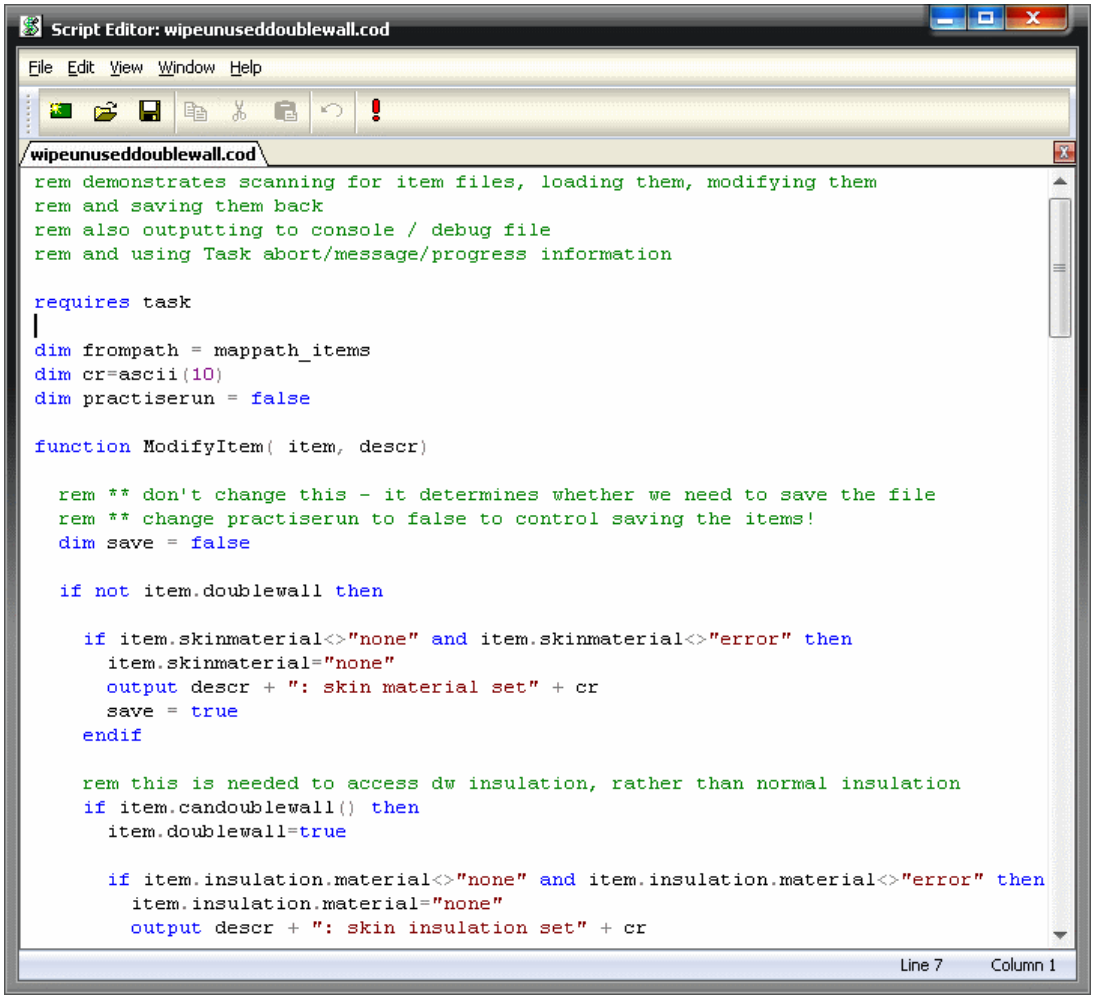
Scripting Definitions

The scripting language is very much like Visual Basic - most of the commands, functions and syntax are the same or very similar. To access **Scripting** go to **Window > Scripting**.



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**, and **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 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.

List of Item Properties

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
SPLITTERS	String True/False	Yes	Yes	Can be set by "Name" or Index, True if Locked

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 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 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 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
Locked	True/False	Yes	Yes	Seam Lock Flag

Item Pattern Dimension Object Properties 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:

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")
Num Value	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 Take-Off e.g. "A"
Status	String	Yes	No	Input, Display, NotUsed, etc.

Item Pattern Option Object Properties 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	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 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 p number, defined as 3.14159265358979323846264338327 95

The following are 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