# PentaLogix ActiveX (COM) Interface

*Apr 2015*

The following is a description of the Methods and Properties available for scripting / programming of PentaLogix CAM products as ActiveX objects. Currently this applies only to CAMMaster.

Many methods have multiple option configurations and may not be fully detailed. In this situation you can use the Macro-Record command to capture the operation. The captured script will contain the command and the correct syntax. To do this perform the following steps:

1. "Macro/Record"
2. Do any commands that are of interest to you (via menus, dialogs, keystrokes etc.)
3. "Macro/Record" again (this will stop recording)
4. "Macro/Editor": you will see the syntax of the recorded commands

Also, the meaning of each method/property is not always fully described, as this would require duplicating the User's manual. For this reason, sometimes just the corresponding menu command is given and further details are to be found in the manual.
Finally, not all commands have corresponding methods/properties. We have implemented the most commonly used ones, but if you have a need for one that is not implemented, please contact us at support@pentalogix.com and we will attempt to add it.

Any values or arguments that are dimensions are in units determined by the current "display units" setting. These can be "inch", "mil", "cm" or "mm" and can be manipulated via the property called `.Units`.

When an outside script calls the CAM application, it will attach to a running CAM application. If none is running, a new windowless CAM application will be started. To give it a window one can use the "Minimize" method (`.Minimize( False )`).

One can load and run a script at CAMMaster startup. The syntax is as follows:

```
CAMMaster <path of script file>
CAMMaster /run <path of script file>
```

where "`<path of script file>`" is the name of the Sax Basic file of script commands. The first usage will load the script file while the second will load it and also run it after CAMMaster is started.

There are four types of primitive elements in the database: traces (linear and arcs), pads and polygons. In addition there are two composite types that group primitive types: step-and-repeat blocks and text blocks.

## Tips For Writing Scripts

While running scripts is it always a good idea to set `.MaxUndo` to `0`, to disable the undo mechanism as it is only useful when user interaction is expected and it slows down operations and uses memory.

Also one can use `.MessagesMode` to decrease the number of messages and even stop the main view from being refreshed (as this can take a long time). To stop the refresh, set `.MessagesMode` to "`Background`". Make sure to set it back to a higher state if you want to see what is happening, because in "`Background`" mode there is no screen display.

Some methods return a status or a count of elements processed. Sometimes one can determine from that whether the operation has completed successfully or has been aborted by an error. There is however a general way to check for errors and that is the property `.LastErrorNumber`. It will store the number of the last encountered error. The script should check that it is zero before continuing.

## Calling From Microsoft Visual C++ (Visual Studio 2005)

This assumes that you are using managed mode (.NET, with the /CLR compiler switch), otherwise follow the steps described above for Visual Studio 2003.

First you need to generate the CAMMaster DLL. From Visual Studio 2005 "Tools" pick "Visual Studio 2005 Command Prompt". Type:

`tlbimp` <path of CAMMaster.exe>

"`tlbimp`" is the type library importer. It is safer to surround the path name by quotes, because otherwise blanks in the name will cause problems. This will create CAMMaster.dll (from CAMMaster.exe), which can then be used as a Reference in Visual Studio 2005.

In the Solution Explorer, right click on the project and pick "References". In the dialog that comes up click

"Add New Reference", then pick the "Browse" tab and in finally pick "CAMMaster.dll" (which was generated in the previous step).

This is it. You don't need to do anything else, except use the CAMMaster ActiveX methods. They are available under the name "`CAMMaster::ToolClass`".

The example from the previous section becomes:

```
CAMMaster::ToolClass    ^cam = gcnew CAMMaster::ToolClass;
CString     version( cam->Version() );
```

## More Resources / Documentation

You can browse our FTP site for more info:

ftp://ftp.pentalogix.com/Documents/AppNotes/Scripting/

## Setup Notes

When using Windows 7 or Vista, you may need to run CAMMAster at least once "as and administrator" to have it register the type library with the operating system. Right click on CAMMaster.exe and pick "Run As An Administrator". You need to do this even if your account has administrator privileges.

# Methods and Properties Reference

## AddSnowmanFilleting

Format:

```
.AddSnowmanFilleting( ClearDestLayer, InFrameOnly, CurrentDCOnly,
   FirstNewDCode, DCSizeTolerance, RoundToDecimal, UsePadAndTraceSize,
   ReductionFactor, CenterpointFactor )
```

Performs the filleting with the "snowman" method.
New D Codes will be created for the fillets if *FirstNewDCode* is a D Code number, otherwise (if *FirstNewDCode* is 0) existing D Codes will be used. The D Code size tolerance is *DCSizeTolerance* (a value from 0 to 1, where 0 means exact size and 1 means that the size can of off by a 100%). The operation can fail existing D Codes are chosen and if the desired D Codes do not already exist.
*RoundToDecimal* is a value in the range of [2,6] (inclusive) and is used in computing the size of the new (fillet) D Codes. If *UsePadAndTraceSize* is True then the size of the fillets will be computed by combining the sizes of the pads and incoming traces (see the application dialog for more info and for the meaning of *ReductionFactor* and *CenterpointFactor*).

Returns:

The number of added pads or -1 if error.

Example:

```
.AddSnowmanFilleting( False, False, False, 100, 0.1, 3, True, 0.5, 1 )
```

Corresponding Command:

```
Tools/Add Filleting
```

## AddTeardropFilleting

Format:

```
.AddTeardropFilleting( ClearDestLayer, InFrameOnly, CurrentDCOnly,
   FirstNewDCode, EndpointFactor )
```

Performs the filleting with the "teardrop" method.
New D Codes will be created for the fillets if *FirstNewDCode* is a D Code number, otherwise (if *FirstNewDCode* is 0) the incoming trace D Code is used.

Returns:

The number of added traces or -1 if error.

Example:

```
.AddTeardropFilleting( False, False, False, 0, 1.5 )
```

Corresponding Command:

```
Tools/Add Filleting
```

## AddToDrawnPadsLibrary

Add the old shape (or D Code), and the new shape (or D Code) to the conversion library. The old shape can be a stored shape created with the autorecord option under draw to flash. A library of shapes can be built and mapped to specific D Codes that have been pre-defined in a bin. For further examples of draw to flash options refer to the Draw to flash script.

Example:

```
'Clear any existing shapes from the draw to flash library
.AddToDrawnPadsLibrary( "" )
'Add a recorded shape file name and the D Code to convert to.
.AddToDrawnPadsLibrary( "samplefilename", D10)
```

## AddToFileList

Format:

```
.AddToFileList( NameOfFile )
```

Adds a file name ( *NameOfFile* ) to a list of files that will be user for import. The list can be cleared by using the empty string as the file name. This is useful for initializing a list (starting with an empty list).

Example:

```
.AddToFileList( "" )          'Clear any file names from the list
.AddToFileList( "name1.gbx" )
.AddToFileList( "name2.gbx" )
.Import( "Gerber" )
```

This sequence will import two Gerber files ("name1.gbx" and "name2.gbx"), starting at the current layer.

See Also:

```
.Import
```

## AlignToEndpointsOnly

Format:

```
.AlignToEndpointsOnly = Boolean
```

Boolean property that determines the aligment mode during select and align operations. When `True` alignment will be at trace endpoints and polygon vertices only. Otherwise it will be at the closest point on a trace or edge of a polygon.

Corresponding Command:

```
Setup/Preferences/Align To Enpoints Only
```

## AllLayersVis

Format:

```
.AllLayersVis( Action )
```

Changes the visibility of all layers.
*Action* is one of: "`On`", "`Off`", "`Toggle`".

Example:

```
.AllLayersVis "Off"
.AllLayersVis "On"
```

Corresponding Command:

```
View/Visibility/Layer/All Visible
View/Visibility/Layer/Toggle All
```

## AreaAndCircumferenceOfSelection

Format:
```
AreaAndCircumferenceOfSelection()
```

Computes the area and the total circumference of all copper in the selection. The selection is viewed as a merged copper area and the area and circumference for this is returned. The circumference includes both the outer path and the inner (hole) path of copper areas.

Returns:

A string. Empty if error, otherwise a comma separated list of two numbers. The first is the area, the second the circumference. Both are in the current units. For area it is in aquare units. For instance, if the units are "inches" the area will be returned in "square inches".

See Also:
```
.CopperAreaOfSelection
```

## AutoAlignDrillLayer

Format:
```
.AutoAlignDrillLayer( Tolerance, Options )
```

Attempts to align the drill layer (of type DRI or PTH) to a component layer (CPU or CPL) by trying to match all plated holes in the drill layer to pads in the component layer. Each hole needs to be matched for this operation to succeed, so the component layer needs to have at least as many pads as the drill layer. Only visible elements / layers are considered.

Returns:

X and Y values of the displacement or "?" on error

Example:
```
.AutoAlignDrillLayer( 0.0005, "" )
```

Corresponding Command:
```
Edit/Move Layers XY Position/Auto-Align Drill Layer
```

## Autoblob

Format:
```
Autoblob(ClearDestinationLayer, InFrameOnly, Tolerance, LargestShape)
```

Create flashed entities from drawn features on a layer. (Soldermask consolidation) This will flash all draws on a layer, useful for masks and paste layers for reference.
If *LargestShape* is set to 0, then no limit is imposed on the shape size.

Example:
```
.Autoblob( False, False, 0.002, 0 )
```

## AutoSnapToElement

Format:
```
SelectionInside = Long
```

Property that turns Auto-Snap on or off. When on, will cause the cursor to snap to the closest active element before certain operations: element insertion, move etc.

See Also:
`SnapTolerance`

## BlankDCode

Format:
`BlankDCode( DCode, MakeBlank )`

Change a D Code to or from blank. (B type D Code). Blank or unblank.

Example:
`.BlankDCode(10, true)`
Change D Code 10 to a blank
`.BlankDCode(10, false)`
Remove the blank for D Code 10

## BoardRoutIsOK

Format:
`BoardRoutIsOK( LayerNumber, ToleranceArea )`

Checks the closed routing paths in `LayerNumber` to make sure that the compensation does not distort the result by areas which are larger than `ToleranceArea`.

Returns:

A string. "OK" if the check was successful, otherwise the error message.

Corresponding Command:

None

## BreakTraces

Format:
`.BreakTraces( lower_x, lower_y, upper_x, upper_y )`

Break the traces within the specified area.

Example:
`.BreakTraces(1, 1, 2, 2)`

## BuildCustomFromFrame

Format:
`.BuildCustomFromFrame( DCode )`

Map D Code *DCode* to a custom shape (M) that is built from the elements currently visible inside the frame.

Returns:

`True` if operation completes successfully.

Corresponding Command:

`Setup/D Codes/Dimensions/Build Shape`

## CalculateCopperArea

### Format:

```
.CalculateCopperArea( Method, Resolution, BoardThickness, ShowReport )
```

Calculates the copper area according to the settings for the previously chosen layers (set with `.ChosenLayers`).

Possible Values for `Method` are:

- `"Layer by layer"`
- `"Combined"`
- `"Upper/Lower"`
- `"OnlyUpper"`
- `"OnlyLower"`

Note that `"OnlyUpper"` and `"OnlyLower"` are methods that do not exist via the normal user interface (they exist for scripting only).

The computation is done is raster mode at a resolution given by *`Resolution`* (in units of mils or 0.01mm).

If *`ShowReport`* is `True`, then the text of the copper area calculation report will be displayed at the end of the computation. The report is appended to the end of the file `.ReportFileName` and this happens independent of the setting of *`ShowReport`*.

### Returns:

The value of the percentage of copper in the area of the frame (from 0 to 100%). The value of the copper area can then be computed by multiplying the area of the frame by the percentage. The coordinates of the frame can be accessed using `.Frame`.
This return value is only usable when the computation is done on one layer only, because otherwise (for instance in "Combined" mode with many layers selected) the return value is the sum of the individual percentages (not a very useful value).

Below is sample Basic code for computing the absolute area from the percent value returned ("CopperAbsoluteArea"):

```
Function Coord( S As String ) As Double
'Extracts and returns the next coordinate from "S"
  Dim i As Integer
  i = InStr( S, "," )
  If i > 0 Then
    Coord = CDbl( Left( S, i ) )
    S = Right( S, Len( S ) - i )
  Else
    Coord = CDbl( S )
    S = Empty
  End If
End Function

Function Area( S As String ) As Double
'Returns area from bounding box string
  Dim X As Double
  Dim Y As Double
  X = Coord( S )
  Y = Coord( S )
  X = Coord( S ) - X
  Y = Coord( S ) - Y
  Area = X * Y
End Function

Function CopperAbsoluteArea( Layers As String, Method As String,
  Resolution As Double, BoardThickness As Double )
  CAM.ChosenLayers = Layers
  Dim percent As Double
  percent = CAM.CalculateCopperArea( Method, Resolution,
  BoardThickness, False )
  CopperAbsoluteArea = percent * Area( CAM.Frame ) / 100.0
End Function
```

## Example:

```
.ChosenLayers = "All"
.CalculateCopperArea( "OnlyUpper", 2.0, .062, False)
```

Include the hole wall. Calculate using the CPU/CPL/DRI layers, 2 mil resolution. Board thickness is 0.062 inches. Don't show report.

```
.ChosenLayers = "1-6"
.CalculateCopperArea( "Layer by layer", 2.0, 0, True )
```

Surface area only. Calculate for layer 1 through layer 6 at 2 mil resolution. Show report.

## Corresponding Command:

Tools/Calculate Copper Area

## See Also:

GetCopperAreaValues

CopperAreaOfSelection

ChosenLayers

ReportFileName

Frame

## CancelUserOperation

If any operations or tools are active, they will be cancelled. This is equivalent to the operator pressing the Escape key to terminate a pending operation.

## CAMTEKINI

String property to get or set the name of the CAMTEK .ini file which contains parameters describing the CAMTEK machine being used.

### Corresponding Command:

```
AOI/CAMTEK/CAMTEK.INI
```

## CAMTEKParameters

String property to get or set the parameters for CAMTEK AOI. It is a comma separated list of item assignments. The items are listed in the table below:

| | |
|---|---|
| *Attribute* | String |
| *Calib* | String |
| *Design* | String |
| *Etch_Factor* | Double |
| *Inspected_Material* | String |
| *JobName* | String |
| *JobDirectory* | String |
| *Drill* | Boolean |
| *Mirror* | Integer |
| *Nominal_Line* | Double |
| *Nominal_Space* | Double |
| *Negative* | Boolean |
| *Res Mils* | Double |
| *Res Type* | Integer |
| *Rotate* | Integer |
| *Server* | String |
| *Thick* | Integer |
| *WINSOCK* | Boolean |
| *X Stretch* | Double |
| *Y Stretch* | Double |

### Corresponding Command:

```
AOI/CAMTEK/Write Reference
```

### See Also:

```
CAMTEKWriteReference
```

## CAMTEKWriteReference

### Format:

```
.CAMTEKWriteReference( Layers )
```

Writes the CAMTEK AOI reference files from the specified layers, the parameters specified by `.CAMTEKParameters` and the shape borders specified by `.Frame`.

*Layers* is a comma-separated list of layer ranges.

Example:

```
.CAMTEKWriteReference( "2-4, 7" )
```

Corresponding Command:

```
AOI/CAMTEK/Write Reference
```

See Also:

```
CAMTEKParameters
Frame
```

## CenterDataInFrame

Format:

```
.CenterDataInFrame
```

Moves the data currently in the frame to the center of the current frame. Only entities with both endpoints or center point totally contained in the frame will be moved.

## CenterLayersToFrame

Moves the entire visible layer(s) to the center of the current frame.

Example:

```
.CenterLayersToFrame("2-4", True)
```

(Move layers 2 through 4, Center as a group.)

```
.CenterLayersToFrame("2-4", False)
```

(Move layers 2 through 4, Center each layer individually.)

## CenterSelectedToFrame

Moves the selected elements so that their center coincides with the center of the frame. The selection does not need to be contained inside the frame, it can be anywhere.

Corresponding Command:

```
Edit/Edit Selection/Center To Frame
```

See Also:

```
Frame
```

## ChordsToArcs

Format:

```
.ChordsToArcs( Tolerance )
```

Changes selected polygons by replacing sequences of chords (straight edges) with arcs. *Tolerance* is used to determine which straights can be approximated. The new selection will consist of the changed polygons. If none were changed, the selection remains as before.

Returns:

The number of changed polygons or -1 if error.

Example:
```
.ChordsToArcs( 0.001 )
```

## ChosenLayers

Format:
```
.ChosenLayers = "Layers list"
```

Defines layer(s) for the next operation, such as copper pouring, draw to flash, etc. This is equal to the layer selection tab under these tools. If this is not specified for the current tool the previously chosen layer(s) will be used.
*Layers list* is a comma separated list of layer ranges or the special string "*All*".

Example:
```
.ChosenLayers = "4-7"
```
Sets layers 4, 5, 6, and 7 as the layer selected for the next operation.

## ClearSelection

Format:
```
.ClearSelection
```

Clears any selected elements globally. This will clear the selection on all layers whether visible or not.

## ClipSilkscreen

Format:
```
.ClipSilkscreen( SilkscreenLayer, PadmasterLayer, Clearance,
   ShortestSilk, InsideFrameOnly )
```

Clips silkscreen elements in layer *SilkscreenLayer* that are closer than *Clearance* from elements in the *PadmasterLayer* layer.
If *InsideFrameOnly* is `True`, then only elements inside the frame will be considered. Silkscreen shorter than *ShortestSilk* will be discarded (set this value to 0 if you want to keep all).

Corrections to clearance violations are done by inserting scratch polygons into the silkscreen layer.

Returns:

The number of corrections or -1 if an error ocurred.

Example:
```
.ClipSilkscreen( 2, 1, 0.02, 0.005, False )
```

Corresponding Command:

```
Tools/Clip Silkscreen Traces
```

See Also:

```
.ClipSilkscreenTraces
```

## ClipSilkscreenTraces

Format:

```
.ClipSilkscreenTraces( SilkscreenLayer, PadmasterLayer, Clearance,
   ShortestResult, InsideFrameOnly, DeleteOriginalTraces )
```

Clips silkscreen traces in layer *SilkscreenLayer* that are closer than *Clearance* from pads in the *PadmasterLayer* layer.
If *InsideFrameOnly* is True, then only traces and pads inside the frame will be considered. Resulting traces that are shorter than *ShortestResult* will be discarded (set this value to 0 if you want to keep them all).
If *DeleteOriginalTraces* is True then the affected traces will be replaced with the clipped traces, otherwise the original traces will be preserved and selected (highlighted) and the new clipped traces will be added.

Returns:

The number of changed traces or -1 if an error ocurred.

Example:

```
.ClipSilkscreenTraces( 2, 1, 0.02, 0.005, False, True )
```

Corresponding Command:

```
Tools/Clip Silkscreen Traces
```

See Also:

```
.ClipSilkscreen
```

## ClipToPoly

Format:

```
.ClipToPoly( SelectOutside, Margin, PolygonLayerNumber,
   InputLayerNumber, OutputLayerNumber )
```

Clips elements to a polygon. The polygon should be in *PolygonLayerNumber* and contain no holes. Elements on *InputLayerNumber* that cross the polygon will be clipped and the outside or inside portion of the clipped elements will be selected (determined by *SelectOutside).* The output goes into layer *OutputLayerNumber*. The polygon can be inflated or deflated by the given *Margin.*

Returns:

The number of clipped and selected elements or -1 if an error ocurred.

Example:

```
.ClipToPoly( True, 0.01, 1, 5, 6 )
```

Corresponding Command:

```
None
```

## Color

Format:

```
.Color( Index ) = Long
```

Long property which sets or gets the RCB setting for a given color index. CAMMaster has 15 possible color indices (1 through 15). Layer colors are mapped to these indices. This property control the RGB value of the corresponding color for each index. The RGB value is a 24-bit value

expressed as: R + 256 * G + 65536 * B where R, G and B are 8-bit intensity values for Red, Green and Blue. In hex the value is composed as: R + &H100 * G + &H10000 * B..

Note that if a large number of color indices need to be set, it is better to use `.ColorSetup` which does them in one step and avoid repeatead screen refresh.

Example:

```
.Color( 3 ) = &H00FF00
```

Changes the color index 3 to bright green.

See Also:

`.ColorsSetup`

`.LayerColor`

## ColorsSetup

Format:

```
.ColorsSetup( Restore, FileName )
```

Saves the current layer colors setup to a file or restores the current layer setup from a file. See **Appendix B** for a description of the format of these files.

Returns:

`False` if the operation fails, `True` otherwise.

Example:

```
.ColorsSetup( False, "C:/colors.txt" )
```

Saves the layer colors setup to C:/colors.txt.

Corresponding Command:

`Setup/Layer Colors/Save To File`

`Setup/Layer Colors/Restore From File`

See Also:

`.Color`

`.LayerColor`

## CombineLayersByBoardNumber

Operates on all visible layers. If several layers have the same board layer number, they will be combined into one (merging into the first layer with that board layer number).

Corresponding Command:

`Edit/Layers Stackup/Combine By Board Number`

## CompactLayers

Changes the layer stackup so that there are no empty layers.

Corresponding Command:

`Compact` in the Layers Toolbar context menu

## CompareTwoLayers

### Format:

```
.CompareTwoLayers( Layer1, Layer2, InsideFrameOnly )
```

Compares elements in *Layer1* to elements in *Layer2* and selects any different elements (elements that only exist in one layer and not in the other). If *InsideFrameOnly* is `True` then only elements completely inside the Reference Frame (`.Frame`) will be considered.

Composites (blocked text and step and repeat blocks) are ignored, so to have them considered you need to unblock them before this operation.

### Returns:

The number of differences or -1 if error.

### Example:

```
.CompareTwoLayers( 1, 2, False )
```

Compare layer 1 to layer 2 (whole layers) and select any differences.

### Corresponding Command:

```
Tools/Compare Layers
```

### See Also:

```
.CompareTwoLayerEx
.Frame
```

## CompareTwoLayersEx

### Format:

```
.CompareTwoLayersEx( Layer1, Layer2, Flags, MaxErrors, Tolerance )
```

Compares elements in *Layer1* to elements in *Layer2* and detects differences that are larger than *Tolerance*. Operation stops when *MaxErrors* is reached.
*Flags* is a bit-field defined as shown below:

| *Bit #* | *Mask (Hex)* | *Meaning* |
|---|---|---|
| 0 | 1 | Only elements completely inside the frame will be considered |
| 1 | 2 | Copper area method is used (see below for description) |

There are two methods for doing the comparison (selected by mask 2 in the *Flags*):

- Element by element. Elements are compared individually and differences are selected.
- Copper area. The copper areas of the two layers are compared. Erros are detected only where the copper is different.

Composites (blocked text and step and repeat blocks) are ignored, so to have them considered you need to unblock them before this operation.

### Returns:

The number of differences or -1 if error.

### Example:

```
.CompareTwoLayersEx( 3, 7, 0, 300, 0.0005 )
```

### Corresponding Command:

```
Tools/Compare Layers
```

See Also:

.CompareTwoLayers

.Frame

## ConvertArcsToChords

Format:

```
.ConvertArcsToChords( Method, Length, Angle )
```

Break arcs into traces of specified length or based upon angle. Note that this method works on all arcs that are active and only inside the frame.

Arguments:

| | | |
|---|---|---|
| *Method* | String | One of: "Length", "Angle", "Bounded angle"<br>If Length uses chord length only (Length argument)<br>If Angle uses chord angle only (Angle argument)<br>If Bounded angle uses chord angle (Angle), but does not generate chords shorter than Length. |
| *Length* | Double | Length of chord |
| *Angle* | Double | Angle of chord |

Example:

```
.ConvertArcsToChords( "Length", 0.004, 0.0 )
.ConvertArcsToChords( "Angle", 0.0, 10.0 )
```

Corresponding Command:

```
Tools/Convert Arcs/Approximate By Chords
```

## ConvertDrawnPads

Format:

```
.ConvertDrawnPads( GuideLayer, GuideMargin, FromSize, UpToSize,
   FlashedDCodesStartAt, FlashedDCodesResolution, Tolerance, Flags )
```

Converts drawn pads in the selection to flashed pads using a guide layer. The new selection will consist of the newly generated flashed pads.
Drawn pads are defined as blobs for which the largest dimension is between *FromSize* and *UpToSize* and for which all elements are completely covered by the *GuideLayer*. All paint (dark) areas in the guide layer are used as if inflated (or deflated) by *GuideMargin* (this is the margin of inflation (swelling) is twice this amount).
Elements in the guide layer do not need to be selected.
They will be replaced with flashed pads with D Codes starting at *FlashedDCodesStartAt* and whose shapes are rounded to the resolution given by *FlashedDCodesResolution*.

The following standard shapes will be tried for replacing each blob: C, S, R, O, D, E, Q and will be matched using *Tolerance*. If none matches then custom shapes (of type M) will generated if so indicated by *Flags* (see below), otherwise the drawn pad will be left unchanged. Holes are allowed in C and S shapes only.

Arguments:

| | | |
|---|---|---|
| *GuideLayer* | Long | Layer that is used as a guide layer to find the drawn pads |
| *GuideMargin* | Double | Margin applied to the guide layer |
| *FromSize* | Double | Smallest dimension of a converted pad |
| *UpToSize* | Double | Largest dimensioon of a converted pad |
| *FlashedDCodesStartAt* | Long | First D Code number for newly created D Codes |
| *FlashedDCodesResolution* | Double | Finest resolution for generated D Codes (they will be multiples of this number) |
| *Tolerance* | Double | Amount of error allowed when matching a drawn shape to a flashed shape |
| *Flags* | Long | Bit-field to specify options:<br>Bit 0 (Mask 1) = OK to generated custom shapes (of type M)<br>Bit 1 (Mask 2) = OK to generate concave custom shapes (Bit 0 must also be set for this)<br>Concave shapes are shapes that have concavities: for instance a cross would be concave, whereas an octagon is convex. |

Returns:

The number of newly generated flashed pads, or -1 if error.

Corresponding Command:

```
Tools/Convert Drawn Pads/Global
```

See Also:

```
.ConvertStandaloneDrawnPads
```

## ConvertNegativeToPositive

Format:

```
.ConvertNegativeToPositive( Layers )
```

Converts negative layers in list of *Layers* to positive by using paint and scratch. A paint polygon is inserted in front of all the data and the data will scratch from it.

Corresponding Command:

```
Edit/Convert Negative To Positive
```

## ConvertSelectionToPolygons

Format:

```
.ConvertSelectionToPolygons
```

Replaces all elements in the selection with equivalent polygons.

Corresponding Command:

```
Edit/Edit Selection/Convert To Polygons
```

See Also:

```
.ReplaceSelectionWithOutlines
```

## ConvertStandaloneDrawnPads

Format:

```
.ConvertStandaloneDrawnPads( FromSize, UpToSize, FlashedDCodesStartAt,
   FlashedDCodesResolution, Tolerance, Flags )
```

Converts standalone drawn pads in the selection to flashed pads. The new selection will consist of the newly generated flashed pads.
Standalone drawn pads are defined as blobs for which the largest dimension is between *FromSize* and *UpToSize*.

Arguments:

See `.ConvertDrawnPads`

Returns:

The number of newly generated flashed pads, or -1 if error.

Corresponding Command:

`Tools/Convert Drawn Pads/Global`

See Also:

`.ConvertDrawnPads`

## CopperAreaOfSelection

Format:

```
.CopperAreaOfSelection
```

Computes and displays the copper are of the currently selected elements. The computation is exact as opposed to the raster based computation of `.CalculateCopperArea` and hence is resolution independent.

Returns:

The area as a double in the current units. For instance if the units are "inches" the area will be returned in "square inches".

Corresponding Command:

`View/Selection/Area`

See Also:

`.CalculateCopperArea`

`.CopperAreaOfSelection`

## CopySelected

Copy the selected elements after moving by the specified distance in x and y.

Example:

```
.CopySelected( 1, 1 )
```

Copies the selection 1 inch in x and y from the current location.

## CountActive

Format:

```
.CountActive
```

Counts active elements.

### Returns:

Integer which is the number of elements which are "active". Active elements are the elements that are visible and which match any selection criterion which is active (such as the "Only" buttons).

## CreateRaster

### Format:

`.CreateRaster(`*`LLx, LLy, Resolution, width, height, FileMap, ProcessId`*`)`

Generates a memory resident bitmap of one bit-per-pixel of the visible entities. One can control what is included by making visible/invisible.

### Arguments:

| | | |
|---|---|---|
| *`LLx, LLy`* | Double | Lower-left corner of bitmap in user coordinates. |
| *`Resolution`* | Double | Pixels per inch (if English units are active) or pixels per cm (if metric). |
| *`width, height`* | Long | Width and height of bitmap in pixels. The bitmap has the layout of a standard Windows bitmap, so that (0,0) in the bitmap is the upper-left corner of the image. |
| *`FileMap, ProcessId`* | Long | Used to specify the memory area for the bitmap. It must be allocated and freed by the caller. See below for an example of how to use. |

### Returns:

`False` if the raster bitmap could not be created.

### Notes:

The order of bits in the returned bitmap corresponds to the standard Windows bitmap order, which is as follows:

Vertical: The bottom row of the image is the top row of the bitmap. In other words the bitmap is mirrored on Y.

Horizontal: If we are numbering the bits in a bitmap byte from 0 to 7 with 0 being the low order bit, then the image, going left to right is 7, 6, 5, 4, 3, 2, 1, 0.

The only safe way to share memory between two or more processes is via "file mapping". This is done with the parameters *`FileMap`* and *`ProcessId`*. The Windows library functions `CreateFileMapping`, `MapViewOfFile` and `getCurrentProcessId` should be used to set up these variables prior to the call to CreateRaster.

### Example:

The calling process should include code similar to the sample below:

```
Const INVALID_HANDLE_VALUE = -1
Const PAGE_READWRITE = 4
Const FILE_MAP_ALL_ACCESS = &HF001F

Declare Function    CreateFileMapping Lib "kernel32" Alias "CreateFileMappingA"
       ( ByVal hFile As Long, ByVal lpFileMappigAttributes As Long, ByVal
       flProtect As Long, ByVal dwMaximumSizeHigh As Long, ByVal
       dwMaximumSizeLow As Long, ByVal lpName As Long) As Long
```

```
Declare Function    MapViewOfFile Lib "kernel32" (ByVal hFileMappingObject As
       Long, ByVal dwDesiredAccess As Long, ByVal dwFileOffsetHigh As Long,
       ByVal dwFileOffsetLow As Long, ByVal dwNumberOfBytesToMap As Long) As
       Long
Declare Function    GetCurrentProcessId Lib "kernel32" () As Long
Declare Function    CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long
Declare Function    UnmapViewOfFile Lib "kernel32" (lpBaseAddress As Any) As
       Long

Dim hFileMap As Long, pViewFileMap As Long
Dim lSize As Long, width As Long, height As Long
width = 700         'Example
height = 500
lSize = ((width + 7) / 8) * height
hFileMap = CreateFileMapping( INVALID_HANDLE_VALUE, 0,  PAGE_READWRITE, 0,
       lSize, 0 )
lpViewFileMap = MapViewOfFile( hFileMap, FILE_MAP_ALL_ACCESS, 0, 0, lSize )
.CreateRaster( 4.5, 17.1, 1000.0, width, height, hFileMap,
       GetCurrentProcessId() )
… use the bitmap …
UnmapViewOfFile( lpViewFileMap )
CloseHandle( hFileMap )
```

### Corresponding Command:

This method has no menu-based equivalent command, it is available only as a scripting method.

### See Also:

`.RenderBitmap`

`.PrintToBMPFile`

## CrosshairDragged

Boolean property that controls the dragging of the crosshair with the mouse.

### Example:

```
.CrosshairDragged = True
```

### Corresponding Command:

`View/Drag Cursor With Mouse`

## CSVSeparator

String property to get or set the value of the separator for "comma-separated" lists. This applies only to the top level of a list that may contain items that are themselves comma-separated. By setting this property to some other value, the returned string may be easier to parse. Use with caution as not all methods or properties have been changed to use this convention. An example of one method that does use it is `.SelectedRoutingPaths`, a property that can contain comma-separated lists at the secondary level .

### Example:

```
.CSVSeparator = "|"
```

## CursorRelative

Boolean property to set the cursor coordinates display to relative or absolute.

Example:
```
.CursorRelative = True
```
Corresponding Command:

View/Coordinates Origin

## CurrentCustomProperty

Format:
```
.CurrentCustomProperty = String
```
String property for getting or setting the current custom property (appears in the Current Custom Property Toolbar). The *String* is of the form:

1. A "Name=Value" property: only elements having that property will be considered.
2. "Name=?": only elements that have a property of the given name (independent of its value) will be considered
3. "Name=False": for Boolean properties only. Only elements that do not have Boolean property Name will be considered (these include elements that have no properties at all).

Example:
```
.CurrentCustomProperty = "Impedance=?"
```

## CurrentDCode

Set (or get) the current D Code.

Example:
```
.CurrentDCode = 10
```
Set the current D Code to 10
```
x = .CurrentDCode
```
Assign the current D Code to variable x

## CurrentDirectory

Set (or get) the current directory.

Example:
```
.CurrentDirectory = "C:\Pentalogix"
```

## CurrentLayer

A long property for the current layer.

Example:
```
.CurrentLayer = 1
```
Corresponding Command:

Current Layer drop-list in the main toolbar.

## CurrentNet

Set the current active net.

Example:
```
.CurrentNet = "142"
```
Set the current active net to 142

## CursorX and CursorY

Move the cursor to the specified location, or get the current cursor location. The location is defined from the absolute origin.

Example:

```
.CursorX = 1
```

Moves the cursor to 1 inches in the x (if units are inches)

```
whereiscursorx = .CursorX
```

Returns the current cursor location in the x to `whereisthecursorx`.

See Also:

```
.MoveCursor
```

## CustomProperties

Format:

```
.CustomProperties = String
```

String property for getting or setting the custom properties of the selection. *`String`* is a comma separated list of properties of the form:

```
property name = property value
```

An empty string means "no properties", so assigning an empty string to `.CustomProperties` will remove all current custom properties of the elements in the selection.

Example:

```
.CustomProperties = "DoNotScale=True"
```

## CustomProperty

Format:

```
.CustomProperty( Name ) = Value
```

String property for getting or setting a specific custom property in the selection. *`Value`* is a the value of the custom property *`Name`*. Both are strings.

The table below summarizes the meaning of special settings for *`Value`*:

| *Value* | *Get* | *Set* |
|---|---|---|
| Empty string | The property does not exist in the selection | Delete the property in the selection |
| "?" | The property exists, but either has more than one value or is not set for all elements in the selection | No action |

The property will be set for all elements in the selection. If no elements are selected than this has no effect.

Example:

```
.CustomProperty( "Impedance" ) = "50.2"
```

## DCodeOfShape

Format:

```
.DCodeOfShape( ShapeDefinition, StartingAt )
```

Returns the number of the first D Code of a certain shape, starting at D Code *StartingAt*. *ShapeDefinition* is a string that defines the shape and includes the shape type and other

parameters. Missing parameters are substituted with defaults. If no D Code of the particular shape exists, a new one will be created.

Example:

```
x = .DCodeOfShape( "C, Diameter=0.04, Hole=0", 4000)
x = .DCodeOfShape( "H", 4000)
```

## DCodeShape

String property to set or get the shape of a D Code. The shape is described by a comma separated list of items.

Example:

```
x = .DCodeShape(100)
```

Assign the shape string for D Code 100 to variable x.

```
.DCodeShape(100) = "C, Diameter=0.04, Hole=0"
```

Set D Code 100 to a circle.

```
.DCodeShape(100) = "S, Diameter=0.04, Hole=0"
```

Set D Code 100 to a square.

```
.DCodeShape(100) ="H, Outer=0.06, Inner=0.04, NumCuts=4,
   CutWidth=0.012, Angle=45"
```

Set D Code 100 to a thermal.

## DeleteDCodesIfUnused

Format:

```
.DeleteDCodesIfUnused( DCodes )
```

Removes the *DCodes* in the list which are not mapped to any elements in the current job (unused D Codes). *DCodes* is a comma-separated list of D Codes or D Code ranges.

Returns:

Number of deleted D Codes.

Example:

```
.DeleteDCodesIfUnused "10, 20-25, 33, 1001-10014"
```

See Also:

```
.DeleteUnusedDCodes
```

## DeleteSelected

Delete the currently selected visible elements.

Example:

```
.DeleteSelected
```

## DeleteUnusedDCodes

Deletes all D Codes that are unused (no elements currently loaded mapped to them).

See Also:

```
.DeleteDCodesIfUnused
```

# DFM_Drill

### Format:

```
.DFM_Drill( Layers, Options, What, AnnularRingOuter, AnnularRingInner,
   ClearanceOuter, ClearanceInner, ClearanceNonPlated, MinDrillSize,
   FileName )
```

Finds drill violations in *Layers*.

Sets `.LastItemCount` with the number of violations found.

### Arguments:

| | | |
|---|---|---|
| *Layers* | String | Comma separated list of layers or layer ranges to process |
| *Options* | Long | Bitmask of options:<br>Bit 0: 1=check only inside frame, 0=check the whole layer |
| *What* | Long | Bitmask<br>Bits that are set indicate that the specific check is to be performed<br>Bit 0: Annular Ring<br>Bit 1: Clearance<br>Bit 2: Nonplated Holes Clearance<br>Bit 3: Min drill size |
| *FileName* | String | Path name of the report file. The file will contain one line for each layer in the format shown below. If this string is the empty string (“”), then no output file will be generated. |

The rest of the arguments are double values that set the named values.

The format of a line in the report file (*FileName*) is:

*<layer number>, <X1>, <Y1>, <X2>, <Y2>*

Where the X,Y coordinates are the location of the violation. Note that they may represent a vector or a point (*X1*=*X2* and *Y1*=*Y2*) in which case it is the center of the violation (for cases when the vector is difficult to compute).

### Corresponding Command:

```
Custom DFM/Drill Violations
```

### See Also:

```
.LastItemCount
```

# DFM_FindMinimumSpacing

### Format:

```
.DFM_FindMinimumSpacing( Layers, Options, FileName )
```

Finds the minimum spacing between copper areas in the specified layers and generates a report file with the results.

### Arguments:

| | | |
|---|---|---|
| *Layers* | String | Comma separated list of layers or layer ranges to process. |
| *Options* | Long | Bitmask of options:<br>Bit 0: 1=check only inside frame, 0=check the whole layer<br>Bit 1: 1=also compute spacing to board outline (BOL layer, if present). The report lines that correspong to measurement against the BOL layer will end with ("to BOL") |
| *FileName* | String | Path name of the report file. The file will contain one line for each layer in the format shown below. If this string is the empty string (“”), then no output file will be generated. |

The format of a line in the report file (*FileName*) is:

*<layer number>, <X1>, <Y1>, <X2>, <Y2>*

The X, Y coordinates point to the spacing on the layer.
Example:

```
.DFM_FindMinimumSpacing( "1-2, 6", 1, "C:/jobs/Spacing Report.txt" )
```

Computes minimum spacing on layers 1, 2 and 6, considering only elements inside the frame. The report will be in the written to “C:/jobs/Spacing Report.txt”. An example of the output is:

```
 1)   1   0.2794      (31.20391, 15.05712) (31.20391, 15.33651)
 2)   1   0.4528      (44.05376, 8.76351) (44.50658, 8.76351) (to BOL)
 3)   2   0.2286      (36.2839, 20.72132) (36.2839, 20.94992)
 4)   2   1.3061      (45.32609, 20.21048) (44.42792, 21.1587) (to BOL)
 5)   3   0.38        (39.5955, 8.64052) (39.32679, 8.37181)
 6)   3   1.2045      (45.32609, 20.21048) (44.49779, 21.08494) (to BOL)
 7)   4   0.38        (39.5955, 8.64052) (39.32679, 8.37181)
 8)   4   1.2045      (45.32609, 20.21048) (44.49779, 21.08494) (to BOL)
 9)   6   0.9896      (39.81102, 8.85604) (39.11127, 8.15629)
10)   6   1.8669      (46.2788, 22.47392) (44.4119, 22.47392) (to BOL)
```

### Corresponding Command:

```
Custom DFM/Find Minimum Spacing
```

### See Also:

```
.DFM_FindMinimumWidth
```

## DFM_FindMinimumWidth

### Format:

```
.DFM_FindMinimumWidth( Layers, Options, FileName )
```

Finds the minimum copper width in the specified layers and generates a report file with the results.

The arguments are the same as for `.DFM_FindMinimumSpacing`, so please consult the manual entry for `.DFM_FindMinimumSpacing`.

### Corresponding Command:

```
Custom DFM/Find Minimum FeatureSize
```

### See Also:

```
.DFM_FindMinimumSpacing
```

# DFM_SpacingViolations

## Format:

```
.DFM_SpacingViolations( Layers, Options, Spacing, OpenTrap,
   ClosedTrap, Swell, AspectRatio, FileName )
```

Finds spacing violations in *Layers*.

Sets `.LastItemCount` with the number of violations found.

## Arguments:

| | | |
|---|---|---|
| *Layers* | String | Comma separated list of layers or layer ranges to process |
| *Options* | Long | Bitmask of options:<br>Bit 0: 1=check only inside frame, 0=check the whole layer |
| *Spacing* | Double | If spacing between two different copper areas is smaller than this value, a violation will be returned. Set to 0.0 if you don't want this check performed. |
| *OpenTrap* | Double | Open acid traps narrower than this value will be flagged as a violation. Set to 0.0 if you don't want this check performed. |
| *ClosedTrap* | Double | Closed acid traps narrower than this value will be flagged as a violation. Set to 0.0 if you don't want this check performed. |
| *Swell* | Double | Corrections will be inflated by this amount |
| *AspectRatio* | Double | Value between 0.0 and 1.0 used in the detection of acid traps. The closer this value is to 0.0, the more it will limit detection to elongated shapes. A setting of 1.0 will detect all acid traps. |
| *FileName* | String | Path name of the report file. The file will contain one line for each layer in the format shown below. If this string is the empty string (""), then no output file will be generated. |

The format of a line in the report file (*FileName*) is:

*<layer number>, <X1>, <Y1>, <X2>, <Y2>*

Where the X,Y coordinates are the location of the violation. Note that they may represent a vector or a point (*X1*=*X2* and *Y1*=*Y2*) in which case it is the center of the violation (for cases when the vector is difficult to compute).

## Example:

```
.DFM_SpacingViolations( "3", 0, 0, 0.12, 0, 0.02, 0.8, "traps.txt" )
```

Finds open acid traps only in layer 3. Acid traps smaller than 0.12 mm will be caught (assuming the units are set to mm). Corrections will be inflated by 0.02 mm. Only acid traps of aspect ratio smaller than 0.8 will be considered. The report will go into file "traps.txt" in the current directory. An example of the output is:

```
3, 130.90429, 20.84215, 130.90429, 20.84215
3, 130.89033, 18.64069, 130.89033, 18.64069
3, 131.51669, 21.3945, 131.51669, 21.3945
3, 131.22147, 21.8167, 131.22147, 21.8167
3, 131.68168, 21.07027, 131.68168, 21.07027
3, 130.90884, 23.59161, 130.90884, 23.59161
```

## Corresponding Command:

```
Custom DFM/Spacing Violations
```

## See Also:

```
.DFM_WidthViolations
.LastItemCount
```

## DFM_WidthViolations

### Format:

```
.DFM_WidthViolations( Layers, Options, FeatureSize, SliverSize,
   Method, FileName )
```

Finds width violations (feature size violations) in *Layers*.

Sets `.LastItemCount` with the number of violations found.

### Arguments:

| | | |
|---|---|---|
| *Layers* | String | Comma separated list of layers or layer ranges to process |
| *Options* | Long | Bitmask of option values. Currently only one option:<br>Bit 0: 1=check only inside frame, 0=check the whole layer |
| *FeatureSize* | Double | Any width smaller than this will be a violation |
| *SliverSize* | Double | Any copper area with width smaller or equal to this will be considered a sliver |
| *Method* | Long | Specifies how to look for violations:<br>0 = Check for feature size violations and ignore slivers<br>1 = Detect slivers only<br>2 = Check for both |
| *FileName* | String | Path name of the report file. The file will contain one line for each layer in the format shown below. If this string is the empty string (“”), then no output file will be generated. |

The format of a line in the report file (*FileName*) is:

*<layer number>, <X1>, <Y1>, <X2>, <Y2>*

Where the X,Y coordinates are the location of the violation. Note that they may represent a vector or a point (*X1*=*X2* and *Y1*=*Y2*) in which case it is the center of the violation (for cases when the vector is difficult to compute).

### Example:

```
.DFM_WidthViolations( "1", 0, 5, 0.5, 0, "width_violations.txt" )
```

Finds locations in layer 1 where the copper width is less than 5 mils (we assume the units were set to mil). Any copper area of width smaller than 0.5 mils will be ignored. The report will go into file “width_violations.txt” in the current directory. An example of the output is:

```
1, 9007.228, 7915.106, 9007.228, 7919.106
1, 9146.172, 7766.118, 9142.172, 7766.118
```

### Corresponding Command:

```
Custom DFM/Feature Size Violations
```

### See Also:

```
.DFM_SpacingViolations
.LastItemCount
```

## DisplayFilledElements

### Format:

```
.DisplayFilledElements = Boolean
```

Boolean property to control zero-width or filled display of elements.

### Corresponding Command:

Shortcut “O”

Example:
```
.DisplayFilledElements = False
```

## DragFrame

Format:
```
.DragFrame( On )
```

Activates or deactivates the dragging of the lower left corner of the frame. `On` is a Boolean.

Example:
```
.DragFrame( True )
```

Corresponding Command:
```
Setup/Frame/Grab LL Corner (W)
```

## DrawnPads

Format:
```
.DrawnPads( Subfigure, AutoRecord, ClearDestLayer )
```

Run the vector draw to flash conversion. This can flash the shape in a frame and export the shape to a file for use with the drawn pads library.

Example:
```
.DrawnPads(False, False, False)
```

Corresponding Command:
```
Tools/Convert Drawn Pads/Single Shape
```

See Also:
```
.DrawnPadsSetupOptions
```

## DrawnPadsFromLibrary

Run the draw to flash using the shapes and settings defined in the drawn pads library.

Example:
```
.DrawnPadsFromLibrary
```

## DrawnPadsSetupDCode

Define the D Code for the draw to flash to go to.

Format:
```
.DrawnPadsSetupDCode(DCode, UseNextUnusedDCode)
```

Example:
```
.DrawnPadsSetupDCode(4000, True)
```
Draw to flash to the first one unused D Code after 4000.
```
.DrawnPadsSetupDCode(4000, False)
```
Draw to flash to D Code 4000, whether used or not.

## DrawnPadsSetupOldShapes

Set method and layer for option.

Example:
```
.DrawnPadsSetupOldShapes( "Transfer", 10)
```
Transfer the old shapes to layer 10.

```
.DrawnPadsSetupOldShapes( "Delete", 10)
```
Delete the old shapes.
```
.DrawnPadsSetupOldShapes( "Highlight", 10)
```
Select the old shapes.

## DrawnPadsSetupOptions

### Format:

```
.DrawnPadsSetupOptions( Options, Tolerance )
```

Set up the options for vector draw to flash.

### Arguments:

*Options* is a bit-field defined as shown below:

| *Bit #* | *Mask (Hex)* | *Meaning* |
|---|---|---|
| 0 | 1 | Clear destination layer first |
| 1 | 2 | Use reference frame |
| 2 | 4 | Find shapes old in rotated orientations |
| 3 | 8 | Insert new shapes rotated |
| 4 | 10 | Identify old shapes by outlines only |
| 5 | 20 | Automatically set the new D Code dimensions |
| 6 | 40 | Automatically determine shape centers |
| 7 | 80 | Insert new shapes filleted |

*Tolerance* is the amount of variation allowed between different instances of the shape.

### Example:

```
.DrawnPadsSetupOptions( 96, .002 )
```

### See Also:

```
.DrawnPads
```

## DRC

### Format:

```
.DRC( Operation )
```

Run a DRC operation. *Operation* can be: `"ClearErrors"`, `"Contour"`, `"Raster"`, `"Vector"`, or `"HistogramVector"`.

Rules and options should be set up prior to this operation.

### Returns:

Number of violations found or -1 if the operation was aborted due to an error.

### Example:

```
.DRC( "Raster" )
```

## DRCErrorsFile

### Format:

```
.DRCErrorsFile( Operation, FileName )
```

Reads or writes DRC error files. These files are a human-readable text files.

*Operation* is an integer with the following possible values :

| | |
|---|---|
| 0 | Save the currently existing DRC errors to the file |
| 1 | Load the errors from the file. Any existing errors are removed. |
| 2 | Add the errors from the file to the existing ones (if any) |

The format of the violations in the file is:

```
<seq #>) <violation code>, <board layer number>, <distamce>, <x1>, <y1>, <x2>, <y2>
```

Where `<violation code>` is given in the table below:

| | |
|---|---|
| 1 | Trace to trace gap |
| 2 | Pad to pad gap |
| 3 | Trace to pad gap |
| 4 | Silk to pad gap (only Raster DRC) |
| 5 | Annular ring outer |
| 6 | Annular ring other |
| 7 | Annular ring plane |
| 8 | Annular ring thermal |
| 9 | Annular ring no pad |
| 10 | Mask to pad |
| 11 | Mask to trace |
| 12 | Mask to mask |
| 13 | Outline to copper |

### Returns:

*False* if operations fails.

### Example:

```
.DRCErrorsFile( 1, "job10_DRC_errors.txt" )
```

### Corresponding Command:

```
Tools/Design Rules Check/Save Errors To File
Tools/Design Rules Check/Load Errors From File
Tools/Design Rules Check/Add Errors From File
```

### See Also:

`.DRC`

## DRCMaxAirGap

Set the maximum air gap to list when doing a vector drc histogram. Any spacing below this will be reported.

### Example:

```
.DRCMaxAirGap = .008
```

List all spaces below .008

## DRCSetupOptions

### Format:

```
.DRCSetupOptions( Options )
```

*Options* is a comma separated list of options and values.

Options are: `InFrameOnly, DisplayFile, OutputReport, MaxErrors, PolygonsAsPads, CheckWithinNet`

Set the options for a DRC operation.

### Example:

```
.DRCSetupOptions( "InFrameOnly=False, OutputReport=True,
  MaxErrors=9999" )
```

## DRCSetupRules

### Format:

```
.DRCSetupRules( Rules, TT, TP, PP, PMask, TMask, DOuter, DOther,
  DPlane, DThermal, Outline, Silk )
```

Set the rules and values for a DRC operation. Note that as more rules and values were added new properties and methods were added in order to keep compatibility with existing scripts.

### Arguments:

*Rules*: bitmask used to define which rules are active.

| *Bit #* | *Mask* | *Rule* | *Vector* | *Raster* | *Contour* |
|---|---|---|---|---|---|
| 0 | 1 | Unterminated traces | X | | |
| 1 | 2 | Trace to trace air gap | X | X | X |
| 2 | 4 | Trace to pad air gap | X | X | X |
| 3 | 8 | Pad to pad air gap | X | X | X |
| 4 | 0x10 | Centerline disconnection | X | | |
| 5 | 0x20 | Pad to pad contact | X | | |
| 6 | 0x40 | Soldermask gap | X | X | X |
| 7 | 0x80 | Annular ring to drill gap | X | X | X |
| 8 | 0x100 | Trace intersections | X | | |
| 9 | 0x200 | Outline to copper area | X | X | X |
| 13 | 0x2000 | Silkscreen to pad gap | | X | X |

The other arguments are the dimensions to be used with the rules:

*TT* = trace to trace air gap, *TP*=trace to pad air gap, *PP*=pad to pad air gap, *PMask*=soldermask to pad, *TMask*=soldermaks to trace, *DOuter*=drill to outer, *DOther*=drill to other, *DPlane*=drill to plane, *DThermal*=drill to thermal, *Outline*=ouline to copper, *Silk*=silkscreen to pad

### Example:

```
.DRCSetupRules( 8, 0.0265, 0.0265, 0.0175, 0.008, 0.008, 0.004,
  0.0025, 0.004, 0.002, 0.005, 0.008 )
```

### See Also:

```
.DRCSilkScreenAgainst
```

## DRCMaskToMask

Value for the “Soldermask gap” / “Mask to mask” rule. One can turn this rule off by setting this value to 0. This rule is only checked during Contour DRC.

Example:

```
.DRCMaskToMask = 0.006
```

## DRCSilkscreenAgainst

Layer type to check the silkscreen against for the DRC. (CPx, MSx)

Example:

```
.DRCSilkscreenAgainst = "CPU"
```

Check the silkscreen against the component upper layer.

```
.DRCSilkscreenAgainst = "MSU"
```

Check the silkscreen against the mask upper layer.

## DrillDataFormat

Format:

```
.DrillDataFormat = Comma-Separated-String
```

Settings for importing drill and rout files. Where the string can contain the items shown below. Not all items need to be present. Items that are not present will keep their old values.

| | |
|---|---|
| *Excellon* | |
| *Sieb & Meyer* | |
| *Wessel* | |
| *Trailing* | |
| *Leading* | |
| *All* | |
| *Decimal* | |
| *Incremental* | |
| *Absolute* | |
| *English* | |
| *Metric* | |
| *Quadrant* | |
| *360-Degree* | |
| *Left* | Assigned integer |
| *Right* | Assigned integer |
| *ASCII* | |
| *EBCDIC* | |
| *EIA* | |
| *ToolsOpposite* | Assigned boolean |

Example:

```
.DrillDataFormat="Left=2, Right=4, Absolute, English, ASCII, Trailing,
  Excellon"
```

Corresponding Command:

File/Import/Drill & Rout/Options: "Data Format" tab.

See Also:

.Import

.DrillImportOptions

.AddToFileList

## DrillExportFormat

Format:

```
.DrillExportFormat = Comma-Separated-String
```

Settings for Exporting drill files. Where the string can contain, in any order the following: Places before the decimal, Places after the decimal, Position coordinates, Data units, Data code, Omit leading/trailing zeroes, language format. Not all items need to be present. Items that are not present will keep their old values.

Example:

```
.DrillExportFormat="Left=2, Right=4, Absolute, English, ASCII,
   Leading, Excellon"
```

Corresponding Command:

```
File/Export/Drill/Options: "Data Format" tab.
```

## DrillImportOptions

String property to get or set import options for NC data (drill and rout). It is a comma separated list of item assignments. The items are listed in the table below:

| | | |
|---|---|---|
| *IgnoreG92* | Boolean | |
| *BlockDelete* | String | *Follow, Read, Delete* |
| *M00* | String | *End, Ignore, Tool* |
| *Format2* | Boolean | |

Example:

```
.DrillImportOptions="IgnoreG92=False, BlockDelete=Read"
```

Corresponding Command:

```
File/Import/Drill & Rout/Options: "Options" tab.
```

See Also:

```
.Import
.DrillDataFormat
```

## DrillPathLengthInLayer

Format:

```
.DrillPathLenthInLayer( LayerNumber )
```

Returns:

Returns as a double value the path length of all the drill holes in layer *LayerNumber*.

## EdgeToEdgeDistance

Returns the distance edge to edge of two selected entities. If the number of selected entities is not exactly 2, returns -1.

Example:
```
x = .EdgeToEdgeDistance
```

## ExePathName

Returns the full path name of the executable, for instance "C:\Program Files\PentaLogix\CAMMaster 9.6\CAMMaster.exe".

Format:
```
.ExePathName
```

## Export

Format:
```
.Export( FileName, FileType, LayerList )
```

Export files. If *FileName* is an empty string then all layers in *LayerList* will be exported, one file for each layer, using the layer names as file names.
Otherwise the layers in *LayerList* will be exported (combined) into one file with the name given by *FileName*.
*LayerList* is a comma-separated list of layer numbers or ranges.

*FileType* is one of: "Basic Gerber", "GerberX", "MDA", "Drill", "HPGL", "DPF", "Netlist" or "PentaLogix". For a description of the PentaLogix format, please see Appendix A.
Set all options for the different output types before export, using the appropriate methods and properties (see for instance ExportDrillOptions below). Note also that only elements and layers that are visible at the time of the export will be exported.

Returns:

False if an error ocurred during the export, True otherwise.

Example:
```
.Export( "", "Drill", "7" )
```
Export the data on layer 7, using the current layer name, as a drill file.
```
.Export( "", "GerberX", "1-3" )
```
Export layers 1 through 3, using the current layer names, layer by layer, gerber extended.
```
.Export( "Combined.gbr", "MDA", "1-3,6,8" )
```
Export layers 1 through 3, layer 6 and layer 8, combined under the filename "Combined.gbr" in MDA format.

See Also:

.Import

## ExportDrillOptions

Comma separated list property to set the drill options for export. Before .Export of a drill file set the .DrillExportFormat and the .ExportDrillOptions.

Example:
```
.ExportDrillOptions="Tool parameters=True, Both X&Y=False, S&R
  codes=False"
```

See Also:

.DrillExportFormat

## ExportGBX…

Format:

```
.ExportGBXThermal = Boolean
.ExportLTIEnv = Boolean
.ExportGBXKnockout = Double
.ExportGBXSR = Boolean
.ExportGBXTimeStamp = Boolean
.ExportGBXOptions = CommadSeparatedList
```

Options that apply to exporting Gerber extended files.

Corresponding Command:

File/Export/Gerber Extended → Options. "RS274-X Options" and "Polygons" tabs.

See Also:

.Export

## ExportLavenirEnv

Format:

```
.ExportLavenirEnv( FileName, VersionNumber )
```

Export a Lavenir type environment (aperture list).

Example:

```
.ExportLavenirEnv( "Sample1.env", 4 )
```

## ExportMDAFilm

Add the current frame size to the header of the file (FSZE)

Example:

```
.ExportMDAFilm = True
```

## ExportMDAMergePaintScratch

Export MDA file with the merge option for paint and scratch. Adds the MRGE and Next to the file header.

Example:

```
.ExportMDAMergePaintScratch = True
```

## ExtractNetlist

Format:

```
.ExtractNetlist( Method, Options )
```

Extract a netlist from the currently loaded data, using the specified options.

`Method` is one of `Vector`, `Raster` or `Contour`.

Example:

```
.ExtractNetlist( "Vector", "GerberTextSize=0.02, GerberDCode=500,
  NetlistFilename=C:\PentaLogix\job\bins\123456ann.f04,
  AlternateTestPoints=True, TraceIntersections=True, InsideFrame=True,
  StepRepeat=False, Report=True, Format0=True, Format1=True,
  Format2=True, Format3=True, Format4=True, GerberRefData=True,
  ExtraTestPointOnly=False, LloydDoyleFormat=False,
  StepRepeat=False" )

.ExtractNetlist( "Raster", "AdjacencyData=True,
  BaseFilename=C:\PentaLogix\job\bins\123456ann, CombineLayers=True,
  ExportFiles=True, IdentifyMidpoints=True, MinNetSeparation=25,
  MinPlaneSize=300, NetCountReport=True, PhaseTestData=True,
  Resolution=2" )

.ExtractNetlist( "Contour",
  "BaseFilename=C:\PentaLogix\job\bins\123456, CombineLayers=True,
  ExportFiles=True, IdentifyMidpoints=True, MinNetSeparation=25,
  MinPlaneSize=300, NetCountReport=True, PhaseTestData=True,
  AddTestPointsAlso=True, Format2=True, Format4=True, IPC356A=True" )
```

Corresponding Command:

```
Tools/Netlist/Extract…
```

## ExportRoutOptions

Comma separated list property to set the rout options for export of a rout NC file.

Corresponding Command:

```
File/Export/Drill & Rout - Options - Rout Options
```

## FillPolygons

Format:

```
.FillPolygons( Method, FillLayer, ClearFillLayer, DCode, Diameter )
```

This fill operation does not affect polygons but rather closed outlines drawn with F-type D Codes and is also called "copper pouring". The outlines will be filled with *DCode,* which will be set to shape C and *Diameter*.

*FillLayer* is a string that specifies the first destination layer (where the fill will go) and can be the string "`Same`" or a layer number. If *ClearFillLayer* is `True` then the destination layers will be deleted before the new fill is generated. The following values are valid for *Method*:

"`Solid`"

"`Outlines Only`"

"`Hatched`"

"`Crosshatched`"

"`Inside Elements`"

Make sure to set all variables relating to the fill (all such methods start with FillSetup…).

Example:

```
.FillPolygons( "Outlines Only", "92", False, 9801, .001 )
```

Fill the outline of the polygon only, fill layer 92, do not clear the fill layer first, use D Code 9801 of size .001.

```
.FillPolygons( "Solid", "94", False, DCodeForFill, 0.05 )
```
Fill the outline of the polygon only, fill layer 94, do not clear the fill layer first, use DCodeForFill variable of size 0.05.

Corresponding Command:

```
Tools/Fill F-Polygonal Outlines
```

See Also:

`FillSetupCircuit, FillSetupGeneral, FillSetupOutline, FillSetupOverlap, FillSetupPolygon.`
`FillSolid` is to be used for filling polygon (rather than F-type outlines).

## FillSetupCircuit

Format:

```
.FillSetupCircuit( AroundElements, Clearance )
```

Method to set up parameters used in filling F-polygonal outlines. These parameters refer to the "circuit" parameters.

Arguments:

| | | |
|---|---|---|
| *AroundElements* | Boolean | If `True` the fill will be around the circuit elements, otherwise over them |
| *Clearance* | Double | Relevant only when `AroundElements` is `True`. Then it represents the distance to keep away from the circuit elements |

Example:

```
.FillSetupCircuit( False, 0 )
```

Corresponding Command:

```
Tools/Fill F-Polygonal Outlines - Circuit
```

See Also:

`FillPolygons`

## FillSetupGeneral

Settings for filling "F" D Codes

Format:

```
.FillSetupGeneral( Angle, Clearance, ShortestFill, TwoDCodes, Outline,
   RoundToDecimal )
```

Example:

```
.FillSetupGeneral( 0, 0.1, 0.001, False, True, 2 )
```

## FillSetupHatch

Crosshatch angle and spacing

Format:

```
.FillSetupHatch( Spacing, CrossHatchAngle )
```

Example:

```
.FillSetupHatch( 0.01, 90 )
```

## FillSetupOutline

When filling with outlines

Format:
```
.FillSetupOutline( PolyOnly, PolyClearance, ElementClearance )
```
Example:
```
.FillSetupOutline( False, 0, 0 )
```

## FillSetupOverlap

Overlap is either set or variable, and how much, when filling "F" D Codes

Format:
```
.FillSetupOverlap( Variable, Overlap )
```
Example:
```
.FillSetupOverlap (True, 0.0005)
```

## FillSetupPolygon

Settings when filling "F" type D Codes. (Polygon selection tab)

Format:
```
.FillSetupPolygon( AddFrame, SelectPolygons, PolygonGaps, Tolerance,
   PolygonNesting, IgnoreDuplicates)
```
Example:
```
.FillSetupPolygon( True, False, 1, 0, 2, True )
```

## FillSolid

Format:
```
.FillSolid( DCode )
```
The selected polygons will be replaced with a solid fill done with D Code *DCode*. The fill is done with horizontal traces. Note that this method acts on the polygons in the selection only (as opposed to FillPolygons).

Returns:

The number of newly inserted traces (for the fill) or -1 if an error ocurred.

Example:
```
.FillSolid( 501 )
```

## FirstNewDCodeStartingAt

Return the first new D Code starting at the specified D Code.

Example:
```
.FirstNewDCodeStartingAt(4000)
```

## FirstUnusedDCode

Return the first unused D Code starting at the specified D Code.

Example:
```
.FirstUnusedDCode(4000)
```

## FirstUnusedToolNumber

Return the first unused tool starting at the specified tool.

Example:
```
.FirstUnusedToolNumber(1)
```

## FourCornersMode

Set the crosshair movement on the frame to 4 corners (or not).

Example:
```
.FourCornersMode = True
```

## Frame

Format:
```
.Frame = "lowerX, lowerY, upperX, upperY"
```

String property for access to the frame coordinates.

Example:
```
.Frame = "0, 0, 12, 18"
```

## FrameAllData

Place a frame around all data at the specified margins. (Like Ctrl+W command)

Example:
```
.FrameAllData( .1, .1 )
```

## FrameVisible

Format:
```
.FrameVisible = Boolean
```

Turns the frame visibility on or off.

Corresponding Command:
```
Setup/Frame/Visible
```

Example:
```
.FrameVisible = True
```

## FtoPolygons

Format:
```
.FToPolygons( Tolerance, Flags )
```

Searches the "F"-type D Codes in the selection for closed trace contours and replaces them with polygons. `Tolerance` is used to close gaps between endpoints.

`Flags` is a bit-mask which sets options for the operation
Bits in `Flags`:

| *Bit* | *Value* | Meaning |
|---|---|---|
| *0-1* | 0-3 | `00`: Polygons are not nested. Only containers result.<br>`01`: If two holes overlap one of them becomes a container.<br>`10`: Two overlapping holes are merged to make a larger hole.<br>`11`: The overlapping area of two holes becomes a new container. |
| *2* | 4 | Unused |
| *3* | *8* | Path can continue at overlap. Will do special processing of overlapping traces |

Example:
```
.FToPolygons( 0, 1 )
```
Corresponding Command:
```
Edit/Edit Selection/Polygons/F To Polygons
```
See Also:
```
PolygonsToF
```

## GenerateOutlines

Format:
```
.GenerateOutlines(Raster, ClearDestLayer, InFrameOnly, CurrentDCOnly)
```
Generate outlines/polygons based on raster or vector method. For vector, set `.OutlinesSettings` first.

Example:
```
.GenerateOutlines( False, False, True, True )
```
Corresponding Command:
```
Tools/Generate Polygon Outlines
```
See Also:
```
.OutlineSettings
```

## GenerateReportFile

Generate a report file.

Example:
```
.GenerateReportFile
```
See Also:
```
.Reportfilename
```

## GenerateSoldermask

Generate a soldermask layer from the specified padmaster. Set `.SolderMaskSetup` variables first.

Format:

```
.GenerateSoldermask( Padmaster, Soldermask, ClearFirst, InFrameOnly,
   CurrentDCOnly )
```

Example:

```
.GenerateSoldermask( 1, 13, False, False, False )
```

See Also:

```
.SolderMaskSetup
```

## GerberDataFormat

Set the input and output variables for Gerber data.

Example:

```
.GerberDataFormat="Places before the decimal, Places after the decimal, Position coordinates,
   Data units, Data Code, Omit leading zeroes, Arc interpolation"
```

Example:

```
.GerberDataFormat="Left=2, Right=4, Absolute, English, ASCII, Leading,
   Quadrant"
```

## GerberExportFormat

Set the output variables for Gerber data.

Example:

```
.GerberExportFormat="Left=2, Right=4, Absolute, English, ASCII,
   Leading, Quadrant"
```

## GerberImportFormat

Set the input variables for Gerber data.

Example:

```
.GerberImportFormat="Left=2, Right=4, Absolute, English, ASCII,
   Leading, Quadrant"
```

## GerberOutputBothXY

Property controlling Gerber export. True = Both X and Y fields are output in every block even when they may repeat the same value from a previous block.

Example:

```
.GerberOutputBothXY=False
```

## GerberOutput2Digits

Property controlling Gerber export. True =G and D commands will be output using two digits, as in D01 instead of D1.

Example:

```
.GerberOutput2Digits= False
```

## GerberOutputDAlways

Property controlling Gerber export. True =draw-type D commands (such as D1) will be output in each block even when they repeat the same value from a previous block.

Example:
```
.GerberOutputDAlways= False
```

## GerberOutputG54

Property controlling Gerber export. True =D code change D commands will be preceded by a G54.

Example:
```
.GerberOutputG54= False
```

## GerberOutputFlash

Property controlling Gerber export. Specifies how the flash commands (normally D3) will be output as shown in the table below:

| Value | How flash command is output |
|---|---|
| 0 | D3 |
| 1 | D2*D3 |
| 2 | D2*G55D3 |
| 3 | D3*D3 *(this is needed by some Allegro software)* |

Example:
```
.GerberOutputFlash=0
```

## GerberOutputNewline

Property controlling Gerber export. True =each block will be followed by a newline character.

Example:
```
.GerberOutputNewline= False
```

## GerberOutputSplitPolygons

Property controlling Gerber export. True: polygons with holes will be output as a collection of polygons without holes that are generated by horizontally splitting the original polygons. False: holes in polygons will be output as scratch entities (on a separate layer).

This is needed because Gerber syntax for polygons (G36/G37) does not allow a way for specifying that some polygons are holes in other (containing) polygons and so some other way of representing them is necessary.

Example:
```
.GerberOutputSplitPolygons = False
```

## GetBoardLayerDimensions

Format:
```
.GetBoardLayerDimensions( BoardLayerNumber, AtZeroWidth )
```

Returns:

The dimensions of the specified visible board layer as a comma separated string: lowerx, lowery, upperx, uppery.

Example:

```
.GetBoardLayerDimensions(1, True)
```

See Also:

.GetLayerDimensions

## GetCopperAreaValues

Format:

```
.GetCopperAreaValues( Method, Resolution, BoardThickness, ShowReport )
```

Calculates the copper area according to the settings in the parameters. Layers must be specified with .ChosenLayers.

For a description of the arguments, please see the manual entry for .CalculateCopperArea.

It computes the same values, but returns a comma separated list as a result.

Returns:

A comma separated list of results per layer.

Values are the same as the ones in the report file and each item in the list has the form:

```
<Board layer-number>) <percent>% area
```

Example:

```
.ChosenLayers="1, 6, 12"
Dim result As String
result = GetCopperAreaValues( "Upper/Lower", 0.5, 0.125, True )
```

Returns in "result" (as an example) the string: "1) 35.9% 16.400, 6) 31.1% 14.299".

Corresponding Command:

Tools/Calculate Copper Area

See Also:

CalculateCopperArea

ChosenLayers

## GetCursorPosition

Format:

```
.GetCursorPosition
```

Returns the cursor position as displayed on the screen toolbar (cursorx, cursory or dist, angle).

Example:

```
.PolarCoordinatesDisplay = True
X = .GetCursorPosition
```

X will be assigned the string "5.248018, 10.254°"

## GetDCodeElementCounts

Format:

```
.GetDCodeElementCounts( DCode )
```

Returns a string with the element counts for the specified D Code (*DCode*). It is in the usual format of a comma separated list of counts: traces, arcs, pads. The traces count includes arcs. If there are no elements mapped to this D Code the returned value is the string "?".

Example:
```
.GetDCodeElementCounts( 45 )
```

## GetHighestLoadedLayer

Returns:

Layer number of highest loaded layer that contains data.

## GetLayerDimensions

Format:
```
.GetLayerDimensions( Layer, AtZeroWidth )
```

Return the dimensions of the specified layer number. If `AtZeroWidth` is `True` then the result is all computed as if the elements were mapped to D Codes of width 0.

Important NOTE: only visible elements are taken into accout. So, for instance, if the whole layer is not visible, this method returns zero dimensions although the layer may contain data.

Returns:

Comma separated string: lowerx, lowery, upperx, uppery.

Example:
```
.GetLayerDimensions( 2, False )
```

See Also:

`.GetLayerDimensionsEx`

`.GetBoarLayerDimensions`

## GetLayerDimensionsEx

Format:
```
.GetLayerDimensionsEx( LayerNumber, Options )
```

Return the X, Y dimensions of the specified layer number. `Options` is a comma separated string of boolean assignments that specify how the computation is done, as explained in the table below:

| *Name* | *Meaning* | *Default* |
|---|---|---|
| `CopperArea` | Computes the dimensions of the actual copper. This requires computation on polygons if the layer is negative or contains scratch elements, so it can take longer to perform. | `False` |
| `ScratchAlso` | Scratch elements are included in the computation. | `True` |
| `VisibleOnly` | Only visible elements are included in the computation. | `False` |
| `AtZeroWidth` | Computation done as if all elements were mapped to D Codes of size 0. | `False` |

Returns:

Comma separated string: `lowerx, lowery, upperx, uppery`.

Example:

```
.GetLayerDimensionsEx( 5, "VisibleOnly=True, ScratchAlso=False" )
.GetLayerDimensionsEx( 2, "CopperArea=True" )
```

See Also:

.GetLayerDimensions

## GetLayerElementCounts

Format:

```
.GetLayerElementCounts( LayerNumber )
```

Returns the element count of the specified layer as a comma separated list of counts: traces, arcs, pads, polygons. The traces count includes arcs. If the layer is empty, an empty string is returned.

Example:

```
.GetLayerElementCounts(12)
```

## GetLayerPolarityChanges

Format:

```
.GetLayerPolarityChanges( LayerNumber )
```

Returns:

The number of changes between groups of paint elements and scratch elements within a layer. For layers of single polarity (where all elements inside the layer are the same polarity – usually paint) the returned value is 0.

See Also:

.PolarityCombineAll

.PolaritySeparateAll

## GetLockingCode

Format:

```
.GetLockingCode
```

Returns:

The locking code of the current session. This is the same string as shown by "Help/Locking Code".

## GetProcessID

Format:

```
.GetProcessID
```

Returns:

The process ID of the process executing CAMMaster as a Long.

## GetSelectionDCodes

Format:

```
.GetSelectionDCodes("Shape Code")
```

*Shape Code* can specify a shape type or can be empty ("") for all shapes.

Returns the D Codes in the current selection ordered by size. Only visible items are reported. Can be shape type or empty for all .

Example:

```
.GetSelectionDCodes("C")
```

Returns D Codes of type C in the selection. For instance: 100,106,108,114

See Also:

```
.GetSelectionTools
```

## GetSelectionDimensions

Format:

```
.GetSelectionDimensions( AtZeroWidth, OnlyVisible )
```

Returns the dimensions of the selection

Example:

```
.GetSelectionDimensions(False, True)
```

## GetSelectionElementCounts

Format:

```
.GetSelectionElementCounts
```

Returns the count of primitive elements in the selection as a string. The format is a comma separated list of numbers as follows: total trace count (linear and arcs), arcs count, pads count, polygons count.
If the selection is empty then an empty string is returned.

Example:

```
x = .GetSelectionElementCounts
```

## GetSelectionToolNumbers

Format:

```
.GetSelectionToolNumbers( ToolType )
```

Returns a comma separated list of tool numbers of the deisred type that exist in the current selection. Only visible items are reported. Note that, as opposed to `.GetSelectionTools`, these are tool numbers (not D Codes) and they also include tools used by routing paths (routing paths are not mapped to D Codes). The list can include ranges, if all tools in the range appear in the selection.

*ToolType* can specify a tool type as a string or the empty string ("") for all types.

Example:

```
.GetSelectionToolNumbers( "Plated" )
```

Can return a string like "`1-7, 15`". This menas that tools 1 through 7 and 15 are used in the selection (by drill holes or routing paths) and are of type "Plated".

See Also:

```
.GetSelectionTools
.GetSelectionDCodes
```

## GetSelectionTools

Format:

```
.GetSelectionTools( ToolType )
```

Returns a comma separated list of D Codes mapped to "N" shape type in the current selection, ordered by size. Only visible items are reported.

*ToolType* can specify a drill tool type as a string or the empty string ("") for all types.

Example:

```
.GetSelectionTools( "Plated" )
```

Returns drill tools of type Plated in the selection the selection (for instance "501, 504, 600", where 501, 504 and 600 are D Code numbers).

See Also:

```
.GetSelectionToolNumbers
.GetSelectionDCodes
```

## GetStepAndRepeatIDs

Format:

```
.GetStepAndRepeatIDs()
```

Returns a comma separated list of ID numbers of step and repeat blocks in the current active layers. An empty string is returned id there are no such blocks.

See Also:

```
.SelectStepAndRepeat
```

## GetVisibleDataDimensions

Format:

```
.GetVisibleDataDimensions( AtZeroWidth )
```

Returns the coordinates of the visible data. If *AtZeroWidth* is `True` then the shape of elements will not be taken into account (only the endpoints).

Returns:

A comma-separated string with the four coordinates of the bounding box of all visible data. The order is lowerx, lowery, upperx, uppery.

Example:

```
.GetVisibleDataDimensions( False )
```

## GetWindowHandle

Format:

```
.GetWindowWandle()
```

Returns:

An integer that is the handle of the main window of the application.

## GotoActive

Go to the selected elements in the specified order

Format:

```
.GoToActive( Which)
0=First, 1=Next, -1=Previous
```

Example:

```
.GoToActive(0)
```

## Grid

`String` property to control grid settings as a comma separated list.

Example:

```
.Grid="On, Fixed, Anchored, X=0.001, Y=0.001"
.Grid="On, Fixed, Dragged, X=0.001, Y=0.001"
```

## GuessFileTypes

Format:

```
.GuessFileTypes( Directory )
```

The file types of the files in folder *Directory* are guessed and returned.

Returns:

Comma separated list of pairs <*filename*>=<*filetype*>. *filename* does not include the full path name (only name and extension). *filetype* is one of the value below:

```
”ENVIRONMENT", "APERTURE", "GERBER", "EXT. GERBER", ”MDA", "HPGL",
"NC-Excellon", ”NC-Wessel", "NC-Sieb & Meyer", "ZIP", "PAK", "ARJ",
"LZH", "DXF", "BIN", "JOB", "IPC-D-356", "F04", "DPF", "ODB++",
"BMP", "Error File", "ADI - netlist format", "ADI - database
format", "ATF", Boeing", "Cadence/Allegro", "Fabmaster - Nail",
"GerbTool", "Luther & Maelzer", "Mentor", "NTD",
"Optrotech/Orbotech", "Pads", "Pcad", Rockwell", "Scicards", "TIF",
"Net Source 1", "Net Source 2"
```

Example:

```
.GuessFileTypes( "C:\PentaLogix\Demo Bins” )
```

Returns:

```
123456.1=EXT. GERBER, 123456.1d=NC-Excellon, 123456.drl=NC-Excellon,
  123456.f04=F04, 123456.ipc=IPC-D-356
```

See Also:

`ImportDirectory`

## HighestPossibleLayerNumber

The highest layer number supported by the current version of the software. This is a readonly property (it cannot be changed).

Example:
```
If .HighestPossibleLayerNumber > 99 Then…
```

## Import

Format:
```
.Import( FileType )
```

Import files of the specified file type. The file names are taken from the list collected (previously) with the `.AddFileToList` method. If that list is empty, nothing happens. The list is cleared after the import is finished.

The import starts at layer `.CurrentLayer` and layers are incremented as each new file is imported.

*`FileType`* is one of: `"Gerber", "Drill", "HPGL", "DPF", "ODB++", "Bitmap"` or `"PentaLogix"`. For a description of the `PentaLogix` format, please see Appendix A. Gerber file import options are found under "File/Import/Gerber/Options". A few are listed below:

```
.ImportD03Modal, .ImportAllowD4D9, .ImportZeroLengthDraw,
.ImportIsolatedD01, .ImportAMRotation, .ImportAutotranscode,
.ImportAutoloadApertures, .ImportAllApSameUnits,
.ImportGerberUnitsSameAsAp, .ImportApsInGerber,
.ImportKeepAllFlashes, ImportEOBCR, .ImportMoveArcCenter
```

Example:
```
.Import( "Gerber" )
```

See Also:

`.AddFileToList`

`.CurrentLayer`

`.ImportFromLayers`

## ImportAMRotation

String property.
Comma-separated list of up to two items that describe how rotation in Gerber extended custom apertures (AM command) is interpreted. One of the items decribes the orientations and has possible values: `"CW", "CCW"`. The other specifies where the center of the rotation is for a primitive that is rotated. The possible values are: `"Primitive", "Aperture", "Automatic"`. `"Primtive"` causes the rotation to happen around the center of each primitive, `"Aperture"` causes the rotation to happen around the center of the whole aperture and `"Automatic"` will pick one of the two methods depending on the type of primitive. `"Automatic"` is the recommended setting.

Example:
```
.ImportAMRotation = "CCW, Automatic"
```

Corresponding Command:

`File/Import/Gerber/Options/RS-274X Options`

See Also:

`.Import`

## ImportAutoTranscode

Boolean property. Controls what happens when importing a Gerber file with an aperture that is mapped to a shape that conflicts with what is already in the database. If set to "True", these D Codes will be transcoded to new D Codes so that the old shapes (as well as the new shapes) are preserved.

Example:

```
.ImportAutoTranscode = "True"
```

Corresponding Command:

```
File/Import/Gerber/Options/Auto Features/Autotranscode D Codes
```

See Also:

```
.Import
```

## ImportDiscardPaths

Boolean property that specifies whether the path part of file names will be discarded when naming the layers during an import operation.

Example:

```
.ImportDiscardPaths = True
```

Corresponding Command:

```
File/Import/Discard Path In Layer Names
```

See Also:

```
.Import
```

## ImportEOBCR

Boolean property that specifies whether the Gerber end-of-clock character is "carriage return" or the asterisk ("*").

Example:

```
.ImportEOBCR = True
```

Corresponding Command:

```
File/Import/Gerber/Options/File Interpretation/End Of Block
Characters
```

See Also:

```
.Import
```

## ImportFromLayers

Import the files as recorded in the layers dialog ("Setup Layers" or F10)

Example:

```
.ImportFromLayers
```

Corresponding Command:

```
File/Import/From Layers
```

## ImportGuess

Format:

```
.ImportGuess( FileName, LayerNumber )
```

Import a file into the specified layer. The file type and settings will be guessed by the software.

Returns:

True if the imports is successful, False otherwise.

Example:

```
.ImportGuess( "C:\Data\test1.dpf", 7 )
```

Corresponding Command:

```
File/Import/Guess
```

## ImportGuessEx

Format:

```
.ImportGuessEx( FileName, LayerNumber, Options )
```

Import a file into the specified layer. The file type and settings will be guessed by the software taking into account the provided *Options*. The syntax for *Options* is the same as for `.ImportDirectory`.

Returns:

True if the imports is successful, False otherwise.

Example:

```
.ImportGuessEx( "Board1.drl", 3, "MaxSizeX=9.0, MaxSizeY=5.0" )
```

Corresponding Command:

```
File/Import/Guess
```

See Also:

```
ImportDirectory
```

## ImportHPGLOptions

Set options for import of HPGL files.

Example:

```
.ImportHPGLOptions="DCode=101, Auto-Select text DCode=True, Ignore IP
  commands=True, Isotropic scaling=True, Ignore Polygon Fills=True,
  Zero-length Draws=Ignore"
```

## ImportDirectory

Format:

```
.ImportDirectory( Options )
```

Import the files in a directory.

### Arguments:

*Options* is a comma separated list of items, which are shown below:

| | | |
|---|---|---|
| `Directory` | String | Path of directory to import |
| `MinSizeX` | Double | Min X dimension when guessing a scale factor for file |
| `MinSizeY` | Double | Min Y dimension when guessing a scale factor for file |
| `MaxSizeX` | Double | Max X dimension when guessing a scale factor for file |
| `MaxSizeY` | Double | Max Y dimension when guessing a scale factor for file |
| `GerberUnits` | String | `English` or `Metric` |
| `DrillUnits` | String | `English` or `Metric` |
| `AptFilesUseSameUnits` | Boolean | If multiple aperture files are being imported, the user will only be prompted once for the unit size |
| `ScaleGivenFiles` | Boolean | Scale unformatted files to formatted file size. Min, Max X & Y will be ignored. |
| `BoardUnits` | String | `Inch, mil, cm` or `mm` |
| `ScanForData` | Boolean | Compare file to know data file types i.e. Gerber, Excellon |
| `ScanForApertures` | Boolean | Compare file to known aperture file types |
| `ScanForNetlists` | Boolean | Compare file to known Netlist file types |
| `ShowImportDialog` | Boolean | Display the scan results before importing the usable file(s) |

### Returns:

Comma separated list of pairs <*filename*>=<*filetype*>. *filename* does not include the full path name (only name and extension). *filetype* is one of the value below:

```
”ENVIRONMENT", "APERTURE", "GERBER", "EXT. GERBER", ”MDA", "HPGL",
"NC-Excellon", ”NC-Wessel", "NC-Sieb & Meyer", "ZIP", "PAK", "ARJ",
"LZH", "DXF", "BIN", "JOB", "IPC-D-356", "F04", ”DPF", "ODB++",
"BMP", "Error File", "ADI - netlist format", "ADI - database
format", "ATF", Boeing", "Cadence/Allegro", "Fabmaster - Nail",
"GerbTool", "Luther & Maelzer", "Mentor", "NTD",
"Optrotech/Orbotech", "Pads", "Pcad", Rockwell", "Scicards", "TIF",
"Net Source 1", "Net Source 2"
```

### Example:

```
.ImportDirectory( "Directory=C:\PentaLogix\Demo Bins, MinSizeX=3,
   MinSizeY=3, MaxSizeX=30, MaxSizeY=30, GerberUnits=English,
   DrillUnits=English, AptFilesUseSameUnits=False,
   ShowImportDialog=True" )
```

Returns:

```
123456.1=EXT. GERBER, 123456.1d=NC-Excellon, 123456.drl=NC-Excellon,
   123456.f04=F04, 123456.ipc=IPC-D-356
```

### Corresponding Command:

`File/Import/Directory`

### See Also:

`GuessFileType`

## ImportIsolatedD01

Format:

```
.ImportIsolatedD01 = String
```

String property that sets an option for importing Gerber files. Determines what to do with a Gerber block that has a D01 (draw) command, but no X, Y, I or J fields. It can have two values:

| | |
|---|---|
| `Ignore` | Block is discarded |
| `Zero-length` | Block is treated as zero-length draw. Behavior depends on `ImportZeroLengthDraw` |

Example:

```
.ImportIsolatedD01 = Ignore
```

Corresponding Command:

```
File/Import/Gerber/Options/File Interpretation/Isolated D01
```

See Also:

```
Import
ImportZeroLengthDraw
```

## ImportJob

Format:

```
.ImportJob( FileName, LayerOffset, OriginX, OriginY )
```

Import a job (.bin file) offset the specified amount in x *(OriginX)* and y *(OriginY)*.

Layer numbers in the jobfile are offset by *LayerOffset*. If 0 then the layer numbers are preserved and data is imported in the same layers as saved in the jobfile. *LayerOffset* can be positive of negative.

Example:

```
.ImportJob( "C:\PentaLogix\Demo Bins\Demo", 9, 0, 0 )
```

Corresponding Command:

```
File/Import/Job
```

## ImportMoveArcCenter

Boolean property used during importing Gerber files. It influences how the software imports arcs that are inexact (the two radii at the endpoints are not the same). If set to `True`, then the center of the arc will be moved. If set to `False`, the center will be preserved, one of the endpoints will be moved closed to the center (to make the radii the same) and then a connecting linear bridge is added to go to the original endpoint (in order to preserve electrical connectivity).

Example:

```
.ImportMoveArcCenter = True
```

Corresponding Command:

```
File/Import/Gerber/Options/File Interpretation/Inexact Arcs
```

See Also:

```
.Import
```

## ImportNetlist

Format:

```
.ImportNetlist( FileName, LayerOffset )
```

Import a netlist file and offset the layers by the specified amount.

Example:

```
.ImportNetlist( "C:\PentaLogix\Demo Bins\Demo.F04", 5 )
```

## ImportX

Format:

```
.ImportX( FileName, FileType, Layer, OriginX, OriginY )
```

Import different filetypes to the specified layer and origin. Filetype = Gerber, Drill, HPGL

Example:

```
.ImportX("1234561.plt", "Gerber", 84, 0, 0)
```

## ImportZeroLengthDraw

String property. Controls what happens to zero-length draws when importing Gerber files. Possible values:

| | |
|---|---|
| `Ignore` | Silently ignore |
| `Pads` | Convert the to pads (a pads will replace the zero-length trace) |
| `Ask` | Prompt the user to pick one of the above |

Example:

```
.ImportZeroLengthDraw = "Ignore"
```

Corresponding Command:

```
File/Import/Gerber/Options/File Interpretation/Zero-Length Draws
```

See Also:

```
.Import
```

## ImportZip

Format:

```
.ImportZip( Options )
```

Imports the files contained in a zipfile.

Arguments:

There are two formats for the *Options* argument.

1. Just the name of the zipfile to import
2. A comma separated list of items, which include (among others) the name of the zipfile

The second form is only available in software versions 9.5 and up. The possible items in the comma separated list are shown below:

| | | |
|---|---|---|
| Zip= | String | Name of zipfile to import |
| MinSizeX= | Double | Min X dimension when guessing a scale factor for file |
| MinSizeY= | Double | Min Y dimension when guessing a scale factor for file |
| MaxSizeX= | Double | Max X dimension when guessing a scale factor for file |
| MaxSizeY= | Double | Max Y dimension when guessing a scale factor for file |
| GerberUnits= | String | English or Metric |
| DrillUnits= | String | English or Metric |
| AptFilesUseSameUnits= | Boolean | If multiple aperture files are being imported, the user will only be prompted once for the unit size |
| ScaleGivenFiles= | Boolean | Scale unformatted files to formatted file size. Min, Max X & Y will be ignored. |
| BoardUnits= | String | Inch, mil, cm or mm |
| ScanForData= | Boolean | Compare file to know data file types i.e. Gerber, Excellon |
| ScanForApertures= | Boolean | Compare file to known aperture file types |
| ScanForNetlists= | Boolean | Compare file to known Netlist file types |
| ShowImportDialog= | Boolean | Display the scan results before importing the usable file(s) |

Note that all the items are optional except for the Zip= which must be present.

Example:

```
.ImportZip( "C:\PentaLogix\job\123456.zip" )
.ImportZip( "Zip=C:\m42.zip, GerberUnits=English,
   ScanForNetlists=True, ShowImportDialog=True" )
```

## ImportZeroLengthDraw

Format:

```
.ImportZeroLengthDraw = String
```

String property that sets an option for importing Gerber files. Determines what to do zero-length draws in Gerber files (draws where the two endpoints are the same). It can have two values:

| | |
|---|---|
| Ignore | Draw command is ignored |
| Pads | The user is prompted whether to ignore or generate a pad |

Example:

```
.ImportZeroLengthDraw = Ignore
```

Corresponding Command:

```
File/Import/Gerber/Options/File Interpretation/Zero-Length Draws
```

See Also:

```
Import
ImportIsolatedD01
```

## InsertArc

Format:

```
.InsertArc(FromCursor, x1, y1, x2, y2, xcenter, ycenter)
```

Insert an arc of the current D Code in the active layer(s).

Example:

A 0.1″ radius 360 degrees arc where cursor x and y are the centerpoint:

```
.InsertArc( False, .CursorX + .1, .CursorY, .CursorX + .1,
  .CursorY, .CursorX, .CursorY)
```

## InsertAsPaint

Format:

```
.InsertAsPaint = Boolean
```

Boolean property that specifies the polarity of newly inserted elememts (paint or scratch). They will be inserted as paint if the value of this property is `True`.

Example:

```
.InsertAsPaint = True
```

Corresponding Command:

```
Insert/Paint
Insert/Scratch
```

See Also:

```
.InserPad
.InsertTrace
.InsertArc
.InsertText
```

## InsertFrameSizedContour

Inserts a contour consisting of 4 traces, taking as model the current frame and using the current insertion settings.

Corresponding Command:

```
Insert/Frame-Sized/Contour
```

See Also:

```
.InserAsPaint
```

## InsertFrameSizedPolygon

Inserts a polygon taking as model the current frame and using the current insertion settings.

Corresponding Command:

```
Insert/Frame-Sized/Polygon
```

See Also:

```
.InserAsPaint
```

## InsertPad

Format:

```
.InsertPad( FromCursor, x, y )
```

Insert a pad of the current D Code at the specified location in the active layer(s).

Example:
```
.InsertPad( False, 3.74, .5906 )
```

## InsertText

Format:
```
.InsertText( Text, DCode, Select )
```

Insert a text string, using specified D Code. If *Select* is `True` then inserted text will be selected and can be further edited. Text is inserted at the current cursor location into the active layer(s).

Returns:

`True` if there were no errors during insertion.

Example:
```
.InsertText( "Text", 381, False )
```

See Also:

`.SetTextParameters`

`.UngroupSelectedText`

## InsertTrace

Format:
```
.InsertTrace( FromCursor, x1, y1, x2, y2 )
```

Insert a linear trace of the current D Code. If *FromCursor* is `False` then the X and Y coordinates are absolute, otherwise they are relative to the current cursor position. Traces are inserted into the active layer(s).

Example:
```
.InsertTrace( False, 0, 0, 1, 0 )
```
A 1-inch trace in x starting at zero.

See Also:

`.InserAsPaint`

## IntersectLayers

Format:
```
.IntersectLayers( TargetLayerNumber, SourceLayerNumber )
```

*TargetLayerNumber* will be replaced with polygons resulting from intersecting the "copper" of the two layers. Intersection is defined as the copper that is common to both layers.

Returns:

*False* if an error happened.

Corresponding Command:

`Edit/Intersect Current Layer`

## InvertSelection

Format:
```
.InvertSelection
```

Swap selected and unselected. Elements that were selected become unselected. And elements that were not selected become selected. Note that this operation applies only to visible elements (layers that are not visible, for instance, will be left unchanged).

## IsBusy

Boolean read-only property that is `True` when the application is executing an operation that can take longer time. While `IsBusy` is `True` one should not issue other script commands.

## JobName

String property that returns the name of the currently loaded job.

## LastItemCount

General purpose integer property to return info about the last method executed.
This is set by a limited number of methods and has different meanings for each of them.

## LastErrorNumber

Integer property for the number of the last error.
These are errors in the CAM software.
Setting it to a particular value is only useful when one wants to clear the error number (by setting the value to 0). Setting to other values will have no effect.
After an error has happened in the software, `.LastErrorNumber` will contain the number of the error, so to check whether an error happened during an operation, set `.LastErrorNumber` to 0, then execute the operation, then check for `.LastErrorNumber` being non-zero.

## LastWarningNumber

Integer property for the number of the last warning message.
Setting it to a particular value is only useful when one wants to clear it (by setting the value to 0). Setting to other values will have no effect.

After a warning message was issued, `.LastWarningNumber` will contain the number of the warning.

## LayerBoardNum

Format:
```
.LayerBoardNum( LayerNumber ) = Long
```

Set (get) the board layer number for the specified layer number.

Example:
```
.LayerBoardNum(2) = 1
```

## LayerColor

Format:
```
.LayerColor( LayerNumber ) = Long
```

Set (get) the layer color index for a give layer. CAMMaster has 15 layer color indices, 1 through 15, which correspond to a color which can be mapped via `.Color`.

Example:
```
.LayerColor(1) = 12
```
Set layer 1 to color number 12 ( usually bright red).

See Also:

`.Color`

## LayerCopy

Format:
```
.LayerCopy( FromLayer, ToLayer )
```

Copy the contents of layer *FromLayer* to layer *ToLayer*.

Example:
```
.LayerCopy(1, 2)
```

## LayerDelete

Format:
```
.LayerDelete( LayerNumber, What, LeaveName )
```

Delete the specified layer (or parts of it) from the job.
*What* can be one of the following: `All, Traces, Pads, Polygons.`
If *LeaveName* is True then the name of the layer will not be deleted.

Example:
```
.LayerDelete( 12, "All", False )
```

## LayerExtension

Format:
```
.LayerExtension( Layer )
```

Returns a string containing the layer extension (the part of the layer name past the dot) for the specified layer number.

Example:
```
X = .LayerExtension(8)
```
Get layer extension for layer 8.

## LayerFill

Set (get) layer number as "filled" in the F10 layer menu

Format:
```
.LayerFill(_LayerNumber) = _Long
```
Example:
```
.LayerFill(1) = False
```

## LayerMove

Move the contents of one layer to another.

Format:
```
.LayerMove(_FromLayer, _ToLayer)
```
Example:
```
.LayerMove(1, 2)
```

## LayerName

Set (get) the name of the specified layer

Format:
```
.LayerName(_LayerNumber) = _String
```
Example:
```
.LayerName(1) = "layername.gbr"
```

## LayerNegative

Set (get) whether the specified layer is negative.

Format:
```
.LayerNegative(_LayerNumber) = _Long
```
Example:
```
.LayerNegative(1) = False
```

## LayerScratch

Set (get) whether the specified layer is a scratch

Format:
```
.LayerScratch( LayerNumber ) = Long
```
Example:
```
.LayerScratch(1) = False
```

## LayersForInsert

A string property that describes the layers into which elements will be inserted during insert operations. This applies to insert operations done by the user, as insert operations in scripting always go into the active layers.

The valus for this property are either a comma-separated list of layer ranges or the string "Active" (for inserting into active layers).

Example:
```
.LayersForInsert = "1-3, 4, 12-14"
.LayersForInsert = "Active"
```
Corresponding Command:
```
Insert/Into Layers
```

## LayerSwap

Swap the contents of the specified layers.

Format:

```
.LayerSwap( Layer1, Layer2)
```

Example:

```
.LayerSwap(1, 2)
```

## LayerType

Set the layer type of the specified layer number.

Format:

```
.LayerType( LayerNumber ) = String
```

The string value can by any of the codes for layer types listed in F10 layer menu.

Example:

```
.LayerType(1) = "CPU"
```

## LayerVisible

Format:

```
.LayerVisible( LayerNumber ) = Long
```

Property to set (get) the visibility of the layer number.

Example:

```
.LayerVisible(1) = True
```

## ListDCodes

Format:

```
.ListDCodes( Which, LayerNumber )
```

A method to list D Codes currently used (mapped to elements in the database).

Returns:

Returns a comma-separated list of D Code ranges or "?" if there was an error. The D Codes will be listed whether they are visible or not. If no D Codes satisfy the criteria, then an empty string is returned.

Arguments:

*Which*: a string that specifies which D Codes will be listed. It can:
"*Any*" – any D Codes that are used
"*Selected*" – only currently D Codes for elements in the current selection
a one-letter string that denotes a D Code shape ("*C*", "*S*", "*R*", "*O*", "*X*", "*H*", "*T*", "*D*", "*B*", "*F*", "*M*", "*N*", "*E*", "*Q*" and "*?*"). Remember that stands "*?*"for unmapped.

*LayerNumber*: an integer that specifies the layer in which to look for the D Codes. If *LayerNumber* is 0, then the whole database is searched.

Example:

```
.ListDCodes( "C", 10 )
```

Will return a string with all D Codes in layer 10 that have shape "C". An example is "20-25, 44, 60-62".

```
.ListDCodes( "Any", 0 )
```

Will return a string with all D Codes which are being used in the database, in all layers.

## LloydDoyle

Format:

```
.LloydDoyle( Operation, Method, Options )
```

Settings for Lloyd Doyle AOI output

## LoadJob

Format:

```
.LoadJob( name )
```

Load the specified job (.bin file).

Example:

```
.LoadJob( "C:\PentaLogix\Demo Bins\Demo" )
```

## MacroFileExecuting

Format:

```
.MacroFileExecuting
```

This property works only in CAMMaster because it refers to the Sax Basic execution engine. If it returns `True` then CAMMaster is in the process of executing a macro using the Sax Basic engine, started either via the "`Macro/Run`" or "`Macro/Load And Run`" commands or via the `.StartMacroFileExecution` method.

When assigned:

- `True`: currently loaded macro (if any) will start executing
- `False`: if a macro is currently executing, it will be aborted

Example:

```
If .MacroFileExecuting = True Then
```

See Also:

`.StartMacroFileExecution`

## MakeOnlyVisibleLayers

Turn off all layers except those listed.

Format:
```
.MakeOnlyVisibleLayers( Layers )
```
Example:
```
.MakeOnlyVisibleLayers( "1, 3, 5" )
```

## MakeSelectedPolygonsWellFormed

Replaces any malformed polygons in the selection with well formed ones. Well formed polygons have no intersecting edges (except at vertices) and no overlapping edges.

Format:
```
.MakeSelectedPolygonsWellFormed
```
Returns:

Number of generated polygons or -1 if an error has occurred.

Corresponding Command:

```
Edit/Edit Selection/Polygons/Make Well Formed
```

SeeAlso:
```
.WellFormedMethod
```

## Marker

String property for the coordinates and state of the marker.
Values can be "`Off`" or a comma separated list of the X and Y coordinates.

Example:
```
.Marker = "Off"
.Marker = "2.1, 5.5"
```
Corresponding Command:

```
View/Mark
```

## MaxUndo

Format:
```
.MaxUndo = Double
```
Sets the undo memory in K bytes. Set to zero to turn off the undo mechanism. Note that while running scripts is it always a good idea to set this to 0 as the undo is only useful when user interaction is expected and it negatively affects performance.
Setting the max undo memory via a script only affects the value for the current CAMMaster session. After CAMMaster is closed and restarted, the last user set value is restored.

Example:
```
.MaxUndo = 10000
```
Sets undo memory to 10 Meg.

Corresponding Command:

```
Setup/Preferences/Max Undo Memory
```

## MergeColinearTraces

Format:

```
.MergeColinearTraces
```

All the linear traces in the selection are examined and every pair which overlaps each other and is colinear (part of the same line) will be replaced with a new trace that covers the maximum extent of the combination of the two.

Corresponding Command:

```
Edit/Edit Selection/Special/Merge Colinear Traces
```

## MessagesMode

String property to set/get the messages mode.

This controls how much user interaction there is (via dialogs, prompts etc.) as well as whether the view is refreshed during operations.

Possible settings are:

"Background"

"Batch"

"Interactive"

"Detailed".

The higher settings will cause more user interaction and more messages to be displayed. In addition, in "Background" mode the main view will not be refreshed (this can save a lot of time when running scripts during which the intermediate displays are irrelevant).

Corresponding Command:

```
Setup/Messages Mode
```

Example:

```
.MessagesMode = "Interactive"
```

## Milestone

Format:

```
.Milestone( Save )
```

Values for *Save* : `True` = save a milestone, `False` = restore last milestone.

Corresponding Command:

```
File/Milestone
```

Example:

```
.Milestone( True )
```

SeeAlso:

```
.SaveJobData
```

## Minimize

Minimize or restores the application window. Minimizing can be useful to avoid the delay of lengthy redraws sometimes caused by intermediate actions in the script.
Also note that when the CAM application is started via the script (no previously running instance exists), the application will be windowless until `.Minimize( False )` is called.

Format:
```
.Minimize( Yes )
```
Example:
```
.Minimize( True )
```
Minimizes window.
```
.Minimize( False )
```
Restores window.

## MirrorSelected

Mirror the selection in x or y, around cursor or center of selection.

Format:
```
.MirrorSelected( Horizontal, AroundCursor)
```
For *_AroundCursor*, false=center of selection.

Example:
```
.MirrorSelected(True, False)
```

## MoveActiveLayers

Move the visible layers the specified amount.

Format:
```
.MoveActiveLayers( DeltaX, DeltaY )
```
Example:
```
.MoveActiveLayers(1, 1)
```
Move the layers 1 inch in x and y.

## MoveCursor

Move the cursor position to the specified position.

Example:
```
.MoveCursor( 1.0, 1.0 )
```
Move the cursor 1 inch in x and y.

See Also:
```
.CursorX
.CursorY
```

## MoveCursorToCenterOfFramedData

Move the cursor position to the center of the framed data.

Example:
```
.MoveCursorToCenterOfFramedData
```

## MoveLayers

Move the listed layers the specified amount.

Format:
```
.MoveLayers(LayerList, DeltaX, DeltaY)
```
Example:
```
.MoveLayers("1,3", 3, 3)
```

## MoveSelected

Move the selection the specified amount.

Format:
```
.MoveSelected( DeltaX, DeltaY )
```
Example:
```
.MoveSelected(3, 3)
```

## MToCSelected

Format:
```
.MToCSelected
```
Converts M-type (custom) pads in the selection to pairs of C-type pads and a polygon. The polygon is the exact shape (copper) of the original M-type and the round pad is inscribed inside the polygon. This is useful for operations that cannot easily deal with complex pads, such as generating data for electrical test.

Corresponding Command:
```
Edit/Edit Selection/Special/Convert M-type Pads to C-type
```

## NameOfResumeFile

Format:
```
.NameOfResumeFile = String
```
Specify the pathname of the script file to run when the operator selects "Macro/Resume". This is useful when a script needs to be interrupted for the user to interact with the application. When ready, the user can click "Macro/Resume" and the script processing will resume (with the new file specified by this property).

Example:
```
.NameOfResumeFile = "D:\PentaLogix\Scripts\Resume.bas"
```

## NetCompare

Format:
```
.NetCompare( BoardLayers, Options, Tolerance )
```
Compares the netlist data in two sets of nets. Both need to be loaded before this operation. One can load a second set of nets (the reference, or REF netlist) by importing another .bin file (with net info) or a netlist file starting at an empty layer beyond the already loaded data.

Arguments:

| | | |
|---|---|---|
| *BoardLayers* | String | Comma separated list of board layer numbers that will be used in the comparison. One does not need to specify all board layers, but one normally would include the drill layer (or padmaster) |
| *Options* | String | Comma separated list of boolean assignments. The items correspond to the checkmark options in the NetCompare dialog. They are: FlagOnlyNut, UsePadCenter, MismatchedNCs, REFInconsistencies. |
| *Tolerance* | Double | Tolerance value for matching pad centers (relevant only when UsePadCenter is True) |

Returns:

The number of errors found or -1 if an error ocurred.

Example:

```
.NetCompare( "0,1,5", "UsePadCenter=True, REFInconsistencies=True",
  0.001 )
```

Corresponding Command:

```
Tools/Netlist/NetCompare/Compare Two Netlists"
```

## NewJob

Discard current job and all data.

Example:

```
.NewJob
```

## NextBoardNum

Format:

```
.NextBoardNum( BoardNum )
```

Return the first boardnum that is visible and loaded, starting at the specified boardnum. If there is no boardnum visible and loaded after the boardnum specified return –1.

Example:

```
.NextBoardNum( 1 )
```

## OnlyCurrentCustomProperty

Format:

```
.OnlyCurrentCustomProperty = Long
```

Affect only the elements that satisfy the current custom property or all elements.

Example:

```
.OnlyCurrentCustomProperty = True
```

Corresponding Command:

```
View/Toolbars/Custom Properties/Only
```

## OnlyCurrentDCode

Format:

```
.OnlyCurrentDCode = Long
```

Affect only the current D Code or all D Codes.

Example:

```
.OnlyCurrentDCode = True
```

## OnlyCurrentLayer

Format:

```
.OnlyCurrentLayer = Long
```

Affect only the current layer or all layers.

Example:

```
.OnlyCurrentLayer = True
```

## OnlyCurrentNet

Format:

```
.OnlyCurrentNet = Long
```

Affect only the current net, or all nets.

Example:

```
.OnlyCurrentNet = False
```

## OptimizePaintScratch

Format:

```
.OptimizePaintScratch
```

Works on selected elements and tries to optimize to paint-scratch-paint polarity changes. There are often many extraneous ones, meaning that several scratch or several paint blocks can be grouped together without altering the geometry of the data. Also, polygon holes are often imported as scratch polygons whereas the CAM database supports polygons with holes and so adding the scratch polygons as holes to corresponding paint polygons simplifies that data considerably. For instance, Gerber does not support polygons with holes, so the holes are always represented as scratch.

Returns:

The number changes or -1 if an error ocurred.

Corresponding Command:

```
Edit/Paint And Scratch/Optimize Selection
```

## OriginAtZero

Set origin at zero or at the cursor location for export (a Boolean value). Useful when writing out files with differing origins.

Example:
```
.OriginAtZero = False
```

## OutlinesSettings

A string property with settings for generating vector outlines. It is a comma separated list with one or more of the following values:

| | |
|---|---|
| TracesAlso | If "True", traces will also be outlined, otherwise only pads. |
| F | If "True", the ouline will consist of F trace contours, otherwise of polygons |
| MinSize | Dimension of smallest D Code that will be outlined. If 0, then all D Codes will be outlined. |
| XMargin | Swelling value on X axis |
| Ymargin | Swelling value on Y axis |

Example:
```
.OutlinesSettings="TracesAlso=True, MinSize=0, XMargin=0, YMargin=0,
  F=True"
```

Corresponding Command:

```
Tools/Generate Polygon Outlines/Vector
```

See Also:

```
.GenerateOutlines
```

## PadsVisible

Boolean property that turns pad visibility on or off.

Example:
```
.PadsVisible = True
```

## PlotterResolution

Set the plotter resolution for photoplotting. This value is saved in .bin files and is used by Lavenir photoplotters. It can also be exported in Gerber or MDA files that are plotted on other types of plotters.

Format:
```
.PlotterResolution = String
```

Example:
```
.PlotterResolution = "1000 dpi"
```

## PointIsInsideSelection

Format:
```
.PointIsInsideSelection( X, Y )
```

Returns:

True if point (*X, Y*) is inside any of the elements in the current selection.
Note that only visible elements in the selection are considered and that if the point is inside a

hole of a polygon it will be considered outside that polygon. “Inside” means on top of copper area.

Example:

```
If .PointIsInsideSelection( 0.145, 12.44 ) Then MsgBox "Inside"
```

## PolarCoordinatesDisplay

Format:

```
.PolarCoordinatesDisplay = True/False
```

Sets the display to polar (distance and angle), use with GetCursorPosition

## PolarityCombineAll

Looks for adjacent layers that belong to the same board layer (and are visible) and combines them into one layer. The operation is then repeated for all the layers in the job. The result may contain layers with multiple polarity (both paint and scratch elements). Layers which are empty are elimintaed unless they are named (have a layer name).

Corresponding Command:

```
Edit/Paint And Scratch/Combine All
```

See Also:

```
.PolaritySeparateAll
```

## PolarityCombineAtLayer

Format:

```
.PolarityCombineAtLayer( LayerNumber )
```

Layers around `LayerNumber` which have the same board layer number as `LayerNumber` will be combined into one layer (of possibly mixed polarity).

Returns:

The number of layers combined, or -1 on error

Corresponding Command:

```
Edit/Paint And Scratch/Combine Layers
```

## PolaritySeparateAfterImport

A Boolean property that controls how files are imported if they contain several polarities (paint and scratch). If `True`, it will cause a `.PolaritySeparateAll` to be done after every import operation. This is useful for scripts which assume that layers are single polarity only (this used to be the case with older versions of the software). Such scripts should set this property to `True` at the beginning of the script (it is `False` by default).

The commands that are affected by this are: `.Import, .ImportX, .ImportJob, .ImportZip, .ImportDirectory, .ImportFromLayers`.

See Also:

```
.PolaritySeparateAll
.PolarityCombineAll
```

## PolaritySeparateAll

All layers that have both paint and scratch elements will be separated into an equivalent sequence of layers which contain only one polarity.

### Corresponding Command:

```
Edit/Paint And Scratch/Separate All
```

## PolaritySeparateLayer

### Format:

```
.PolaritySeparateLayer( LayerNumber )
```

The layer `LayerNumber` will be separated into an equivalent sequence of layers which contain only one polarity.

### Returns:

The number of resulting layers (including the original), or -1 on error.

### Corresponding Command:

```
Edit/Paint And Scratch/Separate Current Layer
```

## PolygonHoles

### Format:

```
.PolygonsHoles( Group )
```

This command works on the selected polygons and either groups them or ungroups them, depending on the boolean argument `Group`. The selected polygons will be replaced with the result of the operation.
Ungrouping: only applies to polygons that have holes. The holes will be removed from the containing polygons and inserted as separate polygons (containers).
Grouping: Polygons that are included in other polygons will be made holes of those polygons.

### Returns:

The number of resulting polygons.

### Example:

```
.PolygonHoles( True )
```

### Corresponding Command:

```
Edit/Edit Selection/Polygons/Group Holes
Edit/Edit Selection/Polygons/Separate Holes
```

## PolygonsToF

### Format:

```
.PolygonsToF( DCode )
```

Replaces polygons in the selection with traces that follow the outlines of the polygons. The traces will be mapped to `DCode` and set to shape "F".

### Returns:

The number of traces generated.

Example:
```
.PolygonsToF( 3000 )
```
Corresponding Command:
```
Edit/Edit Selection/Polygons/Polygons To F
```
See Also:
```
.FToPolygons
```

## PolygonsToStrips

Format:
```
.PolygonsToStrips
```
Replaces polygons in the selection with polygons without holes. Only polygons which originally had holes will be changed. They will be divided into strips that do not have holes but which cover the same copper area as the original polyongs. This is useful for cases when holes are not desirable.

Returns:

The number of strips generated, -1 on error.

Corresponding Command:
```
Edit/Edit Selection/Polygons/Decompose Into Strips
```

## PolygonsVisible

Turn polygon visibility on or off.

Example:
```
.PolygonsVisible = True
```

## PrintToBMPFile

Format:
```
.PrintToBMPFile( FileName, KindOfPrint, BoardLayers, Colors,
   Resolution )
```
Generate a .bmp file of the current view or of the elements inside the Reference Frame.

Arguments:

| | | |
|---|---|---|
| *FileName* | String | Name of bitmap file (or prefix for "by board layer" operation) |
| *KindOfPrint* | String | "View", "ViewBL", "Frame", "FrameBL". The BL variants print "by board layer" |
| *BoardLayers* | String | Comma separated list of board layers or board layer ranges |
| *Colors* | String | "Black on White", "White on Black" or "Screen Colors" |
| *Resolution* | Double | Dots per inch (English) or dots per cm (metric) |

Example:
```
.PrintToBMPFile( "D:\tmp\f2", "FrameBL", "1", "White on Black", 300 )
```
See Also:
```
.CreateRaster
.RenderBitmap
```

## RasterResolution

Property of type Double for the resolution of "raster" type tools in mils (English) or 0.01mm (metric).

Example:

```
.RasterResolution = 1
```

## REFStartLayer

Property of type Integer for specifying where the reference layers start for a netcompare operation ("Tools/Netlist/NetCompare/Compare Two Netlists").

Example:

```
.REFStartLayer = 15
```

## RemovePads

Format:

```
.RemovePads( ClearDestLayer, InFrameOnly, CurrentDCOnly, ByDeleting,
   Unused, Stacked, CenterOffset, GapValue, CenterOffsetValue)
```

Remove pads function and associated options.

Example:

```
.RemovePads( False, False, False, False, True, True, True, 0.003,
   0.003 )
```

## ReorderLayers

Format:

```
.ReorderLayers( Method )
```

The currently loaded layers will be reordered according to *Method*. Currently only value 0 is implemented for *Method* , meaning by board layer, which will reorder the layers so that the board layer numbers are in ascending order.

Corresponding Command:

`Order layers by B Layer` in the context menu for the Layers Setup (F10) dialog.

## RenderBitmap

Format:

```
.RenderBitmap( center_x, center_y, dots_per_inch, pixel_aspect_ratio,
   width, height, board_layer_list, flags, FileMap, ProcessId )
```

This method has no menu equivalent. It is provided as a tool for generating a memory resident bitmap of one bit-per-pixel of the specified view.

### Arguments:

| | | |
|---|---|---|
| *center_x, center_y* | Double | Center of bitmap in user coordinates. |
| *width, height* | Long | Width and height of bitmap in pixels. *width* must be a multiple of 16. The bitmap has the layout of a standard Windows bitmap, so that (0,0) in the bitmap is the upper-left corner of the image. |
| *dots_per_inch* | Double | Pixels per inch (if English units are active) or pixels per cm (if metric). |
| *board_layer_list* | String | Comma separated list of board layer numbers, for example "2,4" will render board layers 2 and 4. |
| *FileMap, ProcessId* | Long | Used to specify the memory area for the bitmap. It must be allocated and freed by the calling process. |
| *pixel_aspect_ratio* | Double | Unused. |
| *flags* | Long | Unused. Must be set to 0. |

### Returns:

The number of bytes per row in the bitmap, or 0 if the call failed for any reason.

The order of bits in the returned bitmap corresponds to the standard Windows bitmap order, which is as follows:

Vertical: The bottom row of the image is the top row of the bitmap. In other words the bitmap is mirrored on Y.

Horizontal: If we are numbering the bits in a bitmap byte from 0 to 7 with 0 being the low order bit, then the image, going left to right is 7, 6, 5, 4, 3, 2, 1, 0.

The only safe way to share memory between two or more processes is via "file mapping". This is done with the parameters *FileMap* and *ProcessId*. The Windows library functions `CreateFileMapping`, `MapViewOfFile` and `getCurrentProcessId` should be used to set up these variables prior to the call to RenderBitmap. If more than one board layer is specified in the *board_layer_list*, then several bitmaps will be returned, one per board layer. They will follow each other in the memory area provided, so make sure that you allocate enough room for all.

### Example:

The calling process should include code similar to the sample below:

```
Const INVALID_HANDLE_VALUE = -1
Const PAGE_READWRITE = 4
Const FILE_MAP_ALL_ACCESS = &HF001F

Declare Function  CreateFileMapping Lib "kernel32" Alias
      "CreateFileMappingA" ( ByVal hFile As Long, ByVal
      lpFileMappigAttributes As Long, ByVal flProtect As Long, ByVal
      dwMaximumSizeHigh As Long, ByVal dwMaximumSizeLow As Long, ByVal
      lpName As Long) As Long
Declare Function  MapViewOfFile Lib "kernel32" (ByVal
      hFileMappingObject As Long, ByVal dwDesiredAccess As Long, ByVal
      dwFileOffsetHigh As Long, ByVal dwFileOffsetLow As Long, ByVal
      dwNumberOfBytesToMap As Long) As Long
Declare Function  GetCurrentProcessId Lib "kernel32" () As Long
```

```
Declare Function  CloseHandle Lib "kernel32" (ByVal hObject As Long) As
      Long
Declare Function  UnmapViewOfFile Lib "kernel32" (lpBaseAddress As Any)
      As Long

Dim hFileMap As Long, pViewFileMap As Long
Dim lSize As Long, width As Long, height As Long
width = 700       'Example
height = 500
'Bitmap size in bytes rounding up for WORD alignment
lSize = ((width + 15) / 16) * 2 * height
hFileMap = CreateFileMapping( INVALID_HANDLE_VALUE, 0,  PAGE_READWRITE,
      0, lSize, 0 )
lpViewFileMap = MapViewOfFile( hFileMap, FILE_MAP_ALL_ACCESS, 0, 0,
      lSize )
.RenderBitmap( 1.5, 1.0, 1000.0, 0, width, height, "1", 0, hFileMap,
      GetCurrentProcessId() )
'Now lpViewFileMap has the bitmap and can be used in CreateBitmap or
      any other way
UnmapViewOfFile( lpViewFileMap )
CloseHandle( hFileMap )
```

See Also:

`.CreateRaster`

`.PrintToBMPFile`

## ReplaceSelectedPolygonsWithOverlaps

Format:

```
.ReplaceSelectedPolygonsWithOverlaps
```

This operation takes every pair of polygons in the selection and replaces them with their overlap (or intersection), which is the copper area common to both polygons.
Note that if two polygons do not overlap, there will be nothing to replace them with, so polygons in the selection which do not overlap any polygons will be deleted.

Corresponding Command:

`Edit/Edit Selection/Polygons/Replace With Overlaps`

Availability:

CAMMaster only.

## ReplaceSelectionWithOutlines

Format:

```
.ReplaceSelectionWithOutlines( PerBoardLayer )
```

All elements in the selection are replaced with polygons and then the polygons are merged so that overlapping polygons are combined. Scratch polygons will erase from the result. If `PerBoardLayer` is `True` then the operation is done one board layer at a time.

Example:

```
.ReplaceSelectionWithOutlines( True )
```

Corresponding Command:

`Edit/Edit Selection/Replace With Merged Polygons`

`Edit/Edit Selection/…Per Board Layer`

See Also:

```
.ConvertSelectionToPolygons
```

## ReplicateSelected

Format:

```
.ReplicateSelected( LayerNumber )
```

Replicate the selection to the specified layer.

Example:

```
.ReplicateSelected( 3 )
```

## ReportFileName

Format:

```
.ReportFileName = String
```

Set the file name for the report when it is generated.

Example:

```
.ReportFileName = "C:\Jobs\100586\Report.txt"
```

## RevertToSaved

Revert to the last saved file

Corresponding Command:

```
File/Revert To Saved
```

## RoundDCodeSizes

Format:

```
.RoundDCodeSizes( ToValue, DCodes )
```

Rounds the dimensions of D Codes to a specified fraction.

Arguments:

| | | |
|---|---|---|
| *ToValue* | Double | Value to round to in inches or cm (depending on whether CAMMaster is in English or metric mode). One can enter any value, but most useful are decimal fractions. So, for instace, in English mode a value of 0.001 would cause rounding to the closest mil ( 1 mil = 0.001 inches). |
| *DCodes* | String | Comma-separated list of D Code ranges. These are the D Codes to round |

Returns:

The number of D Codes which were rounded, -1 on error.

Example:

```
.RoundDCodeSizes( 0.001, "15-18, 200-244, 300" )
```

Corresponding Command:

```
Setup/D Codes/Operations/Round to closest…
```

## RoutingArcsToChords

Format:

```
.RoutingArcsToChords( Method, Length, Angle )
```

Changes the routing paths in the selection so that arcs (if any) are converted to straight chords that approximate the arcs, using the parameters given as arguments.

Arguments:

| | | |
|---|---|---|
| *Method* | String | One of: "Length", "Angle", "Bounded angle"<br>If Length uses chord length only (Length argument)<br>If Angle uses chord angle only (Angle argument)<br>If Bounded angle uses chord angle (Angle), but does not generate chords shorter than Length. |
| *Length* | Double | Length of chord |
| *Angle* | Double | Angle of chord |

Returns:

The number of changed paths or -1 if error.

Example:

```
.RoutingArcsToChords( "Length", 0.004, 0.0 )
```

Corresponding Command:

```
Tools/Routing/Arcs To Chords
```

## RoutingChamfers

Format:

```
.RoutingChamfers( Add )
```

Changes selected routing paths by adding rounded chamfers (or removing them) . *Add* is a boolean argument. It can be `True` for adding chamfers and `False` for removing chamfers.

Returns:

The number of changed paths or -1 if error.

Example:

```
.RoutingChamfers( True )
```

Corresponding Command:

```
Tools/Routing/Arc Chamfers
```

See Also:

```
.RoutingChamgersRadius
```

## RoutingChamfersRadius

A property of type double for the radius of the chamfers used in the `RoutingChamfers` method.

Corresponding Command:

```
Tools/Routing/Arc Chamfers
```

See Also:

```
.RoutingChamfers
```

## RoutingChordsToArcs

Format:

```
.RoutingChordsToArcs( Tolerance )
```

Changes selected routing paths by replacing sequences of chords (straight edges) which approximate arcs with arc edges. *Tolerance* is used to determine which straights can be approximated.

Returns:

The number of changed paths or -1 if error.

Example:

```
.RoutingChordsToArcs( 0.001 )
```

Corresponding Command:

```
Tools/Routing/ChordsToArcs
```

## RoutingConnectorArcs

Format:

```
.RoutingConnectorArcs = Boolean
```

Boolean property that specifies how routing paths compensation is done at corners. When `True`, arcs will be inserted at corners at sharp angles.

Example:

```
.RoutingConnectorArcs = True
```

Corresponding Command:

```
Tools/Routing/Connector Arcs
```

## RoutingGroupToPaths

Finds routing paths in the selection and replaces those elements with the resulting paths.

- For traces: Finds connected traces and build paths from them. Traces are connected if they touch at endpoints (when expanded by their D Code radius) or if they intersect. If the traces are mapped to N type D Codes, the tool number of that D Code is assigned to the path, otherwise a new tool number is picked and its size set to the size of the D Code (or one of the D Codes if there are multiple D Codes in the path).
- For polygons: each polygon will generate one or more paths, for the container of the polygon and its holes (if any).

Returns:

`False` if an error ocurred.

Corresponding Command:

```
Tools/Routing/Group To Paths
```

See Also:

```
.RoutingUngroupToTraces
```

## RoutingUngroupToTraces

Replaces the routing paths in the selection with equivalent traces.

Returns:

The number of traces replaced or -1 if an error occurred.

Corresponding Command:

```
Tools/Routing/Ungroup To Traces
```

See Also:

```
.RoutingGroupToPaths
```

## RotateSelected

Format:

```
.RotateSelected( Angle, AroundCursor )
```

Rotate the selection, at the specified angle, around the cursor or selection center.

Example:

```
.RotateSelected( 90, False )
```

## SaveByLayer

Save .bin files of each visible layer using the basefilename and (L0#).

Example:

```
.SaveByLayer("Binname")
```

Saves: Binname(L01).bin, Binname(L02).bin, etc.

## SaveCompareLayersErrors

Format:

```
.SaveCompareLayersErrors( FileName )
```

Saves a text file containing the coordinates of the errors found in the last layer comparison operation.

Returns:

`False` if the operation fails or if there has not been a layer comparison operation performed beforehand.

Corresponding Command:

```
Tools/Compare Layers/Save Differences To File
```

See Also:

```
.CompareTwoLayers
.CompareTwoLayersEx
```

## SaveJob

Format:

```
.SaveJob( FileName )
```

Saves the currently loaded job under the specified name.

Returns:

`False` if the operation fails.

Example:
```
.SaveJob("C:\Data\Savedjob.bin")
```
Corresponding Command:

File/Save As

SeeAlso:
```
.SaveJobData
```

## SaveJobData

Format:
```
.SaveJobData( FileName )
```
Saves that job data under the specified *FileName* but leaves the current document name unchanged. Useful for saving backup data.

Returns:

`False` if the operation fails.

SeeAlso:
```
.SaveJob, .Milestone
```

## ScaleSelected

Format:
```
.ScaleSelected( ScaleX, ScaleY, DCodesAlso, Around )
```
Scale the selection in X and Y by given scale factor, D Codes also (or not). *Around* can have the following values:

| | |
|---|---|
| *0* | Scale around center of selection |
| *1* | Scale around current cursor position |
| *2* | Scale around origin (0, 0) |

Example:
```
.ScaleSelected( 1.0003, 1.0002, False, 2 )
```
Corresponding Command:

Edit/Edit Selection/Scale

## Select

Format:
```
.Select( Mode, xmin, ymin, xmax, ymax )
```
Select the available entities within the specified area.

*Mode* is "Add", "Remove", or "New". As for all other simialr selection operations, "New" will create a new selected, while "Add" will add to an existing selection and "Remove" will remove the elements chosen from the existing selection. Note that this method will not select step and repeated elements if they are "blocked".

Example:
```
.Select( "New", 0, 0, 18, 24)
```
See Also:

.SelectGlobalEx

## SelectAll

### Format:

```
.SelectAll
```

Select all of the entities, including step and repeated block entities.

## SelectArc

### Format:

```
.SelectArc( Mode, x1, y1, x2, y2, xcenter, ycenter )
```

*Mode* is "Add", "Remove", or "New".

Selects or deselects (depending on Mode) arc specified by absolute endpoints and center. The arc is counterclockwise from the first endpoint to the second. The usual selection criteria are applied.

### Returns:

FALSE if the operation fails.

### Corresponding Command:

There is no corresponding command.

## SelectByIntersection

### Format:

```
.SelectByIntersection( RefLayer, DestLayer, OpsFlags, Margin )
```

This method narrows an existing selection depending on the intersection of elements in the selection. The selected elements in a reference layer will be checked against the selected elements in a destination layer. Only those elements in the destination layer selection that satisfy the intersection criteria are preserved in the selection (all other elements are removed from the selection, even from other layers). For the purposes of computing the intersection, the elements in the destination layer can be inflated/deflated by a given margin.
The destination layer and the reference layer can be the same in which case the selection will be tested against itself (all elements in the selection versus all others).
Elements that have zero width either as they are in the database or after being deflated by the margin will be ignored during this operation.

Arguments:

*RefLayer*: reference layer number

*DestLayer*: destination layer number

*OpsFlags*: bitmask used to define the intersection criteria and various options. Bit values are as follows (D is the destination layer and R is the reference layer):

| *Bit #* | *Mask (Hex)* | *Selects elements in D…* |
|---|---|---|
| 0 | 1 | …that completely cover at least one element in R |
| 1 | 2 | …that are completely covered by at least one element in R |
| 2 | 4 | …that intersect at least one element in R such that it does not completely cover that element or is completely covered by that element (partial intersection) |
| 3 | 8 | …that are completely outside all elements in R (no intersections) |
| 8 | 100 | Restrict checking to elements of the same polarity (paint/scratch) and the same level in the paint/scratch order. If this bit is off, all elements are checked, regardless of polarity level |

*Margin*: destination layer elements are computed as if inflated or deflated by this margin

Returns:

The number of elements found (selected) or -1 if an error occurred.

Example:

```
.SelectByIntersection( 1, 3, 1+4, 0 )
```

This will keep in the selection in layer 3 only the elements that either completely cover at least one element in the selection in layer 1 or partially intersect at least one element in the selection in layer 1. The margin for the operation is 0.

Corresponding Command:

```
Select/By Intersection
```

## SelectCoveredElements

Format:

```
.SelectCoveredElements
```

Selects active elements (in active layers, visible, etc.) that are completely covered by other elements. If such elements are deleted, the resulting copper (image) remains the same.

Returns:

The number of elements selected or -1 if an error occurred.

Corresponding Command:

```
Select/Covered Active Elements
```

## SelectDuplicateElements

Format:

```
.SelectDuplciateElements
```

Selects active elements (in active layers, visible, etc.) that are duplicates to other elements in the database. If there is a group of `n` duplicate elements, `n - 1` will be selected. If such elements are deleted, the resulting copper (image) remains the same.

Returns:

The number of elements selected or -1 if an error occurred.

Corresponding Command:

```
Select/Duplicate Active Elements
```

## SelectedArcsToQuadrants

Format:

```
.SelectedArcsToQuadrants
```

The arcs in the selection that span across quadrant boundaries will be replaced with equivalent quadrant arcs.

Returns:

The number of arcs that were converted or -1 if an error occurred.

Corresponding Command:

```
Edit/Edit Selection/Arcs/Break Into Quadrants
```

## SelectChain

Format:

```
.SelectChain( Mode, Branching )
```

Select a chain of traces, branching (or not). *Mode* is "Add", "Remove", or "New".

Example:

```
.SelectChain("New", False)
```

## SelectClosest

Format:

```
.SelectClosest( Mode )
```

Select the nearest entity to the cursor position. Note that only elements that are currently visible in the current view are considered (this is similar to how one selects manually when double-clicking).

Arguments:

*Mode* is "Add", "Remove", or "New".

Example:

```
.SelectClosest("Add")
```

See Also:

```
.SelectClosestEx
.SelectClosestAndInfo
```

## SelectClosestAndInfo

Format:

```
.SelectClosestAndInfo( Mode )
```

Select the nearest entity to the cursor position. Note that only elements that are currently visible in the current view are considered (this is similar to how one selects manually when double-clicking).

Arguments:

*Mode* is "Add", "Remove", or "New".

Returns:

A string with information about the selected element or "?" if nothing was selected or the selection is not a primitive element (it may be text, a step-and-repeat block etc). The string is a comma-separated list and a description can be found in Appendix A (PentaLogix Geometry Files). The format follows the syntax of these files.

Example:

```
.SelectClosestAndInfo("New")
```

See Also:

`.SelectClosest`

## SelectClosestEx

Format:

```
.SelectClosest( Mode, X, Y, AtEndpointOnly )
```

Similar to `SelectClosestAndInfo`, but selects the nearest to location (*`X, Y`*) rather than the cursor. Is is also different because it considers all elements, not only those present in the current view. However, the selection is still done only for visible elements (visible layers etc.).

Arguments:

*`Mode`* is `"Add", "Remove", or "New"`.
*`AtEndpointOnly`* is a boolean that works similar to the property `.AlignToEndpointsOnly` (which is used by `SelectClosestAndInfo`).

Returns:

A string with information about the selected element or "?" if nothing was selected or the selection is not a primitive element (it may be text, a step-and-repeat block etc). The string is a comma-separated list and a description can be found in Appendix A (PentaLogix Geometry Files). The format follows the syntax of these files.

Example:

```
.SelectClosestEx( "New", 1.2, 4.2, False )
```

See Also:

`.SelectClosestAndInfo`

## SelectedText

`String` property.
When read returns the text content of the selected text blocks. If there are no text blocks in the selection then the empty string is returned. If there are several text blocks in the selection with different text content, then the string "`<Several>`" is returned.

Assigning a value to this property will modify the text part of the selected text blocks.

Corresponding Command:

`Edit/Edit Selection/Text/Change`

Example:

```
.SelectedText = "New text"
```

## SelectIntersectingTraces

Format:

```
.SelectIntersectingTraces
```

Selects active traces that intersect itself at points other than their endpoints.

Returns:

The number of traces selected or -1 if an error happened.

Corresponding Command:

```
Select/Intersecting Traces
```

## SelectTouching

Format:

```
.SelectTouching
```

Select .elements that touch starting near the cursor. In one layer only. Visible elements only.

Corresponding Command:

```
Select/Connected (touching)
```

## SelectUnusedPads

Format:

```
.SelectUnusedPads
```

Unused pads are those pads that are electrically connected only to other pads (or are completely isolated). Used pads are those that are electrically connected to traces, polygons or to negative copper areas. This method will find all unused pads in the selection and then will leave only these pads selected (the rest of the original selection is unselected). If there are no such pads then the selection will be empty.
You need to select all the elements (both pads and non-pads) that you want considered. In addition, as for all other operations, only visible parts of the selection are processed. Within the selection, the processing is done one board layer at a time, with proper consideration for pain-and-scratch as well as negative layers.

Corresponding Command:

```
Tools/Delete Pads/Select Unused Pads
```

See Also:

```
.RemovePads
```

## SelectedRoutingPaths

`String` property in the form of a comma-separated list.
When read, returns the properties of the routings paths in the selection or an empty string if there are no routing paths in the selection.
When assigned, it will set the properties of the routing paths to match the ones in the comma-separated list. Items that do not need to be changed can be omitted from the list. The possible items are explained in the table below.

| | |
|---|---|
| `Tool` | Tool number(s) in the selection. This could be a comma-separated list itself and when it is, it will be enclosed in parentheses. When used in an assignment, only one tool number can be appear. |
| `Precedence` | Precedence(s) in selection. See observations for `Tool` above. |
| `Orientation` | One of: `<Several>`, `ClockWise`, `CounterClockWise` or `Open`. When used in an assignment only `ClockWise` and `CounterClockWise` can be used. |
| `Compensation` | One of: `<Several>`, `None`, `Left` or `Right`. When used in an assignment `<Several>` cannot be used. |
| `Count` | The count of routing paths in the selection. This can of course not be used in an assignment, it is for reading only. |

Corresponding Command:

```
Tools/Routing/Selection Properties
```

Example:

```
.SelectedRoutingPaths = "Tool=2, Precedence=200"
```

## SelectedVisible

Boolean property for the visibility of selected elements.

Example:

```
.SelectedVisible = True
```

Corresponding Command:

```
View/Visibility/Element/Selected
```

See Also:

```
.UnselectedVisible
```

## SelectGlobal

Select elements depending on the selection criteria. Same as `.SelectGlobalEx( "New" )`. Note that this method ignores step and repeated elements if they are "blocked".

Corresponding Command:

```
Select/Global/New
```

## SelectGlobalEx

Format:

```
.SelectGlobalEx( Mode )
```

*Mode* is `"Add"`, `"Remove"`, or `"New"`.

Select elements depending on the selection criteria, with selection modes. Note that this method ignores step and repeated elements if they are "blocked".

Corresponding Command:

```
Select/Global
```

Example:
```
.SelectGlobalEx( "Add" )
```
See Also:

.Select

.SelectGlobal

## SelectionCriterion

Format:
```
.SelectionCriterion = String
```
Set or get the selection criteria. The *String* can be any of the values found in the dropbox in the Selection Toolbar:
`Any, Pads, Traces, Polygons, Text, Routing Paths, Scratch Elements, Orthogonals, Verticals, Horizontals, 45-Degree, Non-Orthogonals, Shape C, Shape S, Shape R, Shape O, Shape Q, Shape D, Shape H, Shape T, Shape E, Shape N, Shape M, Shape F, Shape ?, Invalid Polygons.`

Example:
```
.SelectionCriterion = "Any"
```
Corresponding Command:

Selection Toolbar: "Additional Selection Criterion" dropbox.

See Also:

.OnlyCurrentNet

.OnlyCurrentLayer

.OnlyCurrentCustomProperty

.OnlyCurrentDcode

.CurrentDCode

.CurrentNet

.CurrentLayer

.CurrentCustomProperty

## SelectionCrossing

Boolean property for how selection behaves for traces and polygons that cross the selection frame.

Example:
```
.SelectionCrossing = True
```
Corresponding Command:

`Select/Options: "Select also elements that cross the frame"`

## SelectDisplayMode

`Integer` property that controls how selected elements are displayed.
It corresponds to the items in "View/Visibility/Selected". The values are:

0 In white
1 In contrasting color
2 By dimming unselected
3 Display unselected in gray

Corresponding Command:

```
View/Visibility/Selected
```

## SelectionInside

Boolean property for selection inside the frame (or outside).

Example:

```
.SelectionInside = True
```

Corresponding Command:

```
Select/Options: "Inside Frame" and "Outside Frame"
```

## SelectionProperties

Returns a string that describes the selection properties. This is the string one sees when doing "View/Selection/Properties". Note that the format of the string may change in future versions of the software, so it is not adviseable to parse it in a script.

Example:

```
x = .SelectionProperties
```

Returns in x:

```
9558 elements selected and visible
   (9558 traces (0 arcs), 0 polygons, 0 pads)
   Layer: 1 Board layer: 1
   D Code: 107   ( C 0.0045 )
   Net: ?
   Limits: (1.37225, 1.04425) (7.76305, 6.07075)
   Size: 6.3908 by 5.0265
```

Corresponding Command:

```
View/Selection/Properties
```

See Also:

```
.SelectClosestAndInfo
```

## SelectLayer

Format:

```
.SelectLayer( Mode, LayerNumber )
```

Select an entire layer using selection criteria, except for D Code and layer (since the layer is specified in `LayerNumber`).

Arguments:

*Mode* is "Add", "Remove", or "New".

Example:

```
.SelectLayer( "Add", 3 )
```

Adds to the existing selection all elements in layer 3 that match the other selection criteria, except D Code and layer.

See Also:

```
.SelectionCriterion
.OnlyCurrentNet
.OnlyCurrentCustomProperty
.SelectLayers
```

## SelectLayers

Format:

```
.SelectLayers( ListOfLayers )
```

Select all the elements in the layers specified in the list. This will become a new selection (previously selected elements will be deselected before this operation).

Arguments:

*ListOfLayers* is a comma-separated list of layer ranges.

Example:

```
.SelectLayers( "2-4, 10, 14, 20-25" )
.SelectLayers( "5" )
```

Corresponding Command:

```
Select/Layers
```

See Also:

```
.SelectLayer
```

## SelectStepAndRepeat

Format:

```
.SelectStepAndRepeat( Id )
```

Selects steps and repeat blocks of the requested *Id* . If *Id* is -1, then the first ID found in the current active layer(s) will be selected.

Arguments:

*Id* The ID number of the step and repeat block, or -1 to select first found in active layers

Returns:

A string that describes the selected step and repeat block(s). There could be several blocks because they can span several layers.

The format of the string is:

<step and repeat info> Layers=<layer info> Bbox=<bounding box corners>

If there were no step and repeat blocks selected a question mark is returned ("?")

A sample string is shown below:

```
2 X 1, 24.6, 26.8, ID=5 Layers=1-3,5-7 Bbox=(0.95, 0.95)(20.55 25.75)
```

Example:

```
.SelectStepAndRepeat( -1 )
.SelectStepAndRepeat( 125 )
```

See Also:

```
.GetStepAndRepeatIDs
.StepAndRepeatSelected
.StepAndRepeatChange
.StepAndRepeatUngroup
```

## SelectTrace

Format:

```
.SelectTrace( Mode, x1, y1, x2, y2 )
```

*Mode* is "Add", "Remove", or "New".

Selects or deselects (depending on Mode) a linear trace specified by absolute endpoints. The usual selection criteria are applied.

Returns:

FALSE if the operation fails.

Corresponding Command:

There is no corresponding command.

## SetAbsoluteOrigin

Format:

```
.SetAbsoluteOrigin( x, y )
```

Change the absolute zero to a new location.

Example:

```
.SetAbsoluteOrigin( 1, 1 )
```

## SetSaveOptions

Format:

```
.SetSaveOptions( Options )
```

Set the options for saving the .bin file (CAM job file). Options is a comma separated list of options. They are shown in the table below:

| *Option Name* | *Meaning* |
|---|---|
| Version | Desired jobfile version number |
| Step And Repeat Blocks | True: Step and repat blocks will be store in concise form, otherwise an implicit "ungroup" will be done while saving and each block will be repeated as needed |

Example:

```
.SetSaveOptions( "Step And Repeat Blocks=True" )
```

Corresponding Command:

File/Save Options/Preferences

See Also:

.SaveJob

## SetTextParameters

Format:

```
.SetTextParameters( Parameters )
```

Set the parameters for the Insert/Text, Insert/Location or Insert/Dimension operations.

### Arguments:

*Parameters* is a string containing a comma separated list of parameter assignments. Only parameters that need to be changed must be included (the others will preserve their previous values).
The parameters are shown in the table below:

| Parameter | Meaning |
|---|---|
| `Font` | Name of the font: `Roman, Italic, Gothic, DMPL, Uppercase, Symbol, ODB, Custom, 4X7, 5X7` |
| `Height` | The height of a character cell. This determines the size of the text since the width is derived from height. |
| `CharSpacing` | Extra character spacing. Default is 0. |
| `LineSpacing` | Extra line spacing. Default is 0. |
| `Angle` | Rotation angle of the text. |
| `Mirrored` | If `True`, then text is mirrored on X. |
| `ExactDigits` | Exact number of digits to the right of decimal for Location and Dimension operations |
| `MaxDigits` | Maximum number of digits to the right of decimal for Location and Dimension operations. Note: use either `MaxDigits` or `ExactDigits` but not both. |
| `ExtendBars` | If `True`, then the Dimension bars will be extended |
| `HorOrVert` | If `True`, then Dimension bars will be forced horizontal or vertical |
| `AppendUnits` | If `True`, then the text for the current units (inch, mm etc.) will be appended after the number(s) for the dimension |

### Example:

```
.SetTextParameters( "Height=0.035, Angle=0, Mirrored=False, Font=DMPL"
  )
```

### Corresponding Command:

```
Insert/Loc and Dim Setup
```

### See Also:

```
.InsertText
```

## SetupDCodesDialog

### Format:

```
.SetupDCodesDialog
```

Displays the dialog for viewing and setting up D Codes (F5).

### Corresponding Command:

```
Setup/D Codes
```

## SetupLayerColorDialog

Format:

```
.SetupLayerColorDialog
```

Displays the dialog for viewing and changing the color of the current layer.

Corresponding Command:

```
Setup/Layer/Color
```

See Also:

```
.Color
.ColorsSetup
.CurrentLayer
.LayerColor
```

## ShowBoardLayer

Turn off all board layers, except the one specified.

Format:

```
.ShowBoardLayer( Next )
```

Example:

```
.ShowBoardLayer(2)
```

## SnapToGridSelected

Snap the selected elements to the grid. The grid must be on.

Format:

```
.SnapToGridSelected( Tolerance )
```

Example:

```
.SnapToGridSelected(.003)
```

## SnapTolerance

Format:

```
.SnapTolerance = Double
```

The tolerance used when AutoSnapToElement is active.

## SnapToPadmaster

Snap elements on the chosen layer(s) to the Padmaster layer. See .ChosenLayers.

Format:

```
.SnapToPadmaster( Padmaster, InFrameOnly, CurrentDCOnly, Tolerance,
   TracesAlso )
```

Example:

Layers to snap (step 1):

```
.ChosenLayers="3"
```

Snap the layers:

```
.SnapToPadmaster( l1, False, False, .003, False )
```

## SoldermaskSetupDCodes

Settings for the D Code options when creating soldermask.

Format:

```
.SoldermaskSetupDCodes( RoundToDecimal, CreateNewDCodes, Tolerance,
   FirstNewDCode )
```

Example:

```
.SoldermaskSetupDCodes( 2, True, 0, startdhere )
```

## SoldermaskSetupSwell

Settings for the swell options when creating soldermask.

Format:

```
.SoldermaskSetupSwell( ByFixedAmount, Percentage, PercentageMethod,
   FixedAmount, HeatReliefsAlso, TracesAlso )
```

Example:

```
.SoldermaskSetupSwell( True, 110, 0, .005, True, False )
```

## SortByProximity

Format:

```
.SortByProximity( UsingDistance )
```

Sort/Optimize the entities on a layer.

Example:

```
.SortByProximity(True)
```

## StartMacroFileExecution

Format:

```
.StartMacroFileExecution( PathName )
```

Applies only to CAMMaster as it needs the Sax Basic execution engine to run the Sax Basic macro file `PathName`. This command cannot be used from inside a macro file that is executed by the Sax Basic engine, as the engine is already busy in this case.
One can use it from an external executable or process to make CAMMaster execute a Sax Basic script. Once started, one can check for the completion by using the `.MacroFileExecuting` property.

Returns:

`False` if the macro file could not be started.

Example:

```
.StartMacroFileExecution( "C:\Scripts\Test.bas" )
```

See Also:

`.MacroFileExecuting`

## StatusLineMessage

### Format:

```
.StatusLineMessage
```

String property to read the last message displayed in the application's status line or to display a new message in the status line.

### Example:

```
.StatusLineMessge = "This is a test"
```

Will display "This is a test" in the status line of the application.

## StencilComponent

### Format:

```
.StencilComponent( reduceExpand, roundedRadius )
```

Applies component stencil transform to selected data. The selected data must consist of a component (1, 2 or 4 rows of landings). The model for this transformation is returned as a string and can be fed into `StencilFromModel` to apply this transform to a selection that consists of more than one component.

| | |
|---|---|
| `reduceExpand` | How the landings are changed.<br>A string that can contain one item or 4 comma separated items. If one item, it will be applied to all the 4 sides. "0" means no change. The items may contain units or the percent sign. |
| `roundedRadius` | The radius of rounded corners. If not needed set to 0 or empty string. |

### Returns:

If the operation succeeds, returns a string describing the model This string can be passed as an argument to `StencilFromModel`.

If the operation fails, returns the empty string.

### Corresponding Command:

```
Tools/StencilApertures/General Component
```

### See Also:

```
.StencilFromModel
```

## StencilDCodeNumber

### Format:

```
.StencilDCodeNumber( DCodeNumber, reduceExpand, roundedRadius )
```

Applies homebase transform to selected data. The resulting shape will match that of the D Code identified by *DCodeNumber*. The selected data must consist of a pair of rectangular pads. The model for this transformation is returned as a string and can be fed into `StencilFromModel` to apply this transform to more than one pair of pads.

| | |
|---|---|
| *DCodeNumber* | the D Code number of the shape that will be used as the replacement shape |
| reduceExpand | Change to the rectangle before the operation is done.<br>A string that can contain one item or 4 comma separated items. If one item it will be applied to all the 4 sides. "0" means no change. The items may contain units or the percent sign. |
| roundedRadius | The radius of rounded corners. If not needed set to 0 or empty string. |

Returns:

If the operation succeeds, returns a string describing the model This string can be passed as an argument to `StencilFromModel`.

If the operation fails, returns the empty string.

Corresponding Command:

`Tools/StencilApertures/D Code Shape`

See Also:

`.StencilFromModel`

## StencilFromModel

Format:

`.StencilFromModel( stencilDescription )`

Applies homebase transform to selected data. The transform is described in *stencilDescription*. This is a string thas was returned by one of `.StencilFromHomebase, .StencilWindows` etc. from a previous operation.

Returns:

If the operation succeeds, returns a count of transforms performed.

If the operation fails, returns -1.

## StencilHomebase

Format:

`.StencilHomebase( nose, shoulder, facingIn, variants, reduceExpand, roundedRadius )`

Applies homebase transform to selected data. The selected data must consist of a pair of rectangular pads. The model for this transformation is returned as a string and can be fed into `StencilFromModel` to apply this transform to more than one pair of pads.

| | |
|---|---|
| nose | the nose dimension, absolute or percent |
| shoulder | the shoulder dimension, absolute or percent |
| facingIn | if `True` facing in, else facing out |
| variants | Comma separated string that can contain: `"Split",` |

| | |
|---|---|
| | "Rounded". It can also be the empty string. |
| reduceExpand | Change to the rectangle before the operation is done.<br>A string that can contain one item or 4 comma separated items. If one item it will be applied to all the 4 sides. "0" means no change. The items may contain units or the percent sign. |
| roundedRadius | The radius of rounded corners. If not needed set to 0 or empty string. |

Returns:

If the operation succeeds, returns a string describing the model This string can be passed as an argument to StencilFromModel.

If the operation fails, returns the empty string.

Example:

```
.StencilHomebase( "40%", "80%", True, "Split", "-10%, -5%, -3%, -3%",
   "0" )
```

Corresponding Command:

Tools/StencilApertures/Homebase

See Also:

.StencilFromModel

## StencilInvertedHomebase

Format:

```
.StencilInvertedHomebase( nose, depth, feet, facingIn, variants,
   reduceExpand, roundedRadius )
```

Applies inverted homebase (C shape) transform to selected data. The selected data must consist of a pair of rectangular pads. The model for this transformation is returned as a string and can be fed into StencilFromModel to apply this transform to more than one pair of pads.

| | |
|---|---|
| nose | the nose dimension, absolute or percent |
| depth | the depth dimension, absolute or percent |
| feet | the feet dimension, absolute or percent |
| facingIn | if True facing in, else facing out |
| variants | Comma separated string that can contain: "Split", "Rounded" or "RoundedSimple". It can also be the empty string. |
| reduceExpand | Change to the rectangle before the operation is done.<br>A string that can contain one item or 4 comma separated items. If one item it will be applied to all the 4 sides. "0" means no change. The items may contain units or the percent sign. |
| roundedRadius | The radius of rounded corners. If not needed set to 0 or |

| | |
|---|---|
| | empty string. |

Returns:

If the operation succeeds, returns a string describing the model This string can be passed as an argument to `StencilFromModel`.

If the operation fails, returns the empty string.

Example:

```
.StencilInvertedHomebase( "100mil", "400mil" "10%", True,
  "RoundedSimple", "-10%, -5%, -3%, -3%", "0" )
```

Corresponding Command:

`Tools/StencilApertures/Inverted Homebase`

See Also:

`.StencilFromModel`

## StencilWindows

Format:

```
.StencilWindows( shape, numX, mumY, gap, diagonalMode, reduceExpand,
  roundedRadious )
```

Will execute the conversion of rectangular pads to stencil apertures using the "windows" aka "split" method. The model for this transformation is returned as a string and can be fed into `StencilFromModel`.

| | |
|---|---|
| `shape` | String. If empty the operation will apply to all rectangular pads in the selection. If not empty, must be the description of an "R" or "S" shape (see example below) and the operation will only apply to rectangles of the given shape |
| `numX` | Number of segments on the X axis |
| `numY` | Number of segments on the Y axis |
| `gap` | Dimension of gap as a string with optional units. Must be larger than 0. |
| `diagonalMode` | One of: `"No", "Crosshatch", "Ascending", "Descending"`.<br>Sets diagonal mode. Use `"No"` for normal, orthogonal mode. |
| `reduceExpand` | Change to the rectangle before the operation is done.<br>A string that can contain one item or 4 comma separated items. If one item it will be applied to all the 4 sides. "0" means no change. The items may contain units or the percent sign. |
| `roundedRadius` | The radius of rounded corners. If not needed set to 0 or empty string. |

Returns:

If the operation succeeds, returns a string describing the model This string can be passed as an argument to `StencilFromModel`.

If the operation fails, returns the empty string.

Example:

```
.StencilWindows( "S 10 10 mm", 3, 2, "2 mm", "No", "-5%", "0.2 mm" )
.StencilWindows( "", 3, 2, "2 mm", "Crosshatch", "10%, -5%, -0.02 mm,
  1 mil", "0" )
```

Corresponding Command:

```
Tools/StencilApertures/Windows
```

See Also:

```
.StencilFromModel
```

## StepAndRepeatChange

Format:

```
.StepAndRepeatChange( Num_x, Num_y, Step_x, Step_y, Id )
```

Change the step and repeat codes for the specified step block id (or selection, if Id is -1)

Example:

```
.StepAndRepeatChange(2, 2, 6.3, 8.3, 1)
```

## StepAndRepeatOneUp

Change the specified step and repeat Id to a one up. Id –1=selection, Id 0=all visible step and repeat blocks.

Format:

```
.StepAndRepeatOneUp(_Id)
```

Example:

```
.StepAndRepeatOneUp(0)
```

## StepAndRepeatSelected

Format:

```
.StepAndRepeatSelected( Num_x, Num_y, Step_x, Step_y, Mode )
```

Step and repeat the selected elements.

*Mode* controls how *Step_x* and *Step_y* are interpreted:

0: To corresponding elements

1: To edges of images

2: To centers of edge elements

Example:

```
.StepAndRepeatSelected(2, 2, 6.3, 8.3, 0)
```

## StepAndRepeatUngroup

Format:

```
.StepAndRepeatUngroup( Id )
```

Ungroup a stepped block. *Id* is the identification number of the block. The following special values can be used instead: -1=selection, 0=all visible step and repeat blocks.

Returns:

FALSE if error or command cannot be completed.

Corresponding Command:

```
Edit/Edit Selection/Step And Repeat/Ungroup
```

Example:

```
.StepAndRepeatUngroup( 0 )
```

## StretchTraces

Format:

```
.StretchTraces( xmin, ymin, xmax, ymax, DeltaX, DeltaY )
```

Stretch the traces crossing the selection, and move all entities contained within, the specified movement in x and y.

Example:

```
.StretchTraces(0, 0, 2, 3, 1, 0)
```

Select from 0,0 to 2,3 and stretch 1 in x, 0 in y.

## SwellSelected

Format:

```
.SwellSelected( SwellValue, TranscodeToFirst, TranscodeToLast, Flags )
```

Swell the selection by *SwellValue.* If *SwellValue* is positive the swell will inflate the selection (deflate if negative). Traces and pads are swelled by swelling their D Code. The swelled elements will be transcoded to new D Codes if the data contains elements of the same D Code that are not in the selection (and because of that need to remain unswelled). If transcoding is done than the range of D Codes *TranscodeToFirst - TranscodeToLast* is used for the new (swelled) D Codes. Polygons are swelled by inflating or deflating the outline.
*Flags* is a bitfield of options as shown below:

| *Bit #* | *Mask* | *Parameter Name* | *Meaning* |
|---|---|---|---|
| 0 | 1 | Transcode all swelled D Codes | Transcode always. This is useful to preserve the integrity of the original D Codes |
| 1 | 2 | Avoid rounded corners for polygons | The extra arcs normally generated at the junction of 2 straight edges are not generated, if possible. |

Returns:

`False` if an error occurred.

Example:

```
.SwellSelected( 0.002, 1200, 2000, 0 )
```

Corresponding Command:

```
Edit/Edit Selection/Swell
```

## ToolCount

Format:

```
.ToolCount( ToolNumber )
```

Returns the count of holes and routing paths for tool number *ToolNumber*.

Example:
```
n = .ToolCount(5)
```

## ToolsDestinationLayer

Format:
```
.ToolsDestinationLayer = String
```

Set the destination layer for different functions from the "Tools" menu. Set before any operation where selecting or transferring affected entities is necessary.

Example:
```
.ToolsDestinationLayer = "32"
```

## ToolShape

Format:
```
.ToolShape( ToolNumber ) = String
```

Property to get or set the tool shape (diameter and type) for the specified tool number. Type is `Tooling`, `Plated`, `Non-plated`, `Via` or `?`.

Example:
```
.ToolShape(1) = "0.126, Tooling"
```

## ToolNumbersTranscode

Format:
```
.ToolNumbersTranscode( Method )
```

Transcodes all used tool numbers so that the tools are ordered by tool diameter, starting from T1. Method is an integer that specifies the method used, as shown in the table below:

| *Method* | *Meaning* |
|---|---|
| 0 | Ascending. Ordered by increasing diameters. |
| 1 | Ascending by type. Ordered by increasing diameters, but tools of the same type are grouped together. |
| 2 | Descending. Ordered by decreasing diameters. |
| 3 | Descending by type. Ordered by decreasing diameters, but tools of the same type are grouped together. |

Returns:

The number of transcoded tools.

Corresponding Command:

```
Setup/Tool Codes/Transcode By Size
```

## ToolNumbersTranscodeAscending

Format:
```
.ToolNumbersTranscodeAscending( ByType )
```

Transcodes all used tool numbers so that the tools are ascending by tool diameter, starting from T1. If *ByType* is `True`, then ordering will be done within tool types so that, for instance, all Plated tools will be grouped together.

Returns:

The number of transcoded tools.

Corresponding Command:

```
Setup/Tool Codes/Transcode By Size
```

See Also:

```
.ToolNumbersTranscode
```

## TracePadOverlap

Trace/pad overlap and it's available options. Operations are 0-10, as listed in the operation tab under Tools-TracePad Overlap.

Format:

```
.TracePadOverlap( ClearDestLayer, InFrameOnly, Operation,
   ElementMargin )
```

Example:

```
.TracePadOverlap( False, False, 0, 0 )
```

## TracesVisible

Boolean property to turn trace visibility on or off.

Example:

```
.TracesVisible = True
```

## TranscodeSelected

Transcode the selection to the specified D Code.

Format:

```
.TranscodeSelected( ToDCode )
```

Example:

```
.TranscodeSelected( .FirstUnusedDCode(4000) )
```

## TranscodeSimilarDCodes

Unique shapes option in F5 D Code table.

Format:

```
.TranscodeSimilarDCodes( SizeTolerance )
```

Example:

```
.TranscodeSimilarDCodes(0)
```

## TransferLoadedLayersRange

Format:

```
.TransferLoadedLayersRange( FromLayerNumber, Delta )
```

Moves a group of layers up or down in the layers stackup. The range of layers to move is a contiguous range of loaded layers starting at `FromLayerNumber`. This range will be moved up or down by `Delta`. So, `FromLayerNumber` will end up in the `FromLayerNumber+Delta` spot, etc.

There must be `Delta` empty layers before or after the range to accomodate the move, otherwise this method return `False`.

Corresponding Command:

The "Move Up" and "Move Down" commands in the context menu (mouse right click) in the "Layers Setup" (F10) dialog

## TransferSelected

Format:

```
.TransferSelected( LayerNumber )
```

Transfer the selected elements to the specified layer.

Example:

```
.TransferSelected( 10 )
```

## UngroupSelectedText

Format:

```
.UngroupSelectedText()
```

Replaces all text blocks in the selection with the ungrouped strokes (ordinary traces, pads or polygons). This is useful when one want to manipulate the strokes in some special way.

Corresponding Command:

```
Edit/Edit Selection/Text/Ungroup
```

See Also:

```
.InsertTextEx
```

## Units

Format:

```
.Units
```

`String` property to set or get the units for the job. Units are `"inch", "mil", "mm", "cm"`.

Example:

```
.Units = "inch"
```

Corresponding Command:

Units combobox in "Screen" toolbar.

## UnselectedVisible

Format:

```
.UnselectedVisible = Boolean
```

Turn unselected elements visibility on or off.

Example:

```
.UnselectedVisible = True
```

## VentPolygons

Format:

```
.VentPolygons( Method, VentLayer, ClearVentLayersFirst, DCode,
  DcodeSize )
```

Vents polygons using the options already set. See vent options setup methods. Method is one of:

```
"Pads",
```

"Positive VPI",
"Negative VPI",
"Starburst",
"Solid",
"Hatched",
"Crosshatched"

Example:
```
.VentPolygons ("Pads", "96", False, 4321, .15)
```

Corresponding Command:

Tools/Venting

See Also:

.VentSetupe
.VentSetuphatch
.VentSetupStarburst

## VentSetup

Format:
```
.VentSetup( Clearance, Gap, ShortestTrace, OutlineOutside,
  OutlineInside, OffsetLayerPairs, OffsetVal, StaggerOddRows )
```

Options for venting.

Example:
```
.VentSetup (0, 0, 0.0005, True, False, False, 0, True)
```

Corresponding Command:

Tools/Venting

See Also:

.VentPolygons

## VentSetupHatch

Format:
```
VentSetupHatch( HatchAngle, CrossHatchAngle )
```

Options for venting.

Example:
```
.VentSetupHatch( 45, -45 )
```

Corresponding Command:

Tools/Venting

See Also:

.VentPolygons

## VentSetupStarburst

Format:
```
VentSetupStarburst( StarburstAngle )
```

Options for venting.

Example:
```
.VentSetupStarburst(45)
```
Corresponding Command:
```
Tools/Venting
```
See Also:
```
.VentPolygons
```

## Version

Format:
```
.Version
```
Returns the application version same as in the Help/About as a string of up to four numbers separated by dots. An example would be "8.1.1".

## ViewCenter

Format:
```
.ViewCenter = String
```
String property for getting or setting the center of the current view. *String* is a comma separated and contains the X and Y coordinates of the center.
Example:
```
.ViewCenter = "1.5, 2.781"
```

## ViewSelectionProperties

Pop-up the Selection Properties dialog.

Corresponding Command:
```
View/Selection/Properties
```
See Also:
```
.SelectionProperties
```

## WaitForClick

Format:
```
.WaitForClick( UserPrompt )
```
Waits for the user to click or press Enter and returns the cursor position as a string. *UserPrompt* is displayed in the help message area of the application. If the user pesses the Escape key an empty string is returned.
Use with caution, as while it is active CAMMaster will use up 100% of the computer processing cycles.

Returns:

Cursor position at time of click (or Enter) as a string or empty string if user presses the Escape key.

Example:

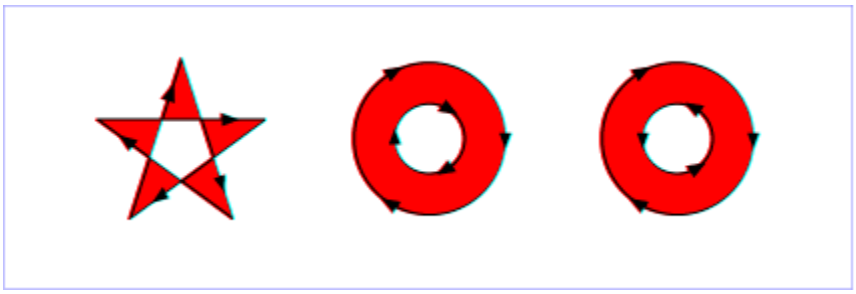
```
.WaitForClick( "Waiting for click" )
```

## WellFormedMethod

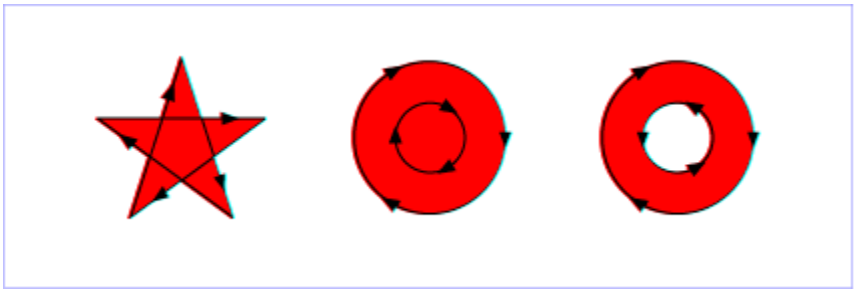
String property for the method used to correct malformed (self-intersecting) polygons. This setting is used when doing the "make well formed" command on selected polygons or when importing data (polygons are corrected automatically).

There are currently two methods available: "`Even-odd`" and "`Non-zero fill`". They can generate different results, so you need to be aware which method fits the data that you are processing. The more common method is "`Even-odd`". The images below illustrate the differences:

(1) "`Even-odd`"



(2) "`Non-zero fill`"



Corresponding Command:

```
Setup/Self-intersecting Polygon Correction
```

See Also:

```
.MakeSelectedPolygonsWellFormed
```

## Workspace

String property for the workspace area (rectangle).

Format:
```
.Workspace = String
```
Example:
```
.Workspace = "0, 0, 20, 26"
```

## ZoomLevel

Set (get) the zoom level.

Format:
```
ZoomLevel = Double
```
Example:
```
.ZoomLevel = 2.5
```

## ZoomOnFrame

Zoom on the frame area and move cursor to frame center.

Example:
```
.ZoomOnFrame
```

## ZoomShowAll

Zoom to show all data.

Example:
```
.ZoomShowAll
```

## ZoomShowAll_Include0

Format:
```
.ZoomShowAll_Include0 = Boolean
```
Controls whether the absolute origin is always showed or not by the `ZoomShowAll` method.

Example:
```
.ZoomShowAll_Include0 = True
```

# Events Reference

## OnDone

### Format:

```
.OnDone( ProcessID )
```

This event will be fired when the user activates the "Macro/Done" command in CAMMaster (Alt+M+D).

### Corresponding Command:

```
Macro/Done
```

# Appendix A

## PentaLogix Geometry Files

PentaLogix geometry files are ASCII files used to communicate information about the geometry of database objects, either from a script to CAMMaster or from CAMMaster to a script. They are part of import and export operations available via scripting commands. Note that, as opposed to other import and export operations, the PentaLogix geometry files can only be manipulated by scripting commands. They are not available through the normal user interface (menu commands).

Geometry files contain CAMMaster primitives: pads, traces (straight and arcs), polygons and routing paths. Only the geometry of these elements is represented (endpoints, vertices , etc.). The shape of pads and traces, the polarity of the layers (positive or negative) and layer info is not represented in geometry files. However, paint/scratch polarity of elements can be represented.

The main use for this file is for quick access to existing elements via script programs and for inserting new elements needed by script operations.

The files are written (output by CAMMaster) using the `.Export` method and read (elements inserted into the CAM database) using the `.Import` method. Please see the descriptions of these commands in the preceding text. The file type name to be used in these methods is `"PentaLogix"`.

For `.Export`, only visible elements are exported.

An example for export is:

```
.Export( "", "PentaLogix", "1-2" )
```

This will export two files, for layers 1 and 2, using the layer names as file names.

To export all selected elements you can do the following:

```
.UnselectedVisible = False
.Export( "selected.txt", "PentaLogix", "All" )
.UnselectedVisible = True
```

An example for import is:

```
.CurrentLayer = 3
.AddToFileList( "" )
.AddToFileList( "C:\data\sample1.txt" )
.AddToFileList( "C:\data\sample2.txt" )
.Import( "PentaLogix" )
```

This will import two files into layers 3 and 4.

### Format Specification

Comments can appear anywhere. They start with ";". Any characters following a ";" are ignored. A comment may be on a line by itself.

File header:

```
PentaLogix geometry file V number
```

Where *number* is the current version number for the format specification (integer).

The next line defines the units for the coordinates in the rest of the file:

```
Units, units-text
```

Where *units-text* is one of: `inch`, `mil`, `cm` or `mm`.

The lines which follow represent primitives. They can appear in any order. The first word in the line determines the type of primitive: `Pad, Hole, Straight, Arc, Polygon, Path`. This keyword can be preceded (optionally) by a "+" or "–" sign to indicate polarity (paint or scratch). If the sign is missing, paint polarity is assumed ("+").

1. **Pads**
   There are two ways to represent pads.

   ```
   Pad, Dnumber, Xcoord, Ycoord
   Hole, Tnumber, Xcoord, Ycoord
   ```

   where *number* is an integer that is either a D Code number or a tool number and the *coords* are floating point coordinates, with an optional decimal point.

2. **Linear Traces**

   ```
   Straight, Dnumber, Xcoord, Ycoord, Xcoord, Ycoord
   ```

3. **Arc Traces**

   ```
   Arc, Dnumber, Xcoord, Ycoord, Xcoord, Ycoord, Xcoord, Ycoord, (Xcoord, Ycoord)
   ```

   Arcs are represented by three points (six pairs of coordinates). The first and last point are the endpoints of the arc and the second point is a point anywhere on the arc, but preferably the midpoint. For a full circle, the first and last point should be the same and the second point should be the point diametrically opposed. The last item, which is parenthesized, is the arc center and is for information purposes only as the center can (and should) be derived from the other three points. The center has been added in version V2 of the PentaLogix geometry files. Previously the definition contained only the arc points.

4. **Polygons**

   Polygons consist of one outside contour (the container) and may also have inner contours (holes). They are represented by first defining the outer contour followed by an arbitrary number of inner contours.

   ```
   Polygon, Start, Xcoord, Ycoord, Enumber, Hnumber
   edges for container
   Polygon, Hole, Xcoord, Ycoord, Enumber
   edges for hole
   Polygon, Hole, Xcoord, Ycoord, Enumber
   ```

*edges for hole*
…
`Polygon, End`

The `X` and `Y` fields are the dimensions of the bounding box. `E` is the number of edges (or vertices) and `H` is the number of holes.
The *edges* are a sequence of straights and arcs:

`Straight, X`*coord*`, Y`*coord*`, X`*coord*`, Y`*coord*
`Arc, X`*coord*`, Y`*coord*`, X`*coord*`, Y`*coord*`, X`*coord*`, Y`*coord*, (`X`*coord*, `Y`*coord*)

similar to the straight and arc traces (see above), but without the D field. The last edge of each contour must close the polygon (go back to the first vertex of the first edge in the contour). If a polarity is specified it appears only before the first `Polygon` (the container).

5. **Routing Paths**

`Path, T`*number*`, X`*coord*`, Y`*coord*`, P`*number*
*edges*
`Path, End`

`T` is the tool number, `X` and `Y` are the dimensions of the bounding box. `P` is the precedence.

## Sample PentaLogix geometry file

```
PentaLogix geometry file V2
Units, inch
Polygon, Start, X1.150001, Y1.15, E52, H2
      Straight, X1.789092, Y3.409205, X1.789092, Y3.548851
      Straight, X1.789092, Y3.548851, X1.794665, Y3.543278
      Straight, X1.794665, Y3.543278, X1.893291, Y3.395338
      Arc, X1.893291, Y3.395338, X1.89392, Y3.394438, X1.894588, Y3.393566, (X1.914193,
      Y3.409276)
      Straight, X1.894588, Y3.393566, X2.09457, Y3.143589
      Arc, X2.09457, Y3.143589, X2.0955, Y3.142492, X2.096492, Y3.141451, (X2.113908, Y3.159041)
      Straight, X2.096492, Y3.141451, X2.196238, Y3.041705
      Arc, X2.196238, Y3.041705, X2.199372, Y3.038997, X2.202911, Y3.036845, (X2.214055,
      Y3.059157)
      Straight, X2.202911, Y3.036845, X2.302911, Y2.986846
      Arc, X2.302911, Y2.986846, X2.308348, Y2.984875, X2.314092, Y2.984206, (X2.314093,
      Y3.009207)
      Arc, X2.314092, Y2.984206, X2.318097, Y2.984529, X2.321998, Y2.985489, (X2.314091,
      Y3.009215)
      Straight, X2.321998, Y2.985489, X2.468149, Y3.034206
      Straight, X2.468149, Y3.034206, X2.514093, Y3.034206
      Arc, X2.514093, Y3.034206, X2.53177, Y3.041528, X2.539092, Y3.059205, (X2.514093,
      Y3.059205)
      Straight, X2.539092, Y3.059205, X2.539092, Y3.059207
      Arc, X2.539092, Y3.059207, X2.53177, Y3.076884, X2.514093, Y3.084206, (X2.514093,
      Y3.059207)
      Straight, X2.514093, Y3.084206, X2.324447, Y3.084206
      Straight, X2.324447, Y3.084206, X2.233027, Y3.175626
      Straight, X2.233027, Y3.175626, X1.948342, Y3.555205
      Straight, X1.948342, Y3.555205, X2.230096, Y3.79
      Arc, X2.230096, Y3.79, X2.236732, Y3.798601, X2.239092, Y3.809205, (X2.214095, Y3.809205)
      Straight, X2.239092, Y3.809205, X2.239092, Y3.809207
      Arc, X2.239092, Y3.809207, X2.237168, Y3.818824, X2.231692, Y3.826961, (X2.214094,
      Y3.809207)
      Straight, X2.231692, Y3.826961, X2.031847, Y4.026806
      Arc, X2.031847, Y4.026806, X2.026171, Y4.031095, X2.019516, Y4.033611, (X2.014096,
      Y4.009214)
```

```
      Straight, X2.019516, Y4.033611, X1.569516, Y4.133611
      Arc, X1.569516, Y4.133611, X1.566822, Y4.134057, X1.564095, Y4.134206, (X1.564095,
      Y4.109213)
      Straight, X1.564095, Y4.134206, X1.564092, Y4.134206
      Arc, X1.564092, Y4.134206, X1.554525, Y4.132303, X1.546415, Y4.126884, (X1.564092,
      Y4.109207)
      Straight, X1.546415, Y4.126884, X1.446415, Y4.026884
      Arc, X1.446415, Y4.026884, X1.440995, Y4.018773, X1.439092, Y4.009206, (X1.464091,
      Y4.009206)
      Straight, X1.439092, Y4.009206, X1.439092, Y4.009205
      Arc, X1.439092, Y4.009205, X1.44339, Y3.99519, X1.454807, Y3.985994, (X1.46409, Y4.009205)
      Straight, X1.454807, Y3.985994, X1.64144, Y3.911341
      Straight, X1.64144, Y3.911341, X1.556186, Y3.882923
      Arc, X1.556186, Y3.882923, X1.543811, Y3.873824, X1.539092, Y3.859206, (X1.564092,
      Y3.859206)
      Straight, X1.539092, Y3.859206, X1.539092, Y3.859205
      Arc, X1.539092, Y3.859205, X1.541016, Y3.849588, X1.546492, Y3.841451, (X1.56409,
      Y3.859205)
      Straight, X1.546492, Y3.841451, X1.589092, Y3.798851
      Straight, X1.589092, Y3.798851, X1.589092, Y3.715107
      Straight, X1.589092, Y3.715107, X1.493544, Y3.524013
      Straight, X1.493544, Y3.524013, X1.396591, Y3.42706
      Arc, X1.396591, Y3.42706, X1.391042, Y3.418889, X1.389091, Y3.409207, (X1.414089,
      Y3.409207)
      Straight, X1.389091, Y3.409207, X1.389091, Y3.409206
      Arc, X1.389091, Y3.409206, X1.396414, Y3.391528, X1.414092, Y3.384205, (X1.414093,
      Y3.409207)
      Arc, X1.414092, Y3.384205, X1.419836, Y3.384874, X1.425272, Y3.386844, (X1.41409,
      Y3.409214)
      Straight, X1.425272, Y3.386844, X1.519994, Y3.434206
      Straight, X1.519994, Y3.434206, X1.658189, Y3.434206
      Straight, X1.658189, Y3.434206, X1.752911, Y3.386846
      Arc, X1.752911, Y3.386846, X1.758348, Y3.384875, X1.764092, Y3.384206, (X1.764093,
      Y3.409207)
      Straight, X1.764092, Y3.384206, X1.764093, Y3.384206
      Arc, X1.764093, Y3.384206, X1.78177, Y3.391528, X1.789092, Y3.409205, (X1.764093,
      Y3.409205)
Polygon, Hole, X0.218601, Y0.1, E4
      Straight, X2.017379, Y3.884206, X1.950713, Y3.784206
      Straight, X1.950713, Y3.784206, X1.798778, Y3.784206
      Straight, X1.798778, Y3.784206, X1.832111, Y3.884206
      Straight, X1.832111, Y3.884206, X2.017379, Y3.884206
Polygon, Hole, X0.778901, Y1.03136, E46
      Straight, X2.053737, Y3.934206, X1.814092, Y3.934206
      Arc, X1.814092, Y3.934206, X1.799474, Y3.929487, X1.790375, Y3.917112, (X1.814092,
      Y3.909206)
      Straight, X1.790375, Y3.917112, X1.740375, Y3.767112
      Arc, X1.740375, Y3.767112, X1.739415, Y3.763211, X1.739092, Y3.759206, (X1.764101,
      Y3.759205)
      Straight, X1.739092, Y3.759206, X1.739092, Y3.759205
      Arc, X1.739092, Y3.759205, X1.746414, Y3.741528, X1.764091, Y3.734206, (X1.764091,
      Y3.759205)
      Straight, X1.764091, Y3.734206, X1.964092, Y3.734206
      Arc, X1.964092, Y3.734206, X1.975888, Y3.737164, X1.984893, Y3.745338, (X1.964092,
      Y3.759207)
      Straight, X1.984893, Y3.745338, X2.084893, Y3.895339
      Arc, X2.084893, Y3.895339, X2.086345, Y3.897814, X2.087504, Y3.900439, (X2.064079,
      Y3.909213)
      Straight, X2.087504, Y3.900439, X2.177057, Y3.810886
      Straight, X2.177057, Y3.810886, X1.898088, Y3.578412
      Arc, X1.898088, Y3.578412, X1.891452, Y3.569811, X1.889092, Y3.559207, (X1.914089,
      Y3.559207)
      Straight, X1.889092, Y3.559207, X1.889092, Y3.559206
      Arc, X1.889092, Y3.559206, X1.890375, Y3.5513, X1.894092, Y3.544206, (X1.914092,
      Y3.559206)
      Straight, X1.894092, Y3.544206, X2.194092, Y3.144206
      Arc, X2.194092, Y3.144206, X2.195242, Y3.142785, X2.196492, Y3.141451, (X2.214253,
      Y3.159346)
      Straight, X2.196492, Y3.141451, X2.287479, Y3.050464
      Straight, X2.287479, Y3.050464, X2.228898, Y3.079755
      Straight, X2.228898, Y3.079755, X2.13275, Y3.175903
```

```
      Straight, X2.13275, Y3.175903, X1.934288, Y3.42398
      Straight, X1.934288, Y3.42398, X1.834893, Y3.573074
      Arc, X1.834893, Y3.573074, X1.833391, Y3.575098, X1.831692, Y3.576961, (X1.814109,
      Y3.55922)
      Straight, X1.831692, Y3.576961, X1.781769, Y3.626884
      Arc, X1.781769, Y3.626884, X1.773659, Y3.632303, X1.764092, Y3.634206, (X1.764092,
      Y3.609207)
      Straight, X1.764092, Y3.634206, X1.764091, Y3.634206
      Arc, X1.764091, Y3.634206, X1.746414, Y3.626884, X1.739092, Y3.609207, (X1.764091,
      Y3.609207)
      Straight, X1.739092, Y3.609207, X1.739092, Y3.449657
      Straight, X1.739092, Y3.449657, X1.675272, Y3.481568
      Arc, X1.675272, Y3.481568, X1.669836, Y3.483538, X1.664092, Y3.484207, (X1.66409,
      Y3.459198)
      Straight, X1.664092, Y3.484207, X1.524447, Y3.484206
      Straight, X1.524447, Y3.484206, X1.531593, Y3.491352
      Arc, X1.531593, Y3.491352, X1.534301, Y3.494487, X1.536454, Y3.498026, (X1.514094,
      Y3.509205)
      Straight, X1.536454, Y3.498026, X1.636453, Y3.698026
      Arc, X1.636453, Y3.698026, X1.638423, Y3.703462, X1.639092, Y3.709206, (X1.614083,
      Y3.709208)
      Straight, X1.639092, Y3.709206, X1.639092, Y3.809207
      Arc, X1.639092, Y3.809207, X1.637168, Y3.818824, X1.631692, Y3.826961, (X1.614094,
      Y3.809207)
      Straight, X1.631692, Y3.826961, X1.610372, Y3.84828
      Straight, X1.610372, Y3.84828, X1.721998, Y3.885489
      Arc, X1.721998, Y3.885489, X1.734373, Y3.894588, X1.739092, Y3.909206, (X1.714092,
      Y3.909206)
      Straight, X1.739092, Y3.909206, X1.739092, Y3.909207
      Arc, X1.739092, Y3.909207, X1.734794, Y3.923222, X1.723377, Y3.932418, (X1.714094,
      Y3.909207)
      Straight, X1.723377, Y3.932418, X1.508578, Y4.018337
      Straight, X1.508578, Y4.018337, X1.572065, Y4.081824
      Straight, X1.572065, Y4.081824, X2.001563, Y3.98638
      Straight, X2.001563, Y3.98638, X2.053737, Y3.934206
Polygon, End
Pad, D50, X0.5, Y0.5
Pad, D50, X1, Y0.5
Pad, D50, X1.5, Y0.5
Pad, D50, X2, Y0.5
Pad, D50, X2.5, Y0.5
Arc, D10, X1, Y1.5, X1, Y2.5, X1, Y1.5, (X1, Y2)
Arc, D10, X1.5, Y2.5, X1.805172, Y2.396068, X1.983476, Y2.12748, (X1.499999, Y1.999999)
Arc, D10, X2.052991, Y2, X2.002462, Y1.769067, X1.860112, Y1.580336, (X1.499994, Y2.000003)
Arc, D10, X2.595432, Y2.224061, X3.460751, Y2.039832, X2.639888, Y1.709832, (X3, Y2)
Path, T5, X3.5, Y2.5, P1
      Straight, X2, Y0.2, X3.7, Y0.2
      Straight, X3.7, Y0.2, X3.7, Y2.7
      Straight, X3.7, Y2.7, X0.2, Y2.7
      Straight, X0.2, Y2.7, X0.2, Y1.1
      Straight, X0.2, Y1.1, X0.2, Y0.2
      Straight, X0.2, Y0.2, X1.5, Y0.2
Path, End
Straight, D10, X0.5, Y1, X1, Y1
Straight, D10, X1.5, Y1, X1.5, Y1.5
Straight, D10, X2, Y1, X2.5, Y1.5
Pad, D11, X3, Y1
Pad, D11, X3.5, Y1
Hole, T1, X3, Y0.5
Hole, T1, X3.5, Y0.5
```

# Appendix B

## Color Setup Files

These are ASCII text files which store information about the color setup: the color index (position) for each layer and the RGB definition of each index. CAMMaster allows for 15 distinct layer indices. Layer colors are then mapped to these indices. Changing the RGB of one index will change the displayed color for all layers which are mapped to that index. These files can be read or written with the script command `.ColorsSetup` and the CAMMaster commands "`Setup/Layer Colors/Save To File`" and "`Setup/Layer Colors/Restore From File`"

### Format Specification

The first line is:

`PentaLogix default colors V`*number*

Where *number* is the current version number for the color file format (integer). This is currently `1`.

Next follow 17 lines of RGB definitions for indices 0 through 16. Only the values for indices 1 through 15 are relevant. Indices 0 and 16 cannot be changed by the user and will be ignored. However, they must be present in the file. The format of each such line is:

*Index: R G B*

*Index* is the color index (integer). It should start from 0 and continue through 16. *R*, *G* and *B* are the red, green and blue component values for the RGB composition of the color (8 bit values). They are in hexadecimal format (`0x00` through `0xFF`).

The final block of 255 lines is the mapping of layer numbers to color indices (positions). Each line is of the form:

*LayerNumber: Index*

Note that incomplete files are accepted, but the part that exists must be in sequence. So, for instance, one could have a file which contains only the RGB mappings of the indices if the mapping of layers to color indices is not to be changed.

### Sample Color Setup file

```
PentaLogix default colors V1
00: 0x00  0x00  0x00
01: 0x00  0x00  0x8F
02: 0x00  0x7F  0x00
03: 0x00  0x7F  0x7F
04: 0x7F  0x00  0x00
05: 0x7F  0x00  0x7F
06: 0x7F  0x7F  0x00
07: 0x4F  0x4F  0x4F
08: 0xC0  0xC0  0xC0
09: 0x00  0x00  0xFF
10: 0x00  0xFF  0x00
11: 0x00  0xFF  0xFF
```

```
12: 0xFF  0x00  0x00
13: 0xFF  0x00  0xFF
14: 0xFF  0xFF  0x00
15: 0xFF  0x80  0x80
16: 0x30  0x30  0x30
1: 10
2: 12
```
… (Layers 3 through 250)
```
251: 9
252: 2
253: 4
254: 1
255: 8
```

## Contact Us:

If you would like more information about the PentaLogix CAM Methods and Properties or the PentaLogix software, we can be contacted at:

4749 Hastings Place

Lake Oswego, OR 97035

Voice: (800) 238-1920 Fax: (503) 828-9408

World wide web: http://www.pentalogix.com

Email: sales@pentalogix.com

For support: support@pentalogix.com

## Copyright, Disclaimer and Trademark Information