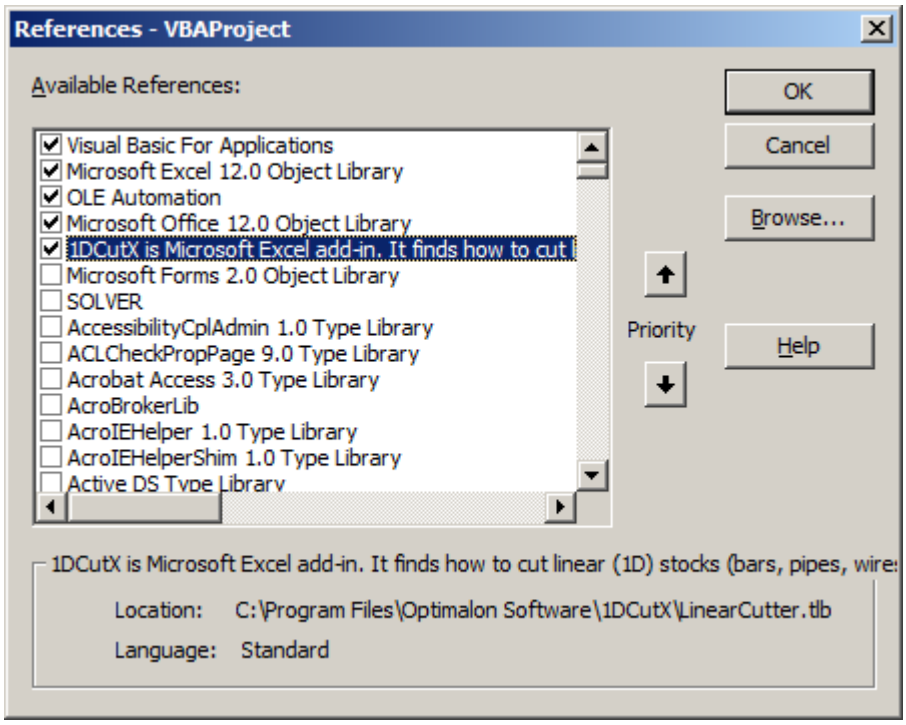
Automation of 1DCutX in Excel (only 32-bit) by VBA.

https://www.optimalon.com

Since version 4.5.0 1DCutX supports automation by Visual Basic for Application (VBA) in Microsoft Excel 32-bit editions. It allows the clients to customize their spreadsheets and run the length cutting optimization without invoking the standard 1DCutX dialog.

If you want to use this possibility then the first think you should add a reference to the ***LinearCutter.tlb*** file in the VBA "**Tools**" -> "**Reference**". You should click on button "Browse" and navigate to the folder when you installed 1DCutX (by default is ***C:\Program Files\Optimalon Software\1DCutX***). There you should select ***LinearCutter.tlb*** file and press Ok.



Pic 1. VBA Preferences with selection of LinearCutter.tlb

This file provides all necessary information to VBA about 1DCutX classes, properties and methods. Once you specified it you can create an instance of linear cutter class, assign required properties and run the calculation.

Creation of the object:

```
Public cut1DObject As New LinearCutter.RuntimeCutter
```

Declaration of the calculator interface:

```
Dim calculator As LinearCutter.IRuntimeCutter
```

Assigning variable to the object:
`Set calculator = cut1DObject`

Now you can setup the calculator properties and run the optimization. All numerical values assigned as string in fixed, scientific or fractional formats like “4.1”, “1.2e-3” or “2/3”

There are following numerical properties available:

`MinOffCut`. Minimal off-cut (waste size). Some cutting machines are unable to make a cut on tiny pieces, because of the technical restriction, like cutting several millimetres from the glass. This property specifies what would be the minimal size of the cut-offs and therefore overcomes such problems.

`Kerf`. Saw kerf (thickness). Cutting produces the gap between parts that shrinks the result part sizes by a saw thickness. This property takes in account the saw kerf during the optimization and generates the layout accordingly.

`TrimBeginning, TrimEnd`. It's not a rear case when the stocks have rough edges that have to be cut before further processing. These properties provide the way to specify the trim sizes for the stocks and take them into account during the cutting optimization

You can tune up the calculation and reports generated by setting the following properties, most of them are boolean values (**True** / **False**):

`bool` **`CompleteMode`**. Complete / incomplete mode. In some cases the supply of the stocks is limited and all parts cannot be cut from it.

`bool` **`MinimizeLayouts`**. This property is very important for woodcutting industries. If all wood stocks have the same layout then they can be placed in a pile and cut simultaneously. That dramatically improves the productivity.

`bool` **`IncludeLayouts`**. If this property is set to **True** then individual layout spreadsheets will be generated. 1DCutX can generate separate spreadsheets for each cutting layouts with names “**1D_*x***” where ***x*** - number of layout. If such information is not required then you can turn off the layout generation by setting this property to **False**.

`bool` **`InsertGraphic`**. If this property is set to **True** then graphical images will be inserted into the report spreadsheets for each layout.

`bool` **`IncludeCutList`**. If this property is set to **True** then each layout spreadsheet will have location of each cut.

`bool` **`IncludePartInfo`**. If this property is set to **True** then each layout spreadsheet will have list of all cut parts, their IDs, sizes and locations.

`int` **`SortColumnSummary`**. Index the column to sort in the summary table. It goes from 0 ("*Stock Length"* column) to 6 ('*Cost*" column).

`bool` **`SortColumnSummaryAscending`**. Sort the column the summary table by ascending (**True**) or descending (**False**) order.

`bool` **`IncludeCutInstruction`**. If this property is set to **True** then the spreadsheet 1D_cutlist will be generated that contains cutting Instructions for each layout/stock. Numbers after column "D" specify the length to cut from the stock. After each cut a stock gets smaller and the numbers indicate where to make the next cut on this smaller stock.

`bool` **`IncludeMatrix`**. If this property is set to **True** then the spreadsheet ***1D_matrix*** will be generated. This spreadsheet includes the matrix of the stocks and number of parts cut from the stocks. For example, if a part **P1** has **2** in a cell for a stock **Stock1** then it means you should cut two parts **P1** from the stock **Stock1**.

`bool` **`MatrixOrderStockPart`**. If this property is set to **True** then the spreadsheet ***1D_matrix*** will contains list of stocks vertically and parts will be listed in horizontal direction. If it's **False** then stocks will be listed horizontally and parts vertically.

`bool` **`IncludeStockOrder`**. If this property is set to **True** then the spreadsheet 1D_stock_order will be generated that contains list of stocks to order for the project.

`bool` **`IncludeWasteList`**. If this property is set to **True** then the spreadsheet 1D_waste_list will be generated that contains list of waste / left overs of stocks for the project.

`bool` **`IncludeUncutPartList`**. If this property is set to **True** then the spreadsheet 1D_uncut_parts will be generated that contains list of parts that were not cut / used for the project.

`bool` **`AscendingOrder`**. If this property is set to **True** then the calculation sorts linear parts in ascending (increasing length) order (from small to large). Otherwise it will sort in descending order.

`bool` **`AllowCombineStock`**. If this property is set to **True** then the calculation will combine several short stocks into one bigger for parts that exceed any of existing stocks.

`int` **`CombineStockLimit`**. Some hardware or logistic limitations can require usage of only few different stocks length. In the worst case scenario only one length is allowed. This option accounts for such requirements and specify how many different stock length can combined into one stock.

`bool` **`AllowCombineStockWaste`**. If you have specified any waste stocks (remnants from previous cuts) you can combine them to actual stocks by selecting this option. Setting to **False** will use strictly actual stocks or waste ones without mixing them.

`bool` **`UseAngle`**. If this property is set to **True** then the calculation will use angle parts information specified.

`bool` **`AllowAngleRotate`**. If this property is set to **True** then the angle cut parts can be turned along X-axis.

`bool` **`AllowAngleFlip`**. If this property is set to **True** then the angle cut parts can be turned along Y-axis (switch start and end).

`bool` **`ExcactAngleOnly`**. If this property is set to **True** then the start and end angles of connected parts must be the same.

After you specified all calculation and report settings you should setup the cell ranges that contain information about your stocks and parts.

All ranges are specified in Excel format as following examples:

- **Sheet1!$A$7** specifies one cell locates on the column "**A**" and row **7** on the worksheet "**Sheet1**".
- **Sheet1!$B$2:$B$5** specifies cells on the column "**B**" from row **2** to row **5** inclusive on the worksheet "**Sheet1**".
- **Sheet2!$C:$C** specifies all cells from the column "**C**" on the worksheet "**Sheet2**".
- **Sheet1!$8:$8** specifies all cells from the row **8** on the worksheet "**Sheet1**".

Some ranges are mandatory and some are optional:

Cells_StockID (*optional*). This range allows specifying the cells that contain text identifiers for each linear stock piece. If this range is omitted then default identifiers "1", "2", etc. are used.

Cells_StockLength (***mandatory***). This range specifies the cells that contain length (size) of the linear stock pieces. These pieces will be cut by smaller pieces, so-called linear parts.

Cells_StockQty (*optional*). If this range left blank then 1DCutX will calculate how many pieces of linear stocks are required to cut all linear parts. If you specified this range it means you already know how many pieces you have and you need to utilize them.

Cells_StockPrice (*optional*). If you specified this range then 1DCutX will calculate the total material cost and report it in the summary table.

Cells_StockDiameter (*optional*). This range of the cells containing the diameter of the linear stocks. If you specified it then you should specified diameters for your parts as well. 1DCutX will match stocks and parts by their diameters and use only such stocks that have the same diameter as parts cut from the stocks.

Cells_StockMaterial (*optional*). This range of the cells containing the material type of the linear stocks. If you specified it then you should specified the material types for your parts as well. 1DCutX will match stocks and parts by their material types and use only such stocks that have the same material type as parts cut from the stocks.

Cells_PartID (*optional*). This range allows specifying the cells that contain text identifiers for each linear part. If this range is omitted then default identifiers "1", "2", etc. are used.

Cells_PartLength (***mandatory***). This range specifies the cells that contain length (size) of the linear parts that will be cut from the linear stocks.

Cells_PartQty (***mandatory***). This range specifies the cells that contain quantity (number) of the linear parts that have to be cut from the linear stocks.

Cells_PartDiameter (*optional*). This range contains the diameter of the linear parts. Only stocks with the same diameter will be considered during the calculation.

Cells_PartMaterial (*optional*). This range contains the material type of the linear parts. Only stocks with the same material type will be considered during the calculation.

Now you can run the calculation by calling the method **Execute**. It returns empty string if the calculation was done successfully. Should any errors happened during the calculation this method returns the text explanation of the error.

Example:

```
' Declaration of the runtime cutter class
Public cut1DObject As New LinearCutter.RuntimeCutter

Sub RunTest1()
' Declaraion of the calculator interface
Dim calculator As LinearCutter.IRuntimeCutter
     Set calculator = cut1DObject
     Dim result As String

     calculator.CompleteMode = True
     calculator.Kerf = "0.2"
     calculator.TrimBeginning = "1/4"
     calculator.IncludeMatrix = True
     calculator.Cells_StockID = "Data!A2:A4"
     calculator.Cells_StockLength = "Data!B2:B4"
     calculator.Cells_StockQty = "Data!C2:C4"
     calculator.Cells_StockPrice = "Data!D2:D4"
     calculator.Cells_PartID = "Data!H2:H6"
     calculator.Cells_PartLength = "Data!I2:I6"
     calculator.Cells_PartQty = "Data!J2:J6"
     result = calculator.Execute
End Sub
```

This example is included into 1DCutX installation.
You can load and run it from "***Start*** -> ***All Programs*** -> ***1DCutX*** -> ***Examples*** -> ***VBA Example.xls***"