Scripting Overview

Scripting is a built in API for accessing and modifying Object data programmatically. Scripting is similar in syntax to VB.NET (Visual Basic.NET), any help reference material would be useful to understand some concepts in Scripting, like **String** and **Math** handling.

Scripts allow you to read and write object data on multiple objects, run tests, and interface with the user results of the script. You can create you own rules and tests for data that would normally be outside of the programs scope, and no code changes to the main application are required (unless a new script object is needed or requested).

What can we Script?

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

- Dimensions
- Options
- Stiffeners
- Connectors
- Seams
- Dampers
- Material
- Gauge
- Insulation (Material, Gauge and Status)
- Pressure Class
- Custom Data
- Supports
- Airturns
- Splitters
- Library
- Number
- Status
- Extra Over
- Cut Type
- Service
- Service Type
- Price Lists
- Fabrication Table
- Install Times
- Filename
- Path
- Weight
- Description
- CID

Additionally some methods are available.

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

Read and Write access for binary and text files is also supported. **Reading Data from an Object**We access the data using the built in " **ITEM**" object, followed by the **DATATYPE** 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.

Examples:

- *Dim cidnumber = item.CID*
- *Dim alias = item.alias*
- *Dim desc = item.description*
- *Dim numb = item.number*
- *Dim lp = item.connectors*
- *Dim ln2 = item.seams*

- *Dim lp2 = item.seams*

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 a " we need to do the following

Dim quote = ASCII(34)

Then concatenate the string using something like **Item.Specification = "2" + quote + " WG"**

To write values to array data, use the **.VALUE** specifier

- *Item.connector[lp].value = "S&D"*
- *Item.customdata["My Text Data"].value = "Hello World"*

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 type of object (or pre-filter, by selecting only the required objects), we can also work on the selection in a loop which will be covered later in this topic.

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

Flow Control "If Then Else", "Else If", and "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

Flow Control "While Loop - Do While Loop"

Dim count = 5

While count < 10

debug count

count = count + 1

End while

Dim count = 10

Do

debug count

count = count - 1

Loop until count = 0

Do while count < 10

debug count

count = count + 1

*

loop

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

There is a built in function called **DEBUG** which you use to display information in a dialogue.

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

DEBUG item.cid

DEBUG item.description

DEBUG item.connector[1].value

You can add data together and then debug as shown in the examples below:

Dim cr = ascii(10)

Dim output = "Item " + item.description + cr + " has " +

item.connectors " Connectors and " + cr +

item.seams " Seams"

Debug outpu

It is also advisable to use the **REM** keyword in your code to add comments, this will help you if you or a colleague has to ammend the script at a later date. The **REM** comments are ignored by the script but allow you to add notes to assist in seeing what the script is trying to do.

Advanced Scripting

The following section will demonstrate:

- Program flow control using "task"
- Displaying the Progress Bar
- Working with Selections
- Working with Files for Read and Write
- Working with Folders
- Read and Write from/to Items in Folders
- Getting user input

Progress bars are used to display to the user the progress of the script operation. Include " **requires task**" at the top of the file to use Progress Bars.

requires task

task.beginprogress(5000)

dim lp = 1

task.progress = lp

for lp = 1 to 5000

task.progress = task.progress + 1

task.message = "Count is now " + lp

if task.aborted then lp = 5000

next lp

task.endprogress()

Use **Task.Selection** to process multiple items within one call from the script, instead of the script being called for each item. We have to use " **requires task.selection**" at the start of the script.If you want to write information to a file you have to work with a **selection**, you would not want the script to run for each item, else you would keep overwriting the file you are trying to create.

requires task.selection

dim loop = 1

while loop <= task.selection.count

dim jobitem = task.selection[loop]

loop = loop + 1

task.message = "Working with " + jobitem.description

if task.aborted then loop = task.selection.count + 1

endwhile

Combining both Progress Bar and Selection