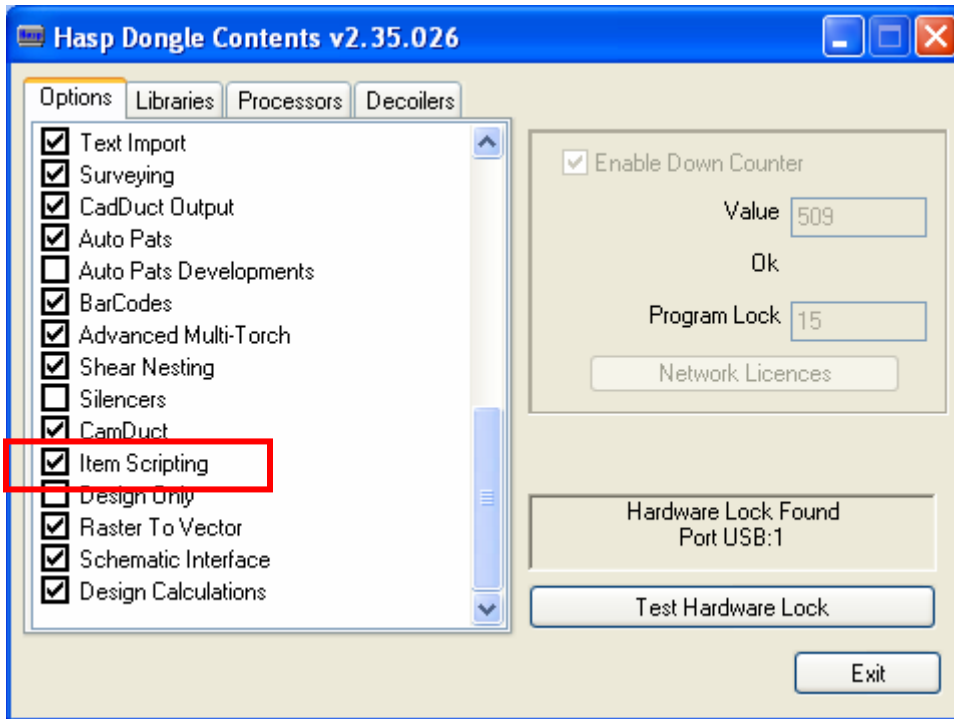


Using Scripting to Modify Item Job/ Drawing Item Files and Master Libraries

Dongle Key Requirements.....	2
Introduction to the Role Scripting Plays in PM Products.....	3
Why Scripting.....	3
Opening and Running Scripts.....	4
Add the Scripting Command to a Button:	6
What can be Scripted.....	7
The Nature of the Scripts and the Scripting Language	9
Sample Script – Writing to Items Selected or in a Job	10
Sample Script – Writing Outputs to a Text File from the Master Item Folder Information	11
Sample Script – Using Flow Control “Select Case End Select”.....	13
Sample Script – Using Flow Control “If Then Else Else If End If”.....	13
Sample Script – Using Flow Control “While Loop – Do While Loop”.....	14
Sample Script – Indexed Dimension Value – Check the Length of Flex	14
Sample Script – Indexed Dimension and Options Values Using Description of the Indexes	15
Sample Script – Changing a Value Based on an Indexed Dimensions Value	15
Basic Commands and Syntax	16
Reading Data from Object	16
Writing Data to the Object.....	16
Reading and Writing Files	17
Variables	17
Keywords and Syntax	17
Item Properties and Methods.....	17
Debugging.....	18
Typedef Documentation – Variable Declaration Types	18
List of Item Properties	19
List of Methods.....	21
Item Sub-Objects	22
Inbuilt General Functions	24
Inbuilt Functions and Variables.....	25
Enumeration Type Documentation	25
The following are used by the File Handling Class	25
Inbuilt Maths Functions	25
Inbuilt String Functions	26
Member Function Documentation.....	27

Dongle Key Requirements

Note: Scripting is a Dongle Key Program Option that requires being enabled. Check your dongle for Scripting by running ViewHasp.exe, from PM Shared/ Utilities



If you do not have this option enabled (Check in the box), consult your sales person.

Introduction to the Role Scripting Plays in PM Products

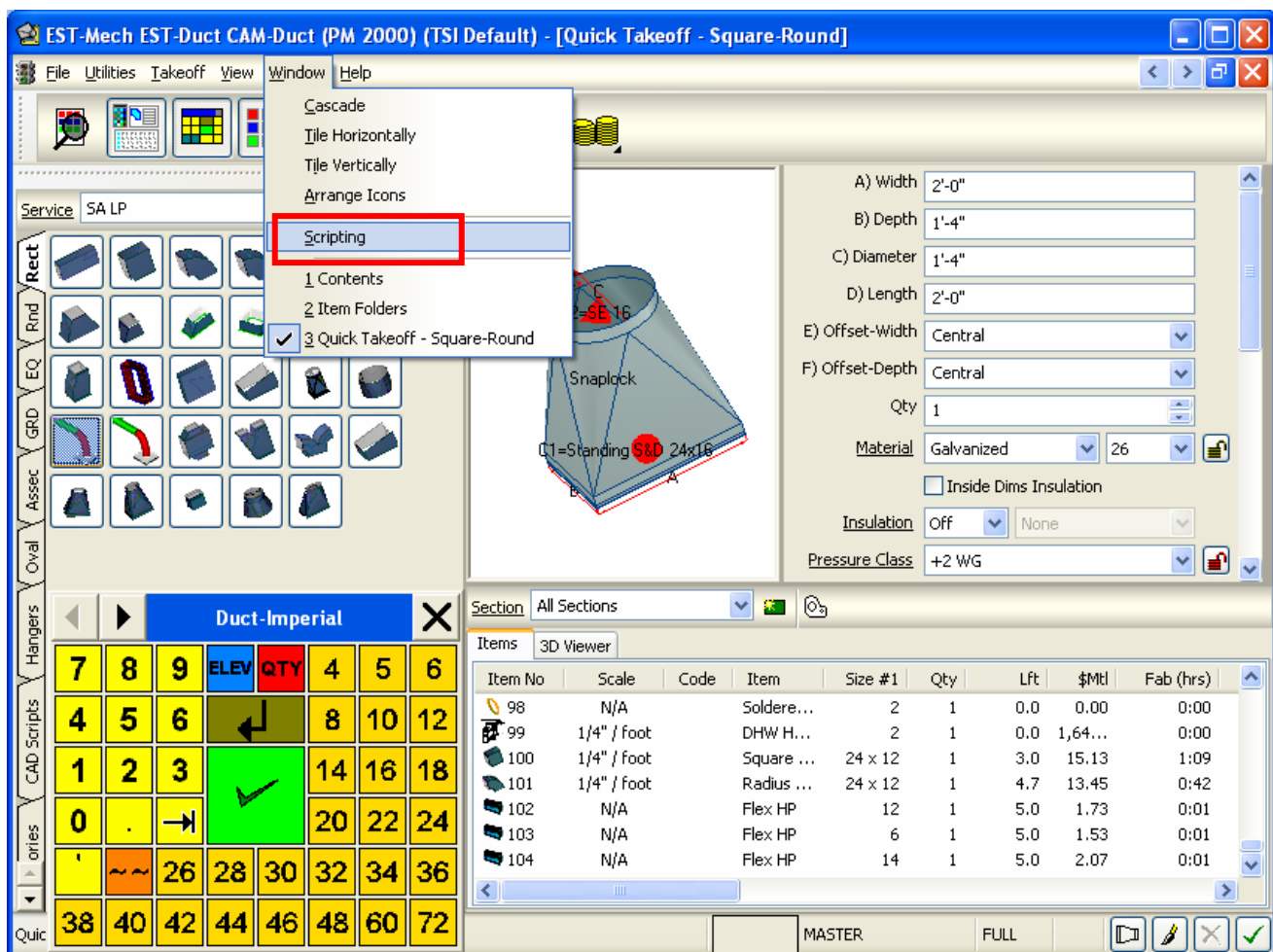
Scripting is a built in API (Application Programming Interface) for accessing and modifying Object data programmatically. Scripting is similar in syntax to (Visual Basic) VB.NET, or the language of almost any help reference. We will be useful to understand some concepts, e.g. String and Math handling.

Scripts allow you to read and write object data on multiple Objects, run tests, interface with the user the results of the script.

Why Scripting

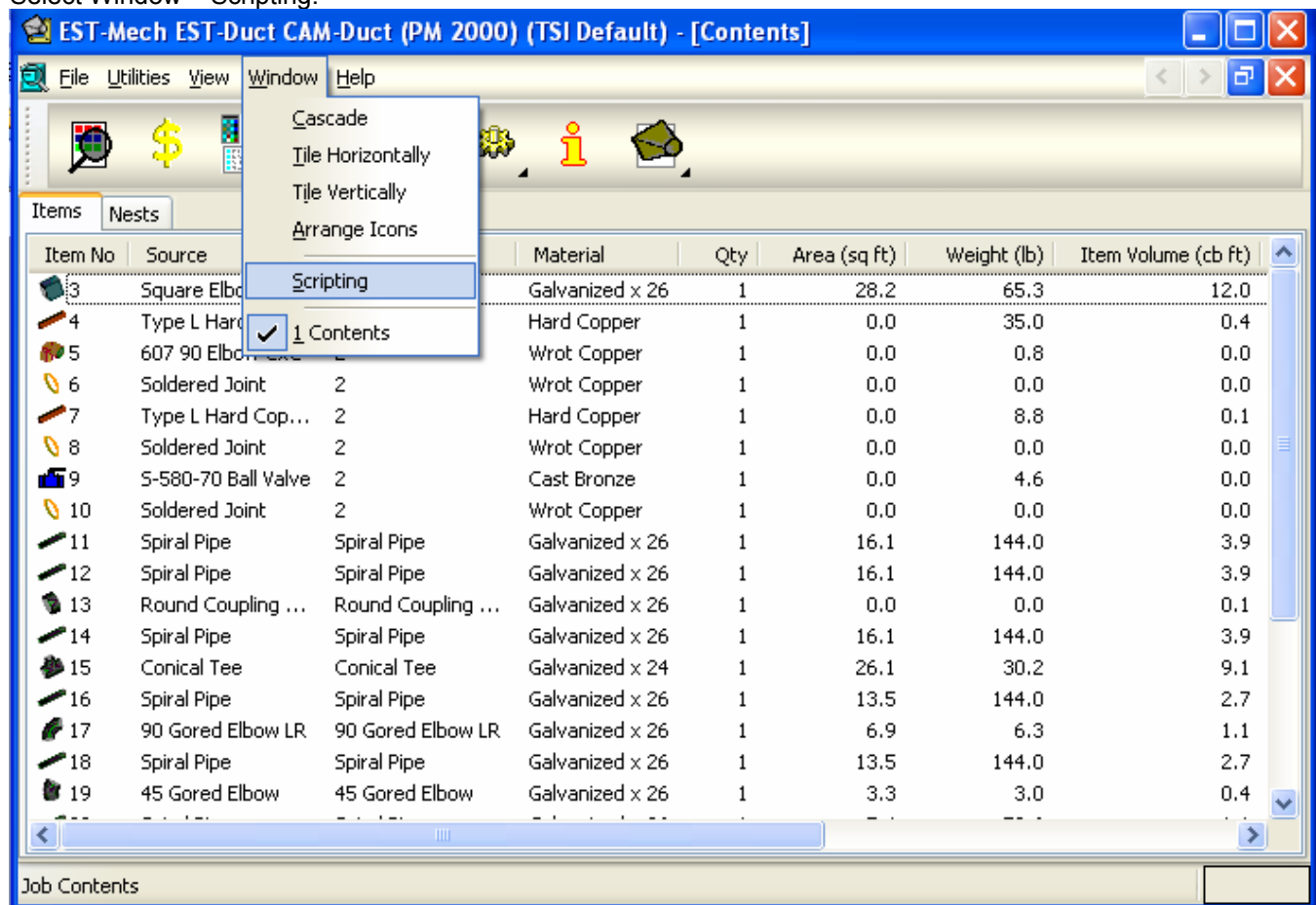
You can create your own rules/tests for data that would normally and be outside of the CAD/ EST or CAMs scope, and no code changes to the Main Application are required (unless a new script object is needed).

Scripting is accessible using the command line command EXECUTESCRIPT in CAD. You must have at least one item in the drawing to select as a component that would be affected by the script even if in reality your script will be applied just to the Master Item Files:

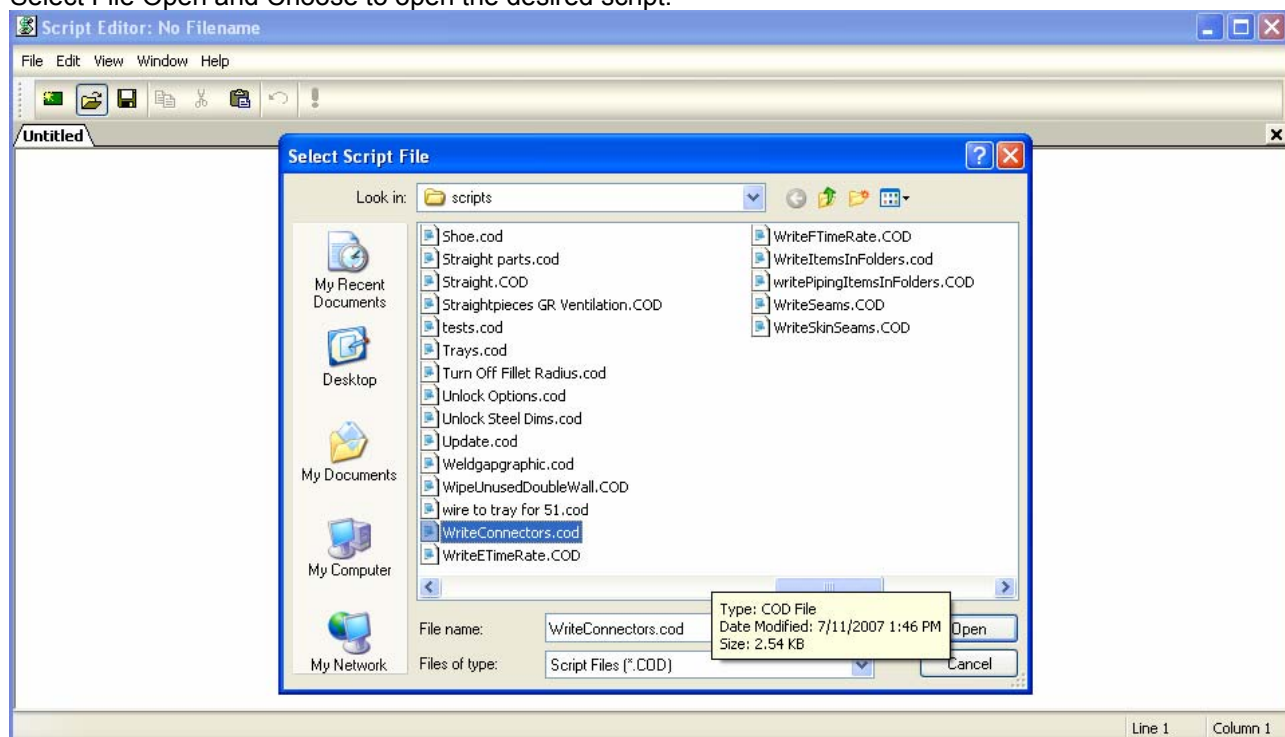


Opening and Running Scripts

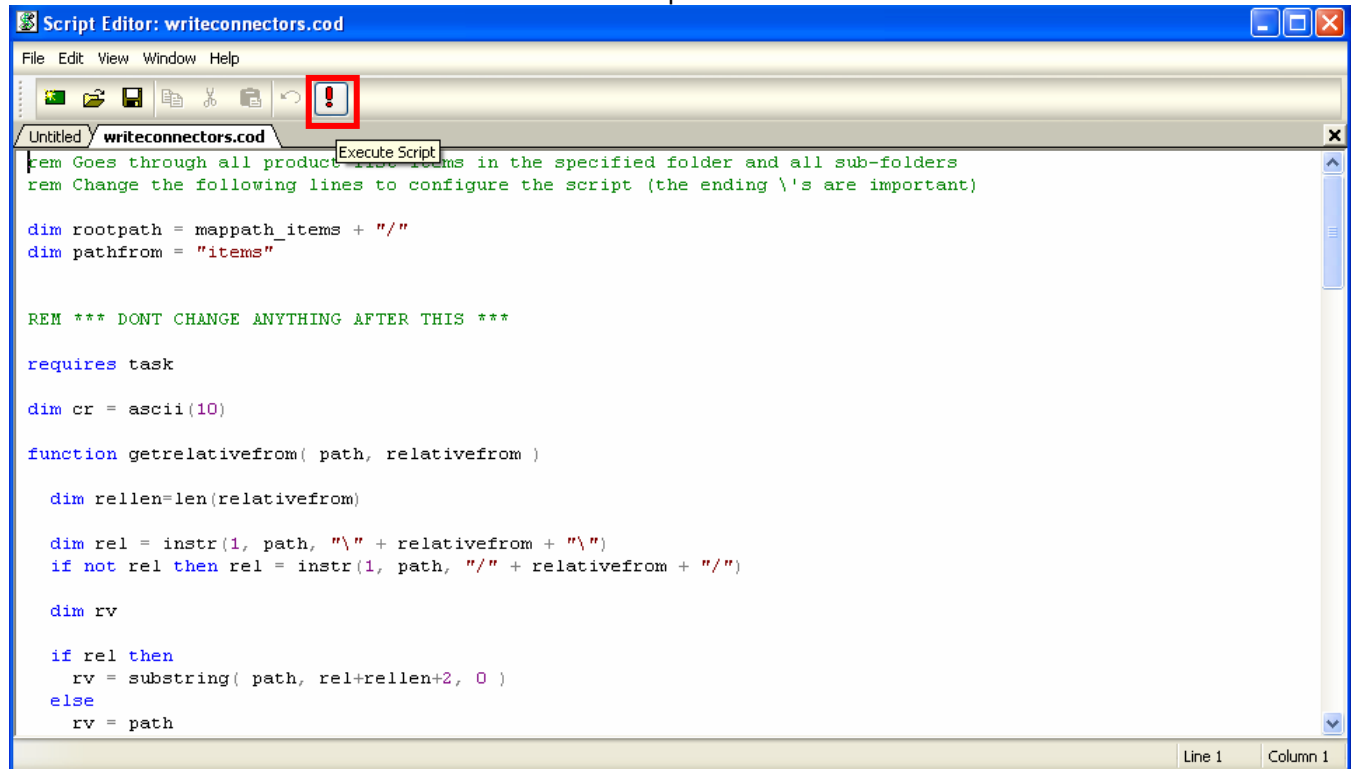
In EST-Duct/ EST-Mech scripts can be run from the pull-down at the top of the Main File Menu, called Window. Select Window – Scripting:



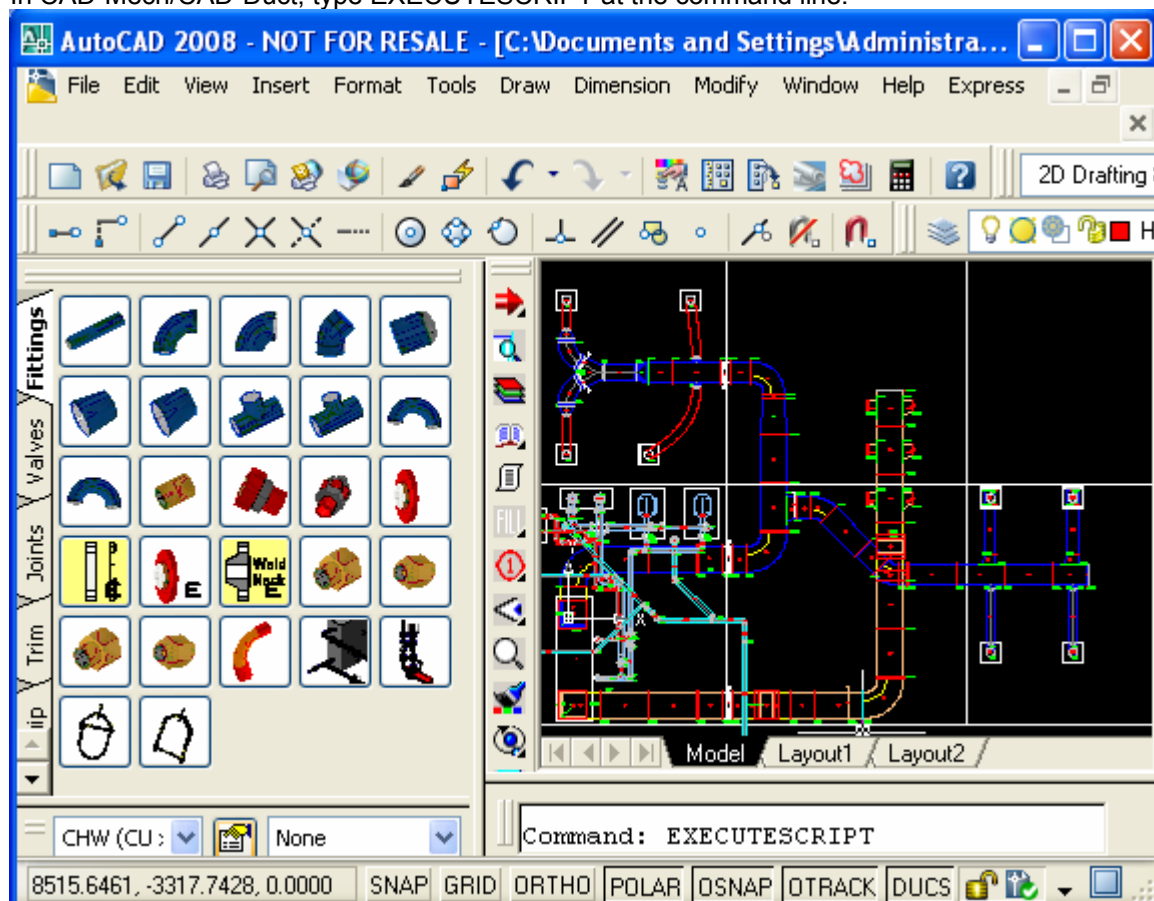
Select File Open and Choose to open the desired script:



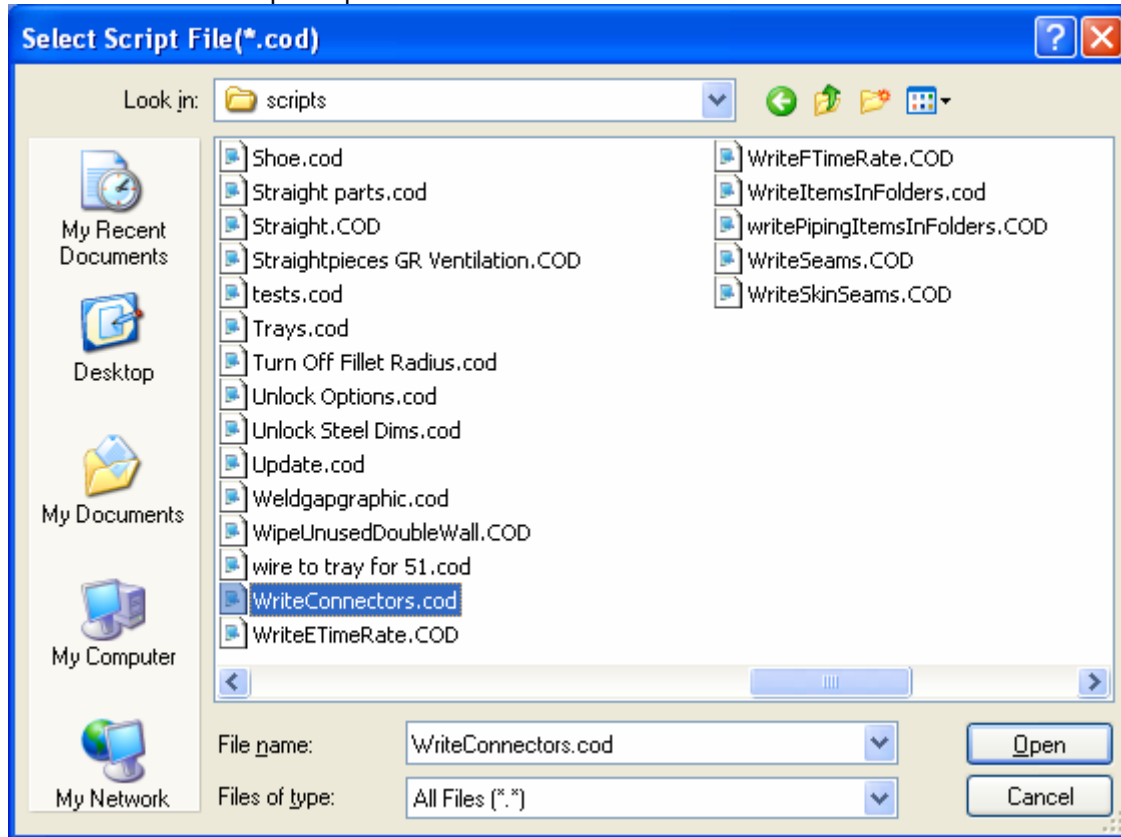
Click on the Red Exclamation Point Icon to Run the Script:



In CAD-Mech/CAD-Duct, type EXECUTESCRIPT at the command line:



Select the desired script to open:



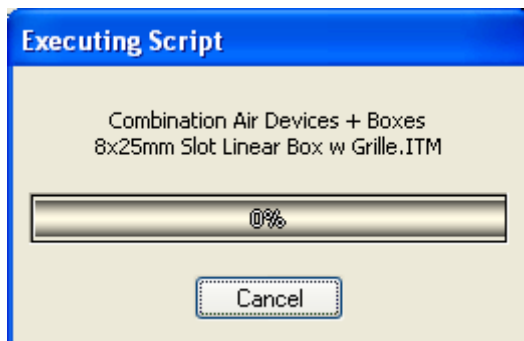
Note: In CAD, even if your script is designed to work on the database itself, or the Raw item folders themselves, you must select objects within the current drawing. Select all objects, as done below, or at least one object to run the script.

Add the Scripting Command to a Button

(EXECUTESCRIPT "SCRIPTFILE.COD") you can specify a Path, if none is specified the SCRIPTS folder is used to locate the file, you will be prompted to "Select Object" The script will run on all the selected objects. You can also pass in a predefined selection set (executescript "test" sset)

Sset would be created using (setq sset (ssget)) in lisp or behind a button.

In either CAD or EST/ CAM, the Script will begin, as begin as below:



The progress bar will show that the script is executing.

What can be scripted

We can access for Read and Write many parts of the Objects Data:

- Dimensions
- Options
- Stiffeners
- Connectors
- Seams
- Dampers
- Material
- Gauge
- Insulation Material, Gauge and Status (Location)
- Pressure Class (Specification)
- Custom Data
- Supports
- Airturns
- LibraryNumber
- Status
- Cut Type
- Service
- Service Type
- Price Lists
- Fab Table
- Install Times
- Extra Fabrication Rate
- Extra Installation Rate
- Filename
- Path
- Weight
- Description
- CID

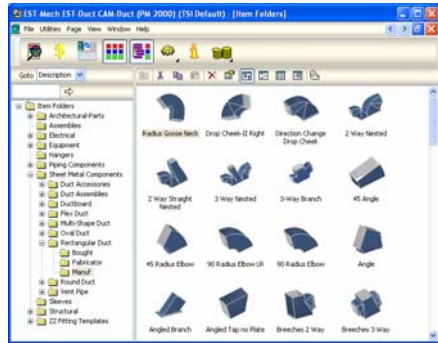
Additionally some methods are available:

- Update()
- Refreshcosts()
- Bitmapfile()
- Load()
- Save()
- AddCustomData()
- Endlocation()

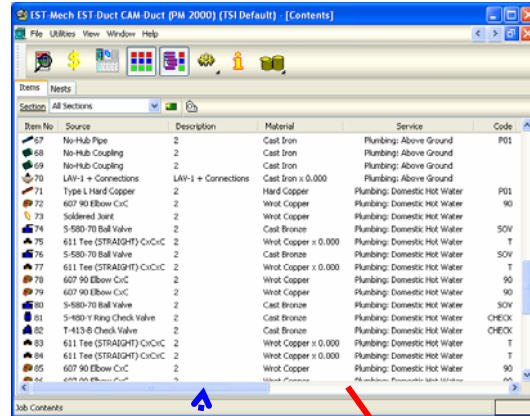
Read and Write for binary and text files also supported.

Using Scripting allows for modification of a large amount of data more easily than the standard program interface. At present, scripting is designed and built around modifying only certain areas of CAD, EST or CAM. The major areas of the program that can be modified with scripting are listed in the top blue circle area of the overall chart below:

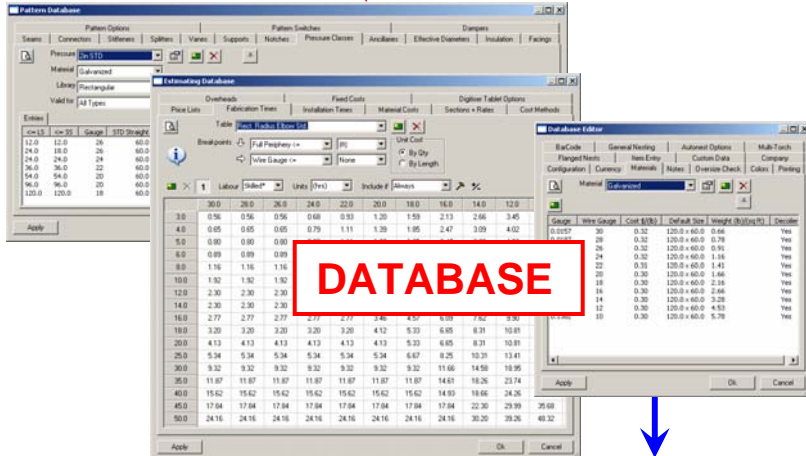
LIBRARIES/ Item Folders



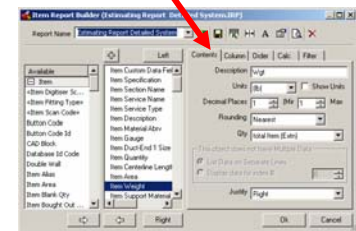
PROJECT/ Job Contents



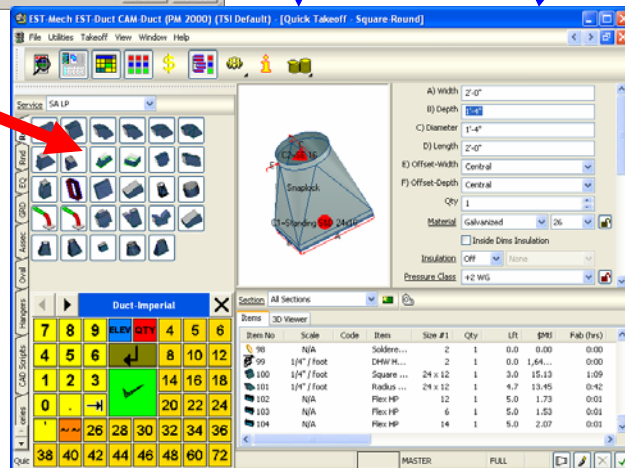
DATABASE



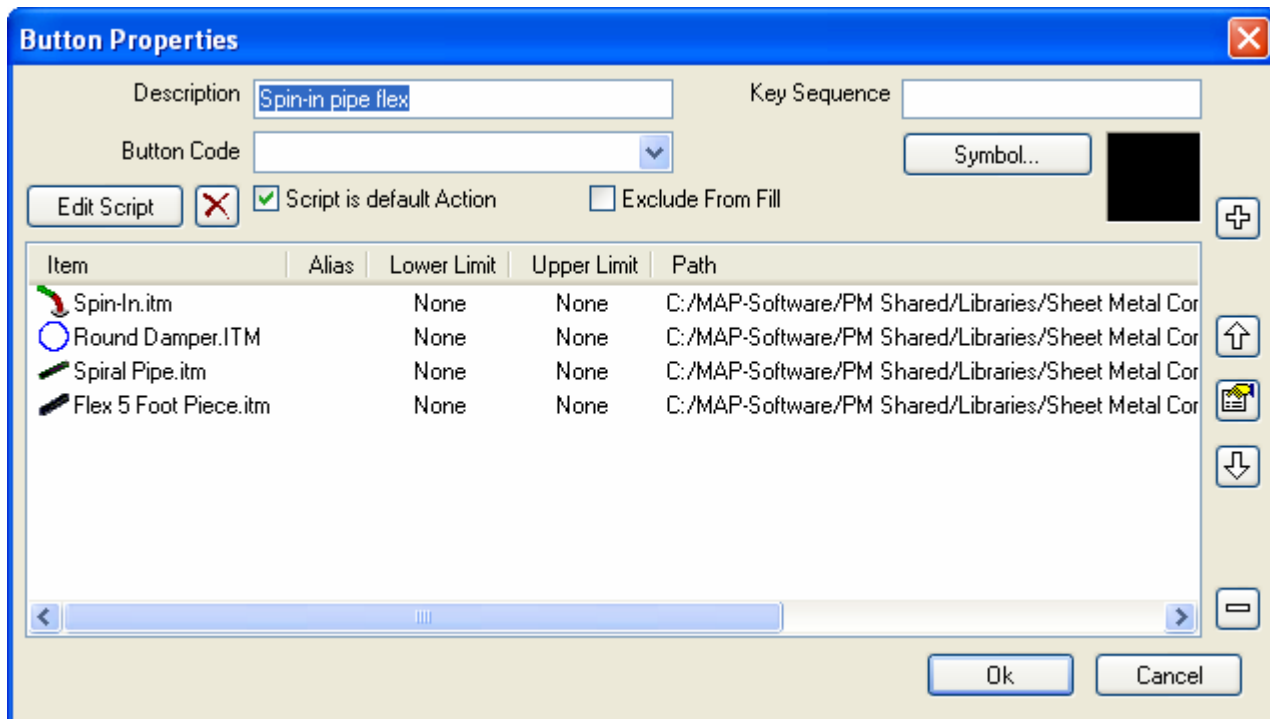
REPORTS



Services Database AND TAKEOFF



Item Files, whether a part of a Job or Drawing, or Master Item Files **can** be modified, viewed or exported in comma delimited text files for the purpose of viewing by using scripting. Scripting in the program is not to be confused with takeoff scripts that appear on buttons in a service that are a component of CAD for Drawing Takeoff Properties:



The script above is defined in the button, using a specific available set of instructions.

The Nature of the Scripts and the Scripting Language

The scripting language is very much like Basic or Visual Basic. Basic Processes are similar, like the use of If and Then statements. The main component, much like old programming language that varies from modern program is that Scripting Files are not **Compiled**. They must be designed to run in order. This means that Dimensioning of Variables must be declared sequentially before the variables are used, not declared public in any spot. Also, any functions or routines must be defined sequentially before they are actually run.

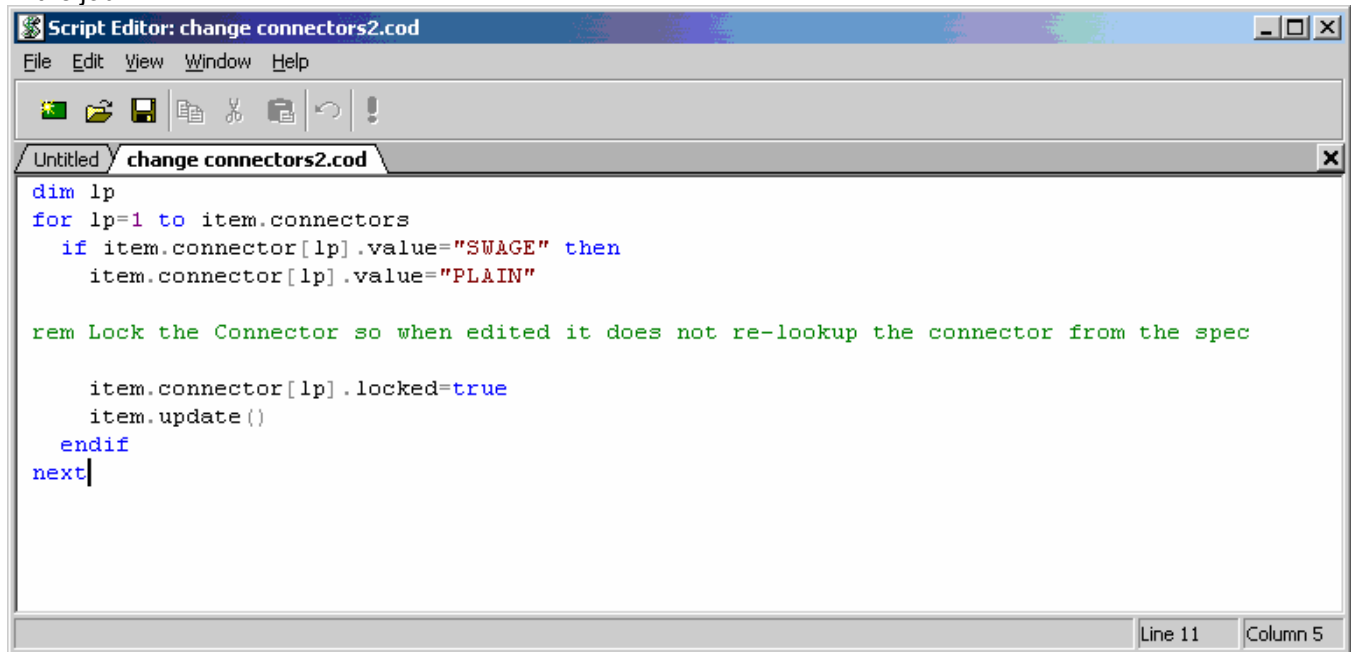
To begin to get familiar with scripting it is suggested that you work with some existing scripts and examine the notes, listed as REM Statements, within them. Observe how they work, and what item variables are modified, or used/ displayed within the script. One should practice attempting on small data sets with making new copies of scripts and choosing new variables.

Some scripts types that are commonly used and have examples pre-built:

- A Script that takes all selected items, or items in the current job and uses if statements to under certain conditions change an item variable (like a connector) from one thing to another
- A Script that takes all items from the Libraries (Master) and uses if statements to under certain conditions change an item variable from one thing to another
- A Script that takes all selected items, or items in the current job and writes some item data to a text file.
- A Script that takes all items from the Libraries (Master) and writes some item data to a text file.
- A Script that Reads a Text or CSV File, and uses its data to modify all selected items, or items in the current job
- A Script that Reads a Text or CSV File, and uses its data to modify all items from the Libraries (Master)

Sample Script – Writing to Items Selected or in a Job:

The following script modifies data on connectors from one type to another based on the selected items or all items in the job:

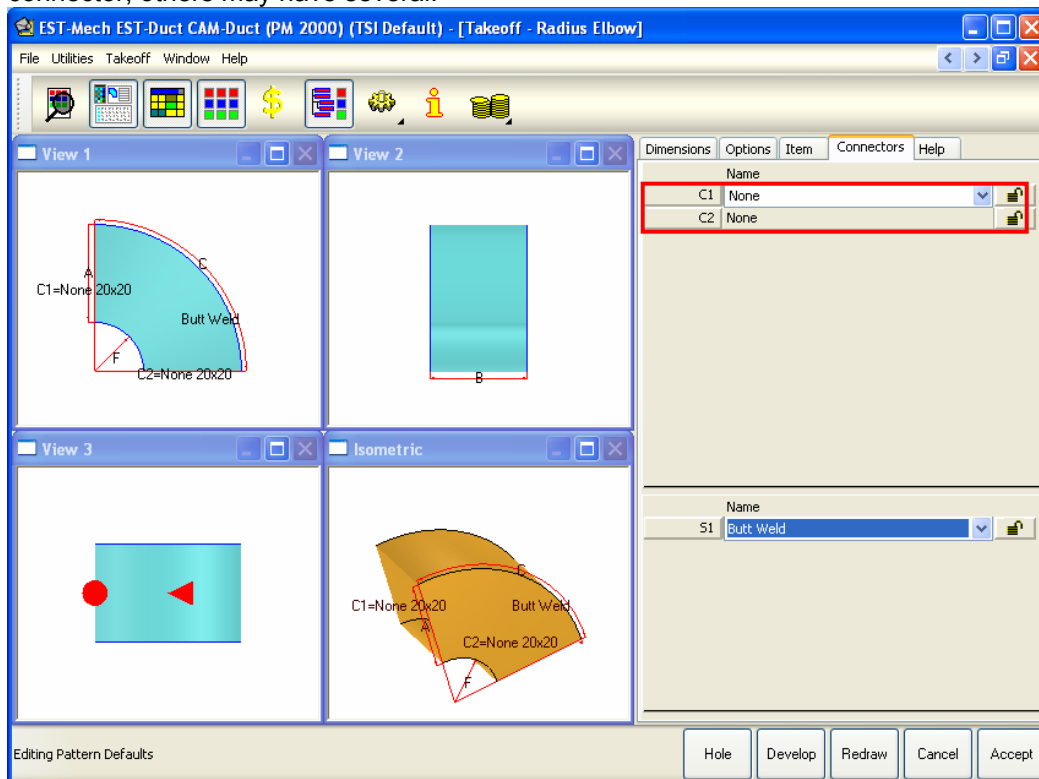


```
dim lp
for lp=1 to item.connectors
  if item.connector[lp].value="SWAGE" then
    item.connector[lp].value="PLAIN"

rem Lock the Connector so when edited it does not re-lookup the connector from the spec

    item.connector[lp].locked=true
    item.update()
  endif
next
```

This script also introduces the uses of multiple variable components, like connectors, where items may have one connector, others may have several.



This item has 2 connectors, a C1 and a C2 which are indexed, and accessed as such above. Other items could have none, 1 or 3 or more connectors.

Sample Script – Writing Outputs to a Text File from the Master Item Folder Information

The next sample script writes out all indexed seam data from all items in the Item Folders.

```
Script Editor: writeseams.cod
File Edit View Window Help

writeseams.cod
rem Goes through all product list items in the specified folder and all su
rem Change the following lines to configure the script (the ending \'s are

dim rootpath = mappath_items + "/"
dim pathfrom = "items"

REM *** DONT CHANGE ANYTHING AFTER THIS ***

requires task

dim cr = ascii(10)

function getrelativefrom( path, relativefrom )

    dim relen=len(relativefrom)

    dim rel = instr(1, path, "\" + relativefrom + "\"")
    if not rel then rel = instr(1, path, "/" + relativefrom + "/")

    dim rv

    if rel then
        rv = substring( path, rel+relen+2, 0 )
    else
        rv = path
    endif

    return rv

endfunction

Line 56 Column 12
```

Dimensioning

1st Function
Defines which
paths in Item
Folders to Use
(Full path to be
written out)

```

function UpdateItem ( path )

    rem scan for all items in the specified folder
    dim filescan as filelocator
    filescan.scan(path, "*.itm", true, false)
    dim folderscan as filelocator
    folderscan.scan(path, ".*", false, true)

    rem debug path + cr + filescan.filecount + " files and " + folderscan.foldercount + " found"

    dim lp, lp2
    dim progressfiles=(folderscan.foldercount = 0)

    if progressfiles then task.beginprogress(filescan.filecount)

    for lp=1 to filescan.filecount
        rem load the item
        dim item as itemstruct
        dim loadfile=filescan.file[lp]
        dim sf=getrelativefrom( path, pathfrom )
        dim relpath = sf
        sf=substring(sf,1,-1)
        sf=GetFilename(sf)
        rem show parent folder name and itm filename
        task.message = sf + cr + GetFilename( loadfile )
        dim ItmName = GetFileName( loadfile )

        if item.load(loadfile) then
            dim Seams = ""
            for lp2=1 to item.seam[lp2].Value + ","
                Seams = Seams + item.seam[lp2].Value + ","
            next lp2

            outfile.writeline( relpath + itmname + "," + Seams)
        endif

        if task.aborted then
            lp=filescan.filecount
        endif

        if task.aborted then
            lp=filescan.filecount
        endif

        if progressfiles then task.progress = task.progress + 1
    next lp

    if progressfiles then
        task.endprogress()
    else
        task.beginprogress( folderscan.foldercount )
    endif

    for lp=1 to folderscan.foldercount
        dim nextpath = folderscan.folder[lp]
        task.message = "Scanning " + cr + nextpath
        UpdateItem( nextpath )
        if task.aborted then
            lp=folderscan.foldercount
        endif
        if not progressfiles then task.progress = task.progress + 1
    next lp

    if not progressfiles then task.endprogress()
end function

```

Dimensioning

1st Function Called here inside this function

Process to call the main Master
Items Files from Libraries – as
opposed to the Job or Selected
Items

Loop to create a variable "Seams" that contains
each seam
[lp2 notes that multiple indexed seams are
looked for] on an item with a comma in between

Output Code Line –Items, Seams

Task Processes from here below

```
task.beginprogress( 1 )
task.progress = 0

object outfile = new file( "C:/ALLITEMSEAMS.TXT", foroutput+istext)

if outfile.isopen then

    UpdateItem( rootpath )
    outfile.close()

endif

task.endprogress()
```

Actual Task Code

Output Code Line –File Save as, and Function “UpdateItem” Called which writes out the Item, Seams

Line 94 Column 1

The Actual Task Code appears sequentially after all dimensioning and Functions are coded, as the scripts are not compiled.

Sample Script – Using Flow Control “Select Case End Select”

The Script will run once for each object selected, so you would need to test some item data to check you are working on the right object (or pre-filter, by selecting only required objects).

The simplest way to check is to use the SELECT command and test the items CID number (but you can use any data).

```
Select ITEM.CID
case 873
    do something here for Flex
case 866,35,36
    do something here with straights
case 3,4
    do something for Square and Radius Bends
End select
```

Sample Script – Using Flow Control “If Then Else Else If End If”

An alternative would be to use an “If then Else” statement, enabling you to test some data and act only if it matches your condition:

```
Dim cid = item.cid
If cid = 873 then
    doflex()
Else if cid = 866 or cid = 35 then
    dostraights()
Else if cid = 3 or cid = 4 then
    dobends()
End if
```

Sample Script – Using Flow Control “While Loop – Do While Loop”

Another type of script uses the Do While Type of Loop, as evidenced by the examples below:

While – End While:

```
Dim count = 5
While count < 10
    debug count
    count = count + 1
End while
```

Do – Loop:

```
Dim count = 10
Do
    debug count
    count = count - 1
Loop until count = 0
```

Do While – Loop:

```
Do while count < 10
    debug count
    count = count + 1
loop
```

While – End While:

```
Dim Found = FALSE
While not myfile.eof and not found
    dim line = myfile.readline()
    if line = "My Data" then
        found = TRUE
    end if
end while
```

Sample Script – Indexed Dimension Value – Check the Length of Flex

This example reads the Length (dim 3) from the dimensions, and the Max Length in the patterns options. If the length is greater than the maximum length, it displays a message telling the user how much too long the flex is.

```
select item.cid
case 873
    dim len = item.Dim[3].value
    dim maxlen = Item.Option["Max Length"].value
    if len > maxlen then
        dim len2 = len - maxlen
        debug "Flex, " + Item.Number + " is " + len2 + " too Long"
    endif
end select
```

Sample Script – Indexed Dimension and Options Values using Description of the Indexes

Example to change the number of parts a shoe is made in.

This example reads the Width and Depth, calculates the periphery, and sets the option for number of parts

```
select item.cid
case 7
  dim width = item.dim ["Top Width"].value
  dim depth = item.dim ["Depth"].value
  dim periphery = width+width+depth+depth
  if (periphery>60) then
    item.Option["2 Parts"].Value="No"
  else
    item.Option["2 Parts"].Value="Yes"
  end if
end select
```

Sample Script – Changing a Value Based on an Indexed Dimensions Value

This example illustrates changing the seam on a part based on the Part's length:

```
select item.Cid
case 28,8
  dim length=item.dim["Length"].Value
  dim seam
  if (length < 300.0) then
    seam="PITTS-S"
  else
    seam="PITTS-L"
  endif

  item.seam[1].value=seam
  item.seam[1].locked=true
end select
```

Basic Commands and Syntax

Reading Data from Object

We access the data from the build in "ITEM" object, followed by the data type we require, you can assign this to an automatically typed variable or use "as is".

Note variable names do not contain spaces, and are not case sensitive.

```
Dim cidnumber = item.alias
Dim alias = item.alias
Dim desc = item.description
Dim numb = item.number
Dim lp = item.connectors
Dim lp2 = item.seams
```

Data can be a single object, or can be an array of data, which requires indexing to access. Arrays are accessed using [], with the index or keyword enclosed.

Array Data

Dim, Option, Connector, Seam, Damper, CustomData

The amount of data can be detected usually by accessing the Data name followed by "s". For instance item.dims would return number of dimensions Item.connectors would return number of connectors.

To access the data, use the .VALUE specifier for the data:

```
item.dim["Width"].value or item.dim[1].value
item.option["override"].value
item.connector[1].value
item.seam[1].value
```

Writing Data to the Object

We write/store data to the object is a very similar way to reading using the ITEM object followed by the .data name. String values are wrapped in Quotation marks.

```
item.alias = "VCD"
item.number = "2"
```

Because strings are "quoted", if you want to write a string value that contains " we need to do the following:

Dim quote = ASCII(34), then concatenate the string using + Item.Specification = "2" + quote + " WG"

To write values to array data, use.VALUE specifier:

```
Item.connector[lp].value = "S&D"
Item.customdata["Spool Name"].value = "Spool2"
```


Reading and Writing Files

First create a File Object using OBJECT NEW and FILE

```
Object ini = new file("TEST.INI", forinput+istext)
```

Here we have specified to open a file called TEST.INI for reading in text mode. We could use new file ("TEST.INI",foroutput+istext) to open For write (overrides existing file), Or forinput + foroutput for reading and writing (appending to):

```
To read from the file    dim line = ini.readline
To Write to the file    ini.writeline("data")
```

Don't forget to close the file when finished:

```
Ini.close()
```

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).

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 which gets called, but they can take variable parameters and return values to indicate, for example, success or failure. All methods must have brackets () which enclose the parameters, if any.

Debugging

To generate check errors there is an **error** command or to just output a debug message (Display the result in a Dialog Box Window), 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).

Any data can be output to the debug window, use widely until the script is complete and working.

```
DEBUG item.cid  
DEBUG item.description  
DEBUG item.connector[1].value
```

You can add data together then debug:

```
Dim cr = ascii(10)  
Dim output = "Item " + item.description + cr + " has " +  
             item.connectors " Connectors and " + cr +  
             item.seams " Seams"
```

Debug output

Also use the REM keyword in your code to add comments.

Typedef Documentation- Variable Declaration Types

Boolean	Scripting Boolean Value (True or False)
NoValue	There is No Return Value. The Value is Null
Numeric	Scripting Numeric Value (Integral or Fractional)
String	Scripting Text Value
Varied	Value can be either a String or a Numeric (not Boolean)

List of Item Properties

Name	Type	Read	Write	Description
AIRTURNS	Numeric	Yes	No	Number of Airturns
ALIAS	String	Yes	Yes	Alias Field
ALTERNATE	Numeric	Yes	Yes	Item Alternate Number
BITMAP	String	Yes	Yes	Item Bitmap - May be full, relative Path, just Filename or blank for default
BOUGHTOUT	Boolean	Yes	Yes	Item Bought Out Flag
BUTTONALIAS	String	Yes	Yes	Button Alias Field from when the item was taken off
CADBLOCK	String	Yes	Yes	AutoCAD Block Associated with this item
CID	Numeric	Yes	Yes	Custom ID Field
COMMENT	String	Yes	Yes	Supports Multi-line text, may contain carriage returns
CONNECTORS	Numeric	Yes	No	Number of Connectors
CUTTYPE	String	Yes	Yes	Cut Type, such as "Decoiled Straight" or "Machine Cut"
DAMPERS	Numeric	Yes	No	Number of Dampers
DATABASEID	String	Yes	Yes	Database ID
DESCRIPTION	String	Yes	Yes	Item Description, Product Name
DIMS	Numeric	Yes	No	Number of Dimensions
DIMSINSIDE	Boolean	Yes	Yes	Inside Dimensions Insulation Flag
DOUBLEWALL	Boolean	Yes	Yes	Double Wall Flag
DRAWING	String	Yes	Yes	Item Drawing Name Field
EXTRAETIME	String	Yes	Yes	Labor Value applied to the Additional Installation Time Extra Install Time (E-Time) Entry
EXTRAETIMERATE	Numeric	Yes	Yes	Labor Rate (Dollars per Hour) From Sections + Rates applied to the Additional Installation Time Extra Install Time (E-Time) Entry
EXTRAETIMEUNITS	TimeUnits	Yes	Yes	Time Units applied to the Additional Installation Time Extra Install Time (E-Time) Entry. Either time_secs, time_mins, or time_hours as entries in the field.
EXTRAFTIME	String	Yes	Yes	Labor Value applied to the Additional Fabrication Time Extra Fab Time (F-Time) Entry
EXTRAFTIMERATE	String	Yes	Yes	Labor Rate (Dollars per Hour) From Sections + Rates applied to the Additional Fabrication Time Extra Fab Time (F-Time) Entry
EXTRAFTIMEUNITS	TimeUnits	Yes	Yes	Time Units applied to the Additional Fabrication Time Extra Fab Time (F-Time) Entry. Either time_secs, time_mins, or time_hours as entries in the field.
FABTABLE	String	Yes	Yes	Item Associated Fabrication Table Name (including Group) eg, "Group: Name"
FABTABLELOCK	Boolean	Yes	Yes	Item Fabrication Cost Locked Flag
FILENAME	String	Yes	Yes	File Name (?????.itm) but without the extension

FIXRELATIVE	Boolean	Yes	Yes	Fix Relative Flag
GAUGE	Numeric	Yes	Yes	Gauge Thickness, if you write to the Gauge the Lock is set.
GAUGELOCK	Boolean	Yes	Yes	Gauge Locked, True if Locked or to Lock
HASPRODUCT	Boolean	Yes	Yes	Exists if the item is Product Listed
INSTALLTABLE	String	Yes	Yes	Item Associated Installation Table Name (including Group) eg, "Group: Name"
INSTALLTABLELOCK	Boolean	Yes	Yes	Item Installation Cost Locked Flag
LIBRARY	String	Yes	No	Fitting Library
MATERIAL	String	Yes	Yes	Item Material Name (including Group) eg, "Group: Name"
NESTPRIORITY	Numeric	Yes	Yes	Higher = Nest First
NOTES	String	Yes	Yes	Notes Field
NUMBER	String	Yes	Yes	Item Number Field
OPTIONS	Numeric	Yes	No	Number of Options
ORDER	String	Yes	Yes	Order Number Field
PALLET	String	Yes	Yes	Pallet Field
PATH	String	Yes	Yes	Location on disk of Item (including trailing '/')
PRICELIST	String	Yes	Yes	Item Associated Price List Name (including Supplier) eg, "Supplier: Name"
PRICETABLELOCK	Boolean	Yes	Yes	Item Material Cost locked Flag
QTY	Numeric	Yes	Yes	Number of Items
SCALE	Numeric	Yes	Yes	The Scale the item was taken off with (if done with digitizer)
SEAMS	Numeric	Yes	No	Number of Seams
SERVICE	String	Yes	Yes	Item Service Name (including Group) eg, "Group: Name"
SERVICETYPE	String	Yes	Yes	Service Type of Item (including service index). Can be Set by "Name" or Numeric Id
SKINCONNECTOR[]	String	Yes	Yes	Access Indexed Skin Connector for Double Wall (If Enabled on the item. Must First Check if Item.Doublewall then)
SKINGAUGE	String	Yes	Yes	Access Skin Gauge for Double Wall (If Enabled on the item. Must First Check if Item.Doublewall then)
SKININSIDE	Boolean	Yes	Yes	Access Flag for Skin is Inside for Double Wall (If Enabled on the item. Must First Check if Item.Doublewall then)
SKINMATERIAL	String	Yes	Yes	Access Skin Material for Double Wall (If Enabled on the item. Must First Check if Item.Doublewall then)
SKINSEAM	String	Yes	Yes	Access Skin Seam for Double Wall (If Enabled on the item. Must First Check if Item.Doublewall then)
SPECIFICATION	String	Yes	Yes	Specification Name
SPECLOCK	Boolean	Yes	Yes	Item Specification is Locked Flag
SPOOL	String	Yes	Yes	Item Spool Name
SPOOLCOLOUR	String	Yes	Yes	Spool Color, AutoCAD Color Index
STATUS	String	Yes	Yes	Current Status Name. Can be Set by Id or "Name"
STIFFENERS	Numeric	Yes	No	Quantity of Stiffeners

SUBITEM[]	Numeric	Yes	Yes	Access Indexed Sub-Assembly Sub Items. Same Functionality as Item, Using Format item.subitem[1].XXX where XXX is any item field in scripts.
SUBITEMS	Numeric	Yes	No	Number of Sub-Items in a Sub-Assembly.
TYPE	String	Yes	No	Combined Libraries, eg. Rectangular/Oval. The Same information as is on Item Properties -> General -> Type, in the same format.
WEIGHT	Numeric	Yes	Yes	Item Weight Based on Pounds. For Items costed by Length, is Pounds/ Foot Values.
WIRE GAUGE	Numeric	Yes	Yes	Wire Gauge of the Item Material. If set then the gauge becomes locked.
ZONE	String	Yes	Yes	Item Zone Field
DIM[]	Sub-Object	Yes	No	Access Indexed Dimension
OPTION[]	Sub-Object	Yes	No	Access Indexed Option
STIFFENER[]	Sub-Object	Yes	No	Access Indexed Stiffener
SUPPORT[]	Sub-Object	Yes	No	Access Indexed Support
CONNECTOR[]	Sub-Object	Yes	No	Access Indexed Connector
SEAM[]	Sub-Object	Yes	No	Access Indexed Seam
INSULATION	Sub-Object	Yes	No	Access Insulation Sub Properties. Returns Separate Objects depending on whether Double Wall is enabled
PRODUCT[]	Sub-Object	Yes	Yes	Access Indexed Product Line Entry Information. May not Exist - always check with "if item.hasproduct then ..."
CUSTOMDATA[]	Sub-Object	Yes	Yes	Custom Data (by String or Index)
DAMPER []	Sub-Object	Yes	No	Access Indexed Damper
AIRTURN[]	Sub-Object	Yes	Yes	Access Indexed Airturn

List of Methods

Name	Return Value	Parameters	Description
UPDATE	True/False	None	Re-develop and Update the Item with any changes to Dims etc. Returns False if the changes invalidate the item.

Item Sub-Objects

Item Pattern Dimension Object Properties (DIM)

Item Pattern Dimensions are an array which is 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	Varied	Yes	Yes	Dim value (e.g. 600,90,"Auto") Access is Numeric Value or Text if optional.
Num Value	Numeric	Yes	No	Actual Value Used (even if Auto). The finished/ calculated value for Auto/ Dependant/ Calculated dims)
Name	String	Yes	No	Dim Description, e.g. "Length"
Locked	Boolean	Yes	Yes	Dim Locked Flag
Annotation	String	Yes	No	Annotation as on Take-Off e.g. "A", "B", "C"
Status	String	Yes	No	("Input", "Display", "Not Used", "Fixed")

Item Pattern Option Object Properties (OPTION)

Item Pattern Options are an array which is 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	Varied	Yes	Yes	Value (e.g. Yes, 12, True, "Auto")
Name	String	Yes	No	Description e.g. "2 Parts"
Locked	Boolean	Yes	Yes	Locked Flag
Status	String	Yes	No	Input, Hidden

Item Stiffener Object Properties (STIFFENER)

Item Stiffeners are an array which is accessed from the item via a 1 -based index. The property is then accessed from using a '.' followed by property name.

Name	Type	Read	Write	Description
Value	String	Yes	Yes	Stiffener Name. Can be set by "Name" or Index
Spacing	Numeric	Yes	No	Spacing Between Stiffener Positions
Quantity	Numeric	Yes	Yes	Number of Stiffeners Required, Excluding End Stiffeners
Locked	Boolean	Yes	Yes	Stiffener Lock Flag

Item Support Object Properties (SUPPORT)

Item Supports are an array which is accessed from the item via a 1 -based index. The property is then accessed from using a '.' followed by property name.

Name	Type	Read	Write	Description
Value	String	Yes	Yes	Support Name. Can be set by "Name" or Index
Spacing	Numeric	Yes	No	Spacing Between Support Positions. If set will alter Quantity.
Quantity	Numeric	Yes	Yes	Number of Supports Required, Excluding End Stiffeners
Locked	Boolean	Yes	Yes	Support Lock Flag

Item Connector Object Properties (CONNECTOR)

Item Connectors are an array which is accessed from the item via a 1 -based index. The property is then accessed from 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. Can be set by "Name" or Index
Type	String	Yes	No	Connector Library
Locked	Boolean	Yes	Yes	Connector Lock Flag

Item Seam Object Properties (SEAM)

Item Seams 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:

Name	Type	Read	Write	Description
Value	String	Yes	Yes	Seam Name. Can be set by "Name" or Index
Locked	Boolean	Yes	Yes	Seam Lock Flag

Item Insulation Object Properties (INSULATION)

Item Insulation Properties are non-indexed, and are accessed from the item, then from using a '.' followed by property name. For example:

Dim Ins1_Material = Item.Insulation.Material

Name	Type	Read	Write	Description
Gauge	Numeric	Yes	Yes	Item Insulation Gauge Thickness
Material	String	Yes	Yes	Item Insulation Material Name (including Group) eg, "Group: Name"
Status	String	Yes	Yes	Item Insulation Field in PM Program, or Item Insulation Location (Status); either ("Inside", "Outside", or "Off")

Item Product Properties (PRODUCT)

Item Product List Data can be access through this entry, which is indexed based on line items in the product list, via a 1 -based index. For example:

Dim Description = item.product.entry[lp2].name

Name	Type	Read	Write	Description
name	String	Yes	Yes	Product List Line Entry Description
hasdatabaseid	Boolean	Yes	Yes	Check for Existence of an id on the Product List Line Entry
hasOrder	Boolean	Yes	Yes	Check for Existence of an Order on the Product List Line Entry
Order	String	Yes	Yes	Order Field when assigned within the Product List
Databaseid	String	Yes	Yes	Id Field (Harrison Code or other) on a Product List Line Entry

Item Custom Data Object Properties (CUSTOMDATA)

Item Custom Data Fields are stored based on an array which is accessed from the item via a 1 -based index. The property is then accessed from using a '.' followed by property name. For example:

Name	Type	Read	Write	Description
Value	Varied	Yes	Yes	Custom Data Field Value in a given instance/ item. Depends on the Setup of the Custom Data Field, String or Numeric
Id	String	Yes	No	Name applied in the database to the Variable of this Custom Data Field

Item Damper Object Properties (DAMPER)

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

Name	Type	Read	Write	Description
Value	String	Yes	Yes	Damper Name. Can be set by "Name" or Index
Locked	Boolean	Yes	Yes	Damper Lock Flag

Item Airturns (Vanes) Object Properties (AIRTURN)

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

Name	Type	Read	Write	Description
Value	String	Yes	Yes	Airturn Name. Can be set by "Name" or index.

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.
---------------------------------------	--

Inbuilt Functions and Variables

There are many inbuilt 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 π number, defined as 3.1415926535897932384626433832795

Enumeration Type Documentation

enum **TimeUnits**

Enumerator:

time_secs : Second Time Units (Numeric Value 1)
time_mins : Minute Time Units (Numeric Value 2)
time_hours : Hours Time Units (Numeric Value 3)

Used by the File Handling Class

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

Inbuilt Maths Functions

number SQRT (number: value)	Returns the square root of the value passed, e.g. SQRT(9) returns 3
number SQR (number: value)	Returns the square of the value passed, e.g. SQR(3) returns 9
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 passed e.g. LOG(100) returns 2
number SIGN (number)	Returns +1 if the number passed is positive and -1 if the number is negative, e.g. SIGN(-2) returns -1
number ABS (number)	Returns the positive 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.
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 <=0 then it is the offset from the end of the string, e.g. end of -1 is the 2nd 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 last 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("HeLo123") returns "HELLO123"
string LOWER (string)	Returns a lower-case version of the string passed, e.g. LOWER("HeLo123") 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"

Member Function Documentation

Boolean ITEM.AddCustomData(**Varied** *which*)

Add Custom Data to the item (for 'User' custom data types)

Parameters:

String ITEM.BitmapFile(**String** *itemfile*)

Get the File Name of the Image Used for an item, given it's original file location

Parameters:

Varied ITEM.EndLocation(**Numeric** *connectorindex*, **String** *xyz*)

Get Connector End Point

Parameters:

[default=none]**Remarks:**

Boolean ITEM.Load(**String** *itemfile*)

Load Item from the Disk

Parameters:

Returns:

No Value ITEM.RefreshCost()

Call after changing any field which will affect the Item's Co

Boolean ITEM.Save(**String** *itemfile*)

Load Item from the Disk

Parameters:

Returns:

Boolean ITEM.Update()

Call after changing any field which will affect the Item's developments, Specification or Model

Returns:

Boolean ITEM.WriteDXF(**String** *filename*, **Boolean** *leads*)

Save Item's Developments as DXF files

Parameters:

Whether or not to Write out Lead Ins to the DXF [default=True]

Remarks:

Each development will be saved with -1 / -2 / -3 postfix to the filename specified.

Returns: