

# Scripting

The scripting language is very much like Visual Basic – most of the commands, functions and syntax are the same or very similar.

## Basic Commands and Syntax

### Variables

to declare a variable use the **DIM** keyword, for example

```
DIM a_string = "a string"
```

```
DIM a_number = 45.8
```

As can be seen, you do not need to specify which type of variable is declared – unlike visual basic's 'As String' or "As Integer". This is automatically defined.

### Keywords and Syntax

The keywords and syntax are almost identical to visual basic – all common keywords like **if**, **then**, **else**, **function**, **endif**, **while**, **loop**, **select**, **for**, **next** are supported in the same way that visual basic does. The scripting language is not case-sensitive when looking for keywords, functions or variables (or indeed when comparing strings).

To generate check errors there is an **error** command or to just output a debug message there is a **debug** command. If you need to ask the user a yes/no type of question to continue there is a **query** command for this also (see the special keywords and inbuilt functions appendices for parameters and return values for these commands).

Scripting also has two commands that visual basic users might not be familiar with – **include** and **run**. Include can be used to insert an external script into the code of the current script. The external script is executed and any functions or variables defined in it will be accessible in the current script after the **include** command. The **run** command is very similar, but any functions and commands defined in the external script will be "forgotten" after execution and will not be accessible from the current script.

## Item properties and methods

the current item being checked is accessed via the "**Item**" variable. To access data or call methods of the item you need to append this with a '.' followed by the **method** / **property** name. For example:

```
Dim Num = Item.Number
```

This accesses the item number and reads it into the temporary variable "**Num**"

```
Item.Number = "1"
```

This accesses the item number and writes to it, changing it to 1

```
Dim ok = Item.Update( )
```

This calls the method “**Update**” on the item. What the methods do is defined by the internal code that gets called, but they can take variable parameters and return values to indicate, for example, success or failure. All methods must have brackets ( ) that enclose the parameters, if any.

Name	Type	Read	Write	Description
NUMBER	String	Yes	Yes	Item Number Field
CID	Number	Yes	Yes	Custom Id Field
DIMS	Number	Yes	No	Number of Dimensions
DIM[ ]	Object	Yes	No	Access Indexed Dimension
OPTIONS	Number	Yes	No	Number of Options
OPTION[ ]	Object	Yes	No	Access Indexed Option
CONNECTORS	Number	Yes	No	Number of Connectors
CONNECTOR[ ]	Object	Yes	No	Access Indexed Connector
SEAMS	Number	Yes	No	Number of Seams
SEAM[ ]	Object	Yes	No	Access Indexed Seam
LIBRARY	String	Yes	Yes	Fitting Library
SPECIFICATION	String	Yes	Yes	Specification Name
SERVICE	String	Yes	Yes	Service Name
MATERIAL	String	Yes	Yes	Material Name
GAUGE	Number	Yes	Yes	Gauge Thickness, if you write to the Gauge, the Lock is set.
ORDER	String	Yes	Yes	Order Number Field
PALLET	String	Yes	Yes	Pallet Field
DATABASEID	String	Yes	Yes	Database Id
CUSTOMDATA[ ]	Various	Yes	Yes	Custom Data (by String or Index)
ALIAS	String	Yes	Yes	Alias Field
NOTES	String	Yes	Yes	Notes Field
STATUS	String	Yes	Yes	Current Status Name
DESCRIPTION	String	Yes	Yes	Item Description, Product Name
FILENAME	String	Yes	Yes	File Name (?????.itm)
PATH	String	Yes	Yes	Location on disk of Item (including trailing '/')
GAUGELOCK	True/False	Yes	Yes	Gauge Locked, True if Locked or to Lock

## Item Sub-Objects

### Item Connector Object Properties

Item Connectors are an array, which is accessed from the item via a 1-based index. The property is then accessed from this using a '.' followed by property name. For example:

```
Dim C1_Name = Item.Connector[1].Value
```

Name	Type	Read	Write	Description
Value	String	Yes	Yes	Connector Name
Type	String	Yes	No	Connector Library
Locked	True/False	Yes	Yes	Connector Lock Flag

### Item Seam Object Properties

Item Seams are an array that is accessed from the item via a 1-based index.

The property is then accessed from this using a '.' followed by property name. For example:

Dim S1\_Name = Item.Seam[1].Value

Name	Type	Read	Write	Description
Value	String	Yes	Yes	Seam Name
Locked	True/False	Yes	Yes	Seam Lock Flag

### Item Pattern Dimension Object Properties

Item Pattern Dimensions are an array that is either accessed from the item via a 1-based index or by description. The property is then accessed from this using a '.' followed by property name. For example:

Dim dim1 = Item.Dim[1].Value  
Dim lengthdim = Item.Dim["length"].Value

Name	Type	Read	Write	Description
Value	String or Number	Yes	Yes	Dim value (e.g. 600,90,"Auto")
NumValue	Number	Yes	No	Actual Value Used (even if auto)
Name	String	Yes	No	Dim Description, e.g. "length"
Locked	True/False	Yes	Yes	Dim Locked Flag
Annotation	String	Yes	No	Annotation as on takeoff, e.g. A
Status	String	Yes	No	Input, Display, NotUsed etc

### Item Pattern Option Object Properties

Item Pattern Options are an array that is either accessed from the item via a 1-based index or by description. The property is then accessed from this using a '.' followed by property name. For example:

Dim option1 = Item.Option[1].Value  
Dim 2parts\_option = Item.Option["2 Parts"].Value

Name	Type	Read	Write	Description
Value	String or Number	Yes	Yes	Value (e.g. Yes, 12, True, "Auto")
Name	String	Yes	No	Description, e.g. "2 Parts"
Locked	True/False	Yes	Yes	Locked Flag
Status	String	Yes	No	Input, Hidden

## Inbuilt Functions and Variables

There are many in-built functions and variables defined to help with basic and simple tasks in scripts. These are predefined in their actions and values and **cannot** or **should not** be changed by the user.

### Inbuilt Variables

- TRUE** Boolean number meaning a positive result
- FALSE** Boolean number meaning a negative result; same as zero
- NULL** Number meaning no value; same as zero
- VOID** Special variable used when wanting to use a default parameter
- PI** Maths number, defined as 3.1415926535897932384626433832795

*The following are used by the File handling class:*

<b>FORINPUT</b>	Open file for input (reading)
<b>FOROUTPUT</b>	Open file for output (writing)
<b>ISTEXT</b>	The file being opened is a text file
<b>FILE_START</b>	Used by the File.Position parameter, meaning start of file
<b>FILE_END</b>	Used by the File.Position parameter, meaning end of file
<b>MAPPATH_HOME</b>	The current working directory (string)

### **Inbuilt maths functions**

number **SQRT** (number: value)

returns the square root of the value passed, e.g. SQRT(4) returns 2

number **SQR** (number : value)

returns the square of the value passed, e.g. SQR(2) returns 4

number **COS** (number: angle in degrees)

returns the cosine of the angle passed, e.g. COS(90) returns 0

number **SIN** (number: angle in degrees)

returns the sine of the angle passed, e.g. SIN(90) returns 1

number **TAN** (number: angle in degrees)

returns the tangent of the angle passed, e.g. TAN(45) returns 1

number **ACOS** (number: between -1 and +1)

returns the arc-cosine of the value passed in degrees, e.g. ACOS(0) returns 90

number **ASIN** (number: between -1 and +1)

returns the arc-sine of the value passed in degrees, e.g. ASIN(1) returns 90

number **ATAN** (number)

returns the arc-tangent of the value passed in degrees, e.g. ATAN(1) returns 45

number **EXP** (number)

returns the base 10 log exponent of the value passed, e.g. EXP(2) returns 100

number **LOG** (number)

returns the base 10 log of the value passed, e.g. LOG(100) returns 2

number **SIGN** (number)

returns +1 if the number passed is +ve and -1 if the number is -ve, e.g. Sign(-2) returns -1

number **ABS** (number)

return the +ve value of the number passed, e.g. Abs(-2) returns 2

number **POW** (number x, number y)

calculates and returns x raised to the power of y

## Inbuilt String Functions

string **ASCII** (number: character code)

returns the character for the code passed, e.g. Ascii(10) returns a carriage return character.

String **CHR** (string a, number: index from 1)

returns the character in the passed string at the index position, e.g. Chr("fred", 3) returns "e".

number **LEN** (string)

returns the number of characters that make up the passed string, e.g. Len("fred") returns 4.

boolean **WILDCARD** (string a, string b)

performs a wildcard test of string a against string b. and returns True if there is a match, e.g. wildcard("Fred was here", "fred\*") returns true.

number **INSTR** (number: start index from 1, string a, string b)

returns the index from 1 that string b appears in string a, starting from the specified index, e.g. Instr(3, "the end was far from the beginning", "the") returns 22. Returns zero if there was no match found.

string **SUBSTRING** (string, number: start, number: end)

returns the sub-string of the string passed, from the "start" character to the "end" character inclusive (1-indexed), e.g. Substring("superlative", 3, 5) returns "per". If end is  $\leq 0$  then it is the offset from the end of the string, e.g., end of  $-1$  is the 2<sup>nd</sup> to last character, 0 is the last character.

string **MID** (string, number start, number length)

returns the sub-string of the string passed, from the start character (1-indexed) and of the specified length, e.g. mid("superlative", 6, 4) returns "lati".

string **LEFT** ( string, number: length)

returns the left portion of a string, of the specified length, e.g. Left("hello", 3) returns "hel".

string **RIGHT** (string, number: length)

returns the right portion of a string, of the specified length, e.g. right("hello", 3) returns "llo".

string **UPPER** (string)

returns an upper-case version of the string passed, e.g. Upper("HelLo123") returns "HELLO123"

string **LOWER** (string)

returns a lower-case version of the string passed, e.g. Upper("HelLo123") returns "hello123"

string **LTRIM** (string)

returns a copy of the string passed, with any spaces at the beginning removed, e.g. Ltrim(" hello ") returns "hello ".

string **RTRIM** (string)

returns a copy of the string passed, with any spaces at the end removed, e.g. Rtrim(" hello ") returns " hello".

string **TRIM** (string)

returns a copy of the string passed, with any spaces at the beginning and end removed, e.g. Trim(" hello ") returns "hello".

### Inbuilt General Functions

boolean **QUERY** (string: prompt)

displays a message box with the prompt that is passed. The user can choose yes or no. If they select yes then True is returned else False is returned.

## Running the Scripts in CadDuct Solids

There are 3 ways a script can be loaded and run in CadDuct.

- Enter **EXECUTESCRIPT** at the command line. You will be offered a "Select Script File (\*.cod)" dialogue, select the script file to run. When you click "Open", you will be prompted to "Select Objects" in AutoCAD. Use standard selection methods to select the objects that you wish the script to work on. It is not necessary to filter out objects in the selection, as this can be done in the script file (e.g. select item.CID).
- Create a LISP routine or Macro on a button that issues the command (executescript "c:/temp/ scripttorun.cod") This will load the script file passed in as the first parameter and you will then be prompted to "Select Objects"
- Create a LISP routine that loads the script file AND passes into the command a Selection Set of Objects to work on.  
i.e. (Executescript "c:/temp/scripttorun.cod" sset)

To do this you must first create the selection set, using standard lisp, for example (setq sset (ssget "X")) will select all objects in the current drawings database and assign them to variable sset.

*So the lisp routine would look something like this:*

```
(defun c:RunScript()  
  (setq sset (ssget "X"))  
  (if sset  
    (Executescript "c:\\temp\\myscript.cod")  
  )  
)
```

The user would type in Runscript at the command line (or pick from a menu or toolbar) and it would automatically run the script if there are any objects in the database.