim Statement" for more information.

Note: While it may be possible in some cases to use variables without declaring them with the Dim statement first, doing so is not supported in Enable BASIC and may cause problems in your code.

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.

For example:

```
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" chapter 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 under "Functions, Statements, Reserved words – Quick Reference" topic in the "Scripting Language Overview" chapter.

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.

Function Return Types

Note: Be aware that type Object is not a valid return type of functions.

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() Flags4

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 functions 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 details)

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 accessible 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 than 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 Integer
```

```
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 elements 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 array 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 ()
```

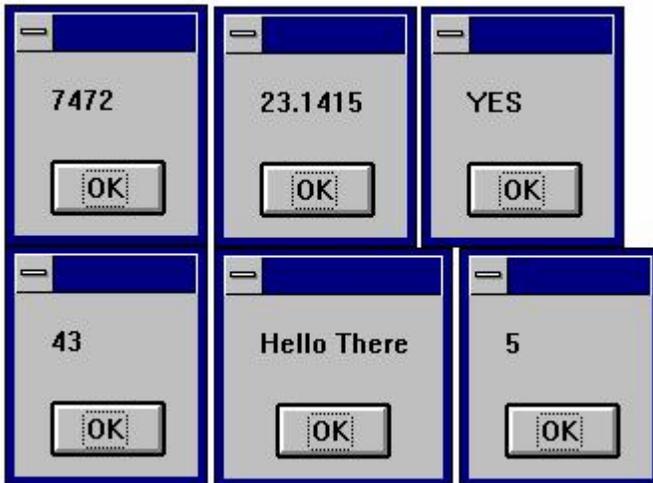
```
    a = 5
```

```
    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

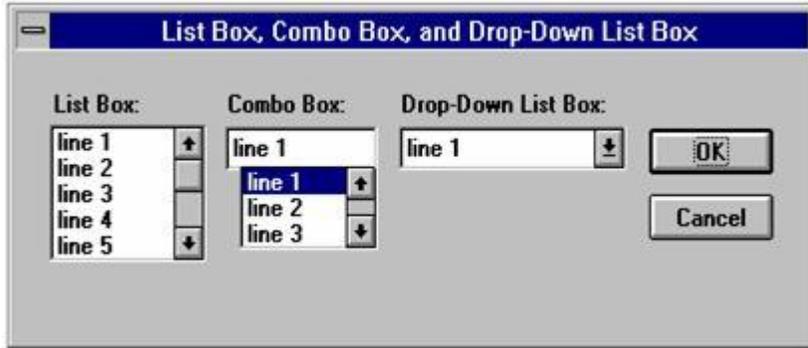
Dim Dlg1 As ButtonSample

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
```

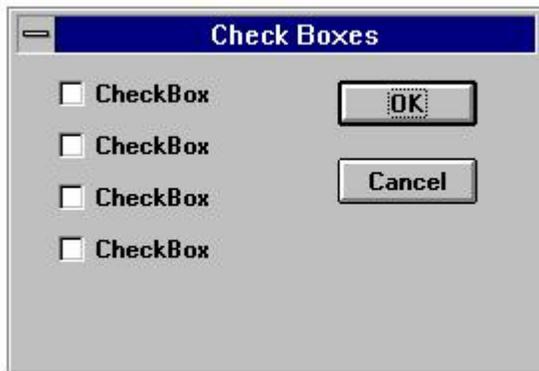
```
Dim Dlg1 As BoxSample
```

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

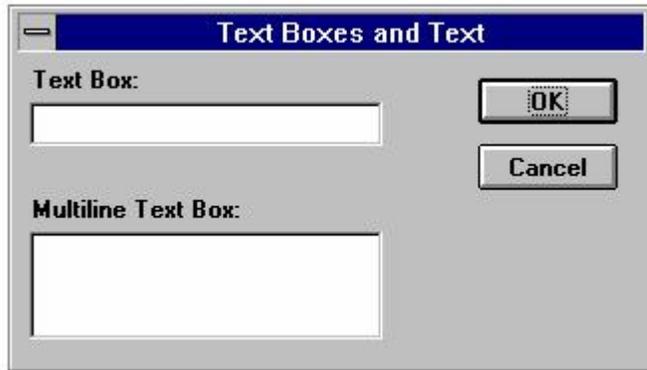
Dim Dlg1 As CheckSample

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 textbox.

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

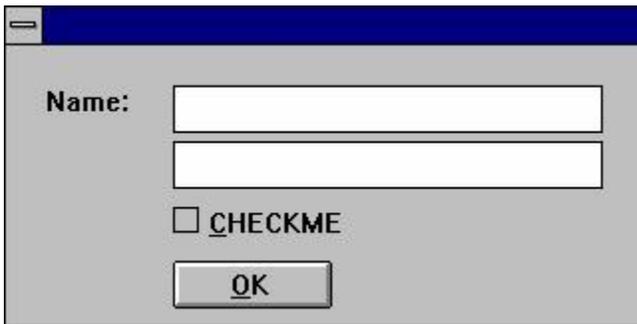
End Dialog

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, supvalue)

  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

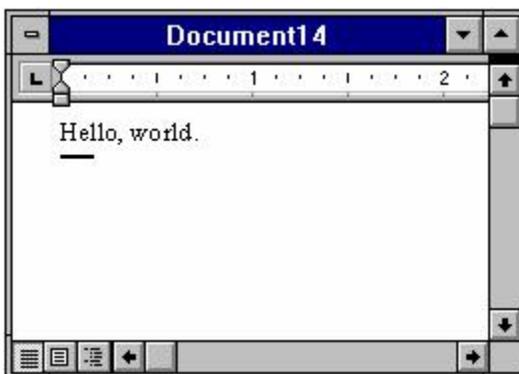
    Set MyObj = CreateObject ("Word.Basic")

    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

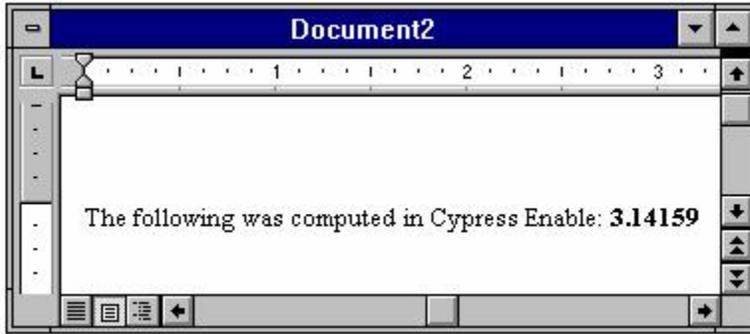
Some applications provide objects that support OLE Automation. You can use Enable Basic to programmatically manipulate the data in these objects. Some objects that support OLE Automation also support linking and embedding. You can create an OLE Automation object by using the CreateObject function.

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 referred 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 of information about the Windows system in a simple sequential listing.

Scripting Language Overview

Introduction

This chapter contains quick reference charts and tables on the following:

- Type/Functions/Statements
- Data Types
- Operators
- Operator Precedence
- Functions, Statements, Reserved Words – Quick Reference

Quick Reference of 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, CDbI, CInt, CInG, 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	$x = y \text{ Mod } z$
+	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 / Multiplication	
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

Introduction

This chapter contains a language reference in alphabetic order of all the BASIC code you can use in your scripts. Statements and functions in the examples are offset with a **boldface** font.

Note: Keep in mind that while most of the examples given in this chapter will work fine inside any Visual Basic editor (such as Visual Basic for Excel), there may be some items that only function inside of the BASIC Script Editor that ships with PC-DMIS.

Abs Function

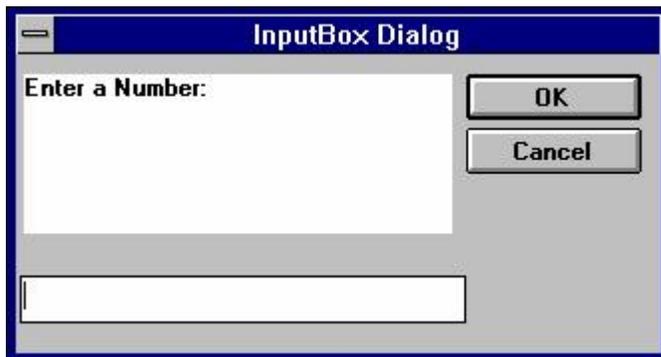
Abs (number)

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)

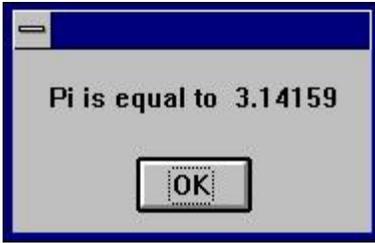
Returns the arc tangent of a number

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
```



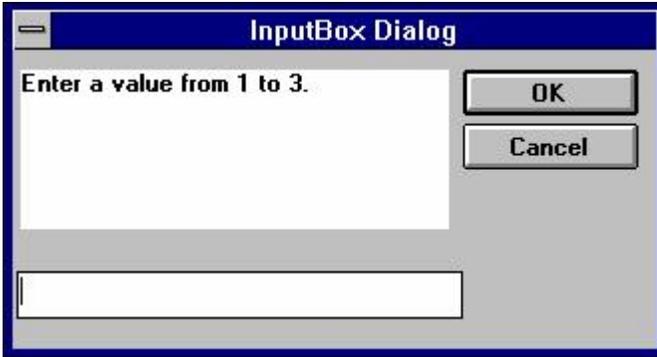
Beep Statement

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*[\i*dir*]...

The parameter *pathname* is a string limited to fewer than 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 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 [[#filename] [, [#]filename],,,
```

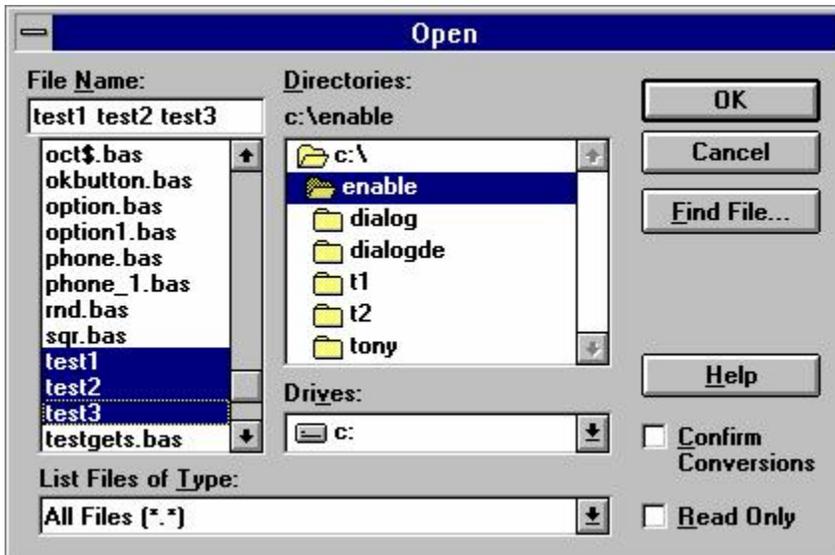
The Close Statement takes one argument *filename*. *Filename* is the number used with the Open Statement to open the file. If the Close Statement is used without any arguments it closes all open files.

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:

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

CreateObject Example

```
Sub Command1_Click ()
    Dim word6 As object
    Set word6 = CreateObject("Word.Basic")
    word6.FileNewDefault
    word6.InsertPara
    word6.Insert "Attn:"
    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:

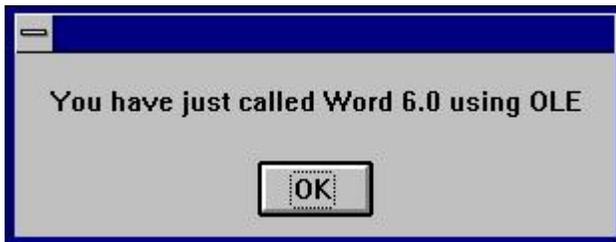
Vender Product:

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
End Sub

```

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

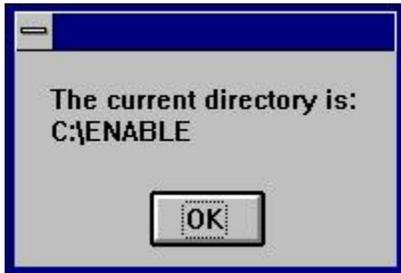
CurDir (*drive*)

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 & ".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 Variant (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 extensions 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 hWnd%
```

```
    Dim str1 As String *51
```

```
    Dim str2 As String * 25
```

```
    hWnd% = GetFocus()
```

```
    print "GetWindowText returned: ", GetWindowText( hWnd%, str1,51 )
```

```
    print "GetWindowText2 returned: ", GetWindowText( hWnd%, 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 preceding `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.

Note: While it may be possible in some cases to use variables without declaring them with the Dim statement first, doing so is not supported in Enable BASIC and may cause problems in your code.

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
    End Select
End Function
```

```
        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*]

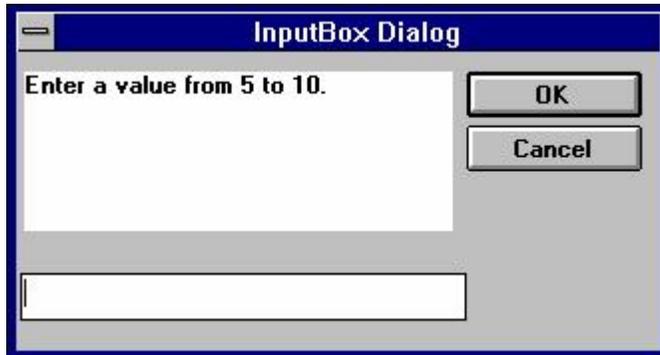
Loop [{While|Until} *condition*]

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)

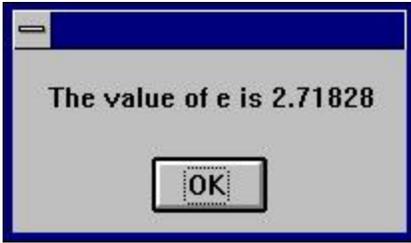
Returns the base of the natural log raised to a power (e^{num}).

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:

```
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.
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.	<p>The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator.</p> <p>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.”</p>
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

- + \$ () Display a literal character.

space 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

One section only

Two

Three

The result is

The format expression applies to all values.

The first section applies to positive values, the second to negative sections values.

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\er\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 tttt. 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).
dddd	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).
tttt	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
mmm-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 Variant (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 [#] *filename*, [*recordnumber*], *variablename*

Reads from a disk file into a variable

The Get Statement has these parts:

Filename 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

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*

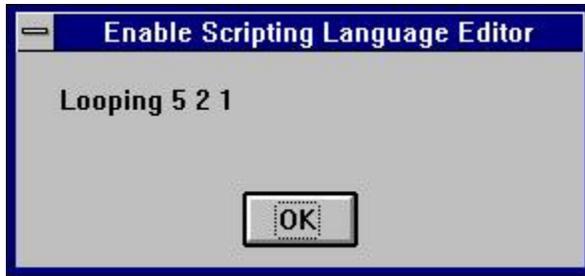
Branches unconditionally and without return to a specified label in a procedure.

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.

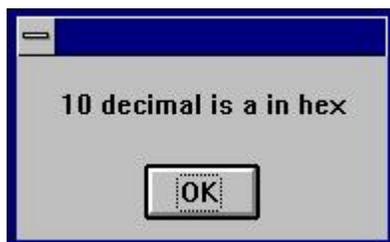
Hex returns a string

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 Main()  
    ' 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 , [ #] filename )
```

Input returns characters from a sequential file.

The input function has two parameters *n* and *filename*. *n* is the number of bytes to be read from a file and *filename* 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.

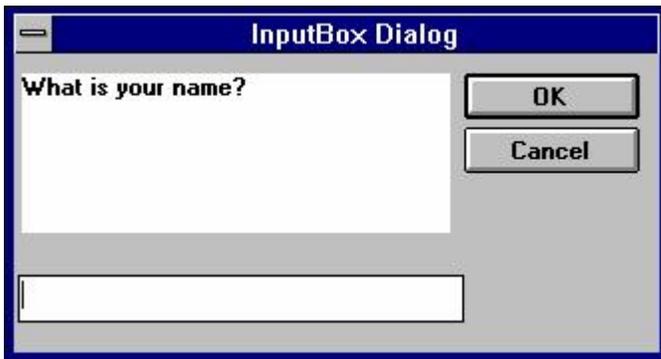
Title is a string that is displayed at the top of the input dialog box.

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

Xpos and Ypos and the x and y coordinates 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*.

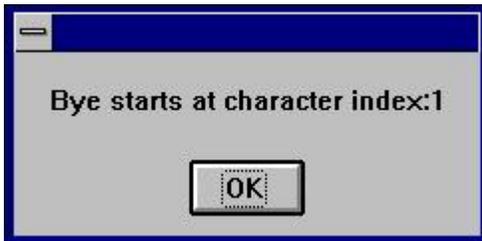
The *numbegin* parameter is not optional and sets the starting point of the search. *numbegin* must be a valid positive integer no greater than 65,535.

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 *variablename* 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)
    MCheckk = IsDate(x)
    MArray1 = CStr(MArray)
    MCheck1 = CStr(MCheck)
    Print MArray1 & " is a date " & Chr(10) & MCheck
    Print x & " is a date" & Chr(10) & MCheckk
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)`

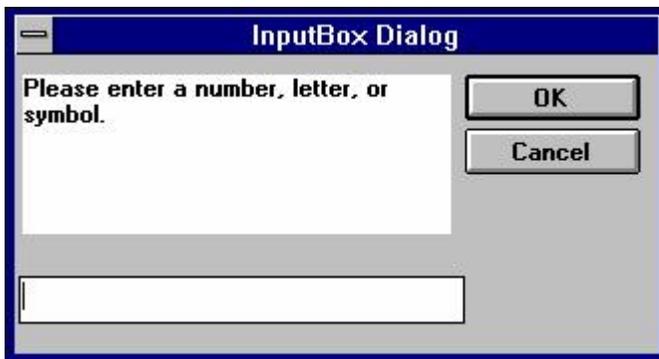
Returns a TRUE or FALSE indicating if the *v* parameter can be converted to a numeric data type.

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.
    End If
End Sub
```

```

        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

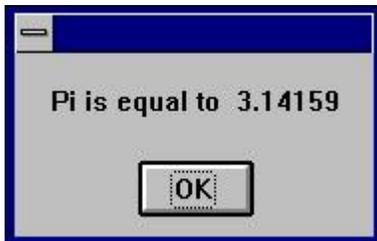
[Let] *variablename* = *expression*

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(filename)

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

The parameter filename 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.

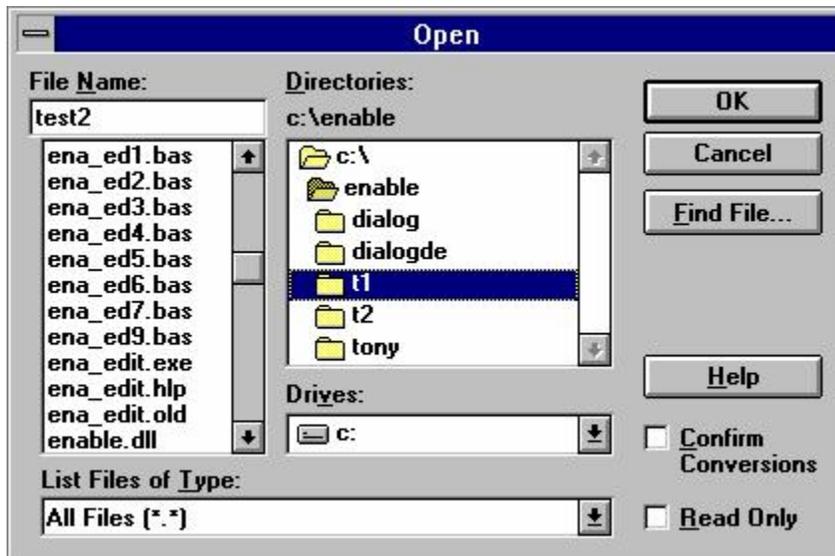
Example:

```

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 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*)

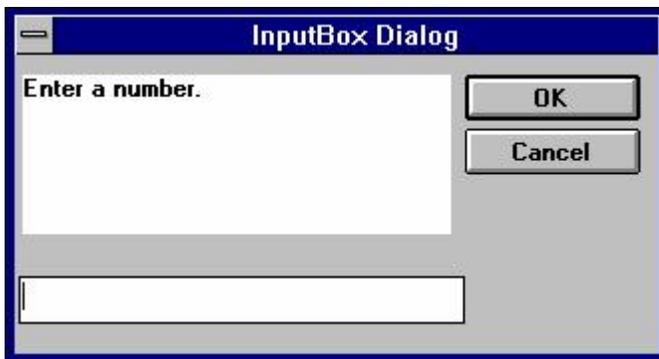
Returns the octal value of the decimal parameter

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:"
    End Dialog
End Sub
```

```

    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 Errors

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

- 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 interrupt 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 the system.";

// Make sure the printer is set 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.
<i>mode</i>	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           ' Print the results in the compiler messages window
End Sub
```



Print # Statement

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

Writes data to a sequential file.

Print statement Description:

filename:

Number used in an Open statement to open

sequential file. It can be any

number of an open file. Note that the

number sign (#) preceding filename 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 *n*th 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 *expression list*, 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:

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

```
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
```

```
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 random 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 an array but not to change the number of dimensions in an array or the type of the elements in the array.

ReDim will only work with single dimensional arrays. Multi-dimensional arrays, like ReDim MyArray (3,5), are invalid.

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, and is skipped by the compiler.
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 filename, position

The parameter *filename* 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"

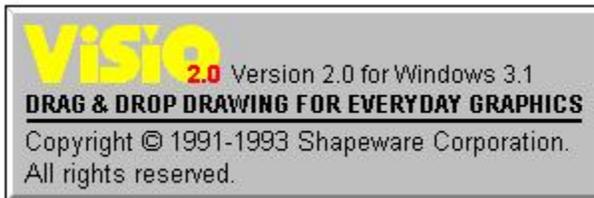
Dim cyan As Object
Set cyan = page.DrawOval (2.5, 8.5, 5.75, 5.25)
cyan.FillStyle = "Cyan fill"

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:

```
' 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:

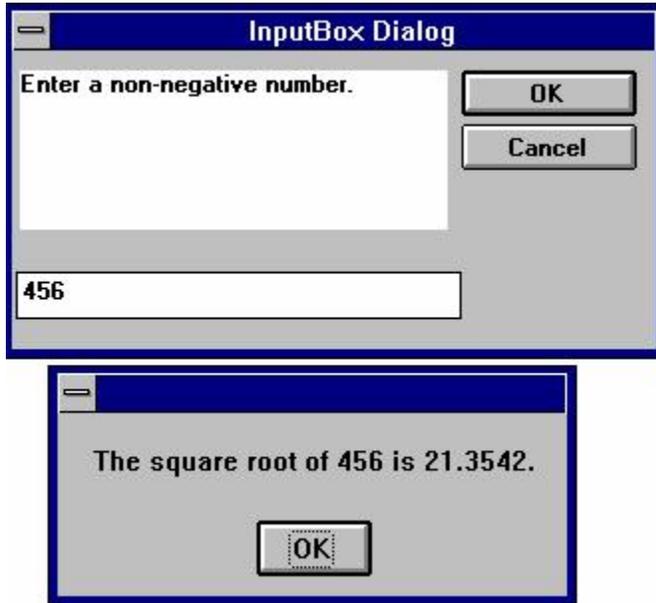
```
Sub Form_Click ()
    Dim Msg, Number          ' Declare variables.
    Msg = "Enter a non-negative number."
    Number = InputBox(Msg)   ' Get user input.
    If Number < 0 Then
        Msg = "Cannot determine the square root of a negative number."
    Else
        Msg = "The square root of " & 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)

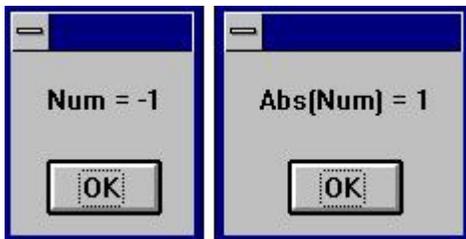
Returns the value of a numeric expression.

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 "*****".
End Sub
```

```

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 Variant (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"
```

```

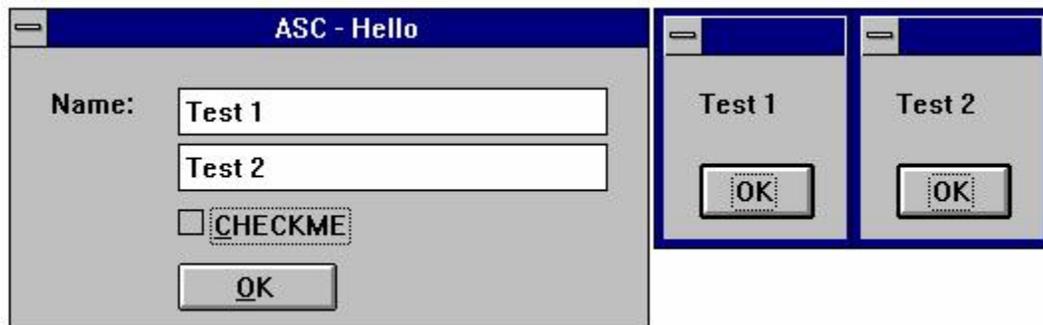
TEXT 10, 10, 28, 12, "Name:"
TEXTBOX 42, 10, 108, 12, .nameStr
TEXTBOX 42, 24, 108, 12, .descStr
CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
OKBUTTON 42, 54, 40, 12

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 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 entered 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
End If
```

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
End Sub
```

With Statement

With object

[STATEMENTS]

End With

The With statement allows you to perform a series of commands or statements on a particular object without again referring 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 member of an object in an outer With block.

Related Topics: While...Wend 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 typela As type1
Dim type2a As type2

Sub Main ()

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

End Sub

```

Write # - Statement

Write #filename [,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.
        PPrint 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

PC-DMIS's Automation gives you the ability to automate repetitive tasks within PC-DMIS or even to use elements of PC-DMIS functionality, within a custom built application.

PC-DMIS Automation contains these benefits:

- *PC-DMIS Automation is computer independent.* You can have a process on one computer automating a process on another computer.
- *PC-DMIS Automation is location independent.* You can run automation scripts within PC-DMIS itself, using the BASIC Script Editor or you can run automation scripts in external Visual Basic Editors. In addition, you can run automation scripts across a network.
- *PC-DMIS Automation is Language independent:* If you don't know BASIC but are familiar with another programming language, you can configure that programming language to use PC-DMIS's library (the examples and descriptions in this manual, however, are written using the BASIC programming language).

This Automation chapter contains a detailed list of methods and properties for each PC-DMIS Automation Object. The various objects are listed in alphabetical order. A **bold** item indicates a default property or method for the object.

To get started with Automation, you should look at "Sample Automation Scripts" on page 2 to get a feel for what's needed, and then look at "Accessing an Object's Properties, Methods and Events" on page 3 for details on how to access the various methods and properties contained in the various objects. For questions beyond the scope of this chapter, consult a Visual BASIC book on automation.

Sample Automation Scripts

Below are some very simple sample automation scripts that will demonstrate some of the necessary components needed to get started automating PC-DMIS.

Note: Notice in the examples below that you must declare and set the proper automation objects before you can access and use PC-DMIS's automation methods and properties. The "Accessing an Object's Properties, Methods and Events" topic on page 3 will be of assistance to you as you create your own scripts.

Sample Automation Script 1

The following example first uses PC-DMIS code to receive an integer value from the user and assigns it to the V1 variable.

```
C1=COMMENT/INPUT,Please type an integer value.  
ASSIGN/V1 = INT(C1.INPUT)  
COMMENT/OPER,BEFORE SCRIPT: Variable is:
```

```
,V1
```

It then calls a BASIC script named TEST2.BAS.

```
CS1=SCRIPT/FILENAME= D:\PROGRAM FILES\PCDMIS35\TEST2.BAS  
FUNCTION/Main,,  
STARTSCRIPT/  
ENDSCRIPT/
```

Here is TEST2.BAS:

```
Sub Main  
  Dim App As Object  
  Set App = CreateObject ("PCDLRN.Application")  
  Dim Part As Object  
  Set Part = App.ActivePartProgram  
  Dim Var As Object  
  Set Var = Part.GetVariableValue ("V1")  
  Dim I As Object  
  If Not Var Is Nothing Then  
    Var.LongValue = Var.LongValue + 1  
    Part.SetVariableValue "V1", Var  
    MsgBox "V1 is now: " & Var  
  Else  
    MsgBox "Could Not find variable"  
  End If  
End Sub
```

This script takes V1 variable and, using the `GetVariableValue` and `SetVariableValue` automation methods, increments the V1 by one and then sets the new value for V1 in the part program.

PC-DMIS then displays the changed variable in an operator comment.

```
COMMENT/OPER,AFTER SCRIPT: Variable is now  
,V1
```

Note: PC-DMIS variables only hold values during execution; at learn time PC-DMIS variables have a value of zero. The `GetVariableValue` and `SetVariableValue` methods only change a variable's value during the script's execution. If you want to permanently change a value of a variable inside PC-DMIS, you should use the `PutText` method instead.

Sample Automation Script 2

The following script receives an operator name from the user and then inserts an `COMMENT/OPER` command into the Edit window for the currently open part program, displaying the name of the operator.

PC-DMIS must be running with an open part program in the background.

```
Sub Main
'Get operator Name And assign it To variable: N$.
N$ = InputBox$("Please enter your name:", "Operator", "", 200, 175)

'The following section adds a comment cmd to the part program
Dim App As Object

'Get the pointer to the PC-DMIS application
Set App = CreateObject("PCDLRN.Application")
Dim Part As Object

'Get the pointer to the current part program
Set Part = App.ActivePartProgram
Dim Cmds As Object

'Get the pointer to the set of commands In the part program
Set Cmds = Part.Commands
Dim Cmd As Object

'Add a COMMENT command
Set Cmd = Cmds.Add(SET_COMMENT, True)

'Set the comment's type to REPT
retvaltype = Cmd.PutText("REPT", COMMENT_TYPE, 0)

'Put the string held in variable N$ into the comment's text
retvaltext = Cmd.PutText(N$, COMMENT_FIELD, 1)

'Redraws the COMMENT command so that the applied changes are applied to
the part program
Cmd.ReDraw
End Sub
```

Accessing an Object's Properties, Methods and Events

Accessing an Object's Properties, Methods and Events

Objects are external classes that contain methods, properties and events.

 **Methods:** Methods are functions that usually perform actions. This usually returns a boolean value to determine whether or not the function succeeded. Methods are analogous to verbs in languages.

 **Properties:** Properties allow you to read or sometime write certain characteristics or attributes of an object or control. Properties are analogous to adjectives in languages.

 **Events:** Events are routines that get called when certain conditions are met. 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, for example supports events. Handling events involves declaring an object of the correct type and then adding handling functions for the different events. Currently events are found in only these objects:

ApplicationObjectEvents

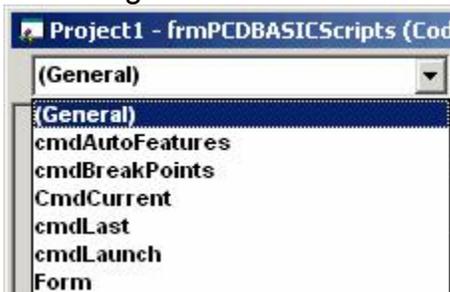
Machine

PartProgram

Accessing Event Subroutines

The easiest way to access an object's event subroutines is by following this procedure.

1. Access a readily available and robust Visual BASIC editor (such as one that ships with Microsoft's Word or Excel products).
2. Select **(General)** from the **Object** list in the code window. This allows you to make global variable declarations.

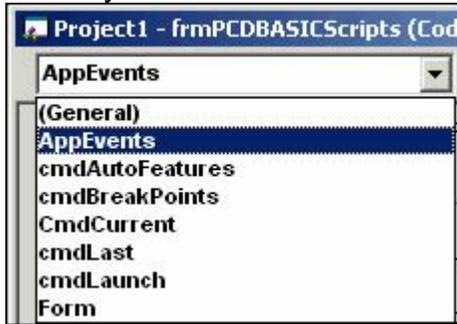


3. Declare a variable using the [WithEvents](#) keyword and specify an object that has events. For example:

```
Dim WithEvents AppEvents As PCDLRN.ApplicationObjectEvents
```

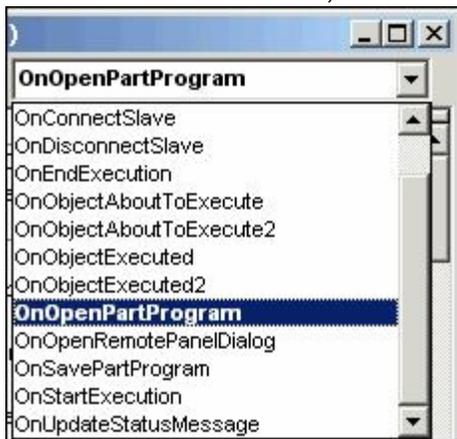
This code would enable the AppEvents variable in the **Object** list

4. Select your declared variable from the **Object** list in the code window.



This enables you to select specific event subroutines from the **Procedure** list.

5. From the code window, select an event subroutine from the **Procedure** list.



The new subroutine appears in the code window.

6. Make modifications to the event's subroutine code as needed. When PC-DMIS meets the specified condition, the event subroutine gets ran along with any code you added.

Accessing Methods and Properties

There are two ways to get to an object's methods, properties, and events.

1. Create objects by their ID
2. Call the object from an existing object

Whether you're creating an object or calling an object from an existing object, you'll need to first create and then set a pointer to the appropriate object.

Step 1: Declare the pointer variable name for the application by using the "DIM" statement. For example:

```
Dim App As Object
```

Step 2: Set the pointer variable to the PCDLRN Application using CreateObject. For example:

```
Set App = CreateObject("PCDLRN.Application")
```

Step 3: Declare and set additional pointer variable names for any needed sub objects found within the Application object. For example if you wanted to access commands available for the active part program, you're code would look something like this:

`Dim Part As Object`

`Set Part = App.ActivePartProgram`

`Dim Cmds As Object`

`Set Cmds = Part.Commands`

`Dim Cmd As Object`

`Set Cmd = Cmds.Add(SET_COMMENT, True)`

Use the hierarchy charts and the Object Browser topics below as guides.

Automation Object Hierarchy Charts

The following are the charts showing the hierarchy of the various charts and how to get to the methods or properties you need.

Chart 1 - Main Overview

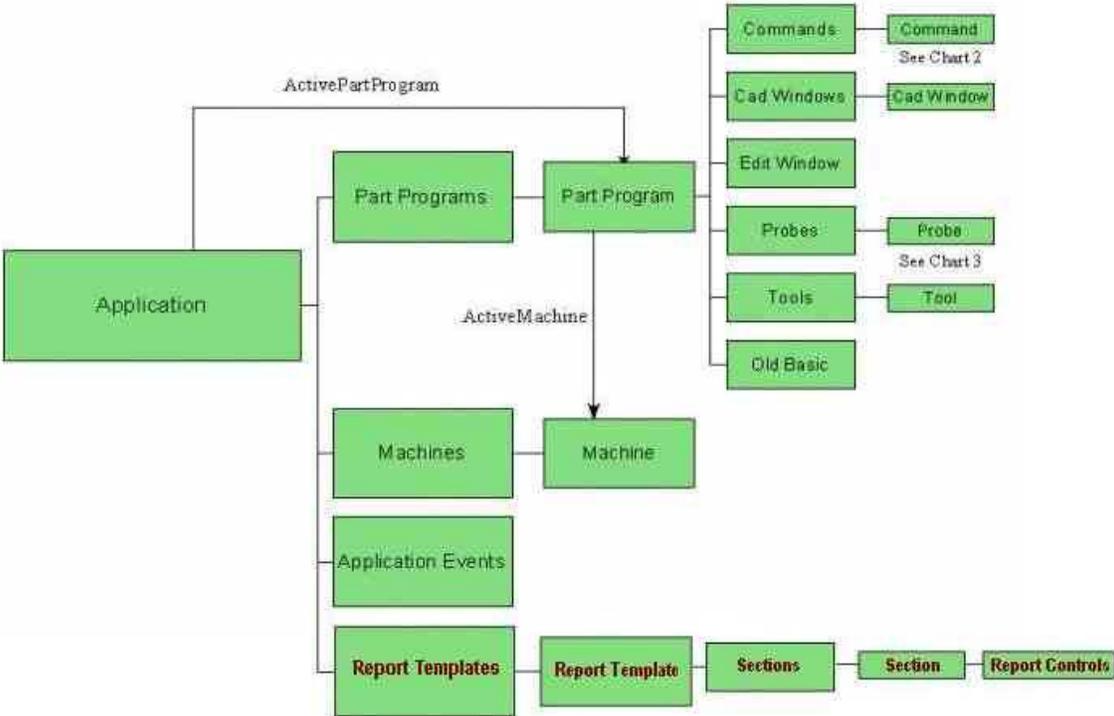


Chart 2 – Command Subobjects

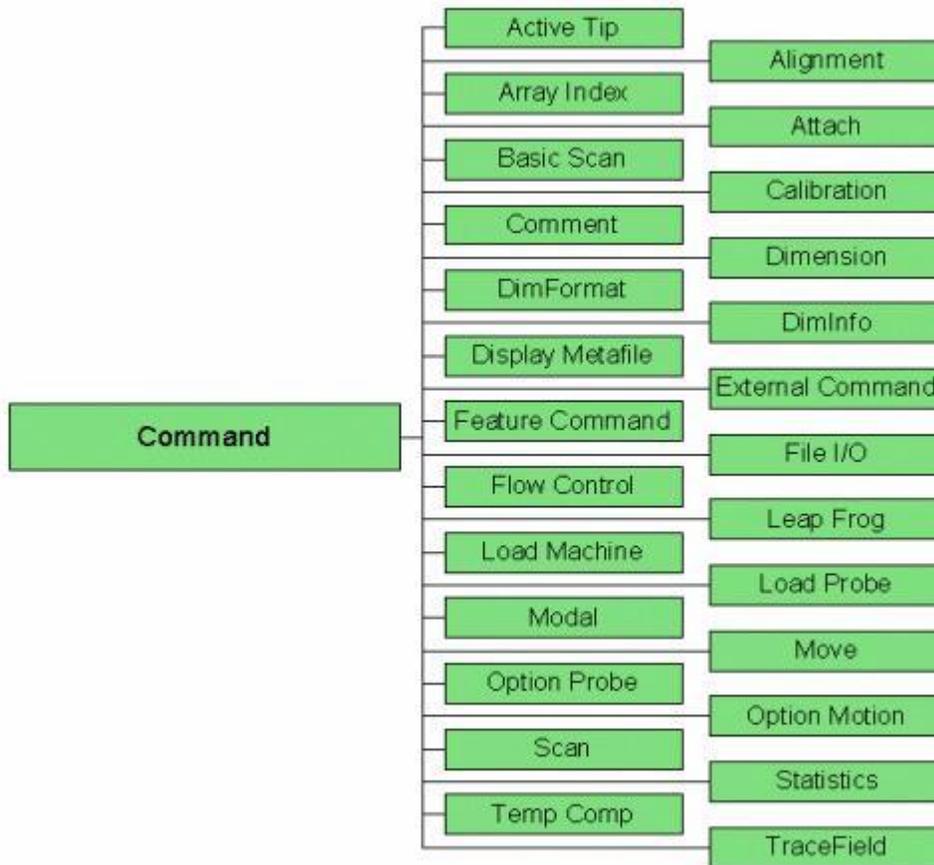


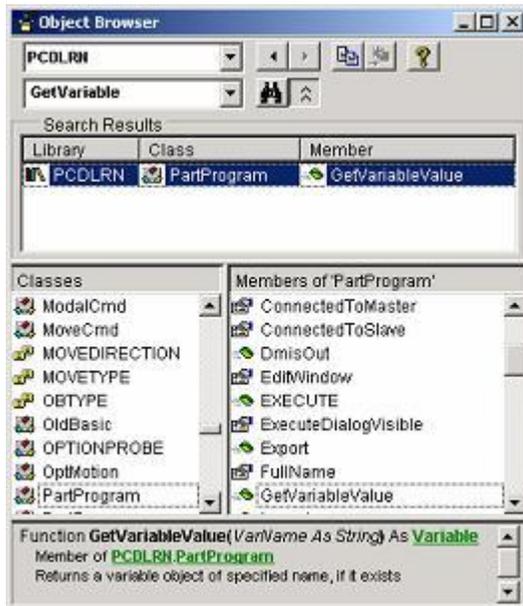
Chart 3 – Probe Subobjects



Using the Object Browser in Other Editors

While the PC-DMIS Basic Script Editor has its uses, it doesn't have a lot of the visual syntax and other programming aids available to you from other common programs that also support automation.

The Object Browser, available in standard Visual Basic Editors, is essential to getting the proper help when writing automation scripts. It contains all the different objects for any library you have chosen to use in your automation project.



Example Object Browser

To set up the object browser with the appropriate libraries, do the following:

1. Open Visual Basic (or you can open the VB editor that ships with MS Word or MS Excel)
2. In Visual Basic, select the **References** menu item. In VB5 this is **Project | References**. (In Excel or Word's VB Editor, select **Tools | References**).
3. The **References** dialog box appears. Items that are checked are libraries already included currently.
4. Scroll down to PC-DMIS X.X Object Library (where X.X is your library version type and select the check box—this should be similar to your version of PC-DMIS).
5. Click **OK**.
6. Access the Object Browser (press F2 within the VB Editor). In the list at the top it should say <All Libraries>. From the list, select the **PCDLRN** library.

You can now browse through all the objects and view their properties, methods, and events. Most of the objects have properties and methods in the Pcdmis object library. Only a few objects have events.

Additionally, when writing code, your Visual Basic Editor will now contain the visual syntax aids for the various PC-DMIS objects and commands.

Note: If new objects, methods, properties, or events are added, the Object Browser will most likely contain the new information first and will subsequently be more up to date than this manual another good reason to use the Object Browser when coding your scripts.

Active Tip Object Overview

The Active Tip object gives access to the properties of the PC-DMIS Set Active Tip command.

Properties:

ActiveTip.Angle

DOUBLE value representing the rotation angle of the tip transformation matrix.

Read/Write **Double**

ActiveTip.TipID

STRING value representing the ID of the tip to be made active.

Read/Write **String**

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.

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.

AlignCommand Object Overview

Objects of type **AlignCommand** are created from more generic **Command** objects to pass alignment information back and forth.

Properties:

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
```

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**.

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.

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
```

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**.

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.

AlignCommand.ExternalFileID

Represents the external filename for recalling external alignments. 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.

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.

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.

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.

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.

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.

AlignCommand.InitID

Represents the initial ID of this alignment object. The initial 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.

AlignCommand.IterativeLevelAxis

Represents the level axis for an iterative alignment. Read/write **PaxisType**.

AlignCommand.IterativeOriginAxis

Represents the origin axis for an iterative alignment. Read/write **PaxisType**.

AlignCommand.IterativeRotateAxis

Represents the rotate axis for an iterative alignment. Read/write **PaxisType**.

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.

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.

AlignCommand.MeasAllFeatAlways

Represents the “Measure All Features Always” 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.

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.

AlignCommand Object Overview

Objects of type **AlignCommand** are created from more generic **Command** objects to pass alignment information back and forth.

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.

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.

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.

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**.

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.

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.

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.

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.

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.

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.

AlignCommand.CalculateMatrices

Forces immediate calculation of alignment matrices.

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("PcdIrn.Application")
```

Launching PC-DMIS With Startup Options

Because of an inherent weakness in the way Microsoft designed the CreateObject function, the CreateObject doesn't allow startup parameters. This means when the code executes it will launch PC-DMIS always in ONLINE mode.

However, there is a way around this. Your code can dynamically create a special startup file that will cause PC-DMIS to launch with specific startup options.

In order to launch PC-DMIS via automation with a startup file you must do the following:

- Create a text file named AutomationStartupOptions.txt.
- Create a single line of text in the file with the available startup options. The PC-DMIS specific startup options include the following:

/f – Launches in Offline mode.

/o – Launches in Operator mode.

/d – Launches in Debug mode.

/r – Launches in Reverse Axes mode

/postin – Launches in import mode. PC-DMIS will automatically import a specified file.

/postout – Launches in export mode. PC-DMIS will automatically export a specified file.

The line of text would look like this: */f /o /d /r /postin /postout*

- Launch PC-DMIS via automation.

When PC-DMIS starts, it checks to see if the AutomationStartupOptions.txt file exists. If it does, then it uses the file to set the necessary flags. However, when PC-DMIS closes, it will delete the text file. This means that the code you use to launch PC-DMIS must also create the needed text file on the fly or must rename an existing file to AutomationStartupOptions.txt. See the "Example Code from C++" below.

Example Code from C++

```
// Create my startup file
```

```

int nIndex;

CString szFileName, szLine(_T("/f"));

GetModuleFileName (AfxGetInstanceHandle(), szFileName.GetBuffer (_MAX_PATH),
_MAX_PATH);

szFileName.ReleaseBuffer();

nIndex = szFileName.ReverseFind('\\');

szFileName = szFileName.Left(nIndex);

szFileName += _T("\\AutomationStartupOptions.txt");

CStdioFile StartupFile;

if( StartupFile.Open(szFileName, CFile::modeCreate|CFile::modeWrite|CFile::typeText) )
{
    StartupFile.WriteString(szLine);

    StartupFile.Close();
}

```

Properties:

Application.ActivePartProgram

Represents the currently active part program. Read/Write **PartProgram**.

Application.ApplicationEvents

Returns the Application Events Object for use in capturing application events.

Application.ApplicationSettings

Returns the Application Settings Object for use in modifying PC-DMIS's settings.

Application.Caption

The text in the title bar of the application. Read/Write **String**.

Application.CurrentUserDirectory

This returns a string showing the directory that contains the current user's setup information. Read-only **String**.

Application.DefaultFilePath

The directory in which the File Open dialog starts. If you set this property to empty it returns the installation path. Read/Write **String**.

Application.DefaultMachineName

The name of the next available machine for attaching to a part program. Read Only **String**

Application.DefaultProbeFile

The name of the last chosen probe file used when creating a new part program. Read Only **String**

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.Height

The height of the PC-DMIS window in screen pixels. Read/Write **Long**.

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.Machines

Returns the read-only **Machines** collection object.

Application.MajorVersion

Returns the major version number of the application. Read only **Long**.

Application.MinorVersion

Returns the minor version number of the application. Read only **Long**.

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 Name property is "PCDLRN.EXE".

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.PartPrograms

Returns the collection of part programs currently active in PC-DMIS. Read-only **PartPrograms** collection.

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 Path property is "C:\PCDMISW".

Application.RemotePanelMode

Indicates that PC-DMIS is in Remote Panel mode. Used by Remote Panel Application (RPA). Read/write **boolean**.

Application.StatusBar

The text on the status bar of the main PC-DMIS window. Read/Write **String**.

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.UserExit

TRUE if the PC-DMIS automation engine is will shut down when the user exits PC-DMIS, otherwise FALSE. Read/Write **Boolean**.

Application.VersionString

Returns the version string for the application. Read only **string**.

Application.Visible

TRUE if PC-DMIS is visible, otherwise FALSE. Read/Write **Boolean**.

Application.Width

The width of the PC-DMIS window in screen pixels. Read/Write **Long**.

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.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.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.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.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.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.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.SpawnNewInstance

Returns Application Object of newly created instance of application.

Application.WaitUntilReady

Waits until online machine has fully initialized or timeout period has elapsed before returning.

Application.WriteRegistryBool

Sets a PC-DMIS registry entry to a TRUE or FALSE value. Write **boolean**.

Syntax:

Function WriteRegistryBool(IpszSection As String, IpszEntry As String, bValue As Boolean, bSetVariable As Boolean) As Boolean

Remarks:

This function takes four parameters.

The first parameter is a string specifies the section in which the registry entry that you want to modify resides.

The second parameter is a string that specifies the entry to which you want to write the value.

The third parameter is the boolean value to write to the registry entry.

The final parameter is a boolean value that determines whether or not an associated global variable in PC-DMIS gets changed to the value of the third parameter. This parameter only works for registry entries pertaining to feature ID prefixes.

This function can also return a boolean value that determines whether or not the value of the third parameter was successfully written to the registry entry.

Example:

This code sample turns off PC-DMIS's start up music:

```
Dim myapp As New PCDLRN.Application  
  
Set myapp = CreateObject("pcdlrn.application")  
  
myval = myapp.WriteRegistryBool("Option", "TurnOffTheAnnoyingStartupMusic",  
False, True)
```

Application.WriteRegistryDouble

Sets a PC-DMIS registry entry to a specified double value. Write **double**.

Syntax:

```
Function WriteRegistryDouble(IpszSection As String, IpszEntry As String, dValue As Double,  
bSetVariable As Boolean) As Boolean
```

Remarks:

This function takes four parameters.

The first parameter is a string specifies the section in which the registry entry that you want to modify resides.

The second parameter is a string that specifies the entry to which you want to write the value.

The third parameter is the double value to write to the registry entry.

The final parameter is a boolean value that determines whether or not an associated global variable in PC-DMIS gets changed to the value of the third parameter. This parameter only works for registry entries pertaining to feature ID prefixes.

This function can also return a boolean value that determines whether or not the value of the third parameter was successfully written to the registry entry.

Example:

This code sample sets an analog probe's lower force to the double value of 0.04.

```
Dim myapp As New PCDLRN.Application  
  
Set myapp = CreateObject("pcdlrn.application")
```

```
myval = myapp.WriteRegistryDouble("ANALOG_PROBING", "ProbeLowerForce",  
0.04, True)
```

Application.WriteRegistryDWORD

Sets a PC-DMIS registry entry to a specified long value. Write **long**.

Syntax:

```
Function WriteRegistryDWORD(IpszSection As String, IpszEntry As String, dwValue As Long,  
bSetVariable As Boolean) As Boolean
```

Remarks:

This function takes four parameters.

The first parameter is a string specifies the section in which the registry entry that you want to modify resides.

The second parameter is a string that specifies the entry to which you want to write the value.

The third parameter is the long value to write to the registry entry.

The final parameter is a boolean value that determines whether or not an associated global variable in PC-DMIS gets changed to the value of the third parameter. This parameter only works for registry entries pertaining to feature ID prefixes.

This function can also return a boolean value that determines whether or not the value of the third parameter was successfully written to the registry entry.

Example:

This code sample sets an auto circle's hits to the long value of 4.

```
Dim myapp As New PCDLRN.Application
```

```
Set myapp = CreateObject("pcdlrn.application")
```

```
myval = myapp.WriteRegistryDWORD("AutoFeatures", "DccCirNumHits", 4, True)
```

Application.WriteRegistryInt

Sets a PC-DMIS registry entry to a specified integer value. Write **Integer**.

Syntax:

```
Function WriteRegistryInt(IpszSection As String, IpszEntry As String, nValue As Long,  
bSetVariable As Boolean) As Boolean
```

Remarks:

This function takes four parameters.

The first parameter is a string specifies the section in which the registry entry that you want to modify resides.

The second parameter is a string that specifies the entry to which you want to write the value.

The third parameter is the integer value to write to the registry entry.

The final parameter is a boolean value that determines whether or not an associated global variable in PC-DMIS gets changed to the value of the third parameter. This parameter only works for registry entries pertaining to feature ID prefixes.

This function can also return a boolean value that determines whether or not the value of the third parameter was successfully written to the registry entry.

Example:

This code sample sets the number of rows for auto feature cylinders to the integer value of 3.

```
Dim myapp As New PCDLRN.Application
```

```
Set myapp = CreateObject("pcdlrn.application")
```

```
myval = myapp.WriteRegistryInt("AutoFeatures", "DccCylNumRows", 3, True)
```

```
MsgBox myval
```

Application.WriteRegistryPoint

Sets a PC-DMIS registry entry to a specified XYZ value. Write **double**.

Syntax:

```
Function WriteRegistryPoint(lpszSection As String, lpszEntry As String, dX As Double, dY As Double, dZ As Double, bSetVariable As Boolean) As Boolean
```

Remarks:

This function takes six parameters.

The first parameter is a string specifies the section in which the registry entry that you want to modify resides.

The second parameter is a string that specifies the entry to which you want to write the value.

The third parameter is the X value, of type **double**, to write to the registry entry.

The fourth parameter is the Y value, of type **double**, to write to the registry entry.

The fifth parameter is the Z value, of type **double**, to write to the registry entry.

The final parameter is a boolean value that determines whether or not an associated global variable in PC-DMIS gets changed to the value of the third parameter. This parameter only works for registry entries pertaining to feature ID prefixes.

This function can also return a boolean value that determines whether or not the value of the third parameter was successfully written to the registry entry.

Application.WriteRegistrySettings

Writes the current PC-DMIS registry entries and their values to the debug.txt file.

Syntax:

```
Sub WriteRegistrySettings()
```

Remarks:

The debug file is a text file with a default name of debug.txt and is located inside your PC-DMIS installation directory. When you have PC-DMIS set up to write data to this text file, you can use it to keep track of every action of PC-DMIS. This is useful data to give to Technical Support so that they can track down possible bugs or problems you may be encountering inside of PC-DMIS.

When you run a script with this method, all the registry entries get appended to the end of your debug file. You will see a line that says:

```
//// Start of registry settings ////
```

Following this line you will find all the registry entries and their values. After the final entry, PC-DMIS shows this line:

```
//// End of registry settings ////
```

Example:

This code sample writes all of PC-DMIS's registry entries and their values to the debug.txt file:

```
Dim myapp As New PCDLRN.Application  
Set myapp = CreateObject("pcdlrn.application")  
myapp.WriteRegistrySettings
```

Application.WriteRegistryString

This writes a specified string value to a valid PC-DMIS registry entry. Write **String**.

Syntax:

Function WriteRegistryString(IpszSection As String, IpszEntry As String, IpszValue As String, bSetVariable As Boolean) As Boolean

Remarks:

This function takes four parameters.

The first parameter is a string specifies the section in which the registry entry that you want to modify resides.

The second parameter is a string that specifies the entry to which you want to write the value.

The third parameter is the string value to write to the registry entry.

The final parameter is a boolean value that determines whether or not an associated global variable in PC-DMIS gets changed to the value of the third parameter. This parameter only works for registry entries pertaining to feature ID prefixes.

This function can also return a boolean value that determines whether or not the value of the third parameter was successfully written to the registry entry.

Example:

The following code sample, sets the PC-DMIS password for the **Setup Options** dialog box to the string: "PCDMIS007".

```
Dim myapp As New PCDLRN.Application
```

```
Set myapp = CreateObject("pcdlrn.application")
```

```
myval = myapp.WriteRegistryString("Option", "Password", "PCDMIS007", True)
```

Application Object Events Object Overview

The **ApplicationObjectEvents** object provides you with a series of events that get called when the PC-DMIS application meets certain conditions.

Events:

The following events are available to the Application Object Events object.

ApplicationObjectEvents.OnAddObject

Syntax:

```
expression.OnAddObject(program, command)
```

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

command: expression that evaluates to a **Command** object to determine the command for which this event should wait.

This event gets launched when the specified *command* gets added to the specified *program*.

ApplicationObjectEvents.OnClosePartProgram

Syntax:

expression.OnClosePartProgram(*program*)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

This event gets launched when the specified *program* gets closed.

ApplicationObjectEvents.OnConnectSlave

Syntax:

expression.OnConnectSlave(*program*)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

This event gets launched when PC-DMIS connects to and launches the specified program on the the slave computer.

ApplicationObjectEvents.OnDisconnectSlave

Syntax:

expression.OnDisconnectSlave(*program*)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

This event gets launched when PC-DMIS disconnects from the slave computer.

ApplicationObjectEvents.OnEndExecution

Syntax:

expression.OnEndExecution(*program*, *type*)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

type: this Long number determines the termination type used by this event.

This event gets launched when PC-DMIS finishes executing the specified program. PC-DMIS determines it has finished execution based on the termination *type*.

ApplicationObjectEvents.OnObjectAboutToExecute

Syntax:

expression.OnObjectAboutToExecute(*program*, *command*)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

command: expression that evaluations to a **Command** object to determine the command about to be executed.

This event gets launched immediately before the specified *command* gets executed.

ApplicationObjectEvents.OnObjectAboutToExecute2

Syntax:

expression.OnObjectAboutToExecute(*program*, *command*,*arm*)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

command: expression that evaluations to a **Command** object to determine the command about to be executed.

arm: Long value representing the arm on a multiple arm machine that is about to execute the *command* causing the event to launch.

This event gets launched immediately before the specified *command* gets executed on a specified *arm* of a multiple arm system.

ApplicationObjectEvents.OnObjectExecuted

Syntax:

```
expression.OnObjectAboutToExecute(program, command)
```

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

command: expression that evaluations to a **Command** object to determine the command that gets executed.

This event gets launched immediately after the specified *command* gets executed.

ApplicationObjectEvents.OnObjectExecuted2

Syntax:

```
expression.OnObjectAboutToExecute(program, command,arm)
```

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

command: expression that evaluations to a **Command** object to determine the command that gets executed.

arm: Long value representing the arm on a multiple arm machine that executes the *command* causing the event to launch.

This event gets launched immediately after the specified *command* gets executed on a specified *arm* of a multiple arm system.

ApplicationObjectEvents.OnOpenPartProgram

Syntax:

expression.OnClosePartProgram(program)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

This event gets launched when the specified *program* gets opened.

ApplicationObjectEvents.OnOpenRemotePanelDialog

Syntax:

expression.OnOpenRemotePanelDialog(program, ID, hwnd, msg, btn1, btn2, btn3, btn4, default)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

ID: A Long value representing a dialog box's ID.

msg: A message displayed in the dialog box.

btn1: A Long value representing button 1.

btn2: A Long value representing button 2.

btn3: A Long value representing button 3.

btn4: A Long value representing button 4.

default: a Long value representing the default button.

This event gets launched when the specified Remote Panel Application dialog box opens.

ApplicationObjectEvents.OnSavePartProgram

Syntax:

expression.OnSavePartProgram(program)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

This event gets launched when the specified *program* gets saved.

ApplicationObjectEvents.OnStartExecution

Syntax:

expression.OnStartExecution(*program*)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

program: expression that evaluates to a **PartProgram** object to determine the part program for which this event should wait.

This event gets launched when the specified part *program* begins execution.

ApplicationObjectEvents.OnUpdateStatusMessage

Syntax:

expression.OnUpdateStatusMessage(*msg*)

expression: Required expression that evaluates to a PC-DMIS **ApplicationObjectEvents** object.

msg: string of the displayed message

This event gets launched when the status bar gets updated with the specified message.

Application Settings Object Overview

The Application Settings object is a class that contains various properties and methods that allow you to work with PC-DMIS settings.

Properties

ApplicationSettings.WarningDefault19

Documentation Pending.

ApplicationSettings.WarningDefault48

Documentation Pending.

ApplicationSettings.WarningDefault60

Documentation Pending.

ApplicationSettings.WarnNoSavePrg

Documentation Pending.

ApplicationSettings.WarnOKPh9

Documentation Pending.

ApplicationSettings.WarnOKRotPh9

Documentation Pending.

ApplicationSettings.WarnOverwritingAlignment

Documentation Pending.

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.

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.

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.

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.

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.

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.

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.

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.

Properties

Attach.AttachedAlign

ID associated with an alignment in the attached program that corresponds with an alignment in the attaching program. Read/Write **String**

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.ID

ID associated with the attached part program. This ID identifies 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.LocalAlign

ID associated with an alignment in the attaching program that corresponds to an alignment in the attached program. Read/Write **String**

Attach.PartName

File name of the attached part program.

Read/Write **String**

Autotrigger Object Overview

The Autotrigger command object automatically takes hits when the probe enters a specified zone.

Properties

Autotrigger.Autotriggeron

Determines whether or not the Autotrigger feature is used when measuring. Read/write BOOLEAN.

Autotrigger.Beepingon

Determines whether or not the Beeping feature is used when the probe approaches the target. The closer you get to your target the more frequently you will hear the beeps. Read/write BOOLEAN.

Autotrigger.Radius

Determines the size of the radius, or tolerance zone, that surrounds the original hit location. When the probe enters this tolerance zone it will automatically take a hit. Read/Write double.

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.

Properties

BasicScan.AutoClearPlane

Determines whether auto clearance planes mode is on or off. Read/Write BOOLEAN.

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

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.

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.

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.

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

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 40.

BasicScan.DisplayHits

Determines whether hits of the scan are displayed in the Edit window or not. Read/Write BOOLEAN.

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 continuous 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.

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	-	-

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, Continuous contact mode.

BSCANMETH_CYLINDER: This method will scan around a Cylinder in High Speed, Continuous contact mode.

BSCANMETH_STRAIGHTLINE: This method will scan a straight line in a plane in High Speed, Continuous 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.

BasicScan.MethodCutPlane

Represents the method's cut plane vector. Read/write **PointData** object.

BasicScan.MethodEnd

Represents the scan's end point. Read/write **PointData** object. BasicScan.MethodEnd

BasicScan.MethodEndTouch

Represents the method's end touch vector. Read/write **PointData** object.

BasicScan.MethodInitDir

Represents the method's initial direction vector. Read/write **PointData** object.

BasicScan.MethodInitTopSurf

Represents the initial Surface Vector for the Edge method. Read/write **PointData** object.

BasicScan.MethodInitTouch

Represents the method's initial touch vector. Read/write **PointData** object.

BasicScan.MethodStart

Represents the scan's start point. Read/write **PointData** object.

Method	Method	Method	Method	Method	Method	Method	Method
	Start	End	CutPlane	InitDir	InitTouch	InitTopSurf	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		-

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.

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

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.

Methods:

BasicScan.AddControlPoint

Syntax

Return Value=expression.AddControlPoint(1,0)

Return Value: This value returns a boolean value. If the value is 1 (True), it adds a control point to the scan.

Expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

BasicScan.CreateBasicScan

Syntax

Return Value=expression.CreateBasicScan(1,0)

Return Value: This value returns a boolean value. If the value is 1 (or True), it causes DCC and Manual Scans to create a basic scan object.

Expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

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(<i>nCrossings</i> , <i>dRadius</i> , <i>dHalfAngle</i>)
Plane	<i>NCrossings</i>
Cone	<i>NCrossings</i> , <i>dHalfAngle</i>
Cylinder	<i>NCrossings</i> , <i>dRadius</i>
Sphere	<i>NCrossings</i> , <i>dRadius</i>

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 boundary point.

Y: Required **Long** variable that will hold the Y value of the boundary point.

Z: Required **Long** variable that will hold the Z value of the boundary point.

Remarks

This function works with patch scans. Use the `boundarypointcount` property to determine how many boundary points are available.

BasicScan.GetControlPoint

Syntax

Return Value=`expression.GetControlPoint(Index)`

Return Value: Returns the control point specified by the index.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

Index: Required **Long** value which indicates which control point to return.

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

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.

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 (<i>bln, bCenteringType, nCenteringDirection, dDiameter, dArcAngle, dDepth, dPitch</i>)
Linear	None
Edge	None
Circle	<i>bln, , , dDiameter, dArcAngle, dDepth</i>
Cylinder	<i>bln, , , dDiameter, dArcAngle, dDepth, dPitch</i>
Str Line	None
Center	<i>, bCenteringType, nCenteringDirection</i>

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 Dimensioned 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.

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.

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 **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.

BasicScan.RemoveControlPoint

Syntax

Return Value=expression.RemoveControlPoint(*Index*)

Return Value: Returns a boolean value. If set to 1 (True), the control point is removed at the specified index.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

Index: Required **Long** value which indicates which control point to remove.

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 (<i>nCrossings</i>, <i>dRadius</i>, <i>dHalfAngle</i>)
Plane	<i>Ncrossings</i>
Cone	<i>NCrossings</i> ,, <i>dHalfAngle</i>
Cylinder	<i>NCrossings</i> , <i>dRadius</i>
Sphere	<i>NCrossings</i> , <i>dRadius</i>

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 boundary point.

Y: Required **Long** that indicates the Y value of the boundary point.

Z: Required **Long** that indicates the Z value of the boundary point.

Remarks

This function works with patch scans. Use the *boundarypointcount* property to set the number of boundary points.

BasicScan.SetControlPoint

Syntax

Return Value=*expression*.SetControlPoint(*Index*)

Return Value: Returns a boolean value. If set to 1 (True), the control point is set at the specified index.

expression: Required expression that evaluates to a PC-DMIS **BasicScanCommand** object.

Index: Required **Long** value which indicates which control point to set.

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

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.

BasicScan.SetMethodParams

Syntax

Return Value=expression.SetMethodParams (bln, 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.

bln: Required variable that sets 0 for Inside scans, 1 for Outside scans, and 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 (<i>bln, bCenteringType, nCenteringDirection, dDiameter, dArcAngle, dDepth, dPitch</i>)
Linear	None
Edge	None
Circle	<i>bln, , , dDiameter, dArcAngle, dDepth</i>
Cylinder	<i>bln, , , dDiameter, dArcAngle, dDepth, dPitch</i>
Str Line	None
Center	<i>, bCenteringType, nCenteringDirection</i>

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 **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.

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.

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 **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.

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).

Related Topics: BasicScan.Method Property, BasicScan.Filter Property, BasicScan.OperationMode Property, BasicScan.HitType Property, BasicScan.NominalMode Property, BasicScan.BoundaryCondition Property

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

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

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

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

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

CadModel Object Overview:

The **CadModel** object allows you to work with the imported CAD model in PC-DMIS' Graphics Display window.

Methods:

CadModel.HighlightElement

Syntax

Return Value=*expression*.HighlightElement(*Name*, *All*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds in highlighting the specified CAD element, false if it fails.

expression: Required expression that evaluates to **CadModel** object.

Name: Required case-sensitive **String** that indicates the CAD element to highlight.

All: Boolean value that indicates whether or not all CAD elements of that have *Name* should be highlighted. If set to TRUE then all elements with *Name* are selected. If set to FALSE, then only the first item in the list that has *Name* is selected.

Remarks:

This method highlights the specified CAD element (or elements) on the CAD model in the Graphics Display window.

Sample Code:

```
Dim App As PCDLRN.Application

Set App = CreateObject("PCDLRN.Application")

Dim Parts As PCDLRN.PartPrograms

Set Parts = App.PartPrograms

Dim Part As PCDLRN.PartProgram

Set Part = App.ActivePartProgram

Dim CADMod As PCDLRN.CadModel

Set CADMod = Part.CadModel

Dim strElement As String

Dim boolYesNo As Boolean

strElement = InputBox("Type the CAD element to highlight", "Highlight CAD")

boolYesNo = MsgBox("Select all?", vbYesNo, "Select All")

If CADMod.HighlightElement(strElement, boolYesNo) = False Then

    MsgBox "Element: " & strElement & " couldn't be highlighted", vbCritical, "No CAD Highlighted"

Else

    MsgBox "Success. Element: " & strElement & " was highlighted", vbOKOnly, "CAD Highlighted"

End If
```

CadModel.UnHighLightElement

Syntax

Return Value=expression.UnHighlightElement(*Name*, *All*)

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds in removing the highlight from the specified CAD element, false if it fails.

expression: Required expression that evaluates to **CadModel** object.

Name: Required case-sensitive **String** that indicates the CAD element from which to remove highlighting.

All: Boolean value that indicates whether or not all CAD elements of that have *Name* should be deselected. If set to TRUE then all elements with *Name* are deselected. If set to FALSE, then only the first item in the list that has *Name* is deselected.

Remarks:

This method removes highlighting from the specified CAD element (or elements) on the CAD model in the Graphics Display window.

CadWindow Object Overview:

The **CadWindow** object is the one and only cad window for a part program.

Related Topics: CadWindows Object Overview

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 **ActivePartProgram** property returns a **PartProgram** object.

CadWindow.Height

The height of the Cad window in screen pixels. Read/Write **Long**.

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.Parent

Returns the parent **CadWindows** object. Read-only.

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.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.Width

The width of the Cad window in screen pixels. Read/Write **Long**.

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.SelectCADObject

Syntax

Return Value=expression.SelectCADObject(long Option, BOOL On)

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: An expression evaluating to a long which represents a valid cad object.

On: An expression that evaluates to a boolean value. This should be true if the cad object is to be selected, and false if the cad object is to be deselected.

This method selects or deselects a cad object.

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.

Related Topics: CadWindow Object Overview

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.

CadWindows.Count

Returns the number of CadWindow objects active in this part program. Read-only **Long**.

Currently, this property always returns one.

CadWindows.Parent

Represents the parent **PartProgram** object. Read-only.

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.

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.

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.SphereID

ID of a sphere command that occurs prior to the calibration command. The sphere should have identical characteristics with the tool identified by ToolID.

Read/Write **String**

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.

Command Object Overview

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.

The Command object is also a "collection object" as it represents:

- the collection of executions of this object in the current execution.
- 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. **Nothing** otherwise. Read-only.

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
```

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.

Command.ArrayIndexCommand

Returns an ArrayIndex Command object if Command is of *Type* ARRAY_INDEX. Returns **Nothing** otherwise. Read-only.

Command.AttachCommand

Returns an Attach Command object if Command is of *Type* ATTACH_PROGRAM. Returns **Nothing** otherwise. Read-only.

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.

Command.CalibrationCommand

Returns a Calibration Command object if Command is of *Type* CALIB_SPHERE. Otherwise it returns **Nothing**. Read-only.

Command.CommentCommand

Returns a Calibration Command object if Command is of *Type* SET_COMMENT. Otherwise it returns **Nothing**. Read-only.

Command.CopyMeastoNom

Property used to indicate/set whether the object should execute in MASTER mode. After executing in MASTER mode, the object copies the measured vector, centroid, and other nominal information to the nominals and turns off MASTER mode. This copies the same information that gets calculated using the CalculateNominals method. Read/Write **Boolean**.

The nominal information that this copies included the following:

- CENTROID
- VECTOR
- DIAMETER
- STARTPOINT
- ENDPOINT
- BALLCENTER
- LENGTH
- ELLIPSEMINORAXIS
- ANGLE
- SURFACEVECTOR
- THICKNESS
- NUMHITS
- SPACER
- INDENT
- AUTO_MOVE_DISTANCE
- DEPTH
- TARG
- SCANROWCOUNT
- ANGLE_VECTOR
- PUNCH_VECTOR
- PIN_VECTOR
- PIN_DIAMETER
- REPORT_VECTOR
- REPORT_SURF_VECTOR
- HEIGHT
- MEASURE_VECTOR

- UPDATE_VECTOR
- CORNER_RADIUS
- ANGLE2

Command.Count

Represents the number of copies of this **Command** which are available. If the part program is currently being executed, it is the number of times it has been executed so far in the current execution cycle. 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.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

Command.DimFormatCommand

Returns a DimFormat Command object if Command is of *Type* DIMENSION_FORMAT. Otherwise it returns **Nothing**. Read-only.

Command.DimInfoCommand

Returns a DimInfo Command object if Command is of *Type* DIMENSION_INFORMATION. Otherwise it returns **Nothing**. Read-only.

Command.DisplayMetaFileCommand

Returns a DispMetaFile Command object if Command is of *Type* DISPLAY_METAFILE. Otherwise it returns **Nothing**. Read-only.

Command.ExternalCommand

Returns an ExternalCommand Command object if Command is of *Type* EXTERNAL_COMMAND. Otherwise it returns **Nothing**. Read-only.

Command.Feature

Represents the kind of feature that this **Command** object is. If it is not a feature it will return F_NONE. Otherwise it will return a value from the following list. Read-only
 ENUM_FEATURE_TYPES.

Type of Feature	Return Value
POINT	F_POINT
CIRCLE	F_CIRCLE
SPHERE	F_SPHERE
LINE	F_LINE
CONE	F_CONE
CYLINDER	F_CYLINDER
PLANE	F_PLANE
CURVE	F_CURVE
SLOT	F_SLOT
SET	F_SET
ELLIPSE	F_ELLIPSE
SURFACE	F_SURFACE

Command.FeatureCommand

Returns this **Command** object as an **FeatCommand** object if it can, **Nothing** otherwise. Read-only.

The **Commands** that have the following *Type* can become **FeatCommand** 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

Command.FileIOCommand

Returns a FileIO Command object if Command is of *Type* FILE_IO_OBJECT. Otherwise it returns **Nothing**. Read-only.

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

```

Command.GetFieldValue

This read-only property returns the value of a field from a command. If you try to access a field that isn't supported by the command, PC-DMIS returns FALSE. This property takes two parameters, the first parameter defines the field item. You can select this item from an enumerated list or use the associated constant number. The second parameter specifies the *TypeIndex*.

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. When using the index property on a field type that can have a variable number of fields, the index must not be greater than the current number of fields (of the type being changed) + 1.

Example:

Suppose your part program has a circle named CIR1. This example would return the theoretical diameter of that circle:

```

Set myapp = CreateObject("pcdlrn.application")
Set myprog = myapp.ActivePartProgram
Dim Cnds As Commands
Set Cnds = myprog.Commands
Dim Cmd As Command
Set Cmd = Cnds.Item("CIR1")
MsgBox Cmd.ID
MsgBox Cmd.GetFieldValue(THEO_DIAM, 0)

```

Command.GetToggleValue

This read-only property checks a command's field and returns 0 if it isn't a toggle field. It also returns 0 if the field doesn't exist. Otherwise, it returns a the current toggle index value, with 1 as the base index value.

It takes two parameters. The first is an enumerated field type value to determine what field to check in a command, and the second is the *TypeIndex*.

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 specified field type. In these cases, the *TypeIndex* must not be greater than the current number of fields of the that type + 1 and *TypeIndex* must be greater than 1. For fields that allow only a single value, the *TypeIndex* is 0.

Example:

Suppose your part program has a circle named CIR1. This example would return a 1 since the COORD_TYPE field is a toggle field:

```
Set myapp = CreateObject("pcdlrn.application")
Set myprog = myapp.ActivePartProgram
Dim Cnds As Commands
Set Cnds = myprog.Commands
Dim Cmd As Command
Set Cmd = Cnds.Item("CIR1")
MsgBox Cmd.ID
MsgBox Cmd.GetToggleValue(COORD_TYPE, 0)
```

Command.HasBreakPoint

Returns TRUE if the current PC-DMIS command has a breakpoint. Read/Write **BOOL**.

Remarks

You can also use this property to automatically set or clear breakpoints on individual commands by setting the HasBreakPoint property for the command to TRUE or FALSE.

Related Topics: [Commands.ClearAllBreakPoints](#)

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.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.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.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.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.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.IsConstructedFeature

Returns TRUE if the command is a constructed feature. Read only **BOOL**.

Command.IsDCCFeature

Returns TRUE if the command is a DCC (Auto) Feature. Read only **BOOL**.

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.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.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.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.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 FlowControlCommand Property.

Command.IsHit

Returns TRUE if the command is a one of the hit command types. Read only **BOOL**.

Command.IsLeapFrog

Returns TRUE if the command is a Leapfrog command. Read only **BOOL**.

Command.IsLeitzMotion

Returns TRUE if the command is a LeitzMot command. Read only **BOOL**.

Remarks

Commands that return TRUE for this property can successfully retrieve a LeitzMotion 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.IsMeasuredFeature

Returns TRUE if the command is a measured feature command. Read only **BOOL**.

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.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.IsOptionProbe

Returns TRUE if the command is an option probe command. Read only **BOOL**.

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.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.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.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.LeapfrogCommand

Returns a Leapfrog Command object if Command is of *Type* LEAPFROG. Otherwise it returns **Nothing**. Read-only.

Command.LeitzMotionCommand

Returns a LietzMotion Command object if Command is of *Type* OPTIONPROBE. Otherwise it returns **Nothing**. Read-only.

Command.LoadMachineCommand

Returns a LoadMachine Command object if Command is of *Type* GET_MACHINE_DATA. Otherwise it returns **Nothing**. Read-only.

Command.LoadProbeCommand

Returns a LoadProbe Command object if Command is of *Type* GET_PROBE_DATA. Otherwise it returns **Nothing**. Read-only.

Command.Marked

Property used to indicate/set whether command is marked in the edit window. Read/Write **Boolean**.

Command.MissedHit

This property checks whether or not a missed hit occurred on the last executed instance of the specified command. Read-only **Boolean**.

PC-DMIS returns TRUE if the command missed a hit; FALSE otherwise.

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

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,

Command.OptionProbeCommand

Returns an OptMotion Command object if Command is of *Type* OPTIONPROBE. Otherwise it returns **Nothing**. Read-only.

Command.OptMotionCommand

Returns an OptMotion Command object if Command is of *Type* OPTIONMOTION. Otherwise it returns **Nothing**. Read-only.

Command.Parent

Returns the parent **Commands** collection object. Read-only.

Command.ScanCommand

Returns a Scan Command object if Command is of *Type* DCCSCAN_OBJECT or *Type* MANSCAN_OBJECT. Otherwise it returns **Nothing**. Read-only.

Command.ShowIDOnCad

Property used to indicate/set whether the command ID should be displayed in the CAD window. Read/Write **Boolean**

Command.Skipped

Property used to indicate whether the a command was skipped over. Read-only **Boolean**.

PC-DMIS returns FALSE if the command was not skipped and TRUE if it was.

Command.SlaveArm

Property used to indicate/set whether command is a slave arm object. Read/Write **Boolean**

Command.StatisticCommand

Returns a Statistics Command object if Command is of *Type* STATISTICS. Otherwise it returns **Nothing**. Read-only.

Command.TempCompCommand

Returns a TempComp Command object if Command is of *Type* TEMP_COMP. Otherwise it returns **Nothing**. Read-only.

Command.TraceFieldCommand

Returns a TraceField Command object if Command is of *Type* TRACEFIELD. Otherwise it returns **Nothing**. Read-only.

Command.TracksErrors

Property used to determine whether or not the script will handle errors for the specified command. Read/Write **Boolean**.

PC-DMIS returns TRUE if error handling is turned on for the specified command; FALSE otherwise.

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.

Command.TypeDescription

Returns a human-readable description of *Type* of the object. Read-only **String**

For example, an object of type CONST_OFF_PLANE has the string “Constructed Offset Plane” returned by this function.

Command.UnexpectedHit

This property checks whether or not an unexpected hit occurred on the last executed instance of the specified command. Read-only **Boolean**.

PC-DMIS returns TRUE if it detects an unexpected hit for the command; FALSE otherwise.

Methods:

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.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.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.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 from which field the expression is being retrieved. 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.

Beginning with PC-DMIS version 3.5, the following are now a part of the **ENUM_FIELD_TYPES** enumeration:

SOLID = 416

FIT = 452

TRACE_VALUE_LIMIT = 473

ROI_DIRECTION = 474

ROI_CENTER_X = 475

ROI_CENTER_Y = 476

CENTER_ROTATION_THEO = 477

CENTER_ROTATION_MEAS = 478

Command.GetText

Syntax

expression.GetText(*FieldType*, *TypeIndex*)

Return Value: **String** which is the text on the given field.

expression: Required expression that evaluates to a PC-DMIS **Command** object.

FieldType: Used to indicate the field from which the text is being retrieved. 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 text of the indicated field of the command.

Remarks

Use this command to get text that is displayed in the edit window 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.

Beginning with PC-DMIS version 3.5, the following are now a part of the **ENUM_FIELD_TYPES** enumeration:

SOLID = 416

FIT = 452

TRACE_VALUE_LIMIT = 473

ROI_DIRECTION = 474

ROI_CENTER_X = 475

ROI_CENTER_Y = 476

CENTER_ROTATION_THEO = 477

CENTER_ROTATION_MEAS = 478

Command.GetToggleString

Syntax

Return Value=*expression*.GetToggleString(*FieldType*,*TypeIndex*)

Return Value: The return value is the string of text if the field is a toggle field, otherwise it returns an empty string.

expression: Required expression that evaluates to a **Command** object.

FieldType: Used to indicate the field from which the text is being retrieved. 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.

Beginning with PC-DMIS version 3.5, the following are now a part of the **ENUM_FIELD_TYPES** enumeration:

SOLID = 416

FIT = 452

TRACE_VALUE_LIMIT = 473

ROI_DIRECTION = 474

ROI_CENTER_X = 475

ROI_CENTER_Y = 476

CENTER_ROTATION_THEO = 477

CENTER_ROTATION_MEAS = 478

Command.GetUniqueID

Syntax

Return Value=expression.GetUniqueID(HiPart, LoPart)

This command retrieves the low and high parts of the 64-bit unique id of the command.

Return Value: This method doesn't have a return value.

Expression: Required expression that evaluates to a PC-DMIS **Command** object.

HiPart: **Long** value used to indicate the high part of the 64-bit unique id of the command.

LoPart: **Long** value used to indicate the low part of the 64-bit unique id of the command.

Command.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 **Command** object.

Expression: Required **String** that is the expression to evaluate for validity.

This function returns TRUE if the expression is valid, and FALSE otherwise.

Command.Item

Syntax

Return value=expression.Item(Num)

Return Value: The Item function returns a **Command** object.

expression: Required expression that evaluates to a **Command** 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.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.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.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.PutText

Syntax

expression.PutText(NewVal,FieldType,TypeIndex)

Return Value: **TRUE** if the field's text was set successfully, **FALSE** otherwise.

expression: Required expression that evaluates to a PC-DMIS **Command** object.

NewVal: **STRING** to put into the indicated field.

FieldType: Used to indicate which field into which the text is being put. 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. When using the index property on a field type that can have a variable number of fields, the index must not be greater than the current number of fields (of the type being changed) + 1.

Puts the text in the indicated field of the command.

Remarks

Use this command to put text that is displayed in the Edit window 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:

1. Creating the desired object in PC-DMIS
2. Inserting the desired expression in the desired field
3. Exporting (posting out) the containing part program to BASIC.

If the field already has an expression in it, the expression is removed.

Beginning with PC-DMIS version 3.5, the following are now a part of the **ENUM_FIELD_TYPES** enumeration:

SOLID = 416

FIT = 452

TRACE_VALUE_LIMIT = 473

ROI_DIRECTION = 474

ROI_CENTER_X = 475

ROI_CENTER_Y = 476

CENTER_ROTATION_THEO = 477

CENTER_ROTATION_MEAS = 478

Command.ReDraw

This method requests that the object be redrawn in the Edit window. This method has no return value.

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.RemoveExpression

Syntax

expression.RemoveExpression(*FieldType*, *TypeIndex*)

This method has no return value.

expression: Required expression that evaluates to a PC-DMIS **Command** object.

FieldType: Used to indicate the field from which the expression is being removed. 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.

Removes the expression from the indicated field of the command.

Remarks

Use this command to remove expressions from 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:

1. Creating the desired object in PC-DMIS
2. Inserting the desired expression in the desired field
3. Exporting (posting out) the containing part program to BASIC.

Beginning with PC-DMIS version 3.5, the following are now a part of the **ENUM_FIELD_TYPES** enumeration:

SOLID = 416

FIT = 452

TRACE_VALUE_LIMIT = 473

ROI_DIRECTION = 474

ROI_CENTER_X = 475

ROI_CENTER_Y = 476

CENTER_ROTATION_THEO = 477

CENTER_ROTATION_MEAS = 478

Command.SetExpression

Syntax

command.SetExpression(Expression, FieldType, TypeIndex)

Return Value: This function returns a **Boolean**, TRUE if the expression was successfully set, FALSE otherwise

command: Required expression that evaluates to a PC-DMIS **Command** object.

Expression: **String** to which to set the expression.

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.

Sets the expression of the indicated field of the command.

Remarks

Use this command to set 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:

1. Creating the desired object in PC-DMIS
2. Inserting the desired expression in the desired field
3. Exporting (posting out) the containing part program to BASIC.

Beginning with PC-DMIS version 3.5, the following are now a part of the **ENUM_FIELD_TYPES** enumeration:

SOLID = 416

FIT = 452

TRACE_VALUE_LIMIT = 473

ROI_DIRECTION = 474

ROI_CENTER_X = 475

ROI_CENTER_Y = 476

CENTER_ROTATION_THEO = 477

CENTER_ROTATION_MEAS = 478

Command.SetToggleString

Syntax:

Boolean *Command.SetToggleString(long ToggleIndex, ENUM_FIELD_TYPES DataType, long TypeIndex)*

Return Value: This method returns a Boolean. It returns True if the underlying Command object exists in PC-DMIS the appropriate DataType and TypeIndex parameters successfully set the toggle string. Otherwise, it returns false.

Command: Required expression that evaluates to a PC-DMIS **Command** object.

ToggleIndex: long value representing the new toggle field that will be set.

- If you pass a number less than 1, it will set the field to the first string.
- If you pass a number larger than the number of possible strings, it will set the field to the last string.

DataType: Indicate the toggle field being changed. Type **ENUM_FIELD_TYPES** enumeration.

TypeIndex: long value indicating the instance of the supplied field type to use when an object has more than one instance of a field type.

Remarks

This method lets you set a toggle field in a numerical, language-independent way.

Example:

The Measured Circle has a BF_MATH_TYPE field that takes one of "QUAD MIN", "SEP MIN", "MAX ISCR", "MIN CIRCOS", or "RAG FISSO" in Italian, or "LEAST_SQR", "MIN_SEP", "MAX_INSC", "MIN_CIRCSC", or "FIXED_RAD" in English. If we do not know which language we are importing into, we can not know which phrase to use.

However, the ENUM_FIELD_TYPES number is the same in both languages, e.g., "MIN CIRCOS" and "MIN_CIRCSC" are both string number 4.

The SetToggleString method can be called in the following fashion to set the BF_MATH_TYPE to use the minimum circumscribed method:

```
retval = DmisCommand.SetToggleString(4, BF_MATH_TYPE,0)
```

Command.SolveExpression

Syntax

Return Value=expr.SolveExpression(Expression)

Return Value: This method returns a variable object if the solved expression is valid.

expr: Required expression that evaluates to a PC-DMIS **Command** object.

Expression: **String** value of the expression to solve.

Remarks

SolveExpression evaluates the expression based on the objects above the command on which SolveExpression gets called.

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.

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.

Commands.Count

Represents the number of Command objects in the parent **PartProgram** object. Read-only **Long**.

Commands.CurrentCommand

Returns a **Command** object representing the current PC-DMIS command. Note that if you use the Commands.Add method prior to this property, the current command returned will be the last added command from the Add method.

Read-only **Command** object.

Commands.LastCommand

Returns a **Command** object representing the last command in the part program. This gives you a faster way of getting the last command. Before you had to use this syntax:

```
Commands.Item(Commands.Count).
```

Read-only **Command** object.

Commands.Parent

Returns the parent **PartProgram** object. Read-only.

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.

Remarks

PC-DMIS only supports one way for adding commands while executing a part program in PC-DMIS: Insert a script command (select **Insert | Basic Script** from within PC-DMIS) that points to the BASIC script containing the **Add** method. Otherwise, you will need to run your script with the **Add** method first and control part program execution from within your script.

Beginning with PC-DMIS version 3.5, the following are now a part of the **OBTYPE** enumeration for this method:

MOVE_ALL = 162

DIMENSION_SYMMETRY = 1115

CONST_SCAN_SEG_ARC = 527

CONST_SCAN_SEG_LINE = 539

CONST_BFRE_ELLIPSE = 580

CONST_BF_ELLIPSE = 581

CONST_PROJ_ELLIPSE = 582

CONST_REV_ELLIPSE = 583

CONST_CAST_ELLIPSE = 584

CONST_INT_ELLIPSE = 585

Commands.ClearAllBreakPoints

Syntax

Return Value=expression.ClearAllBreakPoints

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 the breakpoints on all **Command** objects in the collection. You should use this method if you don't want to step through the execution of a part program.

Related Topics: Command.HasBreakPoint Property, Command Object 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 the collection. ClearMarked always returns TRUE.

Commands.FindByUniqueID

Syntax

Return Value=expression.FindByUniqueID(HiPart,LoPart)

Return Value: The FindByUniqueID function returns the command identified by the *LoPart* and *HiPart* values.

expression: Required expression that evaluates to a PC-DMIS **Commands** object.

LoPart: This parameter is a long value that should come from a call to GetUniqueID made previously on the command object.

HiPart: This parameter is a long value that should come from a call to GetUniqueID made previously on the desired command object.

Commands.GetCommandText

Syntax

Return Value=expression.GetCommandText(Cmd)

Return Value: This method returns a string value of the current command's text if the function succeeds.

expression: Required expression that evaluates to a PC-DMIS **Commands** object.

Cmd: Required **Command** object that indicates the command from which to read the command text.

This command returns all the lines of text for the current command object. If your command occupies more than one line in Edit window's command mode, the function returns text for all the lines in the command. However, this command only returns text for the current command object. For example, if your edit window had a circle:

CIR1 = FEAT/CIRCLE,...

THEO/...

ACTL/...

The returned string would not contain the hits because they are actually different command objects.

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.

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*.

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

Comment Object Overview

The Comment Automation object gives access to the properties of the PC-DMIS Comment command.

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.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.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.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.GetLine

Syntax:

expression.GetLine (Line)

Return Value: String text of the line of the comment specified by the line parameter. 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.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.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.

ControlPoint Object Overview

With the ControlPoint object you can insert control point locations. These locations interrupt the normal scan and alter scan speed, point density or both for defined portions of the scan. The ControlPoint object is used with only the following scans:

- LinearOpen
- LinearClose
- Patch
- Section
- Line (Basic Scan)

Additionally, the ControlPoint object only works on machines that use an analog probe that allows continuous contact scanning.

In the *PC-DMIS Reference Manual*, the Control Points are called Interrupt Points. See "Interrupts" in the "Scanning Your Part" chapter of that documentation.

Properties:

ControlPoint.Crossings

This defines the number of time the scan crosses the specified boundary before the alterations defined by the crossing point take effect. Read/write **Long**.

ControlPoint.I

This specifies the I component of the interrupt location's IJK vector. Read/write **Double**.

ControlPoint.J

This specifies the J component of the interrupt location's IJK vector. Read/write **Double**.

ControlPoint.K

This specifies the K component of the interrupt location IJK vector. Read/write **Double**.

ControlPoint.PointDensity

This defines the density of points per millimeter the scan should take after it encounters the control point. Read/write **Long**.

ControlPoint.Radius

This defines the Radius of circular control point types (Cones, Spheres, Cylinders). Read/write **Double**.

ControlPoint.Type

This specifies the type of control point. Read/write **BSCTRLPT_ENUM**.

There are four types of control points:

1. Plane (uses these elements: Plane, X,Y,Z,I,J,K,Num Crossings, Scan Speed, Point Density)
2. Sphere (uses these elements: Sphere, X, Y, Z, I, J, K, Num Crossings, Scan Speed, Point Density, Diameter)
3. Cone (uses these elements: Cone, X, Y, Z, I, J, K, Num Crossings, Scan Speed, Point Density, Angle)
4. Cylinder (uses these elements: Cylinder, X, Y, Z, I, J, K, Num Crossings, Scan Speed, Point Density, Diameter)

ControlPoint.X

This specifies the X value of the interrupt XYZ location. Read/write **Double**.

ControlPoint.Y

This specifies the Y component of the interrupt XYZ location. Read/write **Double**.

ControlPoint.Z

This specifies the Z component of the interrupt XYZ location. Read/write **Double**.

DataType Object Overview

The DataType Object allows you to return objects of information about a particular data type or field.

Properties:

DataType.Application

This returns the Application Object. Read only.

DataType.Count

This returns the number of instances of this data type in command. Read only **Long**.

DataType.Description

This returns a description of the data type. Read only **String**.

DataType.Parent

This returns the Parent Command Object. Read only.

DataType.Type

This returns the field type of the data type. Read only **DATA_TYPE_TYPES**.

DataType.Value

This returns the Default Property for a Field Type Number. Read only **Long**.

DataTypes Object Overview

The DataTypes Object allows you to return objects of varying data types.

Properties:

DataTypes.Application

This returns the Application Object. Read only.

DataTypes.Count

This returns the number of data type information objects in the data type collection. Read only **Long**.

DataTypes.Parent

This returns the Parent Command Object. Read only.

Related Topics: [Command.Parent](#)

Methods:

DataTypes.GetDataTypeInfo

Syntax

Return Value=expression.GetDataTypeInfo(DataType)

Return Value: This returns the specified data type information object if supported by the data type collection.

expression: Required expression that evaluates to a PC-DMIS **DataTypes** object.

DataType: **ENUM_FIELD_TYPES** value specifying the DataType information object to return.

Remarks

Beginning with PC-DMIS version 3.5, the following are now a part of the **ENUM_FIELD_TYPES** enumeration:

SOLID = 416

FIT = 452

TRACE_VALUE_LIMIT = 473

ROI_DIRECTION = 474

ROI_CENTER_X = 475

ROI_CENTER_Y = 476

CENTER_ROTATION_THEO = 477

CENTER_ROTATION_MEAS = 478

DataTypes.Item

Syntax

Return Value=expression.item(Num)

Return Value: This returns the data type information object specified by the *Num* value from the data type collection.

expression: Required expression that evaluates to a PC-DMIS **DataTypes** object.

Num: **Long** value specifying the data type information object.

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

Properties:

DimData.Bonus

Represents the Bonus member of this object. Read/write **Double**.

DimData.Dev

Represents the Dev member of this object. Read/write **Double**.

Remarks

The Dev member is the default property.

DimData.DevAngle

Represents the DevAngle member of this object. Read/write **Double**.

DimData.Max

Represents the Max member of this object. Read/write **Double**.

DimData.Meas

Represents the Meas member of this object. Read/write **Double**.

DimData.Min

Represents the Min member of this object. Read/write **Double**.

DimData.Minus

Represents the Minus member of this object. Read/write **Double**.

DimData.Nom

Represents the Nom member of this object. Read/write **Double**.

DimData.Out

Represents the Out member of this object. Read/write **Double**.

DimData.Plus

Represents the Plus member of this object. Read/write **Double**.

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.

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.

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.

DimensionCommand.DevAngle

Returns the deviation angle of a dimension. Read/Write **Double**.

DimensionCommand.Deviation

Returns the deviation of a dimension. Read/Write **Double**.

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 affect and getting it returns the empty string. Also, objects of type DIMENSION_KEYIN have no associated features.

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.

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.

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 affect and getting it returns the empty string.

DimensionCommand.IsLocationAxis

boolean value... no help string associated

DimensionCommand.IsTruePosAxis

boolean value... no help string associated

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.

DimensionCommand.Max

Returns the maximum value of a dimension. Read-only **Double**.

DimensionCommand.Measured

Returns the measured value of a dimension. Read-only **Double**.

DimensionCommand.Min

Returns the minimum value of a dimension. Read-only **Double**.

DimensionCommand.Minus

Represents the negative tolerance of a dimension. Read/write **Double**.

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.

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.ParallelPerpendicular

Indicates whether calculations are performed parallel or perpendicular to input for 2-D dimensions. Possible values include the following:

DIM_PERPENDICULAR

DIM_PARALLEL

Read/Write **Enum_Dim_Perp_Parallel Enumeration**.

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.

DimensionCommand.Plus

Returns the positive tolerance of a dimension. Read-only **Double**.

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.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**.

Methods:

DimensionCommand.Evaluate

Syntax:

expression.DimensionCmd.Evaluate

Return Value: Boolean indicating success or failure in evaluating the dimension.

expression: Required expression that evaluates to a PC-DMIS DimensionCmd (dimension command) object.

Evaluates a dimension's data from its feature data.

Remarks:

Some dimension commands exist as command blocks inside of PC-DMIS. Because of this, the Evaluate method only works on a command block if you call the method from the very first item of the block. Calls made from other items of a dimension's command block won't function.

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*.

Properties:

DimFormat.ShowDevSymbols

BOOLEAN value representing whether deviation symbols should be shown in the dimension report text.

Read/Write **Boolean**

DimFormat.ShowDimensionText

BOOLEAN value indicating whether the top two lines of the dimension command should appear or not.

Read/Write **Boolean**

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**

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**

DimFormat.ShowStdDev

BOOLEAN value indicating whether the standard deviation value should appear or not.

Read/Write **Boolean**

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.

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 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.

Properties:

DimInfo.DimensionID

STRING value representing the name of the dimension for which the dimension information object will be showing information.

Read/Write **String**

DimInfo.ShowDimensionID

BOOLEAN value indicating whether the Dimension ID should be shown in the dimension information object.

Read/Write **Boolean**

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**

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.

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.

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.

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

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

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

Display Metafile Object Overview

The Display Metafile Automation object gives access to the comment properties of the PC-DMIS Display Metafile command.

Properties:

DispMetafile.Comment

STRING value representing the comment to be used as a caption for the metafile object.

Read/Write **String**

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.

Properties:

DmisDialog.Visible

Indicates whether the dialog is still visible to the user.

Read Only: **Boolean**

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.

Properties:

DmisMatrix.Copy

Returns a copy of the matrix.

Read Only: **DmisMatrix**

DmisMatrix.Inverse

Returns an inverse matrix of the current matrix.

Read Only: **DmisMatrix**

DmisMatrix.IsIdentity

BOOLEAN property set to true if the matrix is the identity matrix.

Read Only: **Boolean**

DmisMatrix.OffsetAxis

The first set of three doubles in the matrix representing the translation offset of the matrix.

Read/Write: **PointData**

DmisMatrix.PrimaryAxis

The second set of three doubles in the matrix representing the matrix's primary axis.

Read/Write **PointData**

DmisMatrix.SecondaryAxis

The third set of three doubles in the matrix representing the matrix's secondary axis.

Read/Write **PointData**

DmisMatrix.TertiaryAxis

The fourth set of three doubles in the matrix representing the matrix's tertiary axis.

Read/Write **PointData**

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.Multiply

Syntax:

expression.Multiply (SecondMatrix)

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

SecondMatrix: Required parameter of type **DmisMatrix** representing the second matrix.

Return Value:

Matrix that is the result of multiplying the two matrices of type **DmisMatrix**.

DmisMatrix.Normalize

Syntax:

expression.Normalize ()

expression: Required expression that evaluates to a PC-DMIS **DmisMatrix** object.

Remarks

Normalizes the matrix.

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.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 parameter that uses the **ENUM_PLANE_TYPE** enumeration to define which axis to rotate about. Options include PLANE_TOP, PLANE_RIGHT, PLANE_BACK, PLANE_BOTTOM, PLANE_LEFT, and PLANE_FRONT. It defaults to PLANE_TOP. You can also use these numerical **long** values:

- PLANE_TOP = 0
- PLANE_RIGHT = 1
- PLANE_BACK = 2
- PLANE_BOTTOM = 3
- PLANE_LEFT = 4
- PLANE_FRONT = 5

Remarks

Rotates the matrix by the specified angle relative to the workplane.

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 parameter with the **ENUM_PLANE_TYPE** enumeration used to define which axis to rotate about. Options include PLANE_TOP, PLANE_RIGHT, PLANE_BACK, PLANE_BOTTOM, PLANE_LEFT, and PLANE_FRONT. It defaults to PLANE_TOP. You can also use these numerical **long** values:

- PLANE_TOP = 0
- PLANE_RIGHT = 1
- PLANE_BACK = 2
- PLANE_BOTTOM = 3
- PLANE_LEFT = 4
- PLANE_FRONT = 5

Remarks

Rotates the matrix by the calculated angle relative to the workplane.

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 parameter with the **ENUM_PLANE_TYPE** enumeration used to define which axis to rotate about. Options include PLANE_TOP, PLANE_RIGHT, PLANE_BACK, PLANE_BOTTOM, PLANE_LEFT, and PLANE_FRONT. It defaults to PLANE_TOP. You can also use these numerical **long** values:

- PLANE_TOP = 0
- PLANE_RIGHT = 1
- PLANE_BACK = 2
- PLANE_BOTTOM = 3
- PLANE_LEFT = 4
- PLANE_FRONT = 5

Remarks

Rotates the primary axis (as determined by the workplane parameter) to the specified vector.

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.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 modified 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 parameter with the **ENUM_PLANE_TYPE** enumeration used to define which axis to rotate about. Options include PLANE_TOP, PLANE_RIGHT, PLANE_BACK, PLANE_BOTTOM, PLANE_LEFT, and PLANE_FRONT. It defaults to PLANE_TOP. You can also use these numerical **long** values:

- PLANE_TOP = 0
- PLANE_RIGHT = 1
- PLANE_BACK = 2
- PLANE_BOTTOM = 3
- PLANE_LEFT = 4
- PLANE_FRONT = 5

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.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 modified 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 parameter with the **ENUM_PLANE_TYPE** enumeration used to define which axis to rotate about. Options include PLANE_TOP, PLANE_RIGHT, PLANE_BACK, PLANE_BOTTOM, PLANE_LEFT, and PLANE_FRONT. It defaults to PLANE_TOP. You can also use these numerical **long** values:

- PLANE_TOP = 0
- PLANE_RIGHT = 1
- PLANE_BACK = 2
- PLANE_BOTTOM = 3
- PLANE_LEFT = 4
- PLANE_FRONT = 5

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.

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.

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.

EditWindow.Height

The height of the edit window in screen pixels. Read/Write **Long**.

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.Parent

Returns the parent PartProgram of this object. Read-only **PartProgram**.

EditWindow.ShowAlignments

This property is TRUE if alignments are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

EditWindow.ShowComments

This property is TRUE if comments are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

EditWindow.ShowDimensions

This property is TRUE if dimensions are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

EditWindow.ShowFeatures

This property is TRUE if features are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

EditWindow.ShowHeaderFooter

This property is TRUE if headers and footers are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

EditWindow.ShowHits

This property is TRUE if hits are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

EditWindow.ShowMoves

This property is TRUE if moves are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

EditWindow.ShowOutToOnly

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.ShowTips

This property is TRUE if tips are being shown in the edit window, FALSE otherwise. Read/Write **Boolean**.

EditWindow.StatusBar

This property represents the text in the edit window's status bar. Read-Write **String**.

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.Visible

This property is TRUE if the edit window is visible, FALSE otherwise. Read/write **Boolean**.

EditWindow.Width

The width of the edit window in screen pixels. Read/Write **Long**.

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.GetCommandText

Syntax

Return Value = expression.GetCommandText(Cmd)

expression: Required expression that evaluates to a PC-DMIS **EditWindow** object.

Return Value = This function returns a string of the current command text for the specified command in *Cmd*.

Cmd = Required expression that evaluates to a Command object.

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.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.SetDMISOutputOptions

Syntax

expression.SetDMISOutputOptions bEnable, FileName, bOverwrite, bOutputTheos, bOutputFeatWithDimensions

expression: Required expression that evaluates to a PC-DMIS **EditWindow** object.

bEnable: This Boolean value determines whether or not PC-DMIS prints the contents of the Edit window as a DMIS output file.

FileName: This string value identifies the filename and path for the created DMIS output file.

bOverwrite: This parameter determines how PC-DMIS outputs the DMIS file. You can choose to append the DMIS output file to an existing file (PCD_DMIS_FILE_APPEND), overwrite an existing file with the new contents (PCD_DMIS_FILE_OVERWRITE), or append to the existing file name a number (PCD_DMIS_FILE_ADD_INDEX).

bOutputTheos: With this parameter you can choose to not include theoretical values in the output DMIS file (PCD_DMIS_OUTPUT_THEOS_NONE), output all theoretical values along with the measured values (PCD_DMIS_OUTPUT_THEOS_ALL), or to only output theoretical values output by the DMIS program (PCD_DMIS_OUTPUT_THEOS_USE_IMPORTED_SETTING).

bOutputFeatWithDimensions: This Boolean value allows you determine whether or not to output the measured features and associated tolerances together in the output file.

This function sets output options for printing the Edit window contents as a DMIS file.

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 allows you to set Edit window print options.

EditWindow.SetPrintOptionsEx

Syntax

expression.SetPrintOptionsEx long Location, long Draft, long FileMode, long ExtNum, FileName, Format, bHyperReportsInline

expression: Required expression that evaluates to a PC-DMIS **EditWindow** object.

Location: Destination of printed data. Options include Off (PCD___OFF), File (PCD_FILE), or Printer (PCD_PRINTER)

Draft: When destination is printer, specifies if printer should print in draft mode or not. Options include On (DMIS_ON) and Off (DMIS_OFF).

FileMode: When the *Location* is set to PCD_FILE, this specifies file naming and writing parameters. Options include: Append (PCD_APPEND), New File (PCD_NEWFILE), Overwrite (PCD_OVERWRITE), and Auto (PCD_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.

FileName: If PCD_FILE is selected for the *Location* parameter, this string value identifies the filename and path for the created file.

Format: If PCD_FILE is selected for the *Location* parameter, the *Format* parameter specifies the file format for the saved file. You can print the Edit window contents to a file in either the RTF (PCD_RTF) format or the PDF (PCD_PDF) format.

bHyperReportsInline: If PCD_PDF is selected for the *Format* parameter, this True or False parameter determines whether or not data from inline HyperView commands appear in the PDF generated file.

This function allows you to set extended Edit window print options.

ExecutedCommands Object Overview

The ExecutedCommands collection object acts much like the Commands collection object except that it only contains a collection of the *executed* commands from the last part program execution, while the Commands collection object contains all the commands in the part program.

Properties:

ExecutedCommands.Application

Syntax

expression.Application

expression: Required expression that evaluates to a PC-DMIS **ExecutedCommands** object.

The Application property returns the application object.

ExecutedCommands.Count

Syntax

expression.Parent

expression: Required expression that evaluates to a PC-DMIS **ExecutedCommands** object.

The Count property returns a number indicating how many commands were executed.

ExecutedCommands.Parent

Syntax

expression.Parent

expression: Required expression that evaluates to a PC-DMIS **ExecutedCommands** object.

The Parent property returns the parent **PartProgram** object.

Methods

ExecutedCommands.FindByUniqueID

Syntax

Return Value=expression.FindByUniqueID(LoPart, HiPart)

Return Value: The `FindByUniqueID` function returns the command identified by the *LoPart* and *HiPart* values.

expression: Required expression that evaluates to an **ExecutedCommands** object.

LoPart: This parameter is a long value that should come from a call to `GetUniqueID` made previously on the command object.

HiPart: This parameter is a long value that should come from a call to `GetUniqueID` made previously on the desired command object.

Example:

```
Dim App As PCDLRN.Application
' Get the applicaton object via CreateObject
Set App = CreateObject("PCDLRN.Application")
Dim Part As PCDLRN.PartProgram
' assume part program is already open
Set Part = App.ActivePartProgram
' Get the entire list of commands in the part program
Dim Cmds As PCDLRN.Commands
Set Cmds = Part.Commands
Dim Cmd As PCDLRN.Command
```

```

' Declare variables for holding the unique id values

Dim HiPart As Long, LoPart As Long

' Loop through all of the commands in the part program

For Each Cmd In Cmds

    ' find the first assignment command in the part program

    ' And save off the unique id for that command

    If Cmd.Type = ASSIGNMENT Then

        Cmd.GetUniqueID HiPart, LoPart

        Exit For

    End If

Next Cmd

' execute the part program

Part.EXECUTE

Dim ExecutedCmds As PCDLRN.ExecutedCommands

' obtain the set of execute commands

Set ExecutedCmds = Part.ExecutedCommands

' check to see if assignment executed

Set Cmd = ExecutedCmds.FindByUniqueID(HiPart, LoPart)

If Not Cmd Is Nothing Then

    MsgBox "Assignment command executed"

End If

ExecutedCommands.Item

```

Syntax

Return Value=expression.Item(Num)

Return Value: The Item method returns the executed command specified by the provided index number in *Num*.

expression: Required expression that evaluates to an **ExecutedCommands** object.

Num: Required **Long** value that indicates which executed command to return. This is the index number of the executed command in the **ExecutedCommands** collection. For example, if you pass 5 in, the 5th command to execute from the last execution would be returned.

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.

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**

FeatCommand Object Overview

Objects of type **FeatureCommand** are created from more generic **Command** objects to pass information specific to the feature command back and forth.

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.

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.

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.

FeatCommand.AutoMove

Auto Move Flag. **Boolean** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an auto move field.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

FeatCommand.Depth

Depth value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a depth field..

Note: The `Command` object allows you to also use the `Ident` property for setting a feature's depth for compatability purposes with older scripts, but you should use the newer `Depth` property for any current implementation.

FeatCommand.Deviation

Auto sphere deviation value. **Double** Read/Write.

Remarks

This property applies only to the PC-DMIS auto sphere command.

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.

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.

FeatCommand.EdgeThickness

Thickness value for edge points. **Double** Read/Write.

Remarks

This property is only applicable for PC-DMIS edge commands.

FeatCommand.EndAngle

End Angle value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an end angle field.

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.

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.

FeatCommand.FindHole

Flag indicating whether or not to use the Find Hole routine. If this value is true, then the Find Hole routine is used. **Boolean** Read/Write.

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.

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.

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.

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.

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.Increment

Increment value for auto high point. **Double** Read/Write.

Remarks

This property is only applicable for the PC-DMIS auto high point command.

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.

Note: For compatibility purposes with older scripts, the Command object allows you to use `Indent` to set an auto circle's depth, however, for any new scripts you should use the newer `Depth` property to do this.

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.

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.

FeatCommand.InitHits

Number of initial sample hits. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a working initial hits field. These include:

- AUTO Angle
- AUTO Circle
- AUTO Cylinder
- AUTO Edge
- AUTO Ellipse
- AUTO Notch
- AUTO Round Slot
- AUTO Sphere
- AUTO Square Slot
- AUTO Surface
- Angle Hit
- Edge Hit

All other features only allow a read-only zero for initial hits.

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.

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.

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.

FeatCommand.MeasAngle

Measured angle value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have an angle field.

FeatCommand.MeasDiam

Measured diameter value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a diameter field.

FeatCommand.MeasHeight

Measured height value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a height field.

FeatCommand.MeasLength

Measured length value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a length field.

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.

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.

FeatCommand.MeasPinDiam

Measured pin diameter value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a pin diameter field.

FeatCommand.MeasSmallLength

Measured shorter length value. **Double** Read Only.

Remarks

This property applies only to PC-DMIS commands that have a small length field.

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.

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.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.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.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.

FeatCommand.PermHits

Number of permanent sample hits. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a working permanent hits field. These include

- AUTO Angle
- AUTO Circle
- AUTO Cylinder
- AUTO Edge
- AUTO Ellipse
- AUTO Notch
- AUTO Round Slot
- AUTO Sphere
- AUTO Square Slot
- AUTO Surface
- Angle Hit
- Edge Hit

All other features only allow a read-only zero for permanent hits.

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.

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.

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

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

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.

FeatCommand.StartAngle

Start Angle value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a start angle field.

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.

FeatCommand.TheoAngle

Theoretical angle value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have an angle field.

FeatCommand.TheoDiam

Theoretical diameter value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a diameter field.

FeatCommand.TheoHeight

Theoretical height value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a height field.

FeatCommand.TheoLength

Theoretical length value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a length field. These include:

- Lines
- Cylinders
- Cones
- Slots
- Notches
- Generic Features

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.

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.

FeatCommand.TheoPinDiam

Theoretical pin diameter value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a pin diameter field.

FeatCommand.TheoSmallLength

Theoretical shorter length value. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a small length field.

FeatCommand.Thickness

Sheet metal (material) thickness. **Double** Read/Write.

Remarks

This property applies only to PC-DMIS commands that have a thickness field.

FeatCommand.Tolerance

Tolerance value for auto high point. **Double** Read/Write.

Remarks

This property applies only to the PC-DMIS auto high point command.

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.

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.CalculateNominals

Syntax

Return Value=expression.CalculateNominals

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 returns TRUE if the function recalculated the feature nominals. FALSE otherwise.

FeatCommand.CountHits

Syntax

Return Value=expression.CountHits

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 returns TRUE if the function recounted the hits for a measured feature. FALSE otherwise.

FeatCommand.Evaluate

Syntax:

Return Value=expression.FeatCmd.Evaluate (type)

Return Value: Boolean value indicating success (if TRUE) or failure (if FALSE) in evaluating the feature.

expression: Required expression that evaluates to a PC-DMIS FeatCmd (**Feat Command**) object.

type: This specifies the type of evaluation to perform. Possible enumerated types that you can use for this parameter include:

EVAL_NOMINALS – Evaluates the feature’s nominals

EVAL_ACTUALS – Evaluates the feature’s actuals

EVAL_BOTH – Evaluates both the feature’s nominals and actuals

Forces an evaluation of a feature without executing it. This takes one parameter that specifies the type of feature evaluation to perform.

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.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,
 FDATA_SLOT_VECTOR,

SCANSEG_START,

SCANSEG_END

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 parameter with the **ENUM_PLANE_TYPE** enumeration for the PARTMM3 and POLAR coordinate system to denote the workplane to be used. Options include `PLANE_TOP`, `PLANE_RIGHT`, `PLANE_BACK`, `PLANE_BOTTOM`, `PLANE_LEFT`, and `PLANE_FRONT`. It defaults to `PLANE_TOP`. You can also use these numerical **long** values:

- `PLANE_TOP = 0`
- `PLANE_RIGHT = 1`
- `PLANE_BACK = 2`
- `PLANE_BOTTOM = 3`
- `PLANE_LEFT = 4`
- `PLANE_FRONT = 5`

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.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 sample hit.

expression: Required expression that evaluates to a PC-DMIS **FeatCommand** object.

Index: The index number of the desired sample hit object to retrieve.

Data Type: 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`. An empty value of "" will use the current alignment.

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 parameter with the **ENUM_PLANE_TYPE** enumeration for the PARTMM3 and POLAR coordinate system to denote the workplane to be used. Options include `PLANE_TOP`, `PLANE_RIGHT`, `PLANE_BACK`, `PLANE_BOTTOM`, `PLANE_LEFT`, and `PLANE_FRONT`. It defaults to `PLANE_TOP`. You can also use these numerical **long** values:

- `PLANE_TOP = 0`
- `PLANE_RIGHT = 1`
- `PLANE_BACK = 2`
- `PLANE_BOTTOM = 3`
- `PLANE_LEFT = 4`
- `PLANE_FRONT = 5`

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.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.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.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.GetSampleHit

Syntax

Return Value=expression.GetSampleHit(Index, DataType, TheoMeas, CoordSystem, AlignID, Workplane)

Return Value: This method returns a Point Data object with the values of the sample 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. An empty value of "" will use the current alignment.

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 parameter with the **ENUM_PLANE_TYPE** enumeration for the PARTMM3 and POLAR coordinate system to denote the workplane to be used. Options include PLANE_TOP, PLANE_RIGHT, PLANE_BACK, PLANE_BOTTOM, PLANE_LEFT, and PLANE_FRONT. It defaults to PLANE_TOP. You can also use these numerical **long** values:

- PLANE_TOP = 0
- PLANE_RIGHT = 1

- PLANE_BACK = 2
- PLANE_BOTTOM = 3
- PLANE_LEFT = 4
- PLANE_FRONT = 5

Remarks

Use this function to obtain sample hit information from Auto Feature commands.

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.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.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 from which the data is taken to set values in the corresponding object.

Data Type: 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, SCANSEG_START, SCANSEG_END

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 parameter with the **ENUM_PLANE_TYPE** enumeration for the PARTMM3 and POLAR coordinate system to denote the workplane to be used. Options include PLANE_TOP, PLANE_RIGHT, PLANE_BACK, PLANE_BOTTOM, PLANE_LEFT, and PLANE_FRONT. It defaults to PLANE_TOP. You can also use these numerical **long** values:

- PLANE_TOP = 0
- PLANE_RIGHT = 1
- PLANE_BACK = 2
- PLANE_BOTTOM = 3
- PLANE_LEFT = 4
- PLANE_FRONT = 5

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 diameter.

Use the FDATA_THEO flag if you want theoretical data, FDATA_MEAS if you want measured data.

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.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.

K1: **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.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.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.SetHit

Syntax

Return Value=*expression*.SetHit(*Index*, *Data Type*, *MeasOrTheo*, X, Y, Z)

Return Value: This function returns TRUE if the hit or vector gets successfully set, FALSE otherwise.

Index: **Integer** representing the hit number.

Data Type: Enumerated data type that specifies what type of data you are setting. This can be Centroid, Vector, or BallCenter. The can use the following values:

12 FHITDATA_BALLCENTER

0 FHITDATA_CENTROID

1 FHITDATA_VECTOR

MeasOrTheo: Enumerated value that identifies the portion of the hit getting set. The possible enumerated values for all, measured, theoretical, or targets are:

100 FDATA_ALL

3 FDATA_MEAS

27 FDATA_TARG

2 FDATA_THEO

X: The X value for the point / vector being set.

Y: The Y value for the point / vector being set.

Z: The Z value for the point / vector being set.

This method sets the hit data of a specified hit. It uses the part coordinate system relative to the current alignment.

FeatCommand.SetHit2

Syntax

Return Value=*expression*.SetHit(*Index*, *DataType*, *MeasOrTheo*, *CoordSystem*, *Alignment*, *Workplane*, X, Y, Z)

Return Value: This function returns TRUE if the hit or vector gets successfully set, FALSE otherwise.

Index: **Integer** representing the hit number.

DataType: Enumerated data type that specifies what type of data you are setting. This can be Centroid, Vector, or BallCenter. The can use the following values:

12 FHITDATA_BALLCENTER

0 FHITDATA_CENTROID

1 FHITDATA_VECTOR

MeasOrTheo: Enumerated value that identifies the portion of the hit getting set. The possible enumerated values for all, measured, theoretical, or targets are:

100 FDATA_ALL

3 FDATA_MEAS

27 FDATA_TARG

2 FDATA_THEO

CoordSystem: Enumerated value that identifies the type of coordinate system to use (Cad, Machine, Part, PartMM3, or Polar). Possible values are:

5 FDATA_CAD

11 FDATA_MACHINE

13 FDATA_PART

10 FDATA_PARTMM3

4 FDATA_POLAR

Alignment: **string** value representing the ID of the alignment to use.

Workplane: Optional parameter with the **ENUM_PLANE_TYPE** enumeration that denotes the workplane. Options include PLANE_TOP, PLANE_RIGHT, PLANE_BACK, PLANE_BOTTOM, PLANE_LEFT, and PLANE_FRONT. It defaults to PLANE_TOP. You can also use these numerical **long** values:

- PLANE_TOP = 0
- PLANE_RIGHT = 1
- PLANE_BACK = 2
- PLANE_BOTTOM = 3
- PLANE_LEFT = 4
- PLANE_FRONT = 5

X: The X value for the point / vector being set.

Y: The Y value for the point / vector being set.

Z: The Z value for the point / vector being set.

This method works just like the SetHit method—it sets the hit data of a specified hit—but it uses a specified coordinate system and alignment.

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*. The index value must be less than the value returned by the NumHits property (you can use the NumHits property with constructed features to determine the number of inputs. For adding inputs you will need to use the AddInputFeat method).

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.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.

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

Properties:

FeatData.ANGLE

Represents the ANGLE member of this object. Read/write **Double**.

FeatData.DIAM

Represents the DIAM member of this object. Read/write **Double**.

FeatData.EndAngle

Represents the EndAngle member of this object. Read/write **Double**.

FeatData.EndAngle2

Represents the EndAngle2 member of this object. Read/write **Double**.

FeatData.F

Represents the F member of this object. Read/write **Double**.

FeatData.I

Represents the I member of this object. Read/write **Double**.

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.J

Represents the J member of this object. Read/write **Double**.

FeatData.K

Represents the K member of this object. Read/write **Double**.

FeatData.LENGTH

Represents the LENGTH member of this object. Read/write **Double**.

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.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.SmallDiam

Represents the SmallDiam member of this object. Read/write **Double**.

FeatData.StartAngle

FeatData.StartAngle2

Represents the StartAngle2 member of this object. Read/write **Double**.

FeatData.TP

Represents the TP member of this object. Read/write **Double**.

FeatData.X

Represents the X member of this object. Read/write **Double**.

FeatData.Y

Represents the Y member of this object. Read/write **Double**.

FeatData.Z

Represents the Z member of this object. Read/write **Double**.

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 the "Using File Input / Output" chapter in the *PC-DMIS Reference Manual*.

Properties:

FileIO.BufferSize

LONG value representing the buffer size used with the Read Block File I/O command.

Read/Write **Long**

FileIO.Expression

STRING value representing the text to be used in reading from or writing to the opened file.

Read/Write **String**

FileIO.FailIfExists

BOOLEAN value indicating whether a file copy operation should fail or not if the destination file already exists.

Read/Write **Boolean**

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

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**

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**

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

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**

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**

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

FlowControlCommand.ReportAutoPrint

Returns True if you have Hyper Report's **Auto Print** checkbox selected. False otherwise.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

FlowControlCommand.GetLeftSideOfExpression

Syntax

Return Value=expression.GetLeftSideOfExpression

expression: Required expression that evaluates to **FlowControlCommand** object.

Return value: For FlowControlCommand objects of type ASSIGNMENT, this function returns the name of the variable being assigned to. For other types of objects, it returns an empty string.

FlowControlCommand.GetRightSideOfExpression

Syntax

Return Value=expression.GetRightSideOfExpression

expression: Required expression that evaluates to **FlowControlCommand** object.

Return value: For FlowControlCommand objects of type ASSIGNMENT, this function returns the value being assigned to the variable. For other types of objects, it returns an empty string.

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.

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.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 assignment statement), and FALSE otherwise.

FlowControlCommand.IsValidSubroutineArgumentName

Syntax

Return Value=`expr.IsValidSubroutineArgumentName(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 subroutine argument name, and FALSE otherwise.

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.

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.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.

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.

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.

FlowControlCommand.SetLeftSideOfAssignment

Syntax

expr.SetLeftSideOfAssignmentExpression

expr: Required expression that evaluates to **FlowControlCommand** object.

Expression: Required **String** that indicates the expression to be used for the left side of the assignment.

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.

FlowControlCommand.SetRightSideOfAssignment

Syntax

expr.SetRightSideOfAssignmentExpression

expr: Required expression that evaluates to **FlowControlCommand** object.

Expression: Required **String** that indicates the expression to be used for the right side of the assignment.

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.

FPanel Object Overview

The FPanel object contains properties that allow you to work with an F-Panel controller and interface.

Properties:

FPanel.Parent

This read-only property returns the parent Machine object from which the FPanel object was created.

Related Topics: Machine Object Overview

FPanel.PanelSelector

This write-only property allows you to set whether the F-Panel is in automatic or manual mode.

For Manual, set PanelSelector equal to 1

For Automatic, set PanelSelector equal to 2

Two different computers control the FPanel interface. In the original DEA software, these computers talked to each other and passed this parameter around on their own. PC-DMIS, however, only reads from one computer, and so you need to let PC-DMIS know whether or not the other controller is set to manual or automatic mode. You should be able to look and see on the panel in front of them whether they are in manual or automatic mode. Once you let PC-DMIS know by setting this property to either 1 or 2, PC-DMIS passes the information to the other controller and changes its status.

Leapfrog Object Overview

The Leapfrog command object contains three leapfrog properties that will allow you to define how to use PC-DMIS's Leapfrog option (available in PC-DMIS Versions 3.0 and above) to translate along a part as well as the numbers of hits to use for each feature.

For information on Leapfrog, see the "Performing a LeapFrog Operation" topic in the *PC-DMIS Reference Manual*.

Properties:

Leapfrog.Full

BOOLEAN value determines whether or not the leapfrog will be full (TRUE) or partial (FALSE). For more information on this, see the "Creating and Using Alignments" topic in the *PC-DMIS Reference Manual*.

Read/Write **boolean**.

Leapfrog.LeapfrogType

Integer value that defines the type of feature used to translate the CMM along the part.

- 0 Sphere
- 1 Point Sets (Psets)
- 2 Points
- 3 Off

Read/Write **Integer**.

Leapfrog.Numhits

Integer value that determines the number of hits used for the feature types described in the "LeapfrogType" property above. The feature type determines if the number of hits are useful or not.

- If LeapfrogType = 0 then useful values of NumHits are between 5 and 50
- If LeapfrogType = 1 then useful values of NumHits are greater than two
- If LeapfrogType = 2 or 3 then useful values of NumHits are ignored.

Read/Write **Integer**.

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 the "Parameter Settings: Optional Probe Tab" topic in the "Setting Your Preferences" topic in the *PC-DMIS Reference Manual*.

Properties:

LeitzMot.LowForce

Double value used to set or get the low force setting for the probe.

Read/Write **Double**

LeitzMot.MaxForce

Double value used to set or get the max force setting for the probe.

Read/Write **Double**

LeitzMot.PositionalAccuracy

Double value used to set or get the positional accuracy setting.

Read/Write **Double**

LeitzMot.ProbeAccuracy

Double value used to set or get the probe accuracy setting.

Read/Write **Double**

LeitzMot.ReturnData

Double value used to set or get the return data setting.

Read/Write **Double**

LeitzMot.ReturnSpeed

Double value used to set or get the return speed.

Read/Write **Double**

LeitzMot.ScanPointDensity

Double value used to set or get the scan point density.

Read/Write **Double**

LeitzMot.TriggerForce

Double value used to set or get the trigger force setting for the probe.

Read/Write **Double**

LeitzMot.UpperForce

Double value used to set or get the upper force setting for the probe.

Read/Write **Double**

Load Machine Object Overview

The Load Machine object gives access to the machine name property of the PC-DMIS Load Machine command.

Properties:

LoadMachine.MachineName

STRING value representing the name of the machine to be loaded.

Read/Write **String**

Load Probes Object Overview

The Load Probes object gives access to the filename property of the PC-DMIS Load Probes command.

Properties:

LoadProbes.Filename

STRING value representing the name of the probes file to be loaded.

Read/Write **String**

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.

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.

Machine.FPanel

Returns the read-only FPanel object if your controller is indeed an FPanel controller.

Machine.IsFPanel

Returns a read-only TRUE or FALSE value. Returns TRUE if your controller is indeed an FPanel controller, FALSE otherwise.

Machine.Name

Returns the name of the Machine object. Read-only **String**.

Machine.Parent

Returns the read-only **Machines** collection object to which the machine belongs.

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 variable 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.

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.

Machines.Count

Represents the number of Machine objects currently active in PC-DMIS. Read-only **Integer**.

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.

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.

MasterSlaveDlg Object Overview

The MasterSlaveDlg object gets called when the PartProgram.MasterSlaveDlg method is used.

Properties:

Application

This read-only object returns the Application object.

DCC

LONG value that gets or sets whether the Master / Slave calibration is a manual or DCC calibration. The available settings are:

MEASURE_MANUAL = 0

MEASURE_DCC = 1

Read/Write **Long**

MasterProbe

STRING that gets or sets which probe is used on the Master machine. The names are taken from the owning PartProgram object's Probes collection.

Read/Write **String**

MasterTip

STRING that gets or sets which tip is used on the Master machine. The names are taken from the owning PartProgram object's Probes collection.

Read/Write **String**

MeasuringArm

LONG value that gets or sets which arm or arms measure the calibration sphere. The available settings are:

MEASURE_BOTH = 0

MEASURE_MASTER = 1

MEASURE_SLAVE = 2

Read/Write **Long**

Parent

This read-only object returns the PartProgram object.

Position

POINTDATA object that gets or sets the first (or only) sphere position.

Read/Write **PointData**

SlaveProbe

STRING that gets or sets which probe is used on the Slave machine. The names are taken from the owning PartProgram object's Probes collection.

Read/Write **String**

SlaveTip

STRING that gets or sets which tip is used on the Slave machine. The names are taken from the owning PartProgram object's Probes collection.

Read/Write **String**

Tool

STRING that gets or sets which tool is being measured during the calibration process. The names are taken from the owning PartProgram object's Tools collection.

Read/Write **String**

Methods:

Calibrate

Syntax

Return Value=expression.calibrate()

Return Value=The Calibrate function returns an unused and reserved LONG value for future implementation.

expression: Required expression that evaluates to a MasterSlaveDlg object identified.

Remarks

Using this method is the same as clicking the **Calibrate** button on the Master / Slave Calibration dialog box inside the application itself.

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.

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.

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.

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.

Related Topics: Command.Type Property, ModalCommand.Distance2, ModalCommand.PassPlane

ModalCommand.Distance2

Represents the pass-through distance to move for the 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.

Related Topics: Command.Type Property, ModalCommand.Distance, ModalCommand.PassPlane

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.

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.

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.

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.

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.

Related Topics: Command.Type Property, ModalCommand.Distance, ModalCommand.Distance2

ModalCommand.RmeasMode

Represents the current relative measure (or RMEAS) mode.

Read/write **ENUM_RMEAS_MODE** enumeration. These can be 0 for RMEAS_NORMAL or 1 for RMEAS_ABSOLUTE.

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.

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.

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

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.

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.

MoveCommand.IJK

A PointData object that represents the IJK vector to use. 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**.

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.

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.

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.

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**.

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 the "Setting Your Preferences" chapter of the *PC-DMIS Reference Manual*, under the title "Parameter Settings: Acceleration tab".

Properties:

OptMotion.MaxTAcceleration

Double value used to set or get the maximum acceleration in T setting.

Read/Write **Double**

OptMotion.MaxTSpeed

Double value used to set or get the maximum speed in T setting.

Read/Write **Double**

OptMotion.MaxXAcceleration

Double value used to set or get the maximum acceleration in X setting.

Read/Write **Double**

OptMotion.MaxYAcceleration

Double value used to set or get the maximum acceleration in Y setting.

Read/Write **Double**

OptMotion.MaxZAcceleration

Double value used to set or get the maximum acceleration in Z setting.

Read/Write **Double**

OptMotion.MovePositionalAccuracy

Double value used to set or get the move positional accuracy setting.

Read/Write **Double**

OptProbe Object Overview

This automation object provides support for the Optional Probe command. Through this object you can get and set various properties of an option probe command in PC-DMIS. For more information on the option probe command, see the "Parameter Settings: Optional Probe tab" topic of the "Setting Your Preferences" chapter of the *PC-DMIS Reference Manual*.

Properties:

OptProbe.LowForce

Double value used to set or get the probe low force setting.

Read/Write **Double**

OptProbe.MaxForce

Double value used to set or get the probe max force setting.

Read/Write **Double**

OptProbe.PositionalAccuracy

Double value used to set or get the positional accuracy setting.

Read/Write **Double**

OptProbe.ProbeAccuracy

Double value used to set or get the probe accuracy setting.

Read/Write **Double**

OptProbe.ReturnData

Double value used to set or get the probe return data value.

Read/Write **Double**

OptProbe.ReturnSpeed

Double value used to set or get the probe return speed value.

Read/Write **Double**

OptProbe.ScanPointDensity

Double value used to set or get the probe scan point density setting.

Read/Write **Double**

OptProbe.TriggerForce

Double value used to set or get the probe trigger force setting.

Read/Write **Double**

OptProbe.UpperForce

Double value used to set or get the probe upper force setting.

Read/Write **Double**

PartProgram Object Overview

The **PartProgram** object represents a part program currently available in PC-DMIS. This is the main object used to manipulate part programs.

Properties:

PartProgram.ActiveMachine

Returns the **Machine** object associated with this part program. Read-only.

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.

PartProgram.CadWindows

Returns the **CadWindows** object associated with this part program. Read-only.

PartProgram.Commands

Returns the **Commands** collection object of this part program. Read-only.

PartProgram.ConnectedToMaster

New in Version 3.6. Returns TRUE if the part program is on the master computer but is running as the slave part program. Returns FALSE otherwise. Read only.

PartProgram.ConnectedToSlave

New in Version 3.6. Returns TRUE if the part program is on the slave computer but is running as the master part program. Returns FALSE otherwise. Read only.

PartProgram.EditWindow

Returns the **Editwindow** object associated with this part program. Read-only.

PartProgram.EditWindowTextAll

Returns a single string containing all the text of the entire Edit window as seen in the Edit window's command mode. Read only.

PartProgram.ExecutedCommands

The ExecutedCommands property returns the ExecutedCommands object. This object contains a collection class of those commands last executed for the current part program.

PartProgram.ExecutionWasCancelled

Returns TRUE if, during part program execution, the execution is cancelled. Otherwise it returns FALSE. The default value is FALSE. Read only.

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.IsProbeAnalog

Returns TRUE if the current loaded probe in a part program is an analog probe; it returns FALSE otherwise. If you have multiple probe types defined for a part program, the return value will depend, of course, on the location of the insertion point in the part program.

PartProgram.Name

Returns the part program's file name. Read only **String**. If the file name of the part program is C:\PCDMISW\PARTS\1.PRG, the Name returns "1.PRG".

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.

PartProgram.Parent

Returns the **PartPrograms** collection object to which this part program belongs. Read-only.

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.Path

Returns the part program's file path. Read only **String**. If the file name of the part program is C:\PCDMISW\PARTS\1.PRG, the Path returns "C:\PCDMISW\PARTS\".

PartProgram.Probes

The **Probes** property returns this part program's **Probes** collection object. Read-only.

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.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.StatsCount

Returns or sets the stats count for the current part program. Read/Write **Long**.

Remarks

You can view and set the stats count 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.Tools

The Tools property returns this part program's **Tools** collection object. Read-only.

PartProgram.Units

Returns the measurement unit type used in the part program. Either inches or millimeters. Read only **UNITTYPE**.

PartProgram.Visible

Represents the part program's visibility status. Read/Write **Boolean**.

Methods:

PartProgram.AsyncExecute

Syntax

Return Value=expression.AsyncExecute

Return Value: This method returns a boolean value. Boolean returns true if the function succeeds, false if it fails. TRUE if the execution of the part program and the return of control was successful, FALSE otherwise.

expression: Required expression that evaluates to a **PartProgram** object.

This function starts execution of the part program and then returns immediately, allowing for asynchronous execution.

PartProgram.ClearExecutionBlock

Syntax

expression.ClearExecutionBlock()

expression: Required expression that evaluates to a **PartProgram** object.

This clears the start and end commands set by the SetExecutionBlock method.

PartProgram.Close

Syntax

expression.Close

expression: Required expression that evaluates to a **PartProgram** object.

This subroutine saves, closes, and deactivates the part program.

PartProgram.DmisOut

Syntax

Return Value=expression.DmisOut(bExecOrder,FileName)

Return Value: This returns a boolean value. Boolean returns TRUE if the expression succeeds, FALSE otherwise.

expression: Required expression that evaluates to a **PartProgram** object.

bExecOrder:

FileName:

This function outputs DMIS results to a file.

PartProgram.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. TRUE if the execution of the part program was successful, FALSE otherwise.

expression: Required expression that evaluates to a **PartProgram** object.

This function executes the part program.

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.GetVariableValue

Syntax

Return Value=expression.GetVariableValue(VarName)

Return Value: This method returns a variable object specified by the string in *VarName*.

expression: Required expression that evaluates to a **PartProgram** object.

VarName: String value representing a variable object.

Note: PC-DMIS variables only hold values during execution; at learn time PC-DMIS variables have a value of zero. The `GetVariableValue` and `SetVariableValue` methods only change a variable's value during the script's execution. If you want to permanently change a value of a variable inside PC-DMIS, you should use the `PutText` method instead.

Related Topics: [Sample Automation Script 1](#)

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.LoadLayout

Syntax:

Return Value = `expression.LoadLayout(layout)`

Return Value: This method returns TRUE if the function succeeds, FALSE if it fails.

expression: Required expression that evaluates to a PC-DMIS PartProgram object

layout: Required String that indicates the file the layout should be read from. This can be an absolute path, a relative path, or the name the user typed in when creating the layout in PC-DMIS.

The LoadLayout method loads a customized PC-DMIS user-interface layout as if it were selected from the **Windows Layout** toolbar inside PC-DMIS. Also, if a layout has been created and moved to a different directory, you can access it by specifying the absolute or relative file name. For information on using this toolbar, see the "Using Toolbars" chapter inside your *PC-DMIS Reference Manual*.

Examples:

Dim Part As PCDLRN.PartProgram

Part.LoadLayout "layout1.dat" ' loads the first layout from the current user's setup information directory

Part.LoadLayout "c:\mylayout.dat" ' loads the layout in the specified file

Part.LoadLayout "Learn" ' loads the layout named "Learn", if it exists

PartProgram.MasterSlaveDlg

Return Value: This returns a read-only pointer to the **Master/Slave Calibration** dialog box, opening the dialog box if necessary.

expression: Required expression that evaluates to a **PartProgram** object.

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 omitted, 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 omitted, 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.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.

PartProgram.RefreshPart

Syntax

`expression.RefreshPart`

expression: Required expression that evaluates to a **PartProgram** object.

Refreshes the display of the Part in the Edit window and in the CAD window.

PartProgram.RunJournalFile

Syntax

Return Value=expression.RunJournalFile(journal)

Return Value: This method returns TRUE if the part program successfully executes; FALSE otherwise.

expression: Required expression that evaluates to a **PartProgram** object.

journal: string representing the journal file to use.

This executes the part program using point data collected from a journal file. This is the same as a normal execution of a part program except that the point data comes from the specified journal file. Journal files are used with PC-DMIS/NC, a specialized version of PC-DMIS that executes part programs with CNC (Computer Numeric Control) machines. For more information see the *PC-DMIS/NC Reference Manual*.

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.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.SetExecutionBlock

Syntax

Return Value=expression.SetExecutionBlock(StartCommand, EndCommand)

Return Value: This method returns TRUE if the block of commands successfully executes; FALSE otherwise.

expression: Required expression that evaluates to a **PartProgram** object.

StartCommand: This is the first command to execute (as a **Command** object) in the command block.

EndCommand: This is the last command to execute (as a **Command** object) in the command block.

This method defines a block of commands to execute. Calls are made to the Execute or AsyncExecute functions until the execution block is cleared with the ClearExecutionBlock method.

PartProgram.SetVariableValue

Syntax

Return Value=expression.SetVariableValue(VarName, Value)

Return Value: This method sets the value defined in *Value* for the variable specified by the string in *VarName*. This method returns a boolean value: TRUE if the variable was set, FALSE otherwise.

expression: Required expression that evaluates to a **PartProgram** object.

VarName: **String** value representing a variable.

Value: **Variable** value representing the new value the variable will take.

Note: PC-DMIS variables only hold values during execution; at learn time PC-DMIS variables have a value of zero. The `GetVariableValue` and `SetVariableValue` methods only change a variable's value during the script's execution. If you want to permanently change a value of a variable inside PC-DMIS, you should use the `PutText` method instead.

PartProgram.WaitUntilExecuted

Important: This method is currently unavailable.

Syntax:

Return Value = *expression*.WaitUntilExecuted (*Command*, *Timeout*)

Return Value: This methods immediately returns FALSE if the part program is not executing. It returns TRUE if the specified *Command* executes before *Timeout* seconds pass. It returns FALSE if the *Command* does not execute before *Timeout* seconds pass. If *Timeout* is a non-positive value, and the *Command* never executes, this function never returns a value.

expression: Required expression that evaluates to a **PartProgram** object.

Command: This is an expression that evaluates to a **Command** object. This is the command that is waited for.

Timeout: This is the number of seconds (a Long value) to wait for the command to finish execution. If you have a non-positive value, the method waits indefinitely.

This method waits until the specified *Command* object executes, or until Timeout seconds pass.

Events:

The following events are available to the PartProgram object.

PartProgram.OnAddObject

Syntax:

expression.OnAddObject(*command*)

expression: Required expression that evaluates to a PC-DMIS **PartProgram** object.

command: expression that evaluates to a **Command** object to determine the command for which this event should wait.

This event gets launched when the specified *command* gets added to the part program.

PartProgram.OnEndExecution

Syntax:

expression.OnEndExecution(*type*)

expression: Required expression that evaluates to a PC-DMIS **PartProgram** object.

type: this Long number determines the termination type used by this event.

This event gets launched when PC-DMIS finishes executing the part program. PC-DMIS determines it has finished execution based on the termination *type*.

PartProgram.OnExecuteDialogErrorMsg

Syntax:

expression.OnExecuteDialogErrorMsg(*message*)

expression: Required expression that evaluates to a PC-DMIS **PartProgram** object.

message: string representing the error message sent to the **Machine Errors** list in the **Execution Mode Options** dialog box.

This event gets launched when the **Execution Mode Options** dialog box displays *message*.

PartProgram.OnExecuteDialogStatusMsg

Syntax:

expression.OnExecuteDialogStatusMsg(*message*)

expression: Required expression that evaluates to a PC-DMIS **PartProgram** object.

message: string representing the status message sent to the **Machine Commands** list in the **Execution Mode Options** dialog box.

This event gets launched when the **Execution Mode Options** dialog box displays *message*.

PartProgram.OnObjectAboutToExecute

Syntax:

expression.OnObjectAboutToExecute(*command*)

expression: Required expression that evaluates to a PC-DMIS **PartProgram** object.

command: expression that evaluations to a **Command** object to determine the command about to be executed.

This event gets launched immediately before the specified *command* gets executed.

PartProgram.OnObjectAboutToExecute2

Syntax:

expression.OnObjectAboutToExecute2(*command,arm*)

expression: Required expression that evaluates to a PC-DMIS **PartProgram** object.

command: expression that evaluations to a **Command** object to determine the command about to be executed.

arm: Long value representing the arm on a multiple arm machine that is about to execute the *command* causing the event to launch.

This event gets launched immediately before the specified *command* gets executed on a specified *arm* of a multiple arm system.

PartProgram.OnObjectExecuted

Syntax:

```
expression.OnObjectExecuted(command)
```

expression: Required expression that evaluates to a PC-DMIS **PartProgram** object.

command: expression that evaluations to a **Command** object to determine the command that gets executed.

This event gets launched immediately after the specified *command* gets executed.

PartProgram.OnObjectExecuted2

Syntax:

```
expression.OnObjectExecuted2(command,arm)
```

expression: Required expression that evaluates to a PC-DMIS **PartProgram** object.

command: expression that evaluations to a **Command** object to determine the command that gets executed.

arm: Long value representing the arm on a multiple arm machine that executes the *command* causing the event to launch.

This event gets launched immediately after the specified *command* gets executed on a specified *arm* of a multiple arm system.

PartProgram.OnWorkOffset

```
expression.OnWorkOffset(dX, dY, dZ, Rx, Ry, Rz, file, program, gcode)
```

expression: Required expression that evaluates to a PC-DMIS **PartProgram** object.

dX, dY, dZ. These are double values that represent the change in translation from the base alignment to the finished alignment.

Rx, Ry, Rz. These are double values representing the change in rotation from the base alignment to the finished alignment.

file. This is the NC formatted file that gets created if the NC interface is unidirectional.

program. This is the program name assigned to the generated program

gcode. This is the G code to send to the controller. This code can be G54 through G59. To represent these codes, you should use a Long number of 0 to 5 respectively:

0—G54

1—G55

2—G56

3—G57

4—G58

5—G59

This event gets called when PC-DMIS/NC executes WorkOffset command. PC-DMIS/NC is a specialized version of PC-DMIS that allows you to execute part programs using your CNC (Computer Numeric Control) machine. For more information on PC-DMIS/NC, see the *PC-DMIS/NC Reference Manual*.

PartProgram Settings Object Overview

The **PartProgramSettings** object allows you to get or set various part program settings.

Properties:

PartProgramSettings.AutoAdjustPH9

Returns the check box value or sets the **Automatically Adjust Probe Head Wrist** check box from the **General** tab of the **SetUp Options** dialog box. Read/Write **Long**.

If you set this property a *non-zero value* or **True**, then PC-DMIS selects this check box.

If you set this property to a *zero value* or **False**, then PC-DMIS deselects this check box.

PartProgramSettings.AutoLabelPosition

Returns the check box value or sets the **Automatic Label Positioning** check box from the **General** tab of the **SetUp Options** dialog box. Read/Write **Long**.

If you set this property a *non-zero value* or **True**, then PC-DMIS selects this check box.

If you set this property to a *zero value* or **False**, then PC-DMIS deselects this check box.

PartProgramSettings.WarnLoadProbe

Returns the check box value or sets the **Please Load Probe = %s** warning check box found in the **Warnings Display Options** dialog box. Read/Write **Long**.

If you set this property a *non-zero value* or **True**, then PC-DMIS selects this check box.

If you set this property to a *zero value* or **False**, then PC-DMIS deselects this check box.

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.

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.

PartPrograms.Count

Returns the number of part programs active in PC-DMIS. Read-only **Long**.

PartPrograms.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.

Methods:

PartPrograms.Add

Syntax

Return Value=expression.Add(FileName, Units, Machine, ProbeFile)

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, anything else for millimeters.

Machine: Required **String**. The identifying string for your machine. If you're running in offline mode, use "Offline".

ProbeFile: Required **String**. The identifying string for the probe (.prb) file to use in the part program.

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.

\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.

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.

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.

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.

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

Properties

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.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.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.X

Represents the X member of this object. Read/write **Double**.

PointData.Y

Represents the Y member of this object. Read/write **Double**.

PointData.Z

Represents the Z member of this object. Read/write **Double**.

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.

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

```
m_app.ActivePartProgram.Probes.Add(prbFile)
Dim prb As PCDLRN.probe =
m_app.ActivePartProgram.Probes.Item(prbFile)
prb.ActiveComponent = 0
For i As Integer = 0 To prb.ConnectionCount - 1
    If (prb.ConnectionDescription(i) = "PROBEPH9") Then
        prb.ActiveConnection = i
    End If
Next i
prb.ActiveComponent = prb.ComponentCount() - 1
For i As Integer = 0 To prb.ConnectionCount - 1
    If (prb.ConnectionDescription(i) = "PROBETP2") Then
        prb.ActiveConnection = i
    End If
Next i
prb.ActiveComponent = prb.ComponentCount() - 1
For i As Integer = 0 To prb.ConnectionCount - 1
    If (prb.ConnectionDescription(i) = "TIP5BY50MM") Then
        prb.ActiveConnection = i
    End If
Next i
```

Probe.ActiveConnection

Represents the highlighted probe connection in PC-DMIS's Probe dialog's connection drop-down list. Read/write **Long**.

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.

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.ConnectionCount

Returns the number of connections in the connection drop-down list. The contents of this list depend on which component is active.

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.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.Parent

Returns the **Probes** collection object to which this object belongs.

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.QualificationSettings

Returns the Qualify Settings object that can be modified and passed into the `Qualify2` method. It supports these parameters:

- `StartA` – Returns the starting A angle of the probe
- `EndA` – Returns the ending A angle of the probe
- `IncrementA` – Returns the increment value for automatically generated A angles between the starting A angle of the probe and the ending A angle of the probe.
- `StartB` – Returns the starting B angle of the probe

- EndB – Returns the ending B angle of the probe
- IncrementB – Returns the increment value for automatically generated B angles between the starting B angle of the probe and the ending B angle of the probe.

Probe.Tips

Returns the **Tips** object associated with this **Probe** object.

Probe.UseWristMap

Syntax

Return Value=expression.UseWristMap

Return Value: This method returns a boolean value. Boolean returns TRUE if the probe uses a wrist map, FALSE otherwise.

expression: Required expression that evaluates to a PC-DMIS **Probe** object.

Determines whether or not a wrist map is used. Read/Write **boolean**.

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" function on page 173 to select all tips. Use the "Tip.Selected" property of the tip object on page 195 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.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.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.Qualify2

Syntax

Return Value=expression.Qualify2(settings)

Return Value: This method returns a boolean value. Boolean returns TRUE if the expression succeeds, FALSE otherwise.

expression: Required expression that evaluates to a PC-DMIS **Probe** object.

Settings: Values passed in from the QualificationSettings object.

Calibrates probes using settings passed in via the QualificationSettings object.

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 172 to clear all selected tips. Use the "Tip.Selected" property of the tip object on page 195 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.

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.

Probes.Count

Represents the number of Machine objects currently active in PC-DMIS. Read-only **Integer**.

Probes.Parent

Returns the parent PartProgram of this object. Read-only **PartProgram**.

Probes.Visible

Gets or sets the current visible state of the Probes object. You can use this property to show or hide the current probe inside the Graphics Display window of PC-DMIS.

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.CancelChanges

Syntax

expression.CancelChanges

expression: Required expression that evaluates to a **Probes** object.

The CancelChanges function cancels changes made to a probes collection and then closes the probes collection.

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.

Related Topics: Probe Overview

QualificationSettings Object Overview

The QualificationSettings object specifies how to calibrate your probe. The calibration process tells PC-DMIS the location and diameter of the probe tip. For more information on calibrating the probe, see the "Defining Probes" topic in the PC-DMIS manual.

Properties

QualificationSettings.CreateReplaceMap

This determines whether or not a map should be created or replaced. Read/Write. Type: **ENUM_QUAL_CREATE_REPLACE**.

The enumerated values you can set this property equal to are:

- 0 CREATE_NEW_MAP
- 1 REPLACE_CLOSEST_MAP

QualificationSettings.EndA

This determines the ending A angle to use during qualification. Type: **Double**.

QualificationSettings.EndAngle

This determines the End Angle to use on the qualification tool. Type **Double**.

QualificationSettings.EndB

This determines the ending B angle to use during qualification. Type: **Double**.

QualificationSettings.ExecuteMode

This determines the type of calibration action to take. Type **ENUM_CALIBRATION_EXECUTE_MODE**.

QualificationSettings.IncrementA

This determines the angle increment to use for the A angle during qualification. Type: **Double**.

QualificationSettings.IncrementB

This determines the angle increment to for the B angle during qualification. Type: **Double**.

QualificationSettings.Mode

This determines the Calibration Mode. Either DCC or Manual. Type **DCCMODE**.

QualificationSettings.MoveSpeed

This determines the speed the probe moves during calibration. Type **Double**.

QualificationSettings.NumHits

This determines the number of hits to take around the calibration tool. Type **Long**.

QualificationSettings.NumLevels

This determines the number of levels that to use in the calibration process. PC-DMIS divides the number of hits by the number of levels to determine how many hits will be taken on each level of the qualification tool. Type **Long**.

QualificationSettings.Offset

This determines the distance (or length) up from the tip of the shank that PC-DMIS will take the next set of qualification hits. Type **Double**.

QualificationSettings.PHSAPriority

This indicates whether angle A or B for a PHS system receives priority during the calibration process. Type **Boolean**.

TRUE means Angle A gets priority. FALSE means Angle B gets priority.

QualificationSettings.PHSTol

For PHS systems, this determines the PHS tolerance value. Type **Double**.

QualificationSettings.Prehit

This determines the Prehit distance to use during calibration. Type **Double**.

QualificationSettings.ShankCheck

This determines whether or not you'll calibrate the shank of the probe as well. Type **Boolean**.

QualificationSettings.ShankHits

This determines the number of hits used to measure the shank. Type **Long**.

QualificationSettings.StartA

This determines the starting A angle to use during qualification. Type: **Double**.

QualificationSettings.StartAngle

This determines the Start Angle to use on the qualification tool. Type **Double**.

QualificationSettings.StartB

This determines the starting B angle to use during qualification. Type: **Double**.

QualificationSettings.ToolMoved

This indicates whether a tool has moved or not or if the user should be asked. Type **ENUM_TOOL_MOVED**.

QualificationSettings.ToolOnRotaryTable

This indicates whether or not the tool is located on a rotary table. Type **Boolean**.

QualificationSettings.ToolOverrideI

This determines the I value for the Tool Override's IJK vector. Type **Double**.

QualificationSettings.ToolOverrideJ

This determines the J value for the Tool Override's IJK vector. Type **Double**.

QualificationSettings.ToolOverrideK

This determines the K value for the Tool Override's IJK vector. Type **Double**.

QualificationSettings.TouchSpeed

This determines the Touch Speed to use during calibration. Type **Double**.

QualificationSettings.UserDefinedCalibrationMode

This determines whether End Angle, Start Angle, and NumLevels are used in the calibration process, or if they're ignored. Type **Boolean**.

QualificationSettings.UserDefinedCalibrationOrder

This determines whether or not the calibration order is user defined. Type **Boolean**.

Methods:

QualificationSettings.GetTool

Syntax

expression.GetTool

expression: Required expression that evaluates to a **QualificationSettings** object.

This method returns a **tool** object.

QualificationSettings.SetTool

Syntax

expression.SetTool(*tool*)

expression: Required expression that evaluates to a **QualificationSettings** object.

tool: Required expression that evaluates to a **tool** object.

This method sets the current qualification tool to *tool*. This method returns a boolean value: TRUE if the function succeeds, FALSE otherwise.

ReportData Object Overview

This object lets you access data sent to reports during the **EventReportData** event. This event can only be accessed inside the **Properties** dialog box inside the Label and Report Template Editors inside PC-DMIS versions 4.0 and higher.



Properties dialog box

Using this object in conjunction with the **EventReportData** event, you can access the desired information.

For Example:

Suppose in PC-DMIS's Label Template Editor, you use a **Border** object to change its background color to match the current dimension out of tolerance color. You can do this using the ReportData object. The following code works inside of the PC-DMIS's label template editor in the **EventReportData** event of a **Border** object.

```
Dim Count As Integer
Dim I As Integer
Dim MaxIndex As Integer
MaxIndex = 1
Dim MaxDev As Double
Dim CurrentDev As Variant
Dim CurrentBottom As Integer
Count = ReportData.GetCount(132)
If Count > 1 Then
  Border1.Bottom = 66 + ((Count-1)*23)
Else
  Border1.Bottom = 66
End If
I = 1
While I <= Count
  CurrentDev = ReportData.GetValue(340, I)
  ' MsgBox "Deviation for Axis " & I & " Equals " & CurrentDev
  If CurrentDev > MaxDev Then
    MaxDev = CurrentDev
    MsgBox "MaxDev=" & MaxDev & ", I=" & I
    MaxIndex = I
  End If
  I = I + 1
Wend
```

```
Dim Dev As Variant
Dim PTol As Variant
Dim MTol As Variant
Dev = ReportData.GetValue(340, MaxIndex)
PTol = ReportData.GetValue(167, MaxIndex)
MTol = ReportData.GetValue(168, MaxIndex)
```

ReportData Properties

None available.

Methods:

ReportData.GetColorList

Syntax

Return Value=expression.GetColorList()

Return Value: The current global dimension color list.

expression: Required expression that evaluates to a PC-DMIS **ReportData** object.

This returns a the current global dimension color list.

ReportData.GetCommand

Syntax

Return Value=expression.GetCommand()

Return Value: This method returns a Command object if the report data has command interface.

expression: Required expression that evaluates to a PC-DMIS **ReportData** object.

This returns a Command Object if report data has command interface.

ReportData.GetCount

Syntax

Return Value=expression.GetCount(DataType)

Return Value: This method returns a Long value representing the number of instances of the specified data type.

expression: Required expression that evaluates to a PC-DMIS **ReportData** object.

Data Type: This parameter specifies the DataType for the command.

This returns the number of instances of the specified data type.

ReportData.GetTolColor

Syntax

Return Value=*expression*.GetTolColor(Deviation,plustol,minustol)

Return Value: Long value representing the current tolerance color.

expression: Required expression that evaluates to a PC-DMIS **ReportData** object.

Deviation: A double value representing the deviation.

plustol: A double value representing the plus tolerance.

minustol: A double value representing the minus tolerance.

This returns a the current tolerance color based on the deviation, plus tolerance, and minus tolerance.

ReportData.GetValue

Syntax

Return Value=*expression*.GetValue(DataType, TypeIndex)

Return Value: This method returns the value of the indicated field of the command.

expression: Required expression that evaluates to a PC-DMIS **ReportData** object.

DataType: This parameter specifies the DataType for the command.

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 specified field type. In these cases, the TypeIndex must not be greater than the current number of fields of the that type + 1 and TypeIndex must be greater than 1. For fields that allow only a single value, the TypeIndex is 0.

This method returns the value of the indicated field of the command.

Report.GetValue2

Syntax

Return Value=*expression*.GetValue2(DataType, TypeIndex, FirstArrayIndex, SecondArrayIndex)

Return Value: This method returns the value of the indicated field of the command.

expression: Required expression that evaluates to a PC-DMIS **ReportData** object.

DataType: This parameter specifies the DataType for the command.

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 specified field type. In these cases, the *TypeIndex* must not be greater than the current number of fields of the that type + 1 and *TypeIndex* must be greater than 1. For fields that allow only a single value, the *TypeIndex* is 0.

FirstArrayIndex: This parameter specifies the index value for an item in the first array.

SecondArrayIndex: This parameter specifies the index value for an item in the second array.

This method returns the value of the indicated field of the command.

Report.HasCommandData

Syntax

Return Value=*expression*.HasCommandData()

Return Value: This method returns a **Boolean** value, showing **True** if the report data has a part program command interface, or **False** otherwise.

expression: Required expression that evaluates to a PC-DMIS **ReportData** object.

This method returns True if report data has a part program command interface, False otherwise.

ReportControls Object Overview

The ReportControls object gives you access to a variety of controls such as buttons, text boxes, and other items that you can add to, remove, and otherwise manipulate on a particular section of a report template.

Properties:

ReportControls.Application

This property returns the PC-DMIS Application object.

ReportControls.Count

This property counts all the controls in the current section/page and returns it as a **Long** value. Be aware that a hidden control called "Report" always exists in the Report template and cannot be deleted or otherwise manipulated. This object is used by PC-DMIS for internal purposes only. Your count value will therefore be one higher than expected due to this hidden object.

ReportControls.Parent

This property returns this object's parent **Section** object.

Methods:

ReportControls.Add

Syntax

Return Value=expression.Add(type, left, right, top, bottom)

Return Value: This function returns an **Object** of the specified object type.

expression: Required. An expression that returns a **ReportControls** object.

type: Can be an a constant value or an enumerated value as shown here:

Constant Value	Equivalent Enumerated Value
ID_HOB_ARC	60478
ID_HOB_BITMAP	60462
ID_HOB_BORDER	60473
ID_HOB_CIRCLE	60471
ID_HOB_GAUGE	60452
ID_HOB_GRAPH	60458
ID_HOB_LINE	60450
ID_HOB_PCD_CAD_REPORT_OBJECT	26514
ID_HOB_PCD_DIMCOLOR_OB	26614
ID_HOB_PCD_GRID_CTRL_OB	26623
ID_HOB_PCD_LABEL_OB	26603
ID_HOB_PCD_LEADERLINE_OB	26618
ID_HOB_PCD_SPC_CHART_OB	26619
ID_HOB_PCD_TEXT_REPORT_OBJECT	26515
ID_HOB_POLYLINE	60472
ID_HOB_PTR	60459
ID_HOB_TEXT	60448

left: This **Long** value sets the location of the left side of the control from the left side of the editor.

top: This **Long** value sets the top location of the control from the top side of the editor.

right: This **Long** value sets the right location of the control from the left side of the editor.

bottom: This **Long** value sets the bottom location of the control from the top side of the editor.

Remarks

The Add method inserts a new control of a defined location and size into the current section of the report template. To find out what properties are available to a control, in PC-DMIS's Report Template editor, insert the control and then access its properties sheet.

For example,

```
Private Sub Add_Arc()
```

```

Set ArcControl = ReportObjects.Add(ID_HOB_ARC, 10, 10, 200, 200)
ArcControl.Bottom = 100
ArcControl.Left = 10
ArcControl.LineStyle = 2
ArcControl.LineWidth = 1
ArcControl.Right = 790
ArcControl.Top = 20
ArcControl.Visible = False
End Sub

```

ReportControls.Item

Syntax

Return Value=*expression*.Item(*NameOrNum*)

Return Value=The Item method returns an Object of the control with the given name or number.

expression: Required expression that evaluates to a **ReportControls** object.

NameOrNum: Required **Variant** that indicates which control to return. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number given the control within the **ReportControls** collection of controls. If it is a **String**, it is the ID, (or ObjectCode property in the template editor) of the control.

Remarks

This method returns an Object of the control identified by the name or number in the *NameOrNum* parameter. Be aware that a hidden control called "Report" always exists in the Report template and cannot be deleted or otherwise manipulated. This object is used by PC-DMIS for internal purposes only. For this reason do not give *NameOrNum* a value of 1, as it will try to select the Report control.

To manipulate existing report objects, you will need use this method. Once you establish a pointer to a report object, you can get or set any of its properties (similar to the ReportControls.Add method). To find the available properties, consult the dockable **Properties** dialog box inside PC-DMIS.

ReportControls.Remove

Syntax

Return Value=*expression*.Remove(*NameOrNum*)

Return Value=The Item method returns a **Boolean** value with the result of the removal. If True, the control was removed. If False, it wasn't.

expression: Required expression that evaluates to a **ReportControls** object.

NameOrNum: Required **Variant** that indicates which control to remove. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number given the control within the **ReportControls**

collection of controls. If it is a **String**, it is the ID, (or ObjectCode property in the template editor) of the control.

Remarks

This deletes the specified control from the current section of the report template. Be aware that a hidden control called "Report" always exists in the Report template and cannot be deleted or otherwise manipulated. This object is used by PC-DMIS for internal purposes only. For this reason *NameOrNum* with a value of 1 will try to delete the Report control and will return False.

ReportTemplates Object Overview

The **ReportTemplates** object contains all open report templates in PC-DMIS's Report Template editor.

Using the ReportTeplates Collection

Use Add.Report to create a new report template and add it to the ReportTemplates collection.

Use ReportTemplates(*index*) where *index* is the report template name or index number to access an individual report template.

Properties:

ReportTemplates.Application

This property represents the read-only PC-DMIS application object. The **Application** object includes properties and methods that return top-level objects.

ReportTemplates.Count

This property returns a read-only number of open report templates.

ReportTemplates.Parent

This returns the read-only PC-DMIS application object which is the parent object of the ReportTemplates object. See the "Automation Objects Hierarchy Charts" for more information.

Methods:

ReportTemplates.Add

Syntax

Return Value=*expression*.Add()

Return Value: This function returns the added **ReportTemplate** object

expression: Required. An expression that returns a **ReportTemplates** object.

Remarks

The Add function creates a new report template in PC-DMIS.

ReportTemplates.Item

Syntax

Return Value=expression.Item(NameOrNum)

Return Value=The Item function returns the ReportTemplate Object with the given name or number.

expression: Required expression that evaluates to a **ReportTemplates** object.

NameOrNum: Required **Variant** that indicates which **ReportTemplates** object to return. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number of the **ReportTemplate** object in the **ReportTemplates** collection. If it is a **String**, it is the ID of the **ReportTemplate** object.

Remarks

Since the Item method is the default, the function name can be omitted as in Syntax 2.

ReportTemplates.Open

Syntax

Return Value=expression.Open(FileName)

Return Value: This function returns the opened **ReportTemplate** object. If the template does not exist, the function returns **Nothing**.

expression: Required. An expression that returns a **ReportTemplates** object.

FileName: Required **String**. The file name of the **ReportTemplate** to open.

Remarks

The Open Function activates the report template stored in the file *FileName*. If the template file does not exist, nothing happens.

ReportTemplate Object Overview

The **ReportTemplate** object allows you to get or set various settings for a report template.

Properties:

ReportTemplate.Application

This property represents the read-only PC-DMIS application object. The **Application** object includes properties and methods that return top-level objects.

ReportTemplate.Visible

This **boolean** property returns or sets the visibility status of the template editor. If True then Visible, if False, then hidden.

ReportTemplate.FullName

This property returns a read-only string of the full path and filename of the report template.

ReportTemplate.Name

This property returns a read-only string of the report template's filename.

ReportTemplate.Parent

This property returns the report template's parent object, which is the read-only Report Templates object.

ReportTemplate.Sections

This property returns a collection of all the report templates sections as a read-only Sections object.

Methods:

ReportTemplate.Close

Syntax

expression.Close

expression: Required expression that evaluates to a **ReportTemplate** object.

This subroutine closes the report template. To first save any unsaved changes, use the Save method.

ReportTemplate.Save

Syntax

expression.Save

expression: Required expression that evaluates to a **ReportTemplate** object.

This subroutine saves the report template with it's already existing name. If the template has not been saved before use the SaveAs method instead, and specify a filename.

ReportTemplate.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 **ReportTemplate** object.

name: Required expression that evaluates to a **String**. This is the pathway and file name to which you will save the report template.

This method saves the report template.

ReportWindow Object Overview

The **ReportWindow** object allows you to get or set various settings for the Report window.

Properties:

ReportWindow.Application

This property represents the read-only PC-DMIS application object. The **Application** object includes properties and methods that return top-level objects.

ReportWindow.Parent

This returns the read-only PC-DMIS application object which is the parent object of the ReportWindow object. See the "Automation Objects Hierarchy Charts" for more information.

ReportWindow.Visible

This returns or sets the visibility state of the Report window. Read/Write **Boolean**.

Methods:

ReportWindow.FullReportMode

Syntax

expression.FullReportMode

expression: Required expression that evaluates to a **ReportWindow** object.

This method switches the report window to Full Report Mode.

ReportWindow.LastExecutionReportMode

Syntax

expression.LastExecutionReportMode

expression: Required expression that evaluates to a **ReportWindow** object.

This method switches the report window to Last Execution Report Mode.

ReportWindow.LoadReportTemplate

Syntax

```
expression.LoadReportTemplate(filename)
```

expression: Required expression that evaluates to a **ReportWindow** object.

filename: This required **String** value specifies the pathway and filename of the template to load into the Report window.

This method loads the specified report template into the Report window.

ReportWindow.PrintReport

Syntax

```
expression.PrintReport
```

expression: Required expression that evaluates to a **ReportWindow** object.

This method prints the contents of the Report window.

ReportWindow.RefreshReport

Syntax

```
expression.RefreshReport
```

expression: Required expression that evaluates to a **ReportWindow** object.

This method reloads the report data into the report template, thereby refreshing the contents of the Report window.

ReportWindow.SetCurrentAsDefaultReport

Syntax

```
expression.SetCurrentAsDefaultReport
```

expression: Required expression that evaluates to a **ReportWindow** object.

This method sets the current report as the default report for the part program.

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.

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.

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.

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.

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.

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

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 continuous 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.

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.

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

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.

Scan.MethodCutPlane

This property represents the method's cut plane vector. Read/write **PointData** object.

Scan.MethodEnd

This property represents the scan's end point. Read/write **PointData** object.

Scan.MethodEndTouch

This property represents the method's end touch vector. Read/write **PointData** object.

Scan.MethodInitDir

This property represents the method's initial direction vector. Read/write **PointData** object.

Scan.MethodInitTopSurf

This property represents the initial Surface Vector for the Edge method. Read/write **PointData** object.

Scan.MethodInitTouch

This represents the method's initial touch vector. Read/write **PointData** object.

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	-	-

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.

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

Methods:

ReadCommBlock

Integer ReadCommBlock port:=(Integer), buffer:=(String), count:=(Integer)

Reads characters from the comm port specified.

RETURN VALUE: 0 if successful, -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.

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.

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

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 value 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 Manual 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

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.

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 **Dimensioned** 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.

Related Topics: Scan.MethodStart, Scan.MethodEnd, Scan.MethodInitTouch, Scan.MethodEndTouch, Scan.MethodInitDir, Scan.MethodCutPlane

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.

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.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

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 value 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

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.

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.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

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.

Section Object Overview

The **Section** object lets you manipulate a particular section from the collection of available **Selections** used by a report template.

Properties:

Section.Application

This property returns the PC-DMIS Application object.

Section.Name

This property Read/Write value returns or sets the name of the section.

Section.Number

This read only value returns a read-only **Long** value of the index number of the current section within the **Sections** object. This is used with the Sections.Item property to access the section later.

Section.Parent

This property returns a read-only parent object of the Section, the **Sections** object.

Section.ReportControls

This property returns the ReportControls object for this section. You can then access the ReportControls object to dynamically add or modify existing objects in the section

Methods:

None currently available.

Sections Object Overview

The **Sections** object contains a collection of all existing Section tabs for a given report template in PC-DMIS's Report Template editor.

Using the Sections Collection

Use `Sections.Add` to create a new section and add it to the Sections collection.

Use `Sections(index)` where *index* is a section's name or index number to access the section.

Properties:

Sections.Application

This property returns the PC-DMIS Application object.

Sections.Count

This property returns a read-only **Long** value of the number of sections.

Sections.Parent

This property returns the parent object for the Sections object, which is the PC-DMIS Application object.

Methods:

Sections.Add

Syntax

Return Value=`expression.Add(name)`

Return Value: This function returns the added **Section** object

expression: Required. An expression that returns a **Sections** object.

name: Required. This string value defines the name of the new section you want to add.

Remarks

This method adds a new section to the end of the template with the given name

Sections.InsertSectionBefore

Syntax

Return Value=`expression.InsertSectionBefore(name)`

Return Value: This function returns the added **Section** object

expression: Required. An expression that returns a **Sections** object.

name: Required. This string value defines the name of the existing section before which the new section will be inserted.

Remarks

This method inserts a new section before the named section. The new section will have the default internal name of "section" followed by a number.

Sections.Item

Syntax

Return Value=expression.Item(NameOrNum)

Return Value=The Item function returns the Section with the given name or number.

expression: Required expression that evaluates to a **Sections** object.

NameOrNum: Required **Variant** that indicates which **Section** object to return. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number of the **Section** object in the **Sections** collection. If it is a **String**, it is the ID, or name, of the **Section** object.

Remarks

This method returns the **Section** object identified by the *NameOrNum* parameter.

Sections.Remove

Syntax

Return Value=expression.Item(NameOrNum)

Return Value=A **Boolean** value that determines whether or not the specified section was removed. If **True** then the section was removed. If **False** then the section either remained or didn't exist in the first place.

expression: Required expression that evaluates to a **Sections** object.

NameOrNum: Required **Variant** that indicates which **Section** object to remove. It can be either a **Long** or a **String**. If it is a **Long**, it is the index number of the **Section** object in the **Sections** collection. If it is a **String**, it is the ID, or name, of the **Section** object.

Remarks

This method removes or deletes the **Section** object identified by the *NameOrNum* parameter.

Statistics Object Overview

The Statistics Automation object gives access to the properties and data members of the PC-DMIS Statistics command.

Properties:

Statistics.CalcMode

LONG value representing whether calculation mode inside of DataPage is turned off or on.

Read/Write **Long**

Statistics.MemoryPages

LONG value representing number of memory pages to be used by DataPage.

Read/Write **Long**

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.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.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.TransferDir

Syntax:

expression.TransferDir

Return Value: **String** value indicating the directory to which to move the stat file.

expression: Required expression that evaluates to a PC-DMIS **Statistics** object.

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**

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.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.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.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.

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 "Compensating for Temperature" in the "Setting Your Preferences" chapter of the *PC-DMIS Reference Manual*.

Properties:

TempComp.HighThreshold

DOUBLE value representing the high temperature threshold.

Read/Write **Double**

TempComp.LowThreshold

DOUBLE value representing the low temperature threshold.

Read/Write **Double**

TempComp.MaterialCoefficient

DOUBLE value indicating the material coefficient.

Read/Write **Double**

TempComp.RefTemp

DOUBLE value representing the reference temperature.

Read/Write **Double**

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”.

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.

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.

Tip Object Overview

The Tip object describes a single tip of a probe. All of its properties are read-only.

Properties:

Tip.A

Returns the A angle of the tip. Read-only **Double**.

Tip.B

Returns the B angle of the tip. Read-only **Double**.

Tip.Date

Returns the PC-DMIS representation of the most recent calibration date of the tip. Read-only **String**.

Tip.Diam

Returns the diameter of the tip. Read/Write **Double**.

Tip.ID

Returns the ID string of the tip. Read-only **String**.

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.MeasDiam

Returns or sets the measured diameter of the tip. Read/Write **Double**.

Tip.MeasThickness

Returns the measured thickness of the tip. Read-only **Double**.

Tip.MeasXYZ

Returns or sets the measured location of the tip. Read/Write **PointData**.

Tip.Parent

Returns the **Tips** collection object that contains this tip. Read-only.

Tip.Selected

Determines whether tip is selected for qualification. Read/Write **Boolean**

Remarks:

Use the "Probe.SelectAllTips" method of the probe object on page 173 to select all tips at once and the "Probe.ClearAllTips" method of the probe object on page 172 to clear all tips at once.

Tip.Thickness

Returns the nominal thickness of the tip. Read-only **Double**.

Tip.Time

Returns the PC-DMIS representation of the most recent calibration time of the tip. Read-only **String**.

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.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
TIPANALGOPTIC = 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.WristOffset

Returns the wrist offset of the tip. Read-only **PointData**.

Tip.WristTipIJK

Returns the wrist tip vector of the tip. Read-only **PointData**.

Tip.XYZ

Returns the location of the tip. Read/Write **PointData**.

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.

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.

Tips.Count

Represents the number of Tip objects in the parent Probe object. Read-only **Integer**.

Tips.Parent

Returns the parent **Probe** object. Read-only.

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.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.Remove

Syntax

expression.RemoveNum

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.

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.

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.

Tool.Diam

Returns the diameter of the tool. Read-only **Double**.

Tool.ID

Returns the ID of the tool. Read-only **String**.

Tool.Parent

Returns the parent **Tools** object. Read-only.

Tool.ShankIJK

Returns the shank vector of the tool as a **PointData**. Read-only.

Tool.ToolType

Returns the type of the tool. Read-only **Long**.

Remarks

There is only one type of tool currently available, TOOLSPHERE.

Tool.Width

Returns the width of the tool. Read-only **Double**.

Tool.XYZ

Returns the location of the tool. Read-only **PointData**.

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.

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.

Tools.Count

Represents the number of Tool objects in the parent **PartProgram** object. Read-only **Integer**.

Tools.Parent

Returns the parent **PartProgram** object. Read-only.

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.

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.

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.

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 "Using Trace Field" in the "Tracking Statistical Data" chapter section of the *PC-DMIS Reference Manual*.

Properties:

Tracefield.Name

STRING value representing the name of the tracefield.

Read/Write **String**

Tracefield.Value

STRING value representing the value for the tracefield.

Read/Write **String**

Variable Object Overview

The properties of the Variable Object allows you to return and set a variable's:

- Type
- Long value
- Double value
- String value
- Point value
- Command value

The methods of of this object return an array's:

- Upper bound if variable is an array
- Lower bound if variable is an array
- Gets the array variable at a specific position
- Sets the array variable at a specified position

Properties:

Variable.CommandValue

This returns / sets the command value of the variable. Read / write.

Variable.DoubleValue

This returns / sets the double value of the variable. Read / write **Double**.

Variable.LongValue

This returns / sets the long value of the variable. Read / write **Long**.

Variable.PointValue

This returns / sets the point value of the variable. Read / write.

Variable.StringValue

This returns / sets the string value of the variable. Read / write **String**.

Variable.VariableType

This returns / sets the current variable type. Read / write **VARIABLE_TYPE_TYPES**.

Methods:

Variable.GetArrayIndexValue

Syntax

Return Value=expression.GetArrayIndexValue

Return Value: This returns the array variable at the specified index position.

expression: required expression for object type **Variable**.

Variable.GetArrayLowerBound

Syntax

Return Value=expression.GetArrayLowerBound

Return Value: This returns the lower bound if the variable is an array. Otherwise it returns zero.

expression: required expression for object type **Variable**.

Variable.GetArrayUpperBound

Syntax

Return Value=expression.GetArrayUpperBound

Return Value: This returns the upper bound if the variable is an array. Otherwise it returns zero.

expression: required expression for object type **Variable**.

Variable.SetArrayIndexValue

Syntax

Return Value=expression.SetArrayIndexValue (Index, Variable)

Return Value: This sets the array variable at the specified index position. Type **Boolean**.

expression: required expression for object type **Variable**.

Index: Long value specifying the index position.

Variable: Array variable to be set.

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.

Important Notes:

Functions that return type Object are invalid.
Only OldBasic classes support optional parameters.

Important Miscellaneous Programming Notes

Using Parentheses in BASIC Scripts: For information on when to use or omit parentheses, please refer to your BASIC Language documentation; generally however, for methods and properties you should only use parentheses if you're receiving a value.

Invalid Function Return Type: Be aware that objects are not a valid return type for functions.

Functions 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

Application

This property returns the owning application object.

ArcCos

ArcCos x:=(Double)

Return the arc cosine of x in degrees.

ArcSin

ArcSin x:=(Double)

Returns the arc sine 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: This parameter specifies the workplane of the 2D alignment. Options come from the WPLANETYPE enumeration. These include PCD_TOP, PCD_BOTTOM, PCD_FRONT, PCD_BACK, PCD_LEFT, or PCD_RIGHT. You can also use these numerical values:

- PCD_TOP = 1
- PCD_BOTTOM = 2
- PCD_FRONT = 4
- PCD_BACK = 8
- PCD_LEFT = 16
- PCD_RIGHT = 32

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. Options come from the WPLANETYPE enumeration. These include PCD_TOP, PCD_BOTTOM, PCD_FRONT, PCD_BACK, PCD_LEFT, or PCD_RIGHT. You can also use these numerical values:

- PCD_TOP = 1
- PCD_BOTTOM = 2
- PCD_FRONT = 4
- PCD_BACK = 8
- PCD_LEFT = 16
- PCD_RIGHT = 32

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.

CloseCommConnection

CloseCommConnection port:=(Integer)

Closes the port opened with the OpenCommConnection command.

port: The comm port to close.

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.

CreateID

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

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

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

This function must be called to end an alignment block.

EndDim

EndDim takes no parameters, but must be called to finish off the dimension block.

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

Use this command to release the memory allocated for use by the StartGetFeatPoint and GetFeatPoint commands.

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

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. It 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_PH9WARNDDELTA, 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_RTSCSTS, PCD_XONXOFF. Optional. Default is PCD_RTSCSTS.

Functions P

Parent

This read-only property returns the owning part program object.

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.

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.
--

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.

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 successful. 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 successful. 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), width:=(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_PH9WARNDDELTA, 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

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.

SetSlaveMode

SetSlaveMode tog:=(Integer)

Tog: Turns slave mode off or on for all subsequent created commands. PCD_ON or PCD_OFF

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. Options come from the WPLANETYPE enumeration. These include PCD_TOP, PCD_BOTTOM, PCD_FRONT, PCD_BACK, PCD_LEFT, or PCD_RIGHT. You can also use these numerical values:

- PCD_TOP = 1
- PCD_BOTTOM = 2
- PCD_FRONT = 4
- PCD_BACK = 8
- PCD_LEFT = 16
- PCD_RIGHT = 32

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 used with PCD_TRIM.

PCD_TRIM: For auto edge points only. Should not be used with PCD_HEM. Default.

PCD_PIN: For auto circles, cylinders, ellipses, and slots. Do not use with PCD_NORM.

PCD_NORM: For auto circles, cylinders, ellipses, and slots. Do not use 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.

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_CALCUS. 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.

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

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: Options come from the WPLANETYPE enumeration. These include PCD_TOP, PCD_BOTTOM, PCD_FRONT, PCD_BACK, PCD_LEFT, or PCD_RIGHT. You can also use these numerical values:

- PCD_TOP = 1
- PCD_BOTTOM = 2
- PCD_FRONT = 4
- PCD_BACK = 8
- PCD_LEFT = 16
- PCD_RIGHT = 32

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.

Index

A

A 421

AboutAxis..... 184

ActiveComponent..... 380

ActiveConnection..... 380

ActiveMachine..... 363

ActivePartProgram..... 192

Add..... 392, 395, 416

AddBoundaryPoint..... 430

AddFeature..... 430

AddIndexSet..... 208

AddLevelFeat..... 189, 431

AddOriginFeat..... 190, 431

AddRotateFeat..... 190, 431

AddSkipNum..... 342

AlignCommand Members

AlignCommand.AboutAxis..... 184

AlignCommand.AddBestFitFeat..... 189

AlignCommand.AddLevelFeat..... 189

AlignCommand.AddOriginFeat..... 190

AlignCommand.AddRotateFeat..... 190

AlignCommand.Angle..... 184

AlignCommand.AverageError..... 184

AlignCommand.Axis..... 184

AlignCommand.BFOffset..... 185

AlignCommand.CadToPartMatrix..... 185

AlignCommand.CalculateMatrices..... 190

AlignCommand.ExternalFileID..... 185

AlignCommand.ExternalID..... 185

AlignCommand.FeatID..... 185

AlignCommand.FeatID2..... 186

AlignCommand.FindCad..... 186

AlignCommand.ID..... 186

AlignCommand.InitID..... 186

AlignCommand.IterativeLevelAxis..... 186

AlignCommand.IterativeOriginAxis..... 187

AlignCommand.IterativeRotateAxis..... 187

AlignCommand.MachineToPartMatrix..... 187

AlignCommand.MeasAllFeat..... 187

AlignCommand.MeasAllFeatAlways..... 187

AlignCommand.NumInputs..... 187

AlignCommand.Offset..... 188

AlignCommand.Parent..... 188

AlignCommand.PointTolerance..... 188

AlignCommand.RepierceCad..... 188

AlignCommand.UseBodyAxis..... 188

AlignCommand.Workplane..... 189

AlignmentCommand..... 237

AlignWorkPlane..... 302

Angle..... 184, 274, 334, 359

AngleOffset.....	338	Application.Restore	196
Application	233, 237, 263, 271, 295, 300, 352, 353, 363, 381, 384, 392, 395, 396, 398, 415, 416, 425	Application.SetActive	197
Application Members		Application.SpawnNewInstance	197
Application.ActivePartProgram	192	Application.StatusBar	194
Application.ApplicationEvents.....	192	Application.Top.....	194
Application.ApplicationSettings.....	192	Application.UserExit.....	195
Application.Caption.....	192	Application.VersionString.....	195
Application.CurrentUserDirectory.....	192	Application.Visible	195
Application.DefaultFilePath.....	193	Application.WaitUntilReady	197
Application.DefaultProbeFile.....	193	Application.Width.....	195
Application.FullName	193	Application.WriteRegistryBool	197, 201
Application.Height	193	Application.WriteRegistryDouble.....	198
Application.Help	195	Application.WriteRegistryDWORD.....	199
Application.Left	193	Application.WriteRegistryInt.....	199
Application.Machines.....	193	Application.WriteRegistryPoint.....	200
Application.MajorVersion	193	Application.WriteRegistrySettings.....	201
Application.Minimize	195	Application Object Events Members	
Application.MinorVersion	193	ApplicationObjectEvents.OnAddObject.....	202
Application.Name	193	ApplicationObjectEvents.OnClosePartProgram	203
Application.OperatorMode	194	ApplicationObjectEvents.OnConnectSlave	203
Application.PartPrograms	194	ApplicationObjectEvents.OnDisconnectSlave ..	203
Application.Path.....	194	ApplicationObjectEvents.OnEndExecution.....	204
Application.Post.....	196	ApplicationObjectEvents.OnObjectAboutToExecute.....	204
Application.Quit.....	196	ApplicationObjectEvents.OnObjectAboutToExecute2	204
Application.RemotePanelMode.....	194	ApplicationObjectEvents.OnObjectExecuted ...	205
		ApplicationObjectEvents.OnObjectExecuted2 .	205

ApplicationObjectEvents.OnOpenPartProgram	205	ArrowMultiplier	274
ApplicationObjectEvents.OnOpenRemotePanelDialog	206	Attach Members	
ApplicationObjectEvents.OnSavePartProgram	206	Attach.AttachedAlign	210
ApplicationObjectEvents.OnStartExecution	207	Attach.Execute	210
ApplicationObjectEvents.OnUpdateStatusMessage	207	Attach.ID	210
Application Settings Members		Attach.LocalAlign	210
ApplicationSettings.WarningDefault19	207	Attach.PartName	210
ApplicationSettings.WarningDefault48	207	AttachCommand	237
ApplicationSettings.WarningDefault60	207	AttachedAlign	210
ApplicationSettings.WarnNoSavePrg	207	AutoAdjustPH9	375
ApplicationSettings.WarnOKPh9	208	AutoCircularMove	303
ApplicationSettings.WarnOKRotPh9	208	AutoClearPlane	211, 303
ApplicationSettings.WarnOverwritingAlignment	208	AutoLabelPosition	375
ApplicationEvents	192	Automation Objects	
ApplicationSettings	192	accessing	177
ArcCos	431	command subobjects heirarchy chart	181
ArcSin	431	main overfiew heirarchy chart	180
Array Index Members		probe subobjects heirarchy chart	181
ArrayIndex.AddIndexSet	208	AutoMove	303
ArrayIndex.GetLowerBound	208	AutoMoveDistance	303
ArrayIndex.GetUpperBound	209	AutoPH9	303
ArrayIndex.RemoveIndexSet	209	AutoReadPos	304
ArrayIndex.SetLowerBound	209	AverageError	184
ArrayIndex.SetUpperBound	209	Axis	184, 274
ArrayIndexCommand	237	AxisLetter	275

B

B 422

Basic Help 7

BasicScanCommand.....237

BasicScanCommand Members

BasicScan.AddControlPoint.....217

BasicScan.AutoClearPlane.....211

BasicScan.BoundaryCondition.....211

BasicScan.BoundaryConditionAxisV.....212

BasicScan.BoundaryConditionCenter.....212

BasicScan.BoundaryConditionEndApproach....212

BasicScan.BoundaryConditionPlaneV.....212

BasicScan.BoundaryPointCount.....213

BasicScan.CreateBasicScan.....217

BasicScan.DisplayHits.....213

BasicScan.Filter.....213

BasicScan.GetBoundaryConditionParams.....218

BasicScan.GetBoundaryPoint.....218

BasicScan.GetControlPoint.....219

BasicScan.GetFilterParams.....219

BasicScan.GetHitParams.....220

BasicScan.GetMethodParams.....220

BasicScan.GetMethodPointData.....221

BasicScan.GetNomsParams.....222

BasicScan.GetParams.....222

BasicScan.HitType.....214

BasicScan.Method.....215

BasicScan.MethodCutPlane.....215

BasicScan.MethodEnd.....215

BasicScan.MethodEndTouch.....215

BasicScan.MethodInitDir.....215

BasicScan.MethodInitTopSurf.....215

BasicScan.MethodInitTouch.....215

BasicScan.MethodStart.....216

BasicScan.NominalMode.....216

BasicScan.OperationMode.....216

BasicScan.RemoveControlPoint.....223

BasicScan.SetBoundaryConditionParams.....223

BasicScan.SetBoundaryPoint.....224

BasicScan.SetControlPoint.....224

BasicScan.SetFilterParams.....225

BasicScan.SetHitParams.....225

BasicScan.SetMethodParams.....226

BasicScan.SetMethodPointData.....227

BasicScan.SetNomsParams.....228

BasicScan.SetParams.....228

BasicScan.SinglePoint.....217

BestFit2D.....432

BestFit3D.....432

BestFitMathType.....304

BFOffset.....185

Bonus.....273, 275

Bound.....304

BoundaryCondition	211, 400	CalculateMatrices	190
BoundaryConditionAxisV	212, 401	CalculateNominals.....	317
BoundaryConditionCenter	212, 401	Calibrate.....	432
BoundaryConditionEndApproach	212, 401	CalibrationCommand.....	238
BoundaryConditionPlaneV	212, 401	CancelChanges	385
BoundaryPointCount	213	Caption.....	192
BoxLength	304	CatchMotionError.....	432
BoxWidth	305	Check.....	31, 433
BufferSize.....	336	Check Boxes.....	31
<hr/>			
C			
CadModel Members			
CadModel.HighLightElement	231	CircularRadiusIn.....	305
CadModel.UnHighLightElement.....	232	CircularRadiusOut	305
CadToPartMatrix.....	185	ClearExecutionBlock	366
CadWindow Members			
CadWindow.Application	233	ClearMarked.....	265
CadWindow.Height.....	233	ClearPlane	356, 433
CadWindow.Left.....	233	Close.....	366, 397
CadWindow.Parent.....	233	CloseAll	377
CadWindow.Print.....	234	CloseCommConnection.....	433
CadWindow.SelectCADObject.....	234	Column132.....	434
CadWindow.Top.....	233	Command Members	
CadWindow.Visible	234	Command.ActiveTipCommand	237
CadWindow.Width.....	234	Command.AlignmentCommand	237
CadWindows.....	363	Command.Application	237
CalcMode	417	Command.ArrayIndex.....	237
		Command.AttachCommand	237
		Command.BasicScanCommand.....	237
		Command.CalibrationCommand.....	238
		Command.CommentCommand.....	238

Command.CopyMeastoNom	238	Command.IsCalibration	245
Command.Count	239	Command.IsComment	245
Command.Dialog	253	Command.IsConstructedFeature	246
Command.Dialog2	253	Command.IsDCCFeature	246
Command.DimensionCommand	239	Command.IsDimension	246
Command.DimFormat	240	Command.IsDimFormat	246
Command.DimInfoCommand	240	Command.IsDimInfo	246
Command.DisplayMetaFileCommand	240	Command.IsDisplayMetaFile	246
Command.Execute	253	Command.IsExpressionValid	256
Command.ExternalCommand	240	Command.IsExternalCommand	246
Command.Feature	240	Command.IsFeature	247
Command.FeatureCommand	240	Command.IsFileIOCommand	247
Command.FileIOCommand	242	Command.IsFlowControl	247
Command.FlowControlCommand	242	Command.IsHit	247
Command.GetExpression	253	Command.IsLeapFrog	247
Command.GetFieldValue	243	Command.IsLeitzMotion	247
Command.GetText	254	Command.IsLoadMachine	248
Command.GetToggleString	255	Command.IsLoadProbe	248
Command.GetToggleValue	243	Command.IsMeasuredFeature	248
Command.GetUniqueID	256	Command.IsModal	248
Command.HasBreakPoint	244	Command.IsMove	248
Command.ID	244	Command.IsOptionProbe	248
Command.IsActiveTip	244	Command.IsOptMotion	248
Command.IsAlignment	244	Command.IsScan	249
Command.IsArrayIndex	245	Command.IsStatistic	249
Command.IsAttach	245	Command.IsTempComp	249
Command.IsBasicScan	245	Command.IsTraceField	249

Command.Item.....	257	Command.TempCompCommand	252
Command.LeapfrogCommand	249	Command.TraceFieldCommand	252
Command.LeitzMotion.....	249	Command.TracksErrors	252
Command.LoadMachineCommand	250	Command.Type	252
Command.LoadProbeCommand	250	Command.TypeDescription.....	252
Command.Mark	257	Command.UnexpectedHit	252
Command.Marked.....	250	Commands	363
Command.MissedHit.....	250	Commands Members	
Command.ModalCommand.....	250	Commands.Add.....	263
Command.MoveCommand	251	Commands.Application.....	263
Command.Next	257	Commands.ClearMarked.....	265
Command.OptionProbeCommand.....	251	Commands.Count	263
Command.OptMotion.....	251	Commands.CurrentCommand	263
Command.Parent.....	251	Commands.FindByUniqueID	265
Command.Prev.....	258	Commands.GetCommandText.....	265
Command.PutText.....	258	Commands.InsertionPointAfter	266
Command.ReDraw	259	Commands.Item.....	266
Command.Remove	259	Commands.LastCommand	263
Command.RemoveExpression	259	Commands.MarkAll.....	267
Command.ScanCommand.....	251	Commands.Parent.....	263
Command.SetExpression	260	CommandValue	428
Command.SetToggleString.....	261	Comment.....	267, 434
Command.ShowIDOnCad	251	Comment Members	
Command.Skipped	251	Comment.AddLine	268
Command.SlaveArm	251	Comment.Comment	267
Command.SolveExpression	262	Comment.CommentType	267
Command.StatisticCommand.....	252	Comment.GetLine.....	268

DefaultFilePath	193	DimFormat.ShowDevSymbols	280
DefaultHits.....	435	DimFormat.ShowDimensionText	280
DefaultMachineName.....	193	DimFormat.ShowDimensionTextOptions	280
DefaultProbeFile	193	DimFormat.ShowHeadings	280
Delete	4	DimFormat.ShowStdDev	280
Depth.....	306	Dimension Information Members	
Description.....	271	DimInfo.GetFieldFormat.....	282
Dev.....	273	DimInfo.GetLocationAxis.....	283
DevAngle.....	273, 275	DimInfo.GetTruePosAxis.....	284
Deviation	275, 306	DimInfo.SetFieldFormat	285
Dialog2.....	253	DimInfo.SetLocationAxis	286
DIAM.....	334, 422, 425	DimInfo.SetTruePosAxis	287
Digits.....	356	DimensionCommand	239
DimData Members		DimensionCommand Members	
DimData.Bonus.....	273	DimensionCommand.Angle	274
DimData.Dev	273	DimensionCommand.ArrowMultiplier.....	274
DimData.DevAngle.....	273	DimensionCommand.Axis	274
DimData.Max	273	DimensionCommand.AxisLetter.....	275
DimData.Meas	273	DimensionCommand.Bonus.....	275
DimData.Min	273	DimensionCommand.DevAngle	275
DimData.Minus.....	273	DimensionCommand.Deviation.....	275
DimData.Nom.....	274	DimensionCommand.Feat1	275
DimData.Out.....	274	DimensionCommand.Feat2	275
DimData.Plus.....	274	DimensionCommand.Feat3	275
Dimension Format Members		DimensionCommand.GraphicalAnalysis.....	276
DimFormat.GetHeadingType.....	280	DimensionCommand.ID.....	276
DimFormat.SetHeadingType	281	DimensionCommand.Length.....	276

DimensionCommand.Max	276	DlgSetPicture.....	40
DimensionCommand.Measured	276	DlgValue DlgValue().....	40
DimensionCommand.Min.....	276	DmisMatrix Members	
DimensionCommand.Minus	276	DmisMatrix.Copy	289
DimensionCommand.Nominal	277	DmisMatrix.Inverse	289
DimensionCommand.OutputMode.....	277	DmisMatrix.IsIdentity.....	289
DimensionCommand.OutTol	277	DmisMatrix.Item	290
DimensionCommand.ParallelPerpendicular	277	DmisMatrix.Multiply	290
DimensionCommand.Parent	277	DmisMatrix.Normalize	290
DimensionCommand.Plus.....	278	DmisMatrix.OffsetAxis.....	289
DimensionCommand.Profile.....	278	DmisMatrix.PrimaryAxis	289
DimensionCommand.RadiusType.....	278	DmisMatrix.Reset.....	291
DimensionCommand.TextualAnalysis	278	DmisMatrix.RotateByAngle.....	291
DimensionCommand.TruePositionModifier	278	DmisMatrix.RotateToPoint	291
DimensionCommand.TruePosUseAxis.....	279	DmisMatrix.RotateToVector	292
DimensionCommand.UnitType.....	279	DmisMatrix.SecondaryAxis	289
DimFormat.....	435	DmisMatrix.SetMatrix	292
DimFormatCommand	240	DmisMatrix.TertiaryAxis	290
DimInfoCommand.....	240	DmisMatrix.TransformDataBack.....	293
Direction.....	359	DmisMatrix.TransformDataForward.....	294
DisplayConeAngle	306	DmisOut.....	366
DisplayHits	213	DoubleValue.....	428
DisplayMetaFileCommand	240		
Distance	356	<hr/>	
Distance2	357	<i>E</i>	
DlgControlId Function.....	38	EdgeMeasureOrder	306
DlgListBoxArray DlgListBoxArray().....	39	EdgeThickness.....	307
		EditWindow	363

EditWindow Members	EndAlign	435
EditWindow.Application	EndAngle	307, 386
EditWindow.CommandMode.....	EndAngle2	307, 334
EditWindow.GetCommandText	EndB.....	386
EditWindow.Height.....	EndDim.....	435
EditWindow.Left.....	EndGetFeatPoint.....	436
EditWindow.Parent	EndScan	436
EditWindow.Print.....	EquateAlign.....	436
EditWindow.ReportMode.....	ErrorMode.....	338
EditWindow.SetDMISOutputOptions	ErrorType	338
EditWindow.SetPrintOptions	Evaluate	318
EditWindow.SetPrintOptionsEx.....	Execute	210, 367
EditWindow.ShowAlignments.....	ExecutedCommands.....	363
EditWindow.ShowComments	ExecutedCommands Members	
EditWindow.ShowDimensions	ExecutedCommands.Application.....	300
EditWindow.ShowFeatures.....	ExecutedCommands.Count	299
EditWindow.ShowHeaderFooter.....	ExecutedCommands.FindByUniqueID	300
EditWindow.ShowHits	ExecutedCommands.Item.....	301
EditWindow.ShowMoves	ExecutedCommands.Parent	299
EditWindow.ShowOutToOnly	ExecuteMode.....	386
EditWindow.ShowTips.....	ExecutionWasCancelled.....	364
EditWindow.StatusBar	Exit	4
EditWindow.Top.....	Export	366, 367
EditWindow.Visible	Expression.....	336, 339
EditWindow.Width.....	ExternalCommand	240
EditWindowTextAll	ExternalFileID	185
EndA	ExternalID.....	185

F

F 334

FailIfExists.....336

Feat1.....275

Feat2.....275

Feat3.....275

FeatCommand Members

FeatCommand.AddInputFeat.....317

FeatCommand.AlignWorkPlane.....302

FeatCommand.AutoCircularMove.....303

FeatCommand.AutoClearPlane.....303

FeatCommand.AutoMove.....303

FeatCommand.AutoMoveDistance.....303

FeatCommand.AutoPH9.....303

FeatCommand.AutoReadPos.....304

FeatCommand.BestFitMathType.....304

FeatCommand.BoxLength.....304

FeatCommand.BoxWidth.....305

FeatCommand.CalculateNominals.....317

FeatCommand.CircularRadiusIn.....305

FeatCommand.CircularRadiusOut.....305

FeatCommand.CornerRadius.....305

FeatCommand.CountHits.....318

FeatCommand.DCCFindNomsMode.....305

FeatCommand.DCCMeasureInMasterMode.....305

FeatCommand.Depth.....306

FeatCommand.Deviation.....306

FeatCommand.DisplayConeAngle.....306

FeatCommand.EdgeMeasureOrder.....306

FeatCommand.EdgeThickness.....307

FeatCommand.EndAngle.....307

FeatCommand.EndAngle2.....307

FeatCommand.Evaluate.....318

FeatCommand.FilterType.....307

FeatCommand.FindHole.....307

FeatCommand.GenerateHits.....318

FeatCommand.GenericAlignMode.....307

FeatCommand.GenericDisplayMode.....308

FeatCommand.GenericType.....308

FeatCommand.GetData.....319

FeatCommand.GetHit.....321

FeatCommand.GetInputFeat.....322

FeatCommand.GetInputOffset.....323

FeatCommand.GetPoint.....323

FeatCommand.GetSampleHit.....324

FeatCommand.GetSurfaceVectors.....325

FeatCommand.GetVector.....325

FeatCommand.HighPointSearchMode.....309

FeatCommand.ID.....309

FeatCommand.Increment.....309

FeatCommand.Indent.....309

FeatCommand.Indent2.....309

FeatCommand.Indent3.....310

FeatCommand.InitHits.....	310	FeatCommand.SetHit2.....	331
FeatCommand.Inner.....	310	FeatCommand.SetInputOffset.....	333
FeatCommand.InteriorHit.....	310	FeatCommand.Spacer.....	315
FeatCommand.Line3D.....	311	FeatCommand.StartAngle.....	315
FeatCommand.MeasAngle.....	311	FeatCommand.StartAngle2.....	315
FeatCommand.MeasDiam.....	311	FeatCommand.TheoAngle.....	315
FeatCommand.MeasHeight.....	311	FeatCommand.TheoDiam.....	315
FeatCommand.MeasLength.....	311	FeatCommand.TheoHeight.....	315
FeatCommand.MeasMajorAxis.....	311	FeatCommand.TheoLength.....	316
FeatCommand.MeasMinorAxis.....	312	FeatCommand.TheoMajorAxis.....	316
FeatCommand.MeasPinDiam.....	312	FeatCommand.TheoMinorAxis.....	316
FeatCommand.MeasSmallLength.....	312	FeatCommand.TheoPinDiam.....	316
FeatCommand.MeasureSlotWidth.....	312	FeatCommand.Thickness.....	316
FeatCommand.NumHits.....	312	FeatCommand.Tolerance.....	317
FeatCommand.NumHitsPerRow.....	312	FeatCommand.UsePin.....	317
FeatCommand.NumRows.....	313		
FeatCommand.Parent.....	313	FeatData Members	
FeatCommand.PermHits.....	313	FeatData.ANGLE.....	334
FeatCommand.Polar.....	313	FeatData.DIAM.....	334
FeatCommand.PutData.....	326	FeatData.EndAngle.....	334
FeatCommand.PutPoint.....	328	FeatData.EndAngle2.....	334
FeatCommand.PutSurfaceVectors.....	328	FeatData.F.....	334
FeatCommand.PutVector.....	329	FeatData.I.....	334
FeatCommand.ReferenceID.....	314	FeatData.ID.....	334
FeatCommand.ReferenceType.....	314	FeatData.J.....	334
FeatCommand.RMeasFeature.....	314	FeatData.K.....	335
FeatCommand.SetHit.....	330	FeatData.LENGTH.....	335
		FeatData.P1.....	335

FeatData.P2.....	335	FilePointerID.....	338
FeatData.SmallDiam.....	335	Filter.....	213, 402
FeatData.StartAngle.....	335	FilterType.....	307
FeatData.StartAngle2.....	335	Find D2HBFind80.....	4
FeatData.TP.....	335	Find Next D2HBFind_Next80.....	5
FeatData.X.....	335	FindByUniqueID.....	265, 300
FeatData.Y.....	335	FindCad.....	186
FeatData.Z.....	336	FindHole.....	307
FeatID.....	185	Flatness.....	436
FeatID2.....	186	FlowControlCommand.....	242
Feature.....	240	FlowControlCommand Members	
FeatureCommand.....	240	FlowControlCommand.AddArgument.....	342
File IO Members		FlowControlCommand.AddSkipNum.....	342
FileIO.BufferSize.....	336	FlowControlCommand.AngleOffset.....	338
FileIO.Expression.....	336	FlowControlCommand.ErrorMode.....	338
FileIO.FailIfExists.....	336	FlowControlCommand.ErrorType.....	338
FileIO.FileIOType.....	336	FlowControlCommand.Expression.....	339
FileIO.FileName1.....	337	FlowControlCommand.FileName.....	339
FileIO.FileName2.....	337	FlowControlCommand.GetArgumentDescription.....	343
FileIO.FileOpenType.....	337	FlowControlCommand.GetArgumentExpression.....	343
FileIO.FilePointerID.....	338	FlowControlCommand.GetArgumentName.....	343
FileIO.VariableID.....	338	FlowControlCommand.GetEndNum.....	339
FileIOCommand.....	242	FlowControlCommand.GetLeftSideOfExpression.....	344
FileIOType.....	336	FlowControlCommand.GetRightSideOfExpression.....	344
FileName1.....	337	FlowControlCommand.GetSkipNum.....	344
FileName2.....	337		
FileOpenType.....	337		

FlowControlCommand.ID	339
FlowControlCommand.IsExpressionValid.....	345
FlowControlCommand.IsValidLeftHandValue.	345
FlowControlCommand.IsValidSubroutineArgumentName	345
FlowControlCommand.Label.....	340
FlowControlCommand.NumArguments	340
FlowControlCommand.RemoveArgument.....	346
FlowControlCommand.RemoveSkipNum	346
FlowControlCommand.ReportAutoPrint	340
FlowControlCommand.SetArgumentDescription	346
FlowControlCommand.SetArgumentExpression	347
FlowControlCommand.SetArgumentName	347
FlowControlCommand.SetLeftSideOfAssignment	348
FlowControlCommand.SetRightSideOfAssignment	348
FlowControlCommand.SkipCount	340
FlowControlCommand.StartNum.....	340
FlowControlCommand.SubName.....	341
FlowControlCommand.XAxisOffset	341
FlowControlCommand.YAxisOffset	341
FlowControlCommand.ZAxisOffset.....	341
FPanel.....	352
Full	349
FullName	193, 364, 381, 396

G

GapOnly.....	436
GenerateHits.....	318
GenericAlignMode	307
GenericDisplayMode	308
GenericType	308
GetArgumentDescription	343
GetArgumentExpression	343
GetArgumentName.....	343
GetArrayIndexValue.....	429
GetArrayLowerBound.....	429
GetBoundaryConditionParams	218
GetBoundaryPoint	218
GetColorList.....	390
GetCommand.....	390
GetCommandText.....	265
GetControlPoint.....	219
GetCount.....	390
GetData.....	319
GetDataTypeInfo	271
GetDimData D2HBGetDimData158.....	437
GetDimOutTol.....	437
GetEndNum.....	339
GetExpression	253
GetFeatID.....	439
GetFeatPoint.....	439

GetFeature	439	GetToggleString	255
GetFieldValue	243	GetToggleValue	243
GetFilterParams	219, 407	GetTolColor	390
GetHit	321	GetTopMachineSpeed	440
GetHitParams	220, 408	GetTruePosAxis	284
GetInputFeat	322	GetType	440
GetInputOffset	323	GetUniqueID	256
GetLeftSideOfExpression	344	GetUnits	441
GetLine	268	GetUpperBound	209
GetLocationAxis	283	GetValue	390
GetLowerBound	208	GetValue2	390
GetMethodParams	220	GetVariableValue	367
GetMethodPointData	221, 409	GetVector	325
GetNomsParams	222, 410	GraphicalAnalysis	276
GetParams	222, 410		
GetPH9Status	440	<hr/>	
GetPoint	323	<i>H</i>	
GetProbeOffsets	440	HasBreakPoint	244
GetProbeRadius	440	HasCommandData	390
GetProgramOption	440	Height	193, 233, 295
GetProgramValue	440	Help	195
GetRightSideOfExpression	344	HighLightElement	231
GetSampleHit	324	HighPointSearchMode	309
GetSkipNum	344	HighThreshold	420
GetStatsDir	419	Hit Function	441
GetSurfaceVectors	325	HitType	214, 403
GetText	254	<hr/>	
		<i>I</i>	
		I 269, 334, 379	

ID	186, 210, 244, 268, 276, 309, 334, 339, 422, 426	IsDimension	246
IgnoreMotionError	441	IsDimFormat	246
IJK	359, 422	IsDimInfo	246
Import	368	IsDisplayMetaFile	246
Increment	309	IsExpressionValid	256, 345
IncrementA	386	IsExternalCommand	246
IncrementB	386	IsFeature	247
Indent	309	IsFileIOCommand	247
Indent2	309	IsFlowControl	247
Indent3	310	IsFPPanel	352
InitHits	310	IsHit	247
InitID	186	IsIdentity	289
Inner	310	IsLeapFrog	247
Input	268	IsLeitzMotion	247
InsertionPointAfter	266	IsLoadMachine	248
InsertSectionBefore	416	IsLoadProbe	248
InteriorHit	310	IsLocationAxis	276
Inverse	289	IsMeasuredFeature	248
IsActiveTip	244	IsModal	248
IsAlignment	244	IsMove	248
IsArrayIndex	245	IsOptionProbe	248
IsAttach	245	IsOptMotion	248
IsBasicScan	245	IsProbeAnalog	364
IsCalibration	245	IsScan	249
IsComment	245	IsStatistic	249
IsConstructedFeature	246	IsTempComp	249
IsDCCFeature	246	IsTraceField	249

IsTruePosAxis.....	276
IsValidLeftHandValue.....	345
IsValidSubroutineArgumentName	345
Item . 257, 266, 301, 377, 385, 392, 395, 416, 424, 427	
Iterate	441
IterativeLevelAxis.....	186
IterativeOriginAxis.....	187
IterativeRotateAxis.....	187

J

J	269, 334, 379
---	---------------

K

K	270, 335, 379
---	---------------

L

Label.....	340
LastCommand.....	263
Leapfrog Members	
Leapfrog.LeanfrogFull.....	349
Leapfrog.LeanfrogNumhits.....	349
Leapfrog.LeanfrogType	349
LeapfrogCommand.....	249
LeapfrogType.....	349
Left.....	193, 233, 295
Leitz Motion Members	
LeitzMot.LowForce.....	350
LeitzMot.MaxForce.....	350
LeitzMot.PositionalAccuracy.....	350

LeitzMot.ProbeAccuracy.....	350
LeitzMot.ReturnData	350
LeitzMot.ReturnSpeed	351
LeitzMot.ScanPointDensity.....	351
LeitzMot.TriggerForce.....	351
LeitzMot.UpperForce.....	351
LeitzMotionCommand	249
Length.....	276, 335
Level	442
Line3D.....	311
LoadLayout	368
LoadMachineCommand.....	250
LoadProbe	442
LoadProbeCommand	250
LocalAlign	210
LongValue.....	429
LowForce	350, 361
LowThreshold	420

M

Machine Members	
Machine.Application.....	352
Machine.FPanel	352
Machine.IsFPanel	352
Machine.Name.....	352
Machine.Parent.....	352
Machines.....	193

Machines Members		MaxTSpeed	361
Machines.Application	353	MaxXAcceleration	361
Machines.Count	353	MaxYAcceleration	361
Machines.Parent	353	MaxZAcceleration	361
MachineToPartMatrix	187	Meas	273
MajorVersion	193	MeasAllFeat	187
Mark	257	MeasAllFeatAlways	187
MarkAll	267	MeasAngle	311
Marked	250	MeasDiam	311, 422
MasterSlaveDlg	369	MeasHeight	311
MasterSlaveDlg Members		MeasLength	311
MasterSlaveDlg.Applications	354	MeasMajorAxis	311
MasterSlaveDlg.DCC	354	MeasMinorAxis	312
MasterSlaveDlg.MasterProbe	354	MeasPinDiam	312
MasterSlaveDlg.MasterTip	354	MeasSmallLength	312
MasterSlaveDlg.MeasuringArm	355	MeasThickness	422
MasterSlaveDlg.Parent	355	Measured	276
MasterSlaveDlg.Position	355	MeasureSlotWidth	312
MasterSlaveDlg.SlaveProbe	355	MeasXYZ	422
MasterSlaveDlg.SlaveTip	355	MemoryPages	418
MasterSlaveDlg.Tool	355	MessageBox	369
MaterialCoefficient	420	Method	215, 404
Max	273, 276	MethodCutPlane	215, 404
MaxForce	350, 362	MethodEnd	215, 404
Maximize	195	MethodEndTouch	215, 404
MaxMineAve	442	MethodInitDir	215, 405
MaxTAcceleration	361	MethodInitTopSurf	215, 405

MethodInitTouch.....	215, 405
MethodStart	216, 405
Min.....	273, 276
Minimize	196
MinorVersion.....	193
Minus	273, 276
MissedHit	250
ModalCommand.....	250
ModalCommand Members	
ModalCommand.ClearPlane	356
ModalCommand.Digits.....	356
ModalCommand.Distance.....	356
ModalCommand.Distance2.....	357
ModalCommand.Mode.....	357
ModalCommand.Name.....	357
ModalCommand.On	357
ModalCommand.Parent	358
ModalCommand.PassPlane.....	358
ModalCommand.RmeasMode.....	358
ModalCommand.Speed.....	358
ModalCommand.WorkPlane.....	358
Mode	357, 386, 442
Move	442
MoveCommand.....	251
MoveCommand Members	
MoveCommand.Angle.....	359
MoveCommand.Direction.....	359

MoveCommand.IJK.....	359
MoveCommand.NewTip.....	359
MoveCommand.OldTip	360
MoveCommand.Parent.....	360
MoveCommand.XYZ.....	360
MovePositionalAccuracy	361
MoveSpeed.....	386, 443
Multiply	290

N

Name.....	193, 352, 357, 364, 381, 396, 415
NameType.....	418
NewTip	359
Next	257
Nom	274
Nominal	277
NominalMode.....	216, 405
Normalize.....	290
NumArguments	340
Number	415
NumHits.....	312, 349, 386
NumHitsPerRow.....	312
NumInputs.....	187
NumLevels.....	386
NumRows.....	313

O

Offset	188, 387
--------------	----------

OffsetAxis.....	289	OperationMode.....	216, 405
OK and Cancel Buttons	29	OperatorMode	194
OldBasic	364	Opt Motion Members	
OldTip	360	OptMotion.MaxTAcceleration.....	361
OLE Automation		OptMotion.MaxTSpeed.....	361
What is OLE Automation?.....	48	OptMotion.MaxXAcceleration	361
OLE Automation	48	OptMotion.MaxYAcceleration	361
On 357		OptMotion.MaxZAcceleration.....	361
OnAddObject.....	202, 372	OptMotion.MovePositionalAccuracy	361
OnClosePartProgram.....	203	Opt Probe Members	
OnConnectSlave.....	203	OptProbe.LowForce	361
OnDisconnectSlave	203	OptProbe.MaxForce	362
OnEndExecution	204, 372	OptProbe.PositionalAccuracy.....	362
OnExecuteDialogErrorMsg.....	373	OptProbe.ProbeAccuracy	362
OnExecuteDialogStatusMsg	373	OptProbe.ReturnData.....	362
OnObjectAboutToExecute.....	204, 373	OptProbe.ReturnSpeed.....	362
OnObjectAboutToExecute2.....	204, 373	OptProbe.ScanPointDensity	362
OnObjectExecuted.....	205, 374	OptProbe.TriggerForce	362
OnObjectExecuted2.....	205, 374	OptProbe.UpperForce	362
OnOpenPartProgram	205	Option Buttons and Group Boxes	34
OnOpenRemotePanelDialog	206	OptionProbeCommand.....	251
OnSavePartProgram	206	OptMotionCommand	251
OnStartExecution	207	Out.....	274
OnUpdateStatusMessage	207	OutputMode	277
OnWorkOffset.....	374	OutTol.....	277
Open.....	378, 395		
OpenCommConnection	443		

P

P1 335

P2 335

ParallelPerpendicular277

Parent 188, 233, 251, 263, 271, 277, 295, 299, 313,
352, 353, 358, 360, 364, 381, 384, 392, 395, 396,
398, 415, 416, 422, 426

PartName 210, 364

PartProgram Members

PartProgram.ActiveMachine363

PartProgram.Application363

PartProgram.AsyncExecute.....366

PartProgram.CadWindows.....363

PartProgram.ClearExecutionBlock.....366

PartProgram.Close.....366

PartProgram.Commands.....363

PartProgram.ConnectedToMaster.....363

PartProgram.ConnectedToSlave.....363

PartProgram.DmisOut366

PartProgram.EditWindow363

PartProgram.EditWindowTextAll363

PartProgram.Execute367

PartProgram.ExecutedCommands363

PartProgram.ExecutionWasCancelled364

PartProgram.Export.....367

PartProgram.FullName364

PartProgram.GetVariableValue.....367

PartProgram.Import.....368

PartProgram.IsProbeAnalog 364

PartProgram.LoadLayout 368

PartProgram.MasterSlaveDlg 369

PartProgram.MessageBox 369

PartProgram.Name..... 364

PartProgram.OldBasic..... 364

PartProgram.OnAddObject..... 372

PartProgram.OnEndExecution 372

PartProgram.OnExecutedDialogErrorMsg 373

PartProgram.OnExecutedDialogStatusMsg..... 373

PartProgram.OnObjectAboutToExecute 373

PartProgram.OnObjectAboutToExecute2..... 373

PartProgram.OnObjectExecuted 374

PartProgram.OnObjectExecuted2 374

PartProgram.OnWorkOffset 374

PartProgram.Parent 364

PartProgram.PartName..... 364

PartProgram.Path..... 364

PartProgram.Probes 365

PartProgram.Quit..... 369

PartProgram.RefreshPart..... 370

PartProgram.RevisionNumber 365

PartProgram.RunJournalFile..... 370

PartProgram.Save 370

PartProgram.SaveAs 370

PartProgram.SerialNumber..... 365

PartProgram.SetExecutionBlock..... 371

PartProgram.SetVariableValue.....	371	PointData.K.....	379
PartProgram.StatsCount.....	365	PointData.X.....	379
PartProgram.Tools.....	365	PointData.Y.....	380
PartProgram.Units.....	365	PointData.Z.....	380
PartProgram.Visible.....	365	PointDensity.....	270
PartProgram.WaitUntilExecuted.....	372	PointTolerance.....	188
PartProgram Settings Members		PointValue.....	429
PartProgramSettings.AutoAdjustPH9.....	375	Polar.....	313
PartProgramSettings.AutoLabelPosition.....	375	PositionalAccuracy.....	350, 362
PartProgramSettings.WarnLoadProbe.....	376	Post.....	196
PartPrograms.....	194	Prehit.....	387, 444
PartPrograms Object Members		Prev.....	258
PartPrograms.Add.....	376	PrimaryAxis.....	289
PartPrograms.CloseAll.....	377	Print.....	3, 234, 297
PartPrograms.Item.....	377	Print Preview.....	3
PartPrograms.Open.....	378	Probe Members	
PartPrograms.Remove.....	378	Probe.ActiveComponent.....	380
PassPlane.....	358	Probe.ActiveConnection.....	380
Paste.....	4	Probe.Application.....	381
Path.....	194, 364, 381	Probe.ClearAllTips.....	382
PermHits.....	313	Probe.ComponentCount.....	381
PHSAPriority.....	387	Probe.ComponentDescription.....	382
PHSTol.....	387	Probe.ConnectionCount.....	381
Plus.....	274, 278	Probe.ConnectionDescription.....	382
PointData Members		Probe.Dialog.....	383
PointData.I.....	379	Probe.FullName.....	381
PointData.J.....	379	Probe.Name.....	381

Probe.Parent.....	381
Probe.Path.....	381
Probe.QualificationSettings.....	381
Probe.Qualify.....	383
Probe.Qualify2.....	383
Probe.SelectAllTips.....	383
Probe.Tips.....	382
Probe.UseWristMap.....	382
ProbeAccuracy.....	350, 362
ProbeComp.....	444
Probes.....	365
Probes Members	
Probes.Add.....	384
Probes.Application.....	384
Probes.CancelChanges.....	385
Probes.Count.....	384
Probes.Item.....	385
Probes.Parent.....	384
Probes.Visible.....	384
Profile.....	278
PutData.....	326
PutFeatData.....	444
PutPoint.....	328
PutSurfaceVectors.....	328
PutText.....	258
PutVector.....	329

Q

QualificationSettings.....	381
QualificationSettings Members	
QualificationSettings.CreateReplaceMap.....	385
QualificationSettings.EndA.....	386, 387
QualificationSettings.EndAngle.....	386
QualificationSettings.EndB.....	386, 387
QualificationSettings.ExecuteMode.....	386
QualificationSettings.GetTool.....	388
QualificationSettings.IncrementA.....	386
QualificationSettings.IncrementB.....	386
QualificationSettings.Mode.....	386
QualificationSettings.MoveSpeed.....	386
QualificationSettings.NumHits.....	386
QualificationSettings.NumLevels.....	386
QualificationSettings.Offset.....	387
QualificationSettings.PHSAPriority.....	387
QualificationSettings.PHSTol.....	387
QualificationSettings.Prehit.....	387
QualificationSettings.ShankCheck.....	387
QualificationSettings.ShankHits.....	387
QualificationSettings.StartAngle.....	387
QualificationSettings.Tool.....	388
QualificationSettings.ToolMoved.....	387
QualificationSettings.ToolOnRotaryTable.....	388
QualificationSettings.ToolOverrideI.....	388

QualificationSettings.ToolOverrideJ.....	388	RemoveInputFeat.....	330
QualificationSettings.ToolOverrideK.....	388	RemoveLine.....	268
QualificationSettings.TouchSpeed.....	388	RemoveSkipNum.....	346
QualificationSettings.UserDefinedCalibrationMod e.....	388	RemoveStatsDir.....	419
QualificationSettings.UserDefinedCalibrationOrde r.....	388	RepierceCad.....	188
Qualify.....	383	ReportAutoPrint.....	340
Qualify2.....	383	ReportControls.....	415
Quit.....	196, 369	ReportControls Members	
<hr/>			
R		ReportControls.Add.....	392
Radius.....	270	ReportControls.Application.....	392
RadiusType.....	278	ReportControls.Count.....	392
ReadCommBlock.....	406, 444	ReportControls.Item.....	392
ReadLock.....	418	ReportControls.Parent.....	392
RecallEx.....	444	ReportControls.Remove.....	392
RecallIn.....	444	ReportData Members	
ReDraw.....	259	Report.GetColorList.....	390
ReferenceID.....	314	Report.GetCount.....	390
ReferenceType.....	314	Report.GetTolColor.....	390
RefreshPart.....	370	Report.GetValue2.....	390
RefTemp.....	420	Report.HasCommandData.....	390
RemotePanelMode.....	194	ReportData.GetCommand.....	390
Remove.....	259, 378, 392, 416, 425, 427	ReportData.GetValue.....	390
RemoveArgument.....	346	ReportMode.....	297
RemoveControlPoint.....	223	ReportTemplate Members	
RemoveExpression.....	259	ReportTemplate.Application.....	396
RemoveIndexSet.....	209	ReportTemplate.Close.....	397
		ReportTemplate.FullName.....	396

ReportTemplate.Name.....	396	RetroOnly.....	445
ReportTemplate.Parent.....	396	ReturnData.....	350, 362
ReportTemplate.Save.....	397	ReturnSpeed.....	351, 362
ReportTemplate.SaveAs.....	397	RevisionNumber.....	365
ReportTemplate.Sections.....	396	RMeasFeature.....	314
ReportTemplate.Visible.....	396	RmeasMode.....	358
ReportTemplates Members		Rotate D2HBRotate166.....	445
ReportTemplates.Add.....	395	RotateByAngle.....	291
ReportTemplates.Application.....	395	RotateCircle.....	445
ReportTemplates.Count.....	395	RotateOffset.....	445
ReportTemplates.Item.....	395	RotateToPoint.....	291
ReportTemplates.Open.....	395	RotateToVector.....	292
ReportTemplates.Parent.....	395	Roundness.....	446
ReportTemplates.Application.....	395	RunJournalFile.....	370
ReportWindow Members		Runout.....	446
ReportWindow.Application.....	398		
ReportWindow.FullReportMode.....	398	S	
ReportWindow.LastExecutionReportMode.....	398	Save.....	370, 396
ReportWindow.LoadReportTemplate.....	398	SaveAlign.....	446
ReportWindow.Parent.....	398	SaveAs.....	370, 396
ReportWindow.PrintReport.....	398	ScanCommand.....	251
ReportWindow.RefreshReport.....	398	ScanCommand Members	
ReportWindow.SetCurrentAsDefaultReport.....	398	Scan.BoundaryCondition.....	400
ReportWindow.Visible.....	398	Scan.BoundaryConditionAxisV.....	401
Reset.....	291	Scan.BoundaryConditionCenter.....	401
Restore.....	196	Scan.BoundaryConditionEndApproach.....	401
Retract.....	445	Scan.BoundaryConditionPlaneV.....	401

Scan.Filter.....	402	Section.Application.....	415
Scan.GetBoundaryConditionParams.....	407	Section.Name	415
Scan.GetFilterParams	407	Section.Number	415
Scan.GetHitParams.....	408	Section.Parent.....	415
Scan.GetMethodPointData.....	409	Section.ReportControls	415
Scan.GetNomsParams	410	Sections.....	396
Scan.GetParams	410	Sections Members	
Scan.HitType	403	Sections.Add	416
Scan.Method	404	Sections.Application	416
Scan.MethodCutPlane	404	Sections.Count.....	416
Scan.MethodEnd	404	Sections.InsertSectionBefore.....	416
Scan.MethodEndTouch.....	404	Sections.Item	416
Scan.MethodInitDir	405	Sections.Parent	416
Scan.MethodInitTopSurf	405	Sections.Remove	416
Scan.MethodInitTouch	405	Select All.....	4
Scan.MethodStart.....	405	SelectAllTips.....	383
Scan.NominalMode.....	405	SelectCADObject	234
Scan.OperationMode.....	405	Selected.....	422
Scan.SetBoundaryConditionParams	411	Sensors.....	420
Scan.SetFilterParams.....	411	SerialNumber.....	365
Scan.SetHitParams	412	SetActive.....	197
Scan.SetMethodPointData	413	SetArgumentDescription	346
Scan.SetNomsParams	413	SetArgumentExpression.....	347
Scan.SetParams.....	406, 414	SetArgumentName.....	347
ScanPointDensity	351, 362	SetArrayIndexValue.....	429
SecondaryAxis	289	SetAutoParams	446
Section Members		SetAutoVector	447

SetBoundaryConditionParams	223, 411	SetProgramOption	448
SetBoundaryPoint.....	224	SetProgramValue.....	448
SetControlPoint.....	224	SetReportOptions.....	449
SetDMISOOutputOptions.....	297	SetRightSideOfAssignment.....	348
SetExecutionBlock	371	SetRmeasMode.....	449
SetExpression.....	260	SetScanHitParams.....	449
SetFieldFormat.....	285	SetScanHit Vectors	449
SetFilterParams	225, 411	SetScanParams	449
SetHeadingType.....	281	SetScanVectors.....	450
SetHit	330	SetSlaveMode.....	450
SetHit2.....	331	SetStatsDir	420
SetHitParams	225, 412	SetTheos.....	450
SetInputFeat.....	332	SetToggleString.....	261
SetInputOffset.....	333	SetTool.....	388
SetLeftSideOfAssignment	348	SetTruePosAxis.....	287
SetLine	269	SetUpperBound	209
SetLocationAxis.....	286	SetVariableValue.....	371
SetLowerBound	209	ShankCheck.....	387
SetMatrix.....	292	ShankHits	387
SetMethodParams.....	226	ShankIJK.....	426
SetMethodPointData.....	227, 413	Shell.....	47
SetNoms	447	ShowAlignments.....	295
SetNomsParams	228, 413	ShowComments.....	295
SetOrigin	421	ShowDevSymbols.....	280
SetParams	228, 414	ShowDimensions	295
SetPrintOptions	298, 448	ShowDimensionText.....	280
SetPrintOptionsEx	298	ShowDimensionTextOptions.....	280

ShowFeatures.....	295	StartGetFeatPoint D2HBStartGetFeat174	454
ShowHeaderFooter.....	295	StartNum.....	340
ShowHeadings	280	StartScan	454
ShowHits	295	StatisticCommand.....	252
ShowIDOnCad.....	251	Statistics Members	
ShowMoves	296	Statistics.AddStatsDir	419
ShowOutTolOnly	296	Statistics.CalcMode	417
ShowStdDev	280	Statistics.GetStatsDir	419
ShowTips.....	296	Statistics.MemoryPages	418
ShowXYZWindow	451	Statistics.ReadLock.....	418
SinglePoint.....	217	Statistics.RemoveStatsDir	419
SkipCount.....	340	Statistics.SetStatsDir.....	420
Skipped.....	251	Statistics.Statistics.NameType	418
SlaveArm.....	251	Statistics.StatMode	418
Sleep.....	451	Statistics.TransferDir	418
SmallDiam.....	335	Statistics.WriteLock	419
SolveExpression.....	262	StatMode.....	418
Spacer.....	315	Stats	455
SpawnNewInstance	197	StatsCount	365
Speed.....	358	StatusBar	194, 296
StartA	387	Straitness.....	455
StartAlign	451	StringValue	429
StartAngle.....	315, 335, 387	SubName.....	341
StartAngle2.....	315, 335		
StartB	387	<hr/>	
StartDim	451	<i>T</i>	
StartFeature.....	452	TempCompCommand	252
		Temperature Compensation Members	

TempComp.GetOrigin.....	421	Tip.IJK.....	422
TempComp.HighThreshold.....	420	Tip.MeasDiam.....	422
TempComp.LowThreshold.....	420	Tip.MeasThickness.....	422
TempComp.Material Coefficient.....	420	Tip.MeasXYZ.....	422
TempComp.RefTemp.....	420	Tip.Parent.....	422
TempComp.Sensors.....	420	Tip.Selected.....	422
TempComp.SetOrigin.....	421	Tip.Thickness.....	423
TertiaryAxis.....	290	Tip.Time.....	423
Text Boxes and Text.....	32	Tip.TipNum.....	423
TextualAnalysis.....	278	Tip.Type.....	423
TheoAngle.....	315	Tip.WristOffset.....	423
TheoDiam.....	315	Tip.WristTipIJK.....	424
TheoHeight.....	315	Tip.XYZ.....	424
TheoLength.....	316	TipNum.....	423
TheoMajorAxis.....	316	Tips.....	382
TheoMinorAxis.....	316	Tips Members	
TheoPinDiam.....	316	Tips.Add.....	424
TheoSmallLength.....	316	Tips.Item.....	424
Thickness.....	316, 423	Tips.Remove.....	425
Time.....	423	Tolerance.....	317
Tip.....	456	Tool Members	
Tip Members		Tool.Application.....	425
Tip.A.....	421	Tool.Diam.....	425
Tip.B.....	422	Tool.ID.....	426
Tip.Date.....	422	Tool.Parent.....	426
Tip.Diam.....	422	Tool.ShankIJK.....	426
Tip.ID.....	422	Tool.ToolType.....	426

Tool.Width.....	426
Tool.XYZ.....	426
ToolMoved.....	387
ToolOnRotaryTable.....	388
ToolOverrideI.....	388
ToolOverrideJ.....	388
ToolOverrideK.....	388
Tools.....	365
Tools Members	
Tools.Add.....	427
Tools.Item.....	427
Tools.Remove.....	427
ToolType.....	426
Top.....	194, 233, 296
TouchSpeed.....	388, 456
TP 335	
Trace.....	456
TraceFieldCommand.....	252
TracksErrors.....	252
TransferDir.....	418
TransformDataBack.....	293
TransformDataForward.....	294
Translate.....	456
TranslateOffset.....	456
TriggerForce.....	351, 362
TruePositionModifier.....	278
TruePosUseAxis.....	279

Type.....	252, 270, 271, 423
TypeDescription.....	252

U

Undo.....	4
UnexpectedHit.....	252
UnHighLightElement.....	232
Units.....	365
UnitType.....	279
UpperForce.....	351, 362
UseBodyAxis.....	188
UsePin.....	317
UserDefinedCalibrationMode.....	388
UserDefinedCalibrationOrder.....	388
UserExit.....	195
UseWristMap.....	382

V

Value.....	271
Variable Members	
Variable.CommandValue.....	428
Variable.DoubleValue.....	428
Variable.GetArrayIndexValue.....	429
Variable.GetArrayLowerBound.....	429
Variable.GetArrayUpperBound.....	429
Variable.LongValue.....	429
Variable.PointValue.....	429
Variable.SetArrayIndexValue.....	429

Variable.StringValue	429
Variable.VariableType	429
VariableID	338
VariableType	429
VersionString	195
Visible	195, 234, 296, 365, 384, 396, 398

W

Wait	456
WaitUntilExecuted	372
WaitUntilReady	197
WarningDefault19	207
WarningDefault48	207
WarningDefault60	207
WarnLoadProbe	376
WarnNoSavePrg	207
WarnOKPh9	208
WarnOKRotPh9	208
WarnOverwritingAlignment	208
Width	195, 234, 296, 426
Workplane	189, 358, 456
WristOffset	423

WristTipIJK	424
WriteCommBlock	457
WriteLock	419
WriteRegistryBool	197
WriteRegistryDouble	198
WriteRegistryDWORD	199
WriteRegistryInt	199
WriteRegistryPoint	200
WriteRegistrySettings	201
WriteRegistryString	201

X

X	270, 335, 379
XAxisOffset	341
XYZ	360, 424, 426

Y

Y	270, 335, 380
YAxisOffset	341

Z

Z	270, 336, 380
ZAxisOffset	341