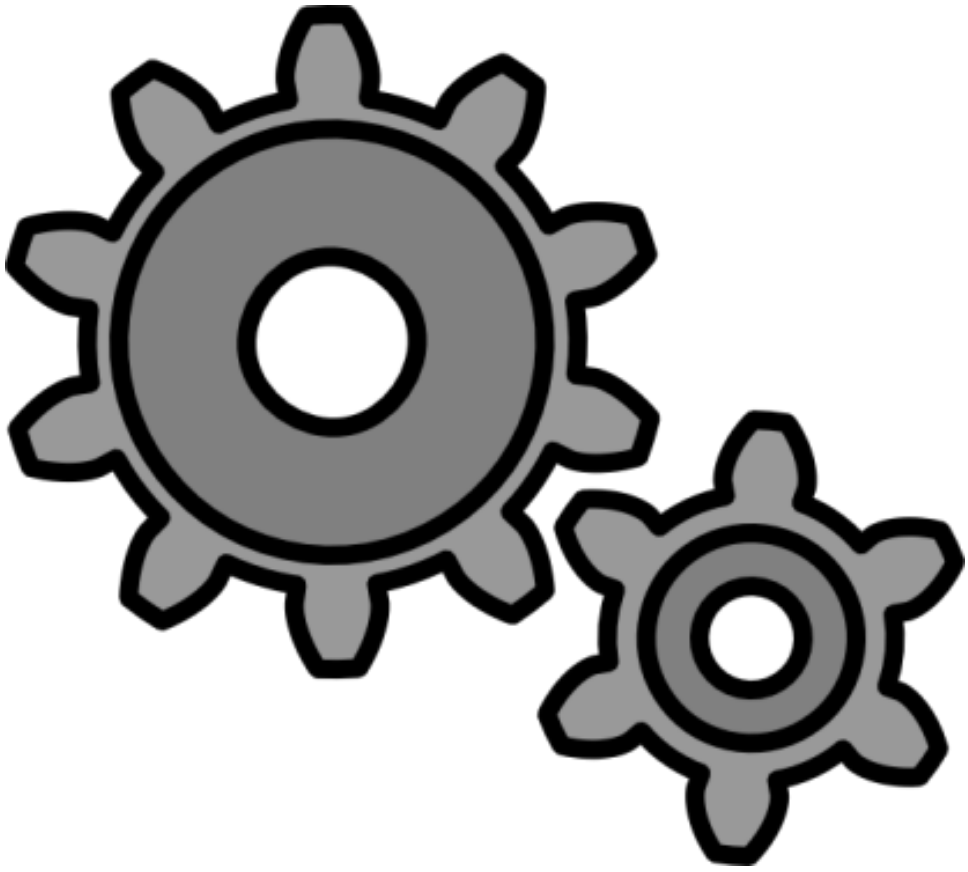
Vectric Lua Interface for Gadgets

The Script interface and associated documentation is designed for people who are familiar with programming and is provided as a development tool for such people. It is extremely important to understand that Vectric cannot offer individual support for people wanting to write scripts. This documentation and the associated example scripts available from the Vectric web site are the only way you can learn about the script interface and develop your own gadgets. The appropriate area on the Vectric forum may also be useful for asking questions of other users.



Version 2.01

Contents

Introduction

The Script interface and associated documentation is designed for people who are familiar with programming and is provided as a development tool for such people. It is extremely important to understand that Vectric cannot offer individual support for people wanting to write scripts. This documentation and the associated example scripts available from the Vectric web site are the only way you can learn about the script interface and develop your own gadgets. The appropriate area on the Vectric forum may also be useful for asking questions of other users.

This document lists the objects (classes) available to Lua scripts (Gadgets) from VCarve Pro V7.5 and Aspire V4.5

An object in Lua can have both methods and properties. A method is called using ':' after the object name e.g. job:Refresh2DView(), a property is accessed via '.' e.g. job.Exists . Properties can be either read only (R/O i.e Get Values) or read / write (R/W - Get/Set Values).

Global Methods

Most of the actions you can perform from Lua are done by calling methods on objects associated with the current job. There are a few methods which operate Globaly on the application and they are documented here .

IsAspire()

Returns true if the script is running inside Aspire.

IsBetaBuild()

returns true if this is a Beta build rather than a release build

GetAppVersion()

Returns application version number as a double e.g 4.004 for Aspire V4.0

GetBuildVersion()

Returns application internal build verion which is the same for both Aspire and VCarve Pro. This is different to the AppVersion above as Aspire 4.004 will report e.g. 7.004 which matches VCarve Pro 7.004 as they are built off the same code base and have the same script level support.

MessageBox (string text)

Displays a message box with passed text to user. The text can include HTML formatting.

text – *string* – text to display in the message box

Job related global methods

CloseCurrentJob ()

Close the current job - same as File - Close, if there are any unsaved changes user will be prompted to save before closing.

CreateNewJob(string name, Box2D bounds, double thickness, bool in_mm, bool origin_on_surface)

Creates a new job with passed name, and settings. Returns true if job created OK else false.

name - string - name for job (without extension)
bounds - Box2D - the 2d bounding box for the job area
thickness - double - thickness of material block
in_mm - bool - true if job is in mm, else in inches
origin_on_surface - true if z zero on material surface, else on machine bed

OpenExistingJob(string pathname)

Opens an existing CRV or CRV3D file. Returns true if file opened OK, else false

pathname - string - path to file

SaveCurrentJob()

Save the current job - same as File - Save, if no path has been set for the job this method will display the File Save As dialog.

Helper methods for creating and accessing objects on layers ...

Vector object related global methods

CastCadObjectToCadBitmap(CadObject obj)

Casts passed CadObject to a CadBitmap - returns a CadBitmap object to work with

obj - CadObject - Object to be cast to a CadBitmap

CastCadObjectToCadContour(CadObject obj)

Casts passed CadObject to a CadContour - returns a CadContour object to work with

obj - CadObject - Object to be cast to a CadContour

CastCadObjectToCadObjectGroup(CadObject obj)

Casts passed CadObject to a CadObjectGroup - returns a CadObjectGroup object to work with

obj - CadObject - Object to be cast to a CadObjectGroup

CastCadObjectToCadPolyline(CadObject obj)

Casts passed CadObject to a CadPolyline - returns a CadPolyline object to work with

obj - CadObject - Object to be cast to a CadPolyline

CastCadObjectToCadToolpathOutline(CadObject obj)

Casts passed CadObject to a CadToolpathOutline - returns a CadToolpathOutline object to work with

obj - CadObject - Object to be cast to a CadToolpathOutline

CastCadObjectToCadToolpathPreview(CadObject obj)

Casts passed CadObject to a CadToolpathPreview - returns a CadToolpathPreview object to work with

obj - CadObject - Object to be cast to a CadBitmap

CastCadObjectToTxtBlock(CadObject obj)

Casts passed CadObject to a TxtBlock - returns a TxtBlock object

obj - CadObject - Object to be cast to a TxtBlock

CreateCircle(double x, double y, double radius, double tolerance, double z_value)

Create a Contour object for a circle consisting of 4 arcs.

x – double – x coord of centre of circle
y – double – y coord of centre of circle
radius – double – radius of circle
tolerance – double – tolerance of contour – use 0.0 for default
z_value – double – z coord for contour

CreateCopyOfSelectedContours(bool smash_beziers, bool smash_arcs, double smash_tol)

Create a ContourGroup containing a copy of the currently selected contours in the job. T

smash_beziers – bool – if true, any Bezier spans in the selection are replaced with a series of straight line spans which match the original Bezier within the specified tolerance.

smash_arc – bool – if true, any Arc spans in the selection are replaced with a series of straight line spans which match the original Arc within the specified tolerance.

I – *double* – tolerance (max allowed deviation) to use when approximating arcs or beziers with a series of straight lines.

Component Related Global Methods – Aspire only

IsTransparent(float value)

Returns true if the specified value is considered transparent

GetTransparentHeight()

Returns as a float the heights in reliefs considered transparent. This is the value used internally to represent a 'transparent' point.

CastComponentToComponentGroup(Component component)

Casts passed Component to a ComponentGroup - returns a ComponentGroup object to work with

Data file locations related global methods

GetDataLocation()

Returns the 'root' data location for the application e.g.

"C:\ProgramData\Vectric\Aspire\V4.0" For V4.0 of Aspire on Windows 7.

"C:\ProgramData\Vectric\VCarve Pro\V7.0" For V7.0 of VCarve Pro on Windows 7.

This is the same folder accessible via "File – OpenApplication Data Folder …" within the program.

GetPostProcessorLocation()

Returns the PostP folder location for the program e.g

"C:\ProgramData\Vectric\Aspire\V4.0\PostP" For V4.0 of Aspire on Windows 7.

GetToolDatabaseLocation()

Returns the Tool Database folder location for the program e.g

"C:\ProgramData\Vectric\Aspire\V4.0\ToolDatabase" For V4.0 of Aspire on Windows 7.

GetGadgetsLocation()

Returns the Gadgets folder location for the program e.g

"C:\ProgramData\Vectric\Aspire\V4.0\Gadgets" For V4.0 of Aspire on Windows 7.

Note that the user can also install Gadgets in their local documents folder under …

"Vectric Files\Gadgets\Aspire V4.0"

GetToolpathDefaultsLocation()

Returns the ToolpathDefaults folder location for the program e.g

"C:\ProgramData\Vectric\Aspire\V4.0\ToopathDefaults" For V4.0 of Aspire on Windows 7.

GetBitmapTexturesLocation()

Returns the BitmapTextures folder location for the program which stores the bitmaps used for displaying different material types e.g

"C:\ProgramData\Vectric\Aspire\V4.0\BitmapTextures" For V4.0 of Aspire on Windows 7.

Note that the user can also install additional textures in their local documents folder under …

"Vectric Files\Material Images"

GetVectorTexturesLocation()

Returns VectorTextures folder location for the program e.g

"C:\ProgramData\Vectric\Aspire\V4.0\VectorTextures" For V4.0 of Aspire on Windows 7.

CreateCadContour (Contour ctr)
Returns a CadContour which owns the passed Contour object.

ctr - Contour - the data which is held by the created CadContour

CreateCadGroup(ContourGroup ctr_group)
Returns a CadContourGroup which owns the passed ContourGroup object.

ctr_group - ContourGroup - the data which is held by the created CadContourGroup

GetDefaultContourTolerance ()
Returns a double value which is the default value for contour tolerances used within the program. This is usually 0.001 if working in metric or 0.00004 if working in inches.

High Level Objects
The objects in this section of the document represent the high level objects within the program. Access to the program normaly starts with the VectricJob, and from there you can access the layers holding all the geometry and the current selection state for the job.

MaterialBlock
This object represents the material block within the program. It holds the width, height and thickness of the job and also the XY and Z origins. There is a single instance of the material block in the program. A number of methods are provided to covert between 'absolute' z values and z values relative to the surface of the material. These are very useful what writing scripts for toolpath generation as scripts need to take account of the fact that the user may change between Z0 on the machine representing the bottom of the material or the top of the material.

Constructors

MaterialBlock() - Constructor
A new object which referes to the single material block within the program.

e.g
```
local mtl_block = MaterialBlock()
```

Properties

. ActualXYOrigin
R/O - Point2D - XY position for user specified origin position on material (see .XY origin)

. BlcOrigin
R/O - Point3D - XYZ position for bottom left corner of material block

.Height
R/O - double - height of material - length in Y

.InMM
R/O - bool - true if the job / job is in MM else it is in inches

.MaterialBox
R/O - Box3D - the 3D bounding box for the material

.MaterialSurfaceZ
R/O - double - Z value for surface of material

.Thickness
R/O - double - thickness of material - length in Z

.Width
R/O - double - width of material - length in X

.XYOrigin
R/O - XYOrigin - Value indicating position in material chosen as XY origin - valid values are ...

MaterialBlock.BLC	- Bottom Left Corner of material
MaterialBlock.BRC	- Bottom Right Corner of material
MaterialBlock.TRC	- Top Right Corner of material
MaterialBlock.TLC	- Top Left Corner of material
MaterialBlock.CENTRE	- Centre of material (Note British spelling!)

.ZOrigin
R/O - ZOrigin - Value indicating position of Z origin - valid values are ...

MaterialBlock.Z_TOP
MaterialBlock.Z_CENTRE
MaterialBlock.Z_BOTTOM

Methods

:CalcAbsoluteZ(double z_value)
Returns an 'absolute' z value from a z value relative to the surface of the block. Z_value can be -ve for values below the surface or +ve for values above the surface.

z_value - double - z value relative to surface of material

:CalcAbsoluteZFromDepth (double z_value)
Returns a depth value below the surface from an absolute z value.

z_value - double - absolute z value.

: CalcDepthFromAbsoluteZ (double z_value)

Returns a depth below the surface of the material from an absolute z value.

z_value - double -Absolute z value

Example Code

```
--[[  ------------------- DisplayMaterialSettings -----------------------------
|
|   Display information about the size and position etc of the material block
|
|
|  Return Values:
|    None
]]
function DisplayMaterialSettings()

    local mtl_block = MaterialBlock()
    local units

    if mtl_block.InMM then
      units  = " mm"
    else
      units  = " inches"
    end

    -- Display material XY origin
    local xy_origin_text = "invalid"
    local xy_origin = mtl_block.XYOrigin

    if xy_origin == MaterialBlock.BLC then
       xy_origin_text = "Bottom Left Corner"
    elseif xy_origin == MaterialBlock.BRC then
       xy_origin_text = "Bottom Right Corner"
    elseif xy_origin == MaterialBlock.TRC then
       xy_origin_text = "Top Right Corner"
    elseif xy_origin == MaterialBlock.TLC then
       xy_origin_text = "Top Left Corner"
    elseif xy_origin == MaterialBlock.CENTRE then  -- NOTE: British spelling for Centre!
       xy_origin_text = "Centre"
    else
       xy_origin_text = "Unknown XY origin value!"
    end

    local z_origin_text = "invalid"
    local z_origin = mtl_block.ZOrigin

    if z_origin == MaterialBlock.Z_TOP then
       z_origin_text = "Top of Material"
    elseif z_origin == MaterialBlock.Z_CENTRE then -- NOTE: British spelling for Centre!
       z_origin_text = "Centre of Material"
    elseif z_origin == MaterialBlock.Z_BOTTOM then
       z_origin_text = "Bottom of Material"
    else
       z_origin_text = "Unknown Z origin value!"
    end

    local xy_origin_pos =  mtl_block.ActualXYOrigin

    -- get 3d box object describing material bounds ....
    local mtl_box = mtl_block.MaterialBox
    local mtl_box_blc = mtl_box.BLC

    -- test methods to conver z values between absolute z and relative depths
    local test_val = 0.125
    local depth = mtl_block:CalcDepthFromAbsoluteZ(test_val)
    local abs_z = mtl_block:CalcAbsoluteZFromDepth(test_val)
```

```
    DisplayMessageBox(
                    "Width     = " .. mtl_block.Width     .. units .."\n" ..
                    "Height    = " .. mtl_block.Height    .. units .. "\n" ..
                    "Thickness = " .. mtl_block.Thickness .. units .. "\n" ..
                    "\n" ..
                    "XY Origin = " .. xy_origin_text .. "\n" ..
                    "  Position = (" .. xy_origin_pos.x .. ", " .. xy_origin_pos.y .. ")\n" ..
                    "Z  Origin = " .. z_origin_text .. "\n" ..
                    "\n" ..
                    "Box Width     = " .. mtl_box.XLength  .. units .."\n" ..
                    "Box Height    = " .. mtl_box.YLength  .. units .. "\n" ..
                    "Box Thickness = " .. mtl_box.ZLength  .. units .. "\n" ..
                    "Box Bottom Left Corner = (" .. mtl_box_blc.x  ..
                                                  "," .. mtl_box_blc.y  ..
                                                  "," ..  mtl_box_blc.z  ..
                                                  ")\n" ..
                    "\n" ..
                    "Test Value = " .. test_val .. "\n" ..
                    "  Depth from absolute test value = " .. depth .. "\n" ..
                    "  Absolute Z from depth test value = " .. abs_z .. "\n" ..
                    "\n"
                    )

end


--[[  --------- main --------------------------------
|
|  Entry point for script
|
]]

function main()

    -- Check we have a job loaded
    job = VectricJob()

    if not job.Exists then
       DisplayMessageBox("No job loaded")
       return false;
    end

    DisplayMaterialSettings()

    return true;
end
```

VectricJob

This is the main object a script writer will interact with and represents the currently open job (file) within the application. The are members of the object which will tell you if there is an existing job open and also to allow you to create a new job.

Properties

.Exists

R/O - bool - returns true if there is an existing job open

.Height

R/O - double - returns height of the job along y - same as YLength

.IsAspire

R/O - bool - returns true if the script is running inside Aspire.

.IsBetaBuild

R/O - bool - returns true if this is a Beta build rather than a release build

.InInches

R/O - bool - returns true if the current units are inches

.InMM

R/O - bool - returns true if the current units are mm

.Job Parameters

R/O - returns the **utParameterList** for the job. The parameter list allows scripts to store persistent data with the job.

.LayerManager

R/O - returns the **CadLayerManager** -for the job. This manages all the layers which in turn hold all the vector objects in the job

.MinX

R/O - double - returns X coordinate of bottom left corner of job

.MinY

R/O - double - returns Y coordinate of bottom left corner of job

.Name

R/O - string - returns the name of the current job WITHOUT the file extension
e.g. ”test” not “test.crv”

.PostProcessorParameters

R/O - returns the **utParameterList** for the last run of a post processor. This will hold information such as the PostP used, the tools used and the actual toolpath files output.

.Selection

R/O - returns the **SelectionList** for the currently selected vectors in the job.

.Width

R/O - double - returns width of the job along x - same as XLength

.XLength
R/O - double - returns length of the job along x - same as XLength

.YLength
R/O - double - returns length of the job along Y - same as Height

Methods

:ClearClipboard()
Clear any data on the clipboard

:CreateHorizontalGuide(double y_value, bool locked)
Create a horizontal guide

y_value - double - y value for guide
locked - bool - if true guide is locked

:CopyToClipboard(string text)
Copies passed text to clipboard. Returns true if copied to clipboard OK else false.

text -string –text to put on clipboard

:CreateVerticalGuide(double x_value, bool locked)
Create a vertical guide

x_value - double - x value for guide
locked - bool - if true guide is locked

:DisplayToolpathPreviewForm()
Display the toolpath preview form for simulation

:ExportSelectionToEps(string pathname)
Exports selected vectors to passed path as an EPS file. Returns true if data exported OK else false.

pathname - string - path to file

:ExportSelectionToDxf(string pathname)
Exports selected vectors to passed path as a DXF file. Returns true if data exported OK else false.

pathname - string - path to file

:GetBuildVersion()
Returns application internal build verion same for Aspire and VCarve Pro e.g. 7.004 for Aspire 4.004 which matches VCarve Pro 7.004

:GetBounds()
Return the 2D bounding box for the drawing area - a **Box2D**

:GetAppVersion()
Returns application version number as a double e.g 4.004 for Aspire V4.0

:GetWrapDiameter()
Returns wrap diameter for a wrapped model

:GroupSelection()
Returns true if the current selection could be grouped

:GuidesVisible()
Returns true if guides are visible else false

:ImportBitmap(string pathname)
Imports a bitmap from passed path. Returns true if file imported OK else false.

pathname - string - path to file

:ImportDxfDwg(string pathname)
Imports a dxf or dwg file from passed path. Returns true if file imported OK else false.

pathname - string - path to file

NOTE: after data is imported it is selected.

:ImportSTLDirect(string pathname)
Directly import (without orientation form) an STL with passed path - automatically creates a component. Returns true if file imported OK else false. NOTE: Aspire only

pathname - string - path to file

:IsWrappedModel()
Return true if this is a wrapped model

:Refresh2DView ()
Refreshes the 2D view, call this after you have created or modified geometry to update the 2d view

:SelectAllVectors()
Returns true if all vectors in the current job could be selected

:SetGuidesVisible(bool visible)
Set visibility of guides

visible - bool - true if guides visible else false

:SetXY_Origin(double origin, double x_offset, double y_offset)
Set the 2D origin for the job - returns true if origin set OK else false.

*origin - **XYOrigin** the four corners or the centre*
x_offset - double - distance in Xof specified point from 0,0
Y_offset - double - distance in Y of specified point from 0,0

NOTE: no data is moved by this call!

:UnGroupSelection(bool deep_ungroup, bool preserve_orig_layers)

Returns true if the current selection could be ungroup

deep_ungroup – boo l – if true ungroup Groups recursively

preserve_orig_layers - bool – if true preserve the original object layers

:WrappingXValues()

Return true if we are wrapping X values - else wrapping Y values

:WrapZOnSurface()

Returns true if z for wrap is on surface of material, else in centre

CadLayerManager

This object is responsible for managing all the layers within the application. A reference to the CadLayerManager is obtained via the *LayerManager* property of the VectricJob. The CadLayerManager maintains a list of all the CadLayers in the job.

Properties

.Count

R/O - integer - Returns the number of layers being managed

.IsEmpty

R/O - bool - Returns true if there are no layers else false

Methods

:FindLayerWithName(string layer_name)

Returns the CadLayer with passed name. If no layer exists with the passed name returns nil

Layer_name - string - Name for layer

:GetActiveLayer()

Returns the current active CadLayer. If no layer exists returns nil

:GetAt(POSITION pos)

Returns the layer at the passed position

pos - POSITION - current position in list

:GetHeadPosition()

Returns a POSITION variable to allow access to the head of the list of layers

:GetLayerWithName(string layer_name)

Returns the CadLayer with passed name. If no layer exists with the passed name a new layer is created

:GetNext(POSITION pos)

Returns the layer at the current position AND a new value for position pointing to the next item in the list (or nil if at end of list)

pos - POSITION - current position in list

Example - note that GetNext(pos) is returning two values ...

```
local pos = layer_manager:GetHeadPosition()
local layer
while pos ~= nil do
    layer, pos = layer_manager:GetNext(pos)
    DO SOMETHING WITH LAYER ....
end
```

:GetPrev(POSITION pos)

Returns the layer at the current position AND a new value for position, pointing to the previous item in the list (or nil if at start of list)

pos - POSITION - current position in list

:GetTailPosition()

Returns a POSITION variable to allow access to the tail of the list of layers

:SetActiveLayer(CadLayer layer)

Sets passed layer to be the current active layer. If passed layer is nil, sets first layer to be active.

Layer - CadLayer - layer to be made the active layer

Example Code

```
function SetBitmapBrightness(job, brightness)

    if not job.Exists then
          DisplayMessageBox("No job loaded")
          return false
     end

    local layer_manager = job.LayerManager


    local pos = layer_manager:GetHeadPosition()
    while pos ~= nil do
       local layer
       layer, pos = layer_manager:GetNext(pos)
       if not layer.IsSystemLayer then
       local layer_pos = layer:GetHeadPosition()
       while layer_pos ~= nil do
          local object
          object, layer_pos = layer:GetNext(layer_pos)
          if object.ClassName == "vcCadBitmap" then
             cad_bitmap = CastCadObjectToCadBitmap(object)
             MessageBox("Found bitmap - brightess = " .. cad_bitmap.Brightness)
             cad_bitmap.Brightness = brightness
             cad_bitmap.Visible = visible
             end
          end
       end  -- end of for each object on layer
    end  -- end of for each layer

    job:Refresh2DView()
```

CadLayer

This object holds all the data for a layer within the application. Layers are created and accessed via the CadLayerManager object. A layer maintains a list of CadObjects.

Properties

.Colour

R/W - 32 bit integer - get / set the colour for a layer as a COLORREF - 32 bit colour value

.Color

R/W - 32 bit integer - get / set the colour for a layer as a COLORREF - 32 bit colour value - this is the U.S spelling version of the method above!

.Count

R/O - integer - Returns the number of CadObjects present on the layer

.IsBitmapLayer

R/O - bool - returns true if this is a bitmap layer. The bitmap layer is used for imported bitmaps and is placed at the start of the layer list so that bitmaps are drawn before vectors.

.IsEmpty

R/O - bool - true if the layer has no objects on it.

.IsSystemLayer

R/O - bool - returns true if this is a system layer. System layers are used for holding items such as toolpath previews.

.Locked

R/W - bool - Get / set the locked property for the layer

.Name

R/W - string - get / set the name for the layer

.Visible

R/W - bool - Get / set the visible property for the layer

Methods

:AddObject (CadObject object, bool on_current_sheet)

Add passed object to this layer

object - CadObject - the object (contour etc) to add to layer - object becomes property of layer

on_current_sheet - bool – this should always be true, and the object is created on the current sheet. If false the sheet from the object would be retained, but as this sheet property is not controllable from Lua, false should not be used in normal operation.

:Find (UUID id)

Returns the position in the list for the object with the passed id. If no object is found the returned position is nil

id - UUID - id for object

:GetHeadPosition()
Returns a POSITION variable to allow access to the head of the list of objects on layer

:GetTailPosition()
Returns a POSITION variable to allow access to the tail of the list of objects on layer

:GetNext(POSITION pos)
Returns the object at the current position AND a new value for position pointing to the next item in the list (or nil if at end of list)

pos - POSITION - current position in list

:GetPrev(POSITION pos)
Returns the object at the current position AND a new value for position, pointing to the previous item in the list (or nil if at start of list)

pos - POSITION - current position in list

:GetAt(POSITION pos)
Returns the object at the passed position

pos - POSITION - position in list

:RemoveObject (CadObject object)
Removes passed object from this layer. Object becomes property of the script. Returns object removed or nil if object was not found on layer.

Object - CadObject - the object (contour ec) to remove from layer - object becomes property of caller

:RemoveAt (POSITION pos)
Removes the CadObject at the passed position and returns it.

pos - POSITION - position in list

:SetColour(double red, double green, double blue)
Set the colour for the layer

red - double - red component of colour in range 0.0 - 1.0
green - double - red component of colour in range 0.0 - 1.0
blue - double - red component of colour in range 0.0 - 1.0

:SetColor (double red, double green, double blue)

Set the color for the layer (U.S spelling)

red - double - red component of colour in range 0.0 - 1.0
green - double - red component of colour in range 0.0 - 1.0
blue - double - red component of colour in range 0.0 - 1.0

CadObject

This object is the base class for all the different types of objects which can appear on a layer within the application. CadObject is derived from the Object class which has only one method - ClassName. The objects derived from CadObject can include polylines, groups, text, bitmaps, toolpath previews etc.

A huge variety of named parameters can be associated with every object within the program and accessed by scripts.

Properties

.ClassName

R/O - string - the returns the name of the class within the application that the object represents.

Values for this property include (but there are many more) ...

vcCadBitmap
vcCadContour
vcCadObjectGroup
etc.

.Id

R/W - luaUUID - The unique identifier for this object - it is EXTREMELY important that you do not set two different objects to have the same id!

.IsBitmap

R/O - bool - true if this object is a bitmap or derived from a bitmap

.IsSelected

R/O - bool - true if this object is selected. The selected state is set when an object is added to the jobs selection list.

.IsVisible

R/W - bool - Get / Set the visibility state for an object. Normally the visibility of objects within the application is controlled via its layer. Be very careful not to create invisible objects from scripts which the user cannot then make visible.

.IsLocked

R/W - bool - Get / Set the locked state for an object. Normally the locked state of objects within the application is controlled via its layer. Be very careful not to create locked objects from scripts which the user cannot then make unlocked.

.LayerId

R/O - luaUUID - Id of layer this object belongs to. This cannot not be modified directly. To add objects to layers or move them between layers the methods of CadLayer must be used.

.RawId

R/W - UUID - The unique identifier for this object - it is EXTREMELY important that you do not set two different objects to have the same id!

.RawLayerId
R/W - UUID - The unique identifier for the layer this object belongs to.

.SheetIndex
R/W - Integer - The id of the sheet this object is on. 0 is the default sheet, the rest are numbered 1 to max number of sheets. Be careful not to set the sheet index of an object to an invalid sheet.

Methods

:CanTransform(integer flags)
Returns true if object can be transformed. The value of flag is made up by combining the following values ...

```
1,      // can object be moved ?
2,      // can object be rotated ?
4,      // can object be mirrored ?
8,      // can object be scaled symetrically ?
16      // can object be scaled asymetrically ?
```

Flags - integer - value indicating the type of valid transform being queried

:Clone ()
Return a new copy of this object, with all new id's. The returned object is not on a layer so is not yet part of the job.

:GetBoundingBox ()
Returns a Box2D representing the 2d bounds of this object

:GetContour ()
Return the Contour representation for this object - not all objects will have a contour representation. E.g a grouped object will return nil for this call.

:InvalidateBounds()
Invalidates the cached bounding box for this object. This should be called if you modify the raw data for the object.

:SetLayer (CadLayer layer)
Adds this object to the passed layer

layer - CadLayer - layer this object should belong to

:Transform(Matrix2D matrix)
Transform the object using the passed transformation matrix. The transformation matrix can move (translate), scale or rotate or a combination of all 3.

matrix - Matrix2D - the transformation matrix to apply to the object

:GetBool(string parameter_name, bool default_value, bool create_if_not_exist)

Retrieve a Boolean flag (true / false) with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
default_value - bool - the value which will be returned if there is no existing value stored
create_if_not_exist - bool - if true the default value will be stored in list if no existing value

:GetDouble(string parameter_name, double default_value, bool create_if_not_exist)

Retrieve a double with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
default_value - double - the value which will be returned if there is no existing value stored
create_if_not_exist - bool - if true the default value will be stored in list if no existing value

:GetInt(string parameter_name, integer default_value, bool create_if_not_exist)

Retrieve an integer with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
value - integer - the value which will be returned if there is no existing value stored
create_if_not_exist - bool - if true the default value will be stored in list if no existing value

:ParameterExists(string parameter_name, utParameterType type)

Returns true if there is an existing parameter with passed name and type.

parameter_name - string - the name of the parameter
type - utParameterType - the type of parameter

:SetBool(string parameter_name, bool value)

Store a Boolean flag (true / false) with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - bool - the value which will be stored in the parameter list

:SetDouble(string parameter_name, double value)

Store a double with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - double - the value which will be stored in the parameter list

:SetInt(string parameter_name, integer value)

Store an integer with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - integer - the value which will be stored in the parameter list

:SetString(string parameter_name, string value)

Store a string with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - string - the value which will be stored in the parameter list

CadContour

This object is derived from CadObject and represents a single vector within the job. A CadContour holds a Contour object which contains the actual data for the vector shape. A CadContour is created from a Contour object using the *CreateCadContour*() method. To access the Contour data from a CadContour object, use the :GetContour() method from the base CadObject class.

Properties

All the properties for this object are documented in the CadObject base class

Methods

This object inherits all documented methods from the CadObject base class with the following additions

:InsertToolpathAtTabAtPoint(Point2D point)

This method inserts a toolpath tab at the point on the CadContour closest to the specified point. It returns a ToolpathTab object.

point - Point2D - The point where to place the toolpath tab

CadObjectGroup

This object is derived from CadObject and represents a group of CadObjects within the job.

Properties

This object inherits all documented properties from the CadObject base class with the following additions

.Count

R/O – integer - returns the number of CadObjects in the group

.IsEmpty

R/O – bool – returns true if the CadObjectGroup is empty

Methods

This object inherits all documented methods from the CadObject base class with the following additions

:GetAt(POSITION pos)

Returns the object at the current position

pos – *POSITION* – current position in list

:GetHead ()

Returns the object at the current position AND a new value for position pointing to the next item in the group (or nil if at end of group)

:GetHeadPosition()

Returns a POSITION variable to allow access to the head of the group

:GetNext(POSITION pos)

Returns the object at the current position AND a new value for position pointing to the next item in the group (or nil if at end of group)

pos – *POSITION* – current position in group

:GetPrev(POSITION pos)

Returns the object at the current position AND a new value for position, pointing to the previous item in the group (or nil if at start of group)

pos – *POSITION* – current position in list

:GetTailPosition()

Returns a POSITION variable to allow access to the tail (end) of the group

CadBitmap

This object represents an imported bitmap in the program. It is identified by the object.ClassName property == “vcCadBitmap”. To convert a ‘CadObject’ to a ‘CadBitmap’ use the helper function CastCadObjectToCadBitmap

Properties

.Brightness

R/W - integer - value in range 0 - 255

External methods

CastCadObjectToCadBitmap(CadObject object)

Return a CadBitmap object for the passed CadObject which must have been identified as being a CadBitmap.

e.g

```
function SetBitmapBrightness(job, brightness)

    if not job.Exists then
          DisplayMessageBox("No job loaded")
          return false
     end

    local layer_manager = job.LayerManager

    local pos = layer_manager:GetHeadPosition()
    while pos ~= nil do
       local layer
       layer, pos = layer_manager:GetNext(pos)
       if not layer.IsSystemLayer then
       local layer_pos = layer:GetHeadPosition()
       while layer_pos ~= nil do
          local object
          object, layer_pos = layer:GetNext(layer_pos)
          if object.ClassName == "vcCadBitmap" then
             cad_bitmap = CastCadObjectToCadBitmap(object)
             MessageBox("Found bitmap - brightess = " .. cad_bitmap.Brightness)
             cad_bitmap.Brightness = brightness
             cad_bitmap.Visible = visible
             end
          end
       end  -- end of for each object on layer
    end  -- end of for each layer

    job:Refresh2DView()
end
```

CadPolyline

This object represents a vector in the program which has been created by the polyline drawing tool. It has all the same properties and methods as a CadContour and can be treated as one. It is identified by the object.ClassName property == “vcCadPolyline”. To convert a ‘CadObject’ to a ‘CadPolyline’ use the helper function CastCadObjectToCadPolyline

CadMarker

This object represents a 'marker' in the program which can be used to label items. CadMarkers are not machinable or selectable. It is derived from CadObject and has all the properties of CadObject available.

Constructor

CadMarker(string text, Point2D pt, integer pixel_size) - Constructor

A CadMarker at passed position

text -string – text for marker object
pt -Point2D – position for marker
pixel_size pt -integer – size marker is drawn at in pixels

Properties

.Text

R/W – string – Text displayed at marker

.Position

R/W – Point2D – Position for marker

Methods

:SetColor(double red, double green, double blue)

Set colour for marker

red - double – red value for colour in range 0-1.0
green - double – gree value for colour in range 0-1.0
blue - double – blue value for colour in range 0-1.0

Example Code

```
--[[  ------------------ MarkContourNodes ----------------------------
|
|  Insert a marker at each node on the passed contour
|
]]
function MarkContourNodes(contour, layer)

   local num_spans = 0
   local ctr_pos = contour:GetHeadPosition()
   while ctr_pos ~= nil do
      local span
      span, ctr_pos = contour:GetNext(ctr_pos)
      -- create a marker at start of span
      local marker = CadMarker("V:" .. num_spans, span.StartPoint2D, 3)
      layer:AddObject(marker, true)
      num_spans = num_spans + 1;
   end
   -- if contour was open mark last point
   if contour.IsOpen then
      local marker = CadMarker("V:" .. num_spans, contour.EndPoint2D, 3)
      layer:AddObject(marker, true)
   end
end
```

utParameterList

This object is used to store strings, doubles and integers as 'properties' of an object such as the job. The parameter has a name such as "DocumentCreator", a type which is either string, double (a floating point number such as 1.2345) or an integer (a whole number such as 7) and the value you assign for the parameter.

For the job you would use the .JobParameters property to access the ParameterList for the job.

Methods

:GetDouble(string parameter_name, double default_value, bool create_if_not_exist)

Retrieve a double with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
default_value - double - the value which will be returned if there is no existing value stored
create_if_not_exist - bool - if true the default value will be stored in list if no existing value

:GetInt(string parameter_name, integer default_value, bool create_if_not_exist)

Retrieve an integer with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
value - integer - the value which will be returned if there is no existing value stored
create_if_not_exist - bool - if true the default value will be stored in list if no existing value

:ParameterExists(string parameter_name, utParameterType type)

Returns true if there is an existing parameter with passed name and type.

parameter_name - string - the name of the parameter
type - utParameterType - the type of parameter

:SetBool(string parameter_name, bool default_value, bool create_if_not_exist)

Retrieve a Boolean flag (true / false) with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
default_value - bool - the value which will be returned if there is no existing value stored
create_if_not_exist - bool - if true the default value will be stored in list if no existing value

:SetBool(string parameter_name, bool value)

Store a Boolean flag (true / false) with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - bool - the value which will be stored in the parameter list

:SetDouble(string parameter_name, double value)

Store a double with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - double - the value which will be stored in the parameter list

:SetInt(string parameter_name, integer value)

Store an integer with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - integer - the value which will be stored in the parameter list

:SetString(string parameter_name, string default_value, bool create_if_not_exist)

Retrieve a string with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
default_value - string - the value which will be returned if there is no existing value stored
create_if_not_exist - bool - if true the default value will be stored in list if no existing value

ParameterList.UTP_DOUBLE
ParameterList.UTP_INT
ParameterList.UTP_BOOL
ParameterList.UTP_STRING

:SetString(string parameter_name, string value)

Store a string with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - string - the value which will be stored in the parameter list

luaUUID

Every object in the job has a unique id refered to as a UUID. As these id's are difficult to save and process from lua, the luUUID object wraps a windows UUID and lets it be treated as a string.

Constructor

luaUUID() - Constructor

A new luaUUID with a new unique id.

Properties

.IsEmpty

R/O - bool – returns true if this is the NULL / Empty id

.RawId

R/O - UUID – returns the raw UUID this object wraps up

Methods

:MakeEmpty()

Makes this id equal to the 'empty' id

:CreateNew()

Make the id hold a new unique value

:AsString()

Return a string representation of the id

:Set(string text_id)

Set the id using a string representation of the id

text_id – *string* – text representation of an id

:SetId(UUID raw_id)

Set the id using raw id

raw_id –*UUID* – id

:IsEqual(luaUUID id)

Return true if passed id is the same as this one

id –*UUID* – id to compare

UUID_List

Object which holds a list of UUID's (note NOT luaUUID's these are 'raw' UUID's).

Constructor

UUID_List() - Constructor

Creates a new empty UUID_List

Local uuid_list = UUID_List()

Properties

.Count

R/O -integer – return the number of ids in the list

.IsEmpty

R/O -bool – return true if the id list is empty

Methods

:AddHead(UUID id)

Add the passed id to the front of the list

id –*UUID* –id to put in list

:AddTail(UUID id)

Add the passed id to the end of the list

id –*UUID* –id to put in list

:GetAt(pos)

Returns the id at the current position

pos – *POSITION* – current position in list

:GetHeadPosition()

Returns a POSITION variable to allow access to the head of the list

:GetNext(POSITION pos)

Returns the id at the current position AND a new value for position pointing to the next item in the list (or nil if at end of list)

pos – *POSITION* – current position in list

Example - note that GetNext(pos) is returning two values ...

```
local pos = uuid_list:GetHeadPosition()
local id
while pos ~= nil do
    id, pos = uuid_list:GetNext(pos)
    DO SOMETHING WITH ID ....
end
```

:GetPrev(POSITION pos)

Returns the toolpath at the current position AND a new value for position, pointing to the previous item in the list (or nil if at start of list)

pos – *POSITION* – current position in list

:GetTailPosition()

Returns a POSITION variable to allow access to the tail (end) of the list toolpaths

:Find(UUID id)

Return the POSITION of passed id in list. Returns nil if id not in list position

id –*UUID* –id to find in list

:FindIndex(UUID id)

Return the index of passed id in list. Returns -1 if id not in list position

id –*UUID* –id to find in list

:InsertAfter(POSITION pos, UUID id)

Insert the id after the passed position

pos – *POSITION* –position in list

id –*UUID* –id to put in list

:InsertBefore(POSITION pos, UUID id)

Insert the id before the passed position

pos – *POSITION* –position in list

id –*UUID* –id to put in list

:RemoveAll()

Remove all ids from the list

:RemoveAt(POSITION pos)

Remove the id at the current position

pos – *POSITION* – current position in list

:RemoveHead()

Remove the id at the front of the list and return it

:RemoveTail()

Remove the id at the end of the list and return it

:SetAt(POSITION pos, UUID id)

Replace the id at the passed position with the new value

pos – *POSITION* –position in list

id –*UUID* –id to put in list

CadObjectList

This object holds a list of CadObjects

Constructor

CadObjectList(bool owns_objects) - Constructor

Creates a new CadObjectList

owns_objects -bool – true if this list owns the objects it contains. If this is true and the list still contains objects when it goes out of scope the objects will be deleted.

Properties

.Count

R/O -integer – return the number of objects in the list

.IsEmpty

R/O -bool – return true if the list is empty

Methods

:AddHead(CadObject obj)

Add the passed object to the front of the list

obj –*CadObject* –object to put in list

:AddTail(CadObject obj)

Add the passed object to the end of the list

obj –*CadObject* –object to put in list

:CanTransform(integer flags)

Return true if object can be transformed

flags –*integer* –value indicating type of transform – these can be combined e.g 3 (1+2) will check for both Move and Rotate

1 = Move
2 = Rotate
4 = Mirror
8 = Scale Symetric
16 = Scale Asymetric

:GetAt(POSITION pos)

Returns the object at the current position

pos – *POSITION* – current position in list

:GetHeadPosition()

Returns a POSITION variable to allow access to the head of the list

:GetNext(POSITION pos)

Returns the object at the current position AND a new value for position pointing to the next item in the list (or nil if at end of list)

pos – *POSITION* – current position in list

Example - note that GetNext(pos) is returning two values ...

```
local pos = obj_list:GetHeadPosition()
local object
while pos ~= nil do
    object, pos = obj_list:GetNext(pos)
    DO SOMETHING WITH OBJECT ....
end
```

:GetPrev(POSITION pos)

Returns the object at the current position AND a new value for position, pointing to the previous item in the list (or nil if at start of list)

pos – *POSITION* – current position in list

:GetTailPosition()

Returns a POSITION variable to allow access to the tail (end) of the list

:RemoveHead()

Remove the id at the front of the list and return it

:RemoveTail()

Remove the id at the end of the list and return it

:Transform(Matrix2D xform)

Transform all objects in list by passed transform

xform – *Matrix2D* – transformation matrix to apply to objects

SelectionList

This object holds a list of CadObjects representing the current selection in the program.

The selection is accessed via the job …

```
-- Check we have a document loaded
local job = VectricJob()

if not job.Exists then
   DisplayMessageBox("No job loaded")
   return false;
end


local selection = job.Selection
if selection.IsEmpty then
    MessageBox("Please select one or more vectors to label")
    return false
end
```

Properties

.Count

R/O -integer – return the number of objects in the list

.IsEmpty

R/O -bool – return true if the list is empty

Methods

:Add (CadObject obj, bool add_tail, bool group_add)

Add the passed object to the list
obj –*CadObject* –object to put in list
add_tail –*bool* –if true add to end of seelction else insert at start
group_add –*bool* –if you are adding a lot of objects to the selection, set this true and call :GroupSelectionFinished() when you have added all the objects. This stops a lot of uneccesary updating while you are updating the selection.

:AddHead(CadObject obj)

Add the passed object to the front of the list
obj –*CadObject* –object to put in list

:AddTail(CadObject obj)

Add the passed object to the end of the list
obj –*CadObject* –object to put in list

:CanTransform(integer flags)

Return true if object can be transformed - these can be combined e.g 3 (1+2) will check for both Move and Rotate

flags –*integer* –value indicating type of transform –

1 = Move
2 = Rotate
4 = Mirror
8 = Scale Symetric
16 = Scale Asymetric

:CanSelect(CadObject obj)

Return true if passed object can be selected

obj –*CadObject* –object to test

:Clear()

Clear the selection (list is emptied)

:GetAt(POSITION pos)

Returns the object at the current position

pos – *POSITION* – current position in list

:GetBoundingBox()

Return a Box2D with the bounds of the selected objects

:GetHeadPosition()

Returns a POSITION variable to allow access to the head of the list

:GetNext(POSITION pos)

Returns the object at the current position AND a new value for position pointing to the next item in the list (or nil if at end of list)

pos – *POSITION* – current position in list

Example - note that GetNext(pos) is returning two values ...

```
local pos = obj_list:GetHeadPosition()
local object
while pos ~= nil do
    object, pos = obj_list:GetNext(pos)
    DO SOMETHING WITH OBJECT ....
end
```

:GetPrev(POSITION pos)

Returns the object at the current position AND a new value for position, pointing to the previous item in the list (or nil if at start of list)

pos – *POSITION* – current position in list

:GetTailPosition()
Returns a POSITION variable to allow access to the tail (end) of the list

:GroupSelectionFinished()
Call this if you have been using :Add(…) with group_add set to true to update the application when you have finished adding objects to the selection.

:RemoveOnLayer(UUID layer_id)
Remove any objects from the selection which are on the layer with the passed id. The objects are just removed from the selection, they are not deleted are changed in any other way.

:Remove(CadObject obj, bool group_remove)
Remove the passed object from the selection
obj –*CadObject* –object to put in list
group_remove –*bool* –if you are removing a lot of objects to the selection, set this true and call :GroupSelectionFinished() when you have removed all the objects. This stops a lot of uneccesary updating while you are updating the selection.

:RemoveHead()
Remove the id at the front of the list and return it

:RemoveTail()
Remove the id at the end of the list and return it

:Transform(Matrix2D xform)
Transform all objects in list by passed transform

xform – *Matrix2D* – transformation matrix to apply to objects

Creating Vectors From Script

The previous objects (VectricJob, CadLayer, CadContour etc) represent high level objects used within the program. When the user wants to create new geometry from within a script, they will be working with lower level entities. These entities are Spans, Contours and Contour Groups.

A Span is the lowest level piece of geometry and it always has a start and end point. It is important to realize that although geometry in the main program is always 2D, the underlying reprentation of Spans and Contours supports full 3d values. This means that when Contours are used to define toolpaths, full 3D tool moves can be represented. If you are simply creating geometry which will appear in the 2D view in the main program, the Z value for all points should be left at 0.0.

Contour

This object represents a single vector within the application. Open and closed vectors, arcs, circles, all are based on a Contour with one or more spans. The Contour is represented as a list of spans, with a span being either a Line, Arc or Bezier.

Constructors

Contour(tolerance) - Constructor

A new contour is created within a Lua script using this contructor method e.g

```
local contour = Contour(0.0)
```

Tolerance - double - tolerance to use when performing operations - use 0.0 for default

Properties

.Area

R/O - double - unsigned area of contour

.BoundingBox2D

R/O - Box2D - 2D bounding box for contour

.BoundingBox3D

R/O - - Box3D - 3D bounding box for contour

.Count

R/O - integer - number of spans in the contour

.ContainsArcs

R/O - bool - true if contour contains 1 or more arcs - may contain other span types as well.

.ContainsBeziers

R/O - bool - true if contour contains 1 or more beziers - may contain other span types as well.

.CentreOfGravity

R/O - Point2D - the centre of gravity for the contour

.EndPoint3D

R/O - Point3D - the 3D position of the contour end point - if contour is closed this will be the same as the start point.

.EndPoint2D

R/O - Point2D - the 2D position of the contour end point - if contour is closed this will be the same as the start point.

.IsClockwise
R/O - bool - Direction of the contour, this is found from the signed area of the contour. For an open contour, it is assumed to be closed with a straight line when calculating the area.

. IsAntiClockwise
R/O - bool - Direction of the contour, this is found from the signed area of the contour. For an open contour, it is assumed to be closed with a straight line when calculating the area.

. IsCW
R/O - bool - Direction of the contour, this is found from the signed area of the contour. For an open contour, it is assumed to be closed with a straight line when calculating the area.

. IsCCW
R/O - bool - Direction of the contour, this is found from the signed area of the contour. For an open contour, it is assumed to be closed with a straight line when calculating the area.

.IsClosed
R/O - bool - True if the contour is closed

.IsEmpty
R/O - bool - true if contour is empty - no spans

.IsOpen
R/O - bool - True if contour is open

.IsSinglePoint
R/O - bool - true if contour is just a single point

.Length
R/O - double - length of contour

.StartPoint3D
R/O - Point3D - the 3D position of the contour start point

.StartPoint2D
R/O - Point2D - the 2D position of the contour start point

.Tolerance
R/W - double - the tolerance value used when polygonising beziers and arcs. This should not be changed without very good reason.

Methods

:AppendPoint(double x, double y)

Set the 2D starting point for the contour. Returns true if point set OK else false. This method should only be called on an EMPTY contour. The Z value for the point defaults to 0.0

x - double - X value for start point of contour
y - double - Y value for start point of contour

:AppendPoint(double x , double y, double z)

Set the 3D starting point for the contour. Returns true if point set OK else false. This method should only be called on an EMPTY contour.

x - double - X value for start point of contour
y - double - Y value for start point of contour
z - double - Z value for start point of contour

:AppendPoint(Point2D pt2d)

Set the 2D starting point for the contour. Returns true if point set OK else false. This method should only be called on an EMPTY contour. The Z value for the point defaults to 0.0

Pt2d - Point2D - 2D coordinate for start point of contour

:AppendPoint(Point3D pt3d)

Set the 3D starting point for the contour. Returns true if point set OK else false. This method should only be called on an EMPTY contour.

Pt2d - Point3D - 3D coordinate for start point of contour

:ArcTo (Point2D end_pt2d, Point2D center_pt2d, boolccw)

Append an arc span from the current end point of the contour to the passed position. Returns true if span added t OK else false. The arc is defined by the existing end point in the contour, the center point of the arc and the direction of rotation. Arcs > 180 degrees will be split into two arcs.

end_pt2d - Point2D - 2D coordinate for end point of arc
center_pt2d - Point2D - 2D coordinate for center point of arc
ccw - bool - true if arc is counter clockwise from start point else clockwise

:ArcTo (Point2D end_pt2d, bool bulge)

Append an arc span from the current end point of the contour to the passed position. Returns true if span added t OK else false. This method should only be called on a **non-empty** contour. Arcs > 180 degrees will be split into two arcs.

end_pt2d - Point2D - 2D coordinate for end point of arc
bulge - double - signed bulge factor for arc

:ArcTo(Point3D end_pt3d, bool bulge)

Append an arc span from the current end point of the contour to the passed position. Returns true if span added OK else false. This method should only be called on a **non-empty** contour. Arcs > 180 degrees will be split into two arcs.

end_pt3d - Point3D - 3D coordinate for end point of arc - arcs must be in XY plane
bulge - double - signed bulge factor for arc

:AppendContour (Contour ctr)

Append passed contour to the end of this contour. Returns true if contour appended OK, false if contour wouldn't append within tolerance

ctr - Contour - Contour to append

:AppendSpan (Span span)

Append passed span to the end of this contour. Returns true if span appended OK, false if span wouldn't append within tolerance

span -Span – Span to append

:AppendSpanAsLines(Span span, double tolerance)

Append passed span as line spans - passed span is NOT modified or adopted. Returns true if span appended OK, false if span wouldn't append within tolerance

span - Span - Span to append polygonised version of
tolerance - double - tolerance to use when converting span to lines

:BezierTo(Point2D end_pt2d, Point2D ctrl_1_2d, Point2D ctrl_2_2d)

Append a bezier span from the current end point of the contour to the passed position. Returns true if span added OK else false. All points must have same z value

end_pt2d - Point2D - 2D coordinate for end point of bezier
ctrl_1_2d - Point2D - 2D coordinate for first control point of bezier
ctrl_2_2d - Point2D - 2D coordinate for second control point of bezier

:BezierTo(Point3D end_pt3d, Point2D ctrl_1_2d, Point2D ctrl_2_2d)

Append a bezier span from the current end point of the contour to the passed position. Returns true if span added OK else false. All points must have same z value

end_pt3d - Point3D - 3D coordinate for end point of bezier
ctrl_1_2d – Point2D - 2D coordinate for first control point of bezier
ctrl_2_2d – Point2D - 2D coordinate for second control point of bezie

:CanAppendContour(Contour ctr)
Return true if passed contour can be appended onto the end of this contour

ctr - Contour - Contour to check if can append

:CanAppendSpan (Span span)
Return true if passed span can be appended onto the end of this contour

span - Span - Span to check if can append

:Clone()
Return a new Contour which is an exact copy of this one.

:CreateTolerancedCopy(double tolerance)
Return a new Contour which has been toleranced (simplified) to the passed value.
NOTE: The contour must consist of just line spans, it must NOT contain beziers or arcs, the properties .ContainsBeziers and .ContainsArcs must be false.

tolerance - double - tolerance to use when simplifying contour.

:CreatePolygonizedCopy(double tolerance, double max_line_len)
Return a new Contour which has been toleranced (simplified) to the passed value. Any bezier or arc spans are polygonized before tolerancing. The max_line_len parameter can be used to limit the maximum length of a tolerance span.

tolerance - double - tolerance to use when simplifying contour. 0.0 means use default
max_line_len - double - maximum length of a tolerance span , 0.0 means no limit

:GetFirstSpan()
Return the first Span in the contour

:GetLastSpan()
Return the last Span in the contour

:GetHeadPosition()
Returns a POSITION variable to allow access to the head of the list of spans in the contour

:GetTailPosition()
Returns a POSITION variable to allow access to the head of the list of spans in the contour

:GetNext(POSITION pos)
Returns the Span at the current position AND a new value for position pointing to the next item in the list (or nil if at end of list). E.g

```
cur_span, pos = offset_vector:GetNext(pos)
```

pos - POSITION - current position in list

:GetPrev (POSITION pos)

Returns the Span at the current position AND a new value for position pointing to the previous item in the list (or nil if at start of list). E.g

```
cur_span, pos = offset_vector:GetPrev(pos)
```

pos - POSITION - current position in list

:GetAt (POSITION pos)

Returns the Span at the current position in the list. E.g .

```
cur_span = offset_vector:GetAt(pos)
```

pos - POSITION - current position in list

:IsPointInside (Point2D point, double tolerance)

Return true if passed point is inside this contour. Beziers and arcs are polygonized for this calculation. If passed tolerance = 0.0 the default tolerance is used, else the specified tolerance. If this is called for an open contour the return value is undefined.

point - Point2D - 2D coordinate if point to check
tolerance - double - tolerance for polygonisation - 0.0 means default

:InvalidateBoundingBox()

If you modify the data in the contour, you must call this method to invalidate the cached bounding box.

:LineTo(double x , double y)

Append a line span from the current end point of the contour to the passed position. Returns true if span added OK else false. This method should only be called on a **non-empty** contour (use :AppendPoint() to start the contour if you want to create a contour consisting of a series of line segments). The Z value for the point defaults to 0.0

x - double - X value for point
y - double - Y value for point

:LineTo(double x , double y, double z)

Append a line span from the current end point of the contour to the passed position. Returns true if span added OK else false. This method should only be called on a **non-empty** contour (use :AppendPoint() to start the contour if you want to create a contour consisting of a series of line segments).

x - double - X value for point
y - double - Y value for point
z - double - Z value for point

:LineTo(Point2D end_pt2d)

Append a line span from the current end point of the contour to the passed position. Returns true if span added t OK else false. This method should only be called on a **non-empty** contour. The Z value for the point defaults to 0.0

end_pt2d - Point2D - 2D coordinate for point

:LineTo(Point3D end_pt3d)

Append a line span from the current end point of the contour to the passed position. Returns true if span added OK else false. This method should only be called on a **non-empty** contour.

end_pt3d - Point3D - 3D coordinate for point

:MakeOffsetsSquare(double offset_dist, double offset_out, double max_dist)

Process contour and change any arcs which match the passed radius into 'square' offset if arc matches passed parameters. This is called AFTER a normal offset offset operation to change normal rounded corners into ‘sharp’ corners.

offset_dist - double - the distance contour was offset by - corners will have this radius
offset_out - bool - true if contour was offset outwards
max_dist - double - maximum distance to allow square offset to extend (acute corners can have square offsets which extend a LONG way out)

:OffsetInZ(double z_offset)

Add the passed z value to the Z coordinates for all spans in the contour

z_val - double - z value to add to all span heights

:ReorderStartPoint(Point2D point)

Reorder the start point of a closed contour to be as close as possible to the passed point. No points are inserted into the contour, the existing spans are just reordered.

point - Point2D - 2D coordinate of point to define contour start

:ReorderStartPoint (integer span_index, double parameter)

Reorder a closed the contour, inserting a point (and hence splitting a span if necessary).

span_index - integer - index of span holding new start point
parameter - double - parameter in range 0-1.0 of point on span

:Reverse ()

Reverse the direction of the contour - the start becomes the end and the direction of all spans are reversed.

:SetZHeight(double z_val)

Set the Z coordinates for all spans in contour to the passed value

z_val - double - z height for contour

:Smash (double tolerance, bool preserve_arcs)

Smash beziers (and optionally arcs) into straight lines within the passed tolerance.

tolerance - double - tolerance to use when polygonizing beziers and arcs
preserve_arcs - bool - if true arcs aren't smashed, only beziers

:Transform (Matrix2D xform)

Transform the contour using the passed transformation matrix. The transformation matrix can move (translate), scale or rotate or a combination of all 3.

matrix - Matrix2D - the transformation matrix to apply to the contour

:RemoveHead ()

Remove the first span in the contour and return it

:RemoveTail ()

Remove the last span in the contour and return it

ContourGroup

This object represents a group of Contour objects. The contours are held in a double linked list which can be iterated through using the GetHeadPosition/ GetTailPosition and GetNext / GetPrev methods.

Constructor

ContourGroup(bool owns_contours) - Constructor

Create a new empty ContourGroup object.

owns_contour -bool -If true, this object will own all the contours placed inside it and will delete them when the group is deleted. If this flag is false the contours in the group will not be deleted when the group is deleted.

Properties

.BoundingBox2D

R/O - Box2D - returns the 2D bounding box for all contours in group

.BoundingBox3D

R/O - Box3D - returns the 3D bounding box for all contours in group

.Count

R/O - integer - returns the number of contours in the group

.ClosedCount

R/O- integer - returns the number of closed contours in the group

.IsEmpty

R/O - bool - true if the group is empty - doesn't contain any contours

.OpenCount

R/O - integer - returns the number of open contours in the group

.OwnsContours

R/O - bool - true if the group owns the contours it holds

Methods

:AppendGroup(ContourGroup group)

Append the contours in the passed group onto the end of this group. Passed group is deleted.

group -ContourGroup – Contour Group object being added to group

:AddHead(Contour ctr)

Add passed contour to the Front / Head / Start of this group

ctr -Contour - Contour object being added to group

:AddTail(Contour ctr)
Add passed contour to the Back / Tail / End of this group

ctr -Contour - Contour object being added to group

:Clone()
Return a copy of this group and its contents. The new group will own its copies off the contours.

:CreateToleranceCopy(double tolerance)
Create a new controur group with all arcs and beziers in the original group replaced with straight line segments which match the original spans within the passed tolerance.

tolerance -double -max deviation from original contour permitted when replacing arcs and Bezier with straight lines.

:ContainsBeziers()
Returns true if one or more contours in the group contains one or more Bezier spans

:DeleteOpenVectors()
Delete all open vectors in group

:GetLength()
Return the length of all contours in the group

:GetHeadPosition()
Return a POSITION object pointing to the first Contour in the group which can be used to iterate through the group contents

:GetTailPosition()
Return a POSITION object pointing to the last Contour in the group which can be used to iterate through the group contents

:GetNext(POSITION pos)
Returns BOTH the contour at the current position and a new pos value pointing to the next contour in the group. If the end of the list has been reached the position will be returned as nil.

pos-POSITION - Current position in the group to get contour from

Example Code

```
-- iterate through the group and contours in group
   local count = 0
   local num_spans = 0
   local span_length = 0.0
   local tolerance = 0.0001

   local pos = group:GetHeadPosition()
   while pos ~= nil do
      local contour
      contour , pos = group:GetNext(pos)

      -- iterate through each span in the contour
      local ctr_pos = contour:GetHeadPosition()
      while ctr_pos ~= nill do
         local span
         span, ctr_pos = contour:GetNext(ctr_pos)
         num_spans = num_spans + 1;
         span_length = span_length + span:GetLength(0.0001)
      end

      count = count + 1
   end
```

NOTE: `contour , pos = group:GetNext(pos)`

- both the **contour** and a new value for **pos** are returned by the method

:GetPrev(POSITION pos)

Returns BOTH the contour at the current position and a new pos value pointing to the previous contour in the group. If the start of the list has been reached the position will be returned as nil.

pos-POSITION - Current position in the group to get contour from

Example Code

```
-- iterate through the group and contours in group
   local span_length = 0.0
   local tolerance = 0.0001

   local pos = group:GetTailPosition()
   while pos ~= nil do
      local contour
      contour , pos = group:GetPrev(pos)

      -- iterate through each span in the contour
      local ctr_pos = contour:GetTailPosition()
      while ctr_pos ~= nill do
         local span
         span, ctr_pos = contour:GetPrev(ctr_pos)
         span_length = span_length + span:GetLength(0.0001)
      end

   end
```

NOTE: `contour , pos = group:GetPrev(pos)`

- both the **contour** and a new value for **pos** are returned by the method

:GetAt(POSITION pos)

Returns the contour at the current position. Pos is left unchanged

pos-POSITION - Current position in the group to get contour from

:GetHead()
Return the contour at the head of the list, the contour remains in the group

:GetTail()
Return the contour at the tail of the list, the contour remains in the group

:MakeOffsetsSquare(double offset_dist, bool offset_out, double max_dist)
Used to 'square' offsets on external corners after an offset operation.

> *offset_dist* -double - distance contours in group were offset
> *offset_out* - bool - if true the offset operation was an offset outwards
> *max_dist* -double - max distance to allow square corners to extend out.
> Offsets of acute corners can extend a long way from the original vector.

:Offset(double dist, double stepover, integer num_offsets, bool preserve_arcs)
Create a new contour group generated by offseting the contours in this group. The group should only contain closed non-overlapping vectors.

> *dist* -double -distance to offset by. +ve values will offset outwards, -ve values inwards
> *stepover* -double -if doing multiple offset, this is the value for the 2nd and subsequent offsetd
> *num_offsets* -integer -number of offsets to perform
> *preserve_arcs* -bool -if true arcs are preserved, else arcs replaced by approximated lines

:OffsetInZ(double dist)
Add the passed z dist to the z height of every span in every contour in group

> *dist*-double -z value to add to every span in every contour in group

:OffsetWithOpenVectors(double dist, double stepover, integer num_offsets, bool preserve_arcs)
Create a new contour group generated by offseting the contours in this group. If the group contains open vectors these are offset individually and merged back into the final offset. When open offsets are merged back in this may result in crossing vectors.

> *dist* -double -distance to offset by. +ve values will offset outwards, -ve values inwards
> *stepover* -double -if doing multiple offset, this is the value for the 2nd and subsequent offsetd
> *num_offsets* -integer -number of offsets to perform
> *preserve_arcs* -bool -if true arcs are preserved, else arcs replaced by approximated lines

:RemoveAt(POSITION pos)
Returns the contour at the current position, removing it from the group. Pos is INVALID after this call and should not be used again.

> *pos*-POSITION - Current position in the group to remove contour from

:RemoveHead()
Remove the first contour in the group and return it

:RemoveTail()
Remove the last contour in the group and return it

:Reverse()
Reverse the direction of every contour in the group

:SetZHeight(double z_val)
Set the z height of every span in every contour to the passed value.

> *z_val* -double -z value for every span in every contour in group

:Smash(double tolerance, bool preserve_arc)
Replaces all beziers (and optionally arcs) in all contours with a straight line representation within passed tolerance.

> *tolerance* -double -max deviation from original contour permitted when replacing arcs and Bezier with straight lines.
> *preserve_arcs* - bool - if true arcs are not smashed, only beziers

:Transform(Matrix2D xform)
Transform all contours in the group using the passed matrix

> *xform*-Matrix2D -matrix to transform all contours with

ContourCarriage

This object is used to move along a contour a fixed distance at a time irrespective of the number of spans in the contour or the type of spans.

Constructor

ContourCarriage(Contour ctr, Point2D pt)

Creates a 'carriage' for the passed contour and positions it at the nearest point on the contour to the passed point.

ctr-Contour -The contour that the carriage travels along
pt-Point2D -The carriage is positioned on the contour at the closest position to this point

ContourCarriage(integer span_index, double parameter)

Creates a 'carriage' for a contour and set its positions for the span with the passed index at the passed parameter position (in range 0-1.0) on the span. These values are normaly specified as 0,0.0 to position the carriage at the start of the first span in the contour.

span_index-integer -Index of span, carriage will be positioned at
parameter-double -Parameter along span carriage will be positioned at

```
e.g       local cursor = ContourCarriage(0, 0.0)
```

Properties

.IsInvalid

R/O - bool - Returns true if current position is invalid

.SpanIndex

R/O - integer - Return span index for current position of carriage

.SpanParameter

R/O - double - Return span parameter (in range 0-1.0) for current position of carriage

Methods

:Move(Contour ctr, double distance)

Moves the carriage the requested distance along the contour. Returns true if manage to move requested distance successfully. If the contour is closed we will wrap around at the end, if it is open and you attempt to move past the end this method will return false.

ctr-Contour -The contour this carriage rides along
distance - double –The distance to move along the contour, negative values will move back towards the start

:Position(Contour ctr)

Returns a Point2D representing the current position on the carriage on the passed contour

ctr-Contour -The contour this carriage rides along

:UpdatePosition(Contour ctr, Point2D pt)

Update the position of the carriage to the nearest position on the contour to the passed point.

ctr-Contour -The contour that the carriage travels along
pt-Point2D -The carriage is positioned on the contour at the closest position to this point

Example Code

```
[[  -------------- MarkContourParameterSteps --------------------------------
|
|  Insert a marker at each parameter step around the contour and display the
|  distance from the start point
|
]]
function MarkContourParameterSteps(num_steps, contour, layer)

   local cursor = ContourCarriage(0, 0.0)

   local contour_length = contour.Length
   local step_dist = contour_length / num_steps

   local end_index = num_steps
   if contour.IsOpen then
      end_index = end_index + 1
   end


   for i = 1, end_index do
     local ctr_pos = cursor:Position(contour)
     local marker = CadMarker("D:" .. (step_dist * (i - 1)), ctr_pos, 3)
     layer:AddObject(marker, true)
     cursor:Move(contour, step_dist)

     end

end
```

ToolpathTab

This object is derived from ContourCarriage and represents a ToolpathTab. A ToolpathTab is created from a CadContour object using the *InsertToolpathTabAtPoint*() method.

Properties

All the properties for this object are documented in the ContourCarriage base class

Methods

All the methods for this object are documented in the ContourCarriage base class

Span

This object is the base class for spans in a Contour. Spans are either lines, arcs or beziers or a single 'point'

Constructors

Span(Point3D pt3d) - Constructor

A new span representing a single point is created within a Lua script using this contructor method

pt3d - Point3D - 3d position for a point

e.g

```
local pt = Point3D(0, 0, 0)
local pt_span = Span(pt)
```

NOTE: most spans are created as either a LineSpan, ArcSpan or BezierSpan.

Properties

.EndPoint3D

R/O - Point3D - The 3D end point of the span

.EndPoint2D

R/O - Point3D - The 2D end point of the span

.IsLineType

R/O - bool - true if span is a line

.IsArcType

R/O - bool - true if span is an arc

.IsBezierType

R/O - bool - true if span is a bezier

.IsLeadInType

R/O - bool - true if span is a lead in

.IsLeadOutType

R/O - bool - true if span is a lead out

.IsOvercutType

R/O - bool - true if span is part of an overcut

.NumberOfControlPoints

R/O - integer - number of control points for span (2 for Bezier, 1 for an arc (the mid point), 0 for a line

.StartPoint3D

R/O - Point3D - The 3D start point of the span

.StartPoint2D

R/O - Point2D - The 2D start point of the span

.Type
R/O - integer - enum identifying type of the span
Usual values are Span.NormalLine, Span.Arc and Span.Bezier

Methods

:ChordLength()
Return the straight line distance between start and end points of the span

:EndVector(bool normalise)
Return a Vector2D representing the direction of the span at the end.

normalise - bool - if true, the returned vector is normalised

:GetControlPointPosition(integer index)
Return a Point2D with position of requested control point (use .NumberOfControlPoints to get count). This call is only valid on a Bezier or an Arc. For a Bezier 0 returns first control point, 1 returns the second. For an arc, 0 returns arc mid point.

index - integer - index of control point

:GetLength(double tolerance)
Return a double holding the length of the span. The passed tolerance is used for beziers to approximate length to given tolerance.

tolerance - double - tolerance to use when polygonizing beziers to find length

:MoveControlPoint(integer index, Point2D pt)
This is only valid for an arc or Bezier and attempts to move the specified control point to the passed position. This call is only valid on a Bezier or an Arc. For a Bezier, index 0 modifies first control point, 1 modifies the second. For an arc, 0 modifies the arc mid point. If the control point can't be moved to the passed position (e.g would make an invalid arc) returns false, else true.

index - integer - index of control point
pt - Point2D - position for control point

:PointAtParameter(double param, double tolerance)
Return a Point2D representing the point on the span at passed parameter in range 0 - 1.0. Passed tolerance is used for polygonisation for Bezier spans.

parameter - double - parameter in range 0 - 1.0 to find point on span at
tolerance - double - tolerance to use when polygonizing beziers

:Reverse()
Reverse the span - swap start and end points and for arcs swap sign of bulge, for beziers swap control points.

:StartVector(bool normalise)
Return a Vector2D representing the direction of the span at the start.

normalise - bool - if true, the returned vector is normalised

:SetStartPoint3D(Point3D pt)
Set start point for span - be EXTREMELY careful if you call this method on spans in a contour as gaps will open between adjoining spans leading to an invalid contour.

pt - Point3D - 3D position for point

:SetStartPoint2D(Point2D pt)
Set start point for span - be EXTREMELY careful if you call this method on spans in a contour as gaps will open between adjoining spans leading to an invalid contour.

pt - Point2D - 2D position for point

:SetEndPoint3D(Point3D pt)
Set end point for span - be EXTREMELY careful if you call this method on spans in a contour as gaps will open between adjoining spans leading to an invalid contour.

pt - Point3D - 3D position for point

:SetEndPoint2D(Point2D pt)
Set end point for span - be EXTREMELY careful if you call this method on spans in a contour as gaps will open between adjoining spans leading to an invalid contour.

pt - Point2D - 2D position for point

LineSpan

This object represents a line span in a Contour. It is derived from the Span class and all methods and properties of the span class are valid on this object as well. As well as creating spans to append to a contour you can use the LineTo methods to avoid span creation.

Constructors

LineSpan(Point3D start_pt, Point3D end_pt) - Constructor

A new span representing a line is created within a Lua script using this contructor method

start_pt - Point3D – 3D position for start point of span
end_pt - Point3D – 3D position for end point of span

e.g

```
local start_pt = Point3D(0,   0, 0)
local end_pt   = Point3D(1.0, 0, 0)

local line_span = LineSpan(start_pt, end_pt)
```

LineSpan(Point2D start_pt, Point2D end_pt) - Constructor

A new span representing aline is created within a Lua script using this contructor method

start_pt - Point2D - 2d position for start point of span
end_pt - Point2D - 2d position for end point of span

e.g

```
local start_pt = Point2D(0,   0)
local end_pt   = Point2D(1.0, 0)

local line_span = LineSpan(start_pt, end_pt)
```

Methods

:ClosestParameterToPoint(Point2D pt)

Return the parameter of the point on the line which is closest to the passed point

pt - Point2D - 2d position of point finding closest parameter to

ArcSpan

This object represents an arc span in a Contour. It is derived from the Span class and all methods and properties of the span class are valid on this object as well. As well as creating spans to append to a contour you can use the ArcTo methods to avoid span creation. Arcs are represented internally using the start and end points and a 'bulge factor' which controls the shape of the arc. The bulge represents a ratio between the chord length of an arc and its height (bulge = (2*arc_height) / chord_length).

Constructors

ArcSpan(Point2D start_pt, Point2D end_pt, Point2D pt_on_arc) - Constructor

A new span representing an arc is created within a Lua script using this contructor method

start_pt – Point2D – 2D position for start point of span
end_pt – Point2D – 2D position for end point of span- must have same Z values as start
pt_on_arc- Point2D – 2D position for a point on arc

NOTE: arcs can be a maximum of 180 degrees

e.g

```
local start_pt  = Point2D(0,   0)
local end_pt    = Point2D(1.0, 0)
local pt_on_arc = Point2D(0.5, 0.3)

local arc_span = ArcSpan(start_pt, end_pt, pt_on_arc)
```

ArcSpan(Point2D start_pt, Point2D end_pt, Point2D centre_pt, bool ccw) - Constructor

A new span representing an arc is created within a Lua script using this contructor method

start_pt – Point2D – 2D position for start point of span
end_pt – Point2D – 2D position for end point of span- must have same Z values as start
centre_pt – Point2D – 2D position for centre point of arc (not mid-point)
ccw - bool - true if arc is Counter Clockwise

e.g

```
local start_pt  = Point2D(0,   0)
local end_pt    = Point2D(1.0, 0)
local center_pt = Point2D(0.5, 0)

local arc_span = ArcSpan(start_pt, end_pt, center_pt, false)
```

NOTE: arcs can be a maximum of 180 degrees

ArcSpan(Point3D start_pt, Point3D end_pt, Point3D centre_pt, bool ccw) - Constructor

A new span representing an arc is created within a Lua script using this contructor method

start_pt - Point3D - 3d position for start point of span
end_pt - Point3D - 3d position for end point of span- must have same Z values as start
centre_pt - Point3D - 3d position for centre point of arc (not mid-point)
ccw - bool - true if arc is Counter Clockwise

e.g

```
local start_pt  = Point3D(0,   0, 0)
local end_pt    = Point3D(1.0, 0, 0)
local center_pt = Point3D(0.5, 0, 0)

local arc_span = ArcSpan(start_pt, end_pt, center_pt, false)
```

NOTE: arcs can be a maximum of 180 degrees

ArcSpan(Point3D start_pt, Point3D end_pt, Point2D pt_on_arc) - Constructor

A new span representing an arc is created within a Lua script using this contructor method

start_pt - Point3D – 3D position for start point of span
end_pt - Point3D – 3D position for end point of span- must have same Z values as start
pt_on_arc- Point2D – 2D position for a point on arc

NOTE: arcs can be a maximum of 180 degrees

e.g

```
local start_pt  = Point3D(0,   0, 0)
local end_pt    = Point3D(1.0, 0, 0)
local pt_on_arc = Point2D(0.5, 0.3)

local arc_span = ArcSpan(start_pt, end_pt, pt_on_arc)
```

ArcSpan(Point2D start_pt, Point2D end_pt, double bulge) - Constructor

A new span representing an arc is created within a Lua script using this contructor method

start_pt – Point2D – 2D position for start point of span
end_pt – Point2D – 2D position for end point of span - must have same Z values as start
bulge - double - buge factor for arc

e.g

```
local start_pt = Point2D(0,   0)
local end_pt   = Point2D(1.0, 0)

local arc_span = ArcSpan(start_pt, end_pt, 1.0)
```

ArcSpan(Point3D start_pt, Point3D end_pt, double bulge) - Constructor

A new span representing an arc is created within a Lua script using this contructor method

start_pt - Point3D – 3D position for start point of span
end_pt - Point3D – 3D position for end point of span - must have same Z values as start
bulge - double - buge factor for arc

e.g

```
local start_pt = Point3D(0,   0, 0)
local end_pt   = Point3D(1.0, 0, 0)

local arc_span = ArcSpan(start_pt, end_pt, 1.0)
```

NOTE: arcs can be a maximum of 180 degrees

Properties

.Bulge

R/O - double - signed bulge for arc

.IsAntiClockwise

R/O - bool - true if arc is anticlockwise (same as .IsCCW)

.IsCCW

R/O - bool - true if arc is counter clockwise (same as . IsAntiClockwise)

.IsClockwise

R/O - bool - true if arc is clockwise (same as . IsCW)

.IsCW

R/O - bool - true if arc is clockwise (same as . IsClockwise)

Methods

:ArcMidPoint()

Returns a Point3D holding the mid point of the arc

:RadiusAndCentre(Point3D ret_centre_pt)

Calculate the centre point of the arc and the radius. The method returns the radius and also sets the position of the passed point to the centre point position of the arc.

ret_centre_pt - Point3D - returned 3d position for centre point of arc (not mid-point)

Example Code

```
local arc = ArcSpan(Point3D(0,0,0), Point3D(1,0,0), 1.0)
local centre = Point3D()
local radius = arc:RadiusAndCentre(centre)
MessageBox("Arc Radius = " .. radius ..
           " Centre Point = " ..  centre.x .. "," .. centre.y )
```

:SetBulge(bulge)

Set the bulge factor for this arc

Bulge - double - bulge factor for arc

Global helper methods

ArcBulgeFromMidPoint(Point2D start, Point2D end, Point2D mid)

Return the bulge factor for an arc through the specified 3 points

start -Point2D -Start point for arc
end -Point2D -End point for arc
mid -Point2D -mid point for arc (NOT centre point)

BezierSpan

This object represents a bezier span in a Contour. It is derived from the Span class and all methods and properties of the span class are valid on this object as well. As well as creating spans to append to a contour you can use the BezierTo methods to avoid span creation.

Constructors

BezierSpan(Point2D start_pt, Point2D end_pt, Point2D ctrl_pt_1, Point2D ctrl_pt_1) - Constructor

A new span representing a bezier is created within a Lua script using this contructor method

start_pt – Point2D – 2D position for start point of span
end_pt – Point2D – 2D position for end point of span- must have same Z values as start
ctrl_pt_1 - Point2D – 2D position for first control point
ctrl_pt_2 - Point2D – 2D position for second control point

e.g

```
local start_pt = Point2D(0,   0)
local end_pt   = Point2D(1.0, 0)
local ctrl_1   = Point2D(0.2, 0.2)
local ctrl_2   = Point2D(0.8, 0.2)

local bez_span = BezierSpan(start_pt, end_pt, ctrl_1, ctrl_2)
```

BezierSpan(Point3D start_pt, Point3D end_pt, Point2D ctrl_pt_1, Point2D ctrl_pt_1) - Constructor

A new span representing a bezier is created within a Lua script using this contructor method

start_pt - Point3D – 3D position for start point of span
end_pt - Point3D – 3D position for end point of span- must have same Z values as start
ctrl_pt_1 - Point2D – 2D position for first control point
ctrl_pt_2 - Point2D – 2D position for second control point

e.g

```
local start_pt = Point3D(0,   0, 0)
local end_pt   = Point3D(1.0, 0, 0)
local ctrl_1   = Point2D(0.2, 0.2)
local ctrl_2   = Point2D(0.8, 0.2)

local bez_span = BezierSpan(start_pt, end_pt, ctrl_1, ctrl_2)
```

Span Helper Methods

The following methods are used to convert a Span object to the specific span type

CastSpanToLineSpan(Span span)

Returns a LineSpan object for the passed span object. The passed span objects .**IsLineType** property must be true.

span - Span - the span we are casting to a line

CastSpanToArcSpan(Span span)

Returns a LineSpan object for the passed span object. The passed span objects. **IsArcType** property must be true.

span - Span - the span we are casting to an arc

CastSpanToBezierSpan(Span span)

Returns aBezierSpan object for the passed span object. The passed span objects .**IsBezierType** property must be true.

span - Span - the span we are casting to a bezier

Low Level Geometry

To make programming scripts easier, Vectric provide access to a range of low level geometric primitives which can greatly simplify writing scripts which involve geometry manipulation.

The Point2D and Point3D primitives are widely used to set the end points of spans and while building geometry. The Box2D and Box3D objects are extremely useful for determining the size and position of objects. The Vector objects can make a lot of tasks which rely on calculation of distances between points and transformation of objects much easier.

Point2D

This object represents a 2D point (X and Y values only). It is used by many other objects and can also be used with other geometry objects such as Vector2D to perform calculations.

Constructors

Point2D() - Constructor

This construct an un-initialised point with invalid x and y values which can be tested for using the .IsInvalid property.

Point2D(double x, double y) - Constructor

A new point with the specified X and Y values

x -double - the value for the x coordinate of the point
y -double - the value for the y coordinate of the point

Point2D(Point2D pt) - Constructor

A new point with the same X and Y values as the passed point

pt -Point2D - an existing point to copy X and Y values from

Properties

.IsInvalid

Returns true if the point is invalid - not initialised or :SetInvalid called

.X

R/W - double -x coordinate for point

.x

R/W - double -x coordinate for point

.Y

R/W - double -y coordinate for point

.y

R/W - double -y coordinate for point

Methods

:IsCoincident(Point2D point, double tol)

Returns true if passed point is coincident with this point within passed tolerance

point -Point2D -point to check for coincidence
tol -double -maximum distance apart point can be and still be considered coincident

:Set(double x, double y)

Set both the x and y values for the point

x -double -new x value for point
y -double -new y value for point

:SetInvalid()

Set the point to invalid state -X and Y will be lost

Operations

Matrix2D * Point2D

Multiplying a matrix by a point returns a transformed point

Point2D - Point2D

Subtracting a point from another point will return a Vector2D with the difference between the 2 points.

Point2D + Vector2D

Adding a vector to a point will return a point displaced from original point by adding the vector

Point2D - Vector2D

Subtracting a vector from a point will return a point displaced from original point by subtracting the vector

Point3D

This object represents a 3D point (X,Y and Z values). It is used by many other objects and can also be used with other geometry objects such as Vector3D to perform calculations.

Constructors

Point3D() - Constructor

This construct an un-initialised point with invalid x, y and z values which can be tested for using the .IsInvalid property.

Point3D(double x, double y, double z) - Constructor

A new point with the specified X, Y and Z values

x -double - the value for the x coordinate of the point
y -double - the value for the y coordinate of the point
z -double - the value for the z coordinate of the point

Point3D(Point3D pt) - Constructor

A new point with the same X, Y and Z values as the passed point

pt -Point3D - an existing point to copy X, Y and Z values from

Point3D(Point2D pt, double z)

Create a new point with x and y values from passed 2D point and new z value

pt -Point2D -2D point holding x and y values for point
z -double -new z value for point

Properties

.X

R/W - double -x coordinate for point

.x

R/W - double -x coordinate for point

.Y

R/W - double -y coordinate for point

.y

R/W - double -y coordinate for point

.Z

R/W - double -z coordinate for point

.z

R/W - double -z coordinate for point

.IsInvalid

Returns true if the point is invalid - not initialised or :SetInvalid called

Methods

:IsCoincident(Point3D point, double tol)

Returns true if passed point is coincident with this point within passed tolerance

point -Point3D -point to check for coincidence
tol -double -maximum distance apart point can be and still be considered coincident

:Set(double x, double y, double z)

Set new x, y and z values for the point

x -double -new x value for point
y -double -new y value for point
z -double -new z value for point

:Set(Point2D pt, double z)

Set new x and y values from passed 2D point and new z value

pt -Point2D -2D point holding new x and y values for point
z -double -new z value for point

:SetInvalid()

Set the point to invalid state –X,Y and Z will be lost

Operations

Matrix2D * Point3D

Multiplying a matrix by a point returns a transformed point. The Z value is unchanged

Point3D - Point3D

Subtracting a point from another point will return a Vector3D with the difference between the 2 points.

Point3D + Vector3D

Adding a vector to a point will return a point displaced from original point by adding the vector

Point3D - Vector3D

Subtracting a vector from a point will return a point displaced from original point by subtracting the vector

Vector2D

This object represents a 2D vector (X and Y values only). It is used by many other objects and can also be used with other geometry objects such as Point2D to perform calculations.

Constructors

Vector2D() - Constructor

This construct an un-initialised vector with invalid x and y values which can be tested for using the .IsInvalid property.

Vector2D(double x, double y) - Constructor

A vector point with the specified X and Y values

x -double - the value for the x coordinate of the vector
y -double - the value for the y coordinate of the vector

Vector2D(Vector2D vec) - Constructor

A new vector with the same X and Y values as the passed vector

vec -Vector2D - an existing vector to copy X and Y values from

Properties

.IsInvalid

Returns true if the vector is invalid - not initialised or :SetInvalid called

.Length

Return the length of the vector

.LengthSq

Return the length squared of the vector. If comparing distance between points etc. using LengthSq instead of Length will be faster as it avoids the square root operation required to find the exact length.

.X

R/W - double -x value for vector

.x

R/W - double -x value for vector

.Y

R/W - double -y value for vector

.y

R/W - double -y value for vector

Methods

:Cross(Vector2D vec)

Return the cross product (a Vector3D) of this vector and passed vector

:Dot(Vector2D vec)
Return the dot product (a double value) of this vector and passed vector

:Normalize()
Normalize this vector to make it a unit vector

:NormalTo()
Return a new vector 'normal' to this vector found by rotating vector 90 degrees clockwise

:Set(double x, double y)
Set both the x and y values for the vector

x -double -new x value for *vector*
y -double -new y value for *vector*

:SetInvalid()
Set the vector to invalid state -X and Y will be lost

Operations

Double * Vector2D
Returns a new vector multiplied by double value

Matrix2D * Vector2D
Multiplying a matrix by a point returns a transformed vector

Vector2D + Vector2D
Adding a vector to a vector will return a new vector.

Vector2D - Vector2D
Subtracting a vector from a point will return a new vector.

- Vector2D
Negates x,y values of vector (reverses vector direction)

Vector3D

This object represents a 3D vector (X, Y and Z values). It is used by many other objects and can also be used with other geometry objects such as Point3D to perform calculations.

Constructors

Vector3D() - Constructor

This construct an un-initialised vector with invalid x, y and z values which can be tested for using the .IsInvalid property.

Vector3D(double x, double y, double z) - Constructor

A vector point with the specified values

x -double - the value for the x coordinate of the vector
y -double - the value for the y coordinate of the vector
z -double - the value for the z coordinate of the vector

Vector3D(Vector3D vec) - Constructor

A new vector with the same X, Y and Z values as the passed vector

vec -Vector3D - an existing vector to copy values from

Properties

.IsInvalid

Returns true if the vector is invalid - not initialised or :SetInvalid called

.Length

Return the length of the vector

.LengthSq

Return the length squared of the vector. If comparing distance between points etc. using LengthSq instead of Length will be faster as it avoids the square root operation required to find the exact length.

.X

R/W - double -x value for vector

.x

R/W - double -x value for vector

.Y

R/W - double -y value for vector

.y

R/W - double -y value for vector

.Z

R/W - double -z value for vector

.z

R/W - double -z value for vector

Methods

:Cross(Vector3D vec)
Return the cross product (a Vector3D) of this vector and passed vector

:Dot(Vector3D vec)
Return the dot product (a double value) of this vector and passed vector

:Dot(Point3D vec)
Return the dot product (a double value) of this vector and passed point. The X,Y,Z values of the point are treated exactly the same as for the Vector3D version of this method.

:Normalize()
Normalize this vector to make it a unit vector

:Set(double x, double y, double z)
Set the x, y and z values for the vector

x -double -new x value for *vector*
y -double -new y value for *vector*
z -double -new z value for *vector*

:SetInvalid()
Set the vector to invalid state -X, Y and Z will be lost

Operations

Double * Vector3D
Returns a new vector multiplied by double value

Vector3D + Vector3D
Adding a vector to a vector will return a new vector.

Vector3D - Vector3D
Subtracting a vector from a point will return a new vector.

- Vector3D
Negates x,y and z values of vector (reverses vector direction)

Box2D

This object represents a 2d bounding box for an object or group of objects.

Constructors

Box2D() - Constructor

This construct an un-initialised box with invalid values which can be tested for using the .IsInvalid property.

Box2D (Point2D p1, Point2D p2) - Constructor

Create a box which bounds the 2 passed in points

P1 -Point2D - first point within the box

P1 -Point2D - second point within the box

Box2D(Box2D box) - Constructor

A new Box2D with the same values as the passed box

box -Box2D - an existing box to copy values from

Properties

.BLC

R/O - Point2D - a point representing the bottom left corner of the bounding box

.BRC

R/O - Point2D - a point representing the bottom right corner of the bounding box

.Centre

R/O - Point2D - a point representing the centre of the bounding box (British spelling)

.Center

R/O - Point2D - a point representing the centre of the bounding box (U.S spelling)

.IsInvalid

R/O - bool - returns true if the box is in an invalid state - not initialised or SetInvalid called

.MinX

R/O - double - the minimum X value for the bounding box

.MinY

R/O - double - the minimum Y value for the bounding box

.MaxX

R/O - double - the maximum X value for the bounding box

.MaxY

R/O - double - the maximum Y value for the bounding box

.XLength

R/O - double - the length in X for the bounding box

.YLength
R/O - double - the length in Y for the bounding box

.MaxLength
R/O - double - the length of the longest side for the bounding box

.MinLength
R/O - double - the length of the shortest side for the bounding box

.TRC
R/O - Point2D - a point representing the top right corner of the bounding box

.TLC
R/O - Point2D - a point representing the top left corner of the bounding box

Methods

:Expand(double offset_dist)
Expand this box on all sides by passed amount

offset_dist -double - distance to offset box by

:IsInside(Point2D point, double tol)
Return true if passed point is inside this box within passed tolerance (which may be 0.0).

point -Point2D - point to test to see if inside box
tol - double - tolerance value to use when checking if inside box

:IsInsideOrOn(Point2D point, double tol)
Return true if passed point is inside or on the edge of this box within passed tolerance (which may be 0.0).

point -Point2D - point to test to see if inside or on box
tol - double - tolerance value to use when checking if inside box

:IsInside(Box2D box, bool on_counts_as_int, double tol)
Return true if passed box intersects this box within passed tolerance (which may be 0.0).

box -Box2D - box to test to see if intersects this box
on_counts_as_int - bool - if true boxes with shared edges count as intersections
tol - double - tolerance value to use when checking if inside box

:Intersects(Box2D box, double tol)
Return true if passed box is inside this box within passed tolerance (which may be 0.0).

box -Box2D - box to test to see if inside this box
tol - double - tolerance value to use when checking if inside box

:Merge(double x, double y)
Merge a point with the passed x and y values into the box updating limits

x -double - x value for point being merged
y -double - y value for point being merged

:Merge(Point2D point)

Merge a point into the box updating limits

point -Point2D - point being merged

:Merge(Box2D box)

Merge another box into this box, updating the limits

box -Box2D - box being merged

:SetInvalid()

Set the box to invalid state

Box3D

This object represents a 3D bounding box for objects.

Properties

.BLC

R/O - Point3D - a point representing the bottom left corner of the bounding box

.BRC

R/O - Point3D - a point representing the bottom right corner of the bounding box

.Centre

R/O - Point3D - a point representing the centre of the bounding box (British spelling)

.Center

R/O - Point3D - a point representing the centre of the bounding box (U.S spelling)

.IsInvalid

R/O - bool - returns true if the box is in an invalid state - not initialised or SetInvalid called

.MaxLength

R/O - double - the length of the longest side for the bounding box

.MaxX

R/O - double - the maximum X value for the bounding box

.MaxY

R/O - double - the maximum Y value for the bounding box

.MaxZ

R/O - double - the maximum Z value for the bounding box

.MinLength

R/O - double - the length of the shortest side for the bounding box

.MinX

R/O - double - the minimum X value for the bounding box

.MinY

R/O - double - the minimum Y value for the bounding box

.MinZ

R/O - double - the minimum Z value for the bounding box

.TRC

R/O - Point3D - a point representing the top right corner of the bounding box

.TLC

R/O - Point3D - a point representing the top left corner of the bounding box

.XLength
R/O - double - the length in X for the bounding box

.YLength
R/O - double - the length in Y for the bounding box

.ZLength
R/O - double - the length in Z for the bounding box

Methods

:Expand(double offset_dist)
Expand this box on all sides by passed amount

offset_dist -double - distance to offset box by

:IsInside(Point3D point, double tol)
Return true if passed point is inside this box within passed tolerance (which may be 0.0).

point -Point3D - point to test to see if inside box
tol - double - tolerance value to use when checking if inside box

:IsInsideOrOn(Point3D point, double tol)
Return true if passed point is inside or on the edge of this box within passed tolerance (which may be 0.0).

point -Point3D - point to test to see if inside or on box
tol - double - tolerance value to use when checking if inside box

:IsInside(Box3D box, bool on_counts_as_int, double tol)
Return true if passed box intersects this box within passed tolerance (which may be 0.0).

box -Box2D - box to test to see if intersects this box
on_counts_as_int - bool - if true boxes with shared edges count as intersections
tol - double - tolerance value to use when checking if inside box

:Intersects(Box3D box, double tol)
Return true if passed box is inside this box within passed tolerance (which may be 0.0).

box -Box2D - box to test to see if inside this box
tol - double - tolerance value to use when checking if inside box

:Merge(double x, double y, double z)
Merge a point with the passed values into the box updating limits

x -double - x value for point being merged
y -double - y value for point being merged
z -double - z value for point being merged

:Merge(Point3D point)
Merge a point into the box updating limits

point -Point3D - point being merged

:Merge(Box3D box)

Merge another box into this box, updating the limits

box -Box3D - box being merged

:SetInvalid()

Set the box to invalid state

Matrix2D

This object represents a matrix which can be used to transform 2D points - translation, rotation scaling etc.

Constructors

Matrix2D objects are usually constructed by the utility methods listed below.

IdentityMatrix2D()

Returns a Matrix2D which is an identity matrix.

ReflectionMatrix2D(Point2D p1, Point2D p1)

Returns a Matrix2D which is performs reflection about a line between the passed points.

P1 -Point2D - the start point of the line about which reflection will take place
P2 -Point2D - the end point of the line about which reflection will take place

RotationMatrix2D(Point2D rotation_pt, double angle)

Returns a Matrix2D which is performs the rotation about the specified point by the specified amount.

rotation_pt -Point2D - the point about which rotation will take place
angle -double - the angle to rotate by in degrees, positive values are CCW.

ScalingMatrix2D(Vector2D scale_vec)

Returns a Matrix2D which performs the scaling around the origin (0,0) by the specified amount.

scale_vec -Vector2D - the X and Y values of the vector specify the amount to scale in X and Y

ScalingMatrix2D(Point2D scale_pt, Vector2D scale_vec)

Returns a Matrix2D which is performs the scaling about the specified point by the specified amount.

scaling_pt -Point2D - the point about which scaling will take place
scale_vec -Vector2D - the X and Y values of the vector specify the amount to scale in X and Y

TranslationMatrix2D(Vector2D vec)

Returns a Matrix2D which is performs the translation specified by the vector.

vec -Vector2D - the X and Y values of the vector specify the distances to translate in X and Y

Operations

Matrix2D * Matrix2D

Multiplying a matrix by a matrix returns a new matrix

Example Code

```
local p1 = Point2D(10 ,10)
local xform = RotationMatrix2D(Point2D(0,0) 90)
local p3 = xform * p1
DisplayMessageBox("Transformed p1 = " .. p3.X .. " , " .. p3.Y)
```

Toolpaths

The objects documented in this section allow the script writer to create toolpaths within the program. There are two types of toolpaths which can be created from scripts.

1) Standard Vectric toolpaths. These are the same toolpaths you would have created within the program using the Profile. Pocketing, VCarvingn etc. forms. From script you can fill out all the same parameters that you would manually in the program, and the user can edit the toolpaths within the interface to change parameters and recalculate. These toolpaths are created for the currently selected vectors (the selection can also be controlled from script).

2) External toolpaths. These are toolpaths which are created entirely from within the script. The script creates contours which represent the movement of the tool and hence has complete freedom to do whatever it wants to do. As these toolpaths are not related to the geometry in the main program, the user can not recalculate them, but would need to delete them and re-run the script to recreate a new version.

ToolpathManager

The ToolpathManager object is used to handle all the toolpaths within the program. Like the MaterialBlock object there is one instance within the program which is accessed by any ToolpathManager object. For the initial release, the ToolpathManager is only used to create new toolpaths, load templates and recalculate all toolpaths, for future releases Vectric may offer more access to existing toolpaths depending on demand.

Constructor

ToolpathManager - constructor

Returns a new object which refers to the single toolpath manager within the program.

e.g

```
local toolpath_mgr = ToolpathManager()
```

Properties

.Count

R/O - integer - the number of toolpaths

.IsEmpty

R/O -bool - true if there are no toolpaths

.NumVisibleToolpaths

R/O - integer - the number of visible toolpaths

Methods

:AddExternalToolpath(ExternalToolpath toolpath)

Adds the passed toolpath to the list of toolpaths in the job. Returns true if the toolpath was added OK, else false. See the documentation for ExternalToolpath for information on how to create the actual toolpath itself.

toolpath -ExternalToolpath - toolpath created in a lua script which is added to job

:CreateProfilingToolpath(

```
String name,
Tool tool,
ProfileParameterData profile_data,
RampingData ramping_data,
LeadInOutData lead_data,
ToolpathPosData pos_data,
GeometrySelector geometry_selector,
bool create_2d_preview,
```

bool interactive
)

Creates a profiling toolpath for the currently selected vectors. Returns the UUID for the toolpath created.

name-string - Name for the toolpath to be created
tool -Tool -Tool to use for the toolpath
profile_data -ProfileParameterData -Settings for profiling depth, inside / outside etc
ramping_data -RampingData -Settings for ramping
lead_data -LeadInOutData-Settings for lead in / out
pos_data-ToolpathPosData -Settings for home position, safe z etc
geometry_selector-GeometrySelector -Can be used to automatically select vectors on layers etc

create_2d_preview-bool -If true create preview vectors in 2d view
interactive-bool -If true display warnings etc to user

Example Code - from Profile_Vectors_On_Layer.lua

```
function CreateLayerProfileToolpath
        (
        layer_name,
        name ,
        start_depth,
        cut_depth,
        tool_dia,
        tool_stepdown,
        tool_in_mm
        )


   -- clear current selection
   local selection = job.Selection
   selection:Clear()

   -- get layer
   local layer = job.LayerManager:FindLayerWithName(layer_name)

   if layer == nil then
      DisplayMessageBox("No layer found with name = " .. layer_name)
      return false
   end


   -- select all closed vectors on the layer
   if not SelectVectorsOnLayer(layer, selection, true, false, true) then
      DisplayMessageBox("No closed vectors found on layer " .. layer_name)
      return false
   end

   -- Create tool we will use to machine vectors
   local tool = Tool(
                   "Lua End Mill",
                   Tool.END_MILL       -- BALL_NOSE, END_MILL, VBIT
                   )

   tool.InMM = tool_in_mm
   tool.ToolDia = tool_dia
   tool.Stepdown = tool_stepdown
   tool.Stepover = tool_dia * 0.25
   tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN, INCHES_SEC ...
   tool.FeedRate = 30
   tool.PlungeRate = 10
   tool.SpindleSpeed = 20000
   tool.ToolNumber = 1
   tool.VBitAngle = 90.0                -- used for vbit only
   tool.ClearStepover = tool_dia * 0.5  -- used for vbit only
```

```
-- Create object used to set home position and safez gap above material surface
local pos_data = ToolpathPosData()

pos_data:SetHomePosition(0, 0, 5.0)
pos_data.SafeZGap = 5.0

-- Create  object used to pass profile options
local profile_data = ProfileParameterData()
   -- start depth for toolpath
   profile_data.StartDepth = start_depth

   -- cut depth for toolpath this is depth below start depth
   profile_data.CutDepth = cut_depth

   -- direction of cut - ProfileParameterData.
   -- CLIMB_DIRECTION or ProfileParameterData.CONVENTIONAL_DIRECTION
   profile_data.CutDirection = ProfileParameterData.CLIMB_DIRECTION

   -- side we machine on - ProfileParameterData.
   -- PROFILE_OUTSIDE,  ProfileParameterData.PROFILE_INSIDE or
   -- ProfileParameterData.PROFILE_ON
   profile_data.ProfileSide = ProfileParameterData.PROFILE_OUTSIDE

   -- Allowance to leave on when machining
   profile_data.Allowance = 0.0

   -- true to preserve start point positions, false to reorder start
   -- points to minimise toolpath length
   profile_data.KeepStartPoints = false

   -- true if want to create 'square' external corners on toolpath
   profile_data.CreateSquareCorners = false

   -- true to perform corner sharpening on internal corners (only with v-bits)
   profile_data.CornerSharpen = false

   -- true to use tabs (position of tabs must already have been defined on vectors)
   profile_data.UseTabs = false
   -- length for tabs if being used
   profile_data.TabLength = 5.0
   -- Thickness for tabs if being used
   profile_data.TabThickness = 1.0
   -- if true then create 3d tabs else 2d tabs
   profile_data.Use3dTabs  = true

   -- if true in Aspire, project toolpath onto composite model
   profile_data.ProjectToolpath = false

 -- Create object used to control ramping
 local ramping_data = RampingData()
   -- if true we do ramping into toolpath
   ramping_data.DoRamping = false
   -- type of ramping to perform RampingData.RAMP_LINEAR , RampingData.RAMP_ZIG_ZAG
   -- or RampingData.RAMP_SPIRAL
   ramping_data.RampType = RampingData.RAMP_ZIG_ZAG
   -- how ramp is contrained - either by angle or distance RampingData.CONSTRAIN_DISTANCE
   -- or RampingData.CONSTRAIN_ANGLE
   ramping_data.RampConstraint = RampingData.CONSTRAIN_ANGLE
   -- if we are constraining ramp by distance, distance to ramp over
   ramping_data.RampDistance = 100.0
   -- if we are contraining ramp by angle , angle to ramp in at (in degrees)
   ramping_data.RampAngle = 25.0
   -- if we are contraining ramp by angle, max distance to travel before 'zig zaging'
   -- if zig zaging
   ramping_data.RampMaxAngleDist = 15
   -- if true we restrict our ramping to lead in section of toolpath
   ramping_data.RampOnLeadIn = false


-- Create object used to control lead in/out
local lead_in_out_data = LeadInOutData()
   -- if true we create lead ins on profiles (not for profile on)
   lead_in_out_data.DoLeadIn = false
   -- if true we create lead outs on profiles (not for profile on)
   lead_in_out_data.DoLeadOut = false
```

```
      -- type of leads to create LeadInOutData.LINEAR_LEAD or LeadInOutData.CIRCULAR_LEAD
      lead_in_out_data.LeadType = LeadInOutData.CIRCULAR_LEAD
      -- length of lead to create
      lead_in_out_data.LeadLength = 10.0
      -- Angle for linear leads
      lead_in_out_data.LinearLeadAngle = 45
      -- Radius for circular arc leads
      lead_in_out_data.CirularLeadRadius = 5.0
      -- distance to 'overcut' (travel past start point) when profiling
      lead_in_out_data.OvercutDistance = 0.0

   -- Create object which can be used to automatically select geometry
   local geometry_selector = GeometrySelector()

   -- if this is true we create 2d toolpaths previews in 2d view, if false we dont
   local create_2d_previews = true

   -- if this is true we will display errors and warning to the user
   local display_warnings = true

   -- Create our toolpath

   local toolpath_manager = ToolpathManager()

   local toolpath_id = toolpath_manager:CreateProfilingToolpath(
                                          name,
                                          tool,
                                          profile_data,
                                          ramping_data,
                                          lead_in_out_data,
                                          pos_data,
                                          geometry_selector,
                                          create_2d_previews,
                                          display_warnings
                                          )

   if toolpath_id  == nil  then
      DisplayMessageBox("Error creating toolpath")
      return false
   end

   return true

end
```

CreatePocketingToolpath(

String name,
Tool tool,
Tool area_clear_tool,
PocketParameterData pocket_data,
ToolpathPosData pos_data,
GeometrySelector geometry_selector,
bool create_2d_preview,
bool interactive
)

Creates a pocketing toolpath for the currently selected vectors. Returns the UUID for the toolpath created.

name-string - Name for the toolpath to be created
tool -Tool -Tool to use for the toolpath - must not be nil
area_clear_tool -Tool -Optional tool for area clearance can be nil
pocket_data -PocketParameterData -Setting for pocketing depth, style etc
pos_data-ToolpathPosData -Settings for home position, safe z etc
geometry_selector-GeometrySelector -Can be used to automatically select vectors on layers etc
create_2d_preview-bool -If true create preview vectors in 2d view
interactive-bool -If true display warnings etc to user

Example Code - from Create_Pocketing_Toolpath.lua

```
-- VECTRIC LUA SCRIPT
require "strict"
--[[  ------------ CreatePocketingToolpath ------------
|
|   Create a Pocketing toolpath within the program for the currently selected vectors
|  Parameters:
|   name,                 -- Name for toolpath
|   start_depth           -- Start depth for toolpath below surface of material
|   cut_depth             -- cut depth for pocket toolpath
|   tool_dia              -- diameter of end mill to use
|   tool_stepdown         -- stepdown for tool
|   tool_stepover_percent -- percentage stepover for tool
|   tool_in_mm            -- true if tool size and stepdown are in mm
|
|  Return Values:
|     true if toolpath created OK else false
|
]]


function CreatePocketingToolpath(
                                name ,
                                start_depth,
                                cut_depth,
                                tool_dia,
                                tool_stepdown,
                                tool_stepover_percent,
                                tool_in_mm
                                )

   -- Create tool we will use to machine vectors
   local tool = Tool(
                    "Lua End Mill",
                    Tool.END_MILL       -- BALL_NOSE, END_MILL, VBIT
                    )
```

```
tool.InMM = tool_in_mm
tool.ToolDia = tool_dia
tool.Stepdown = tool_stepdown
tool.Stepover = tool_dia * (tool_stepover_percent / 100)
tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN, INCHES_SEC,...
tool.FeedRate = 30
tool.PlungeRate = 10
tool.SpindleSpeed = 20000
tool.ToolNumber = 1
tool.VBitAngle = 90.0               -- used for vbit only
tool.ClearStepover = tool_dia * (tool_stepover_percent / 100)  -- used for vbit only

-- Create object used to set home position and safez gap above material surface
local pos_data = ToolpathPosData()
pos_data:SetHomePosition(0, 0, 5.0)
pos_data.SafeZGap = 5.0

-- Create  object used to pass pocketing options
local pocket_data = PocketParameterData()
   -- start depth for toolpath
   pocket_data.StartDepth = start_depth

   -- cut depth for toolpath this is depth below start depth
   pocket_data.CutDepth = cut_depth

   -- direction of cut for offet clearance - ProfileParameterData.CLIMB_DIRECTION or
   -- ProfileParameterData.CONVENTIONAL_DIRECTION - NOTE: enum from ProfileParameterData
   pocket_data.CutDirection = ProfileParameterData.CLIMB_DIRECTION

   -- Allowance to leave on when machining
   pocket_data.Allowance = 0.0

   -- if true use raster clearance strategy , else use offset area clearance
   pocket_data.DoRasterClearance = true
   -- angle for raster if using raster clearance
   pocket_data.RasterAngle = 0
   -- type of profile pass to perform  PocketParameterData.PROFILE_NONE ,
   -- PocketParameterData.PROFILE_FIRST orPocketParameterData.PROFILE_LAST
   pocket_data.ProfilePassType = PocketParameterData.PROFILE_LAST

   -- if true we ramp into pockets (always zig-zag)
   pocket_data.DoRamping = false
   --  if ramping, distance to ramp over
   pocket_data.RampDistance = 10.0

   -- if true in Aspire, project toolpath onto composite model
   pocket_data.ProjectToolpath = false

   -- Create object which can be used to automatically select geometry
   local geometry_selector = GeometrySelector()

-- if this is true we create 2d toolpaths previews in 2d view, if false we dont
local create_2d_previews = true

-- if this is true we will display errors and warning to the user
local display_warnings = true

-- if we are doing two tool pocketing define tool to use for area clearance
local area_clear_tool = nill

-- we just create a tool twice as large for testing here
area_clear_tool = Tool(
                     "Lua Clearance End Mill",
                     Tool.END_MILL       -- BALL_NOSE, END_MILL, VBIT
                     )

area_clear_tool.InMM = tool_in_mm
area_clear_tool.ToolDia = tool_dia * 2
area_clear_tool.Stepdown = tool_stepdown * 2
area_clear_tool.Stepover = tool_dia * 2 *(tool_stepover_percent / 100)
area_clear_tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN, INCHES_SEC..
area_clear_tool.FeedRate = 30
area_clear_tool.PlungeRate = 10
area_clear_tool.SpindleSpeed = 20000
area_clear_tool.ToolNumber = 1
area_clear_tool.VBitAngle = 90.0                -- used for vbit only
area_clear_tool.ClearStepover = tool_dia*2*(tool_stepover_percent/100) -- used for vbit
```

```
   -- Create our toolpath
   local toolpath_manager = ToolpathManager()

   local toolpath_id = toolpath_manager:CreatePocketingToolpath(
                                              name,
                                              tool,
                                              area_clear_tool,
                                              pocket_data,
                                              pos_data,
                                              geometry_selector,
                                              create_2d_previews,
                                              display_warnings
                                              )

   if toolpath_id  == nill  then
      DisplayMessageBox("Error creating toolpath")
      return false
   end

   return true

end

--[[  ------------ main ------------
|
|  Entry point for script
|
]]
function main()
    -- Check we have a job loaded
    job = VectricJob()

    if not job.Exists then
       DisplayMessageBox("No job loaded")
       return false;
    end

   local selection = job.Selection
   if selection.IsEmpty  then
       MessageBox("Please select one or more vectors to pocket")
       return false
    end

    local start_depth = 0.0
    local cut_depth = 5.0
    local tool_dia = 12.0
    local tool_stepdown = 2.5
    local tool_stepover_percent = 50.0
    local tool_in_mm = true

    local success = CreatePocketingToolpath(
                                          "Lua Pocketing Toolpath",
                                          start_depth,
                                          cut_depth,
                                          tool_dia,
                                          tool_stepdown,
                                          tool_stepover_percent,
                                          tool_in_mm
                                          )


    return success;
end
```

:CreateDrillingToolpath(

String name,
Tool tool,
DrillParameterData drilling_data,
ToolpathPosData pos_data,
GeometrySelector geometry_selector,
bool create_2d_preview,
bool interactive
)

Creates a drilling toolpath for the currently selected vectors. Returns the UUID for the toolpath created.

name-string - Name for the toolpath to be created
tool -Tool -Tool to use for the toolpath
drill_data *Drill*ParameterData -Setting for drilling depth, peck etc
pos_data-ToolpathPosData -Settings for home position, safe z etc
geometry_selector-GeometrySelector -Can be used to automatically select vectors on layers etc
create_2d_preview-bool -If true create preview vectors in 2d view
interactive-bool -If true display warnings etc to user

Example Code - from Create_Drilling_Toolpath.lua

```
-- VECTRIC LUA SCRIPT
require "strict"

--[[  ------- CreateDrillingToolpath ------------------------------------------------
|
|   Create a drilling toolpath within the program for the currently selected vectors
|  Parameters:
|     name,                 -- Name for toolpath
|     start_depth       -- Start depth for toolpath below surface of material
|    cut_depth           -- cut depth for drilling toolpath
|    retract_gap        -- distance to retract above surface for pecks
|    tool_dia             -- diameter of drill to use
|    tool_stepdown     -- stepdown for tool
|    tool_in_mm         -- true if tool size and stepdown are in mm
|
|  Return Values:
|     true if toolpath created OK else false
|
]]

function CreateDrillingToolpath( name , start_depth, cut_depth, retract_gap, tool_dia,
tool_stepdown, tool_in_mm)

   -- Create tool we will use to machine vectors
   local tool = Tool(
                   "Lua Drill",
                   Tool.THROUGH_DRILL       -- BALL_NOSE, END_MILL, VBIT, THROUGH_DRILL
                   )

   tool.InMM = tool_in_mm
   tool.ToolDia = tool_dia
   tool.Stepdown = tool_stepdown
   tool.Stepover = tool_dia * 0.25
   tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN, INCHES_SEC,
                                 -- INCHES_MIN, FEET_MIN
   tool.FeedRate = 30
   tool.PlungeRate = 10
   tool.SpindleSpeed = 20000
   tool.ToolNumber = 1
   tool.VBitAngle = 90.0               -- used for vbit only
```

```
   tool.ClearStepover = tool_dia * 0.5  -- used for vbit only


   -- Create object used to set home position and safez gap above material surface
   local pos_data = ToolpathPosData()

   pos_data:SetHomePosition(0, 0, 5.0)
   pos_data.SafeZGap = 5.0

   -- Create  object used to pass profile options
   local drill_data = DrillParameterData()
      -- start depth for toolpath
      drill_data.StartDepth = start_depth

      -- cut depth for toolpath this is depth below start depth
      drill_data.CutDepth = cut_depth

      -- if true perform peck drilling
      drill_data.DoPeckDrill = retract_gap > 0.0

      -- distance to retract above surface when peck drilling
      drill_data.PeckRetractGap = retract_gap

      -- if true in Aspire, project toolpath onto composite model
      drill_data.ProjectToolpath = false

   -- Create object which can be used to automatically select geometry
   local geometry_selector = GeometrySelector()

   -- if this is true we create 2d toolpaths previews in 2d view,
   -- if false we dont
   local create_2d_previews = true

   -- if this is true we will display errors and warning to the user
   local display_warnings = true

   -- Create our toolpath

   local toolpath_manager = ToolpathManager()

   local toolpath_id = toolpath_manager:CreateDrillingToolpath(
                                            name,
                                            tool,
                                            drill_data,
                                            pos_data,
                                            geometry_selector,
                                            create_2d_previews,
                                            display_warnings
                                            )

   if toolpath_id  == nil  then
      DisplayMessageBox("Error creating toolpath")
      return false
   end

   return true

end


--[[  _________________________ main ------------------------------------------------
|
|  Entry point for script
|
]]

function main()

    -- Check we have a job loaded
    job = VectricJob()

    if not job.Exists then
       DisplayMessageBox("No job loaded")
       return false;
    end

   local selection = job.Selection
```

```
   if selection.IsEmpty  then
      MessageBox("Please select one or more vectors to drill")
      return false
    end

    local start_depth = 0.0
    local cut_depth = 5.0
    local tool_dia = 3.0
    local retract_gap = 1.0
    local tool_stepdown = 1.0
    local tool_in_mm = true

    local success = CreateDrillingToolpath(
                                          "Lua Drilling Toolpath",
                                          start_depth,
                                          cut_depth,
                                          retract_gap,
                                          tool_dia,
                                          tool_stepdown,
                                          tool_in_mm
                                          )


    return success;
end
```

:CreateVCarvingToolpath(

String name,
Tool tool,
Tool area_clear_tool,
VCarveParameterData vcarve_data,
PocketParameterData pocket_data,
ToolpathPosData pos_data,
GeometrySelector geometry_selector,
bool create_2d_preview,
bool interactive
)

Creates a vcarving toolpath for the currently selected vectors. Returns the UUID for the toolpath created.

name-string - Name for the toolpath to be created
tool -Tool -Tool to use for the toolpath - must not be nil
area_clear_tool -Tool -Optional tool for area clearance can be nil
vcarve_data -VCarveParameterData -Settings for vcarving
pocket_data -PocketParameterData -Setting for pocketing depth, style etc
pos_data-ToolpathPosData -Settings for home position, safe z etc
geometry_selector-GeometrySelector -Can be used to automatically select vectors on layers etc
create_2d_preview-bool -If true create preview vectors in 2d view
interactive-bool -If true display warnings etc to user

Example Code - from Create_VCarving_Toolpath.lua

```
-- VECTRIC LUA SCRIPT
require "strict"
--[[  -------------- CreateVCarvingToolpath --------------
|
|   Create a VCarving toolpath within the program for the currently selected vectors
|  Parameters:
|    name,                 -- Name for toolpath
|    start_depth       -- Start depth for toolpath below surface of material
|    flat_depth          -- flat depth - if 0.0 assume not doing flat bottom
|    vbit_angle          -- angle of vbit to use
|    vbit_dia             -- diameter of VBit  to use
|    vbit_stepdown     -- stepdown for tool
|    tool_stepover_percent - percentage stepover for tool
|    tool_in_mm         -- true if tool size and stepdown are in mm
|
|  Return Values:
|     true if toolpath created OK else false
|
]]

function CreateVCarvingToolpath( name , start_depth, flat_depth, vbit_angle, vbit_dia,
vbit_stepdown, tool_stepover_percent, tool_in_mm)

   -- Create tool we will use to machine vectors
   local tool = Tool(
                    "Lua VBit",
                    Tool.VBIT       -- BALL_NOSE, END_MILL, VBIT
                    )

   tool.InMM = tool_in_mm
   tool.ToolDia = vbit_dia
   tool.Stepdown = vbit_stepdown
```

```
tool.Stepover = vbit_dia * (tool_stepover_percent / 100)
tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN, INCHES_SEC,
                              -- INCHES_MIN, FEET_MIN
tool.FeedRate = 30
tool.PlungeRate = 10
tool.SpindleSpeed = 20000
tool.ToolNumber = 1
tool.VBitAngle = 90.0                -- used for vbit only
tool.ClearStepover = vbit_dia * (tool_stepover_percent / 100) * 2  -- used for vbit only


-- Create object used to set home position and safez gap above material surface
local pos_data = ToolpathPosData()

pos_data:SetHomePosition(0, 0, 5.0)
pos_data.SafeZGap = 5.0

-- Create  object used to pass pocketing options - used for area clearance only
local vcarve_data = VCarveParameterData()
   -- start depth for toolpath
   vcarve_data.StartDepth = start_depth

   -- flag indicating if we are creating a flat bottomed toolpath
   vcarve_data.DoFlatBottom = flat_depth > 0.0

   -- cut depth for toolpath this is depth below start depth
   vcarve_data.FlatDepth = flat_depth

   -- if true in Aspire, project toolpath onto composite model
   vcarve_data.ProjectToolpath = false

   -- set flag indicating we are using flat tool
   vcarve_data.UseAreaClearTool = true

-- Create  object used to pass pocketing options - used for area clearance only
local pocket_data = PocketParameterData()
   -- start depth for toolpath
   pocket_data.StartDepth = start_depth

   -- cut depth for toolpath this is depth below start depth
   pocket_data.CutDepth = flat_depth

   -- direction of cut for offet clearance - ProfileParameterData.CLIMB_DIRECTION
   -- or ProfileParameterData.CONVENTIONAL_DIRECTION - NOTE: enum from ProfileParameterData
   pocket_data.CutDirection = ProfileParameterData.CLIMB_DIRECTION

   -- if true use raster clearance strategy , else use offset area clearance
   pocket_data.DoRasterClearance = false
   -- angle for raster if using raster clearance
   pocket_data.RasterAngle = 0
   -- type of profile pass to perform  PocketParameterData.PROFILE_NONE ,
   -- PocketParameterData.PROFILE_FIRST orPocketParameterData.PROFILE_LAST
   pocket_data.ProfilePassType = PocketParameterData.PROFILE_LAST


-- if this is true we create 2d toolpaths previews in 2d view, if false we dont
local create_2d_previews = true

-- if this is true we will display errors and warning to the user
local display_warnings = true

-- if we are doing two tool pocketing define tool to use for area clearance
local area_clear_tool = nil

-- we just create a 10mm end mill

area_clear_tool = Tool(
                      "Lua Clearance End Mill",
                      Tool.END_MILL       -- BALL_NOSE, END_MILL, VBIT
                      )

area_clear_tool.InMM = true
area_clear_tool.ToolDia = 10
area_clear_tool.Stepdown = 3
area_clear_tool.Stepover = 3
area_clear_tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN,
                                         -- INCHES_SEC, INCHES_MIN, FEET_MIN
```

```
   area_clear_tool.FeedRate = 30
   area_clear_tool.PlungeRate = 10
   area_clear_tool.SpindleSpeed = 20000
   area_clear_tool.ToolNumber = 2

   -- Create object which can be used to automatically select geometry
   local geometry_selector = GeometrySelector()

   -- Create our toolpath
   local toolpath_manager = ToolpathManager()

   local toolpath_id = toolpath_manager:CreateVCarvingToolpath(
                                            name,
                                            tool,
                                            area_clear_tool,
                                            vcarve_data,
                                            pocket_data,
                                            pos_data,
                                            geometry_selector,
                                            create_2d_previews,
                                            display_warnings
                                            )

   if toolpath_id  == nil  then
      DisplayMessageBox("Error creating toolpath")
      return false
   end

   return true

end


--[[ --------------  main ------------------------------
|  Entry point for script
|
]]

function main()

    -- Check we have a job loaded
    job = VectricJob()

    if not job.Exists then
       DisplayMessageBox("No job loaded")
       return false;
    end

   local selection = job.Selection
   if selection.IsEmpty  then
       MessageBox("Please select one or more vectors to VCarve")
       return false
    end

    local start_depth = 0.0
    local flat_depth = 5.0
    local vbit_angle = 90.0
    local tool_dia = 32.0
    local tool_stepdown = 2.5
    local tool_stepover_percent = 2.0
    local tool_in_mm = true

    local success = CreateVCarvingToolpath(
                                          "Lua VCarving Toolpath",
                                          start_depth,
                                          flat_depth,
                                          vbit_angle,
                                          tool_dia,
                                          tool_stepdown,
                                          tool_stepover_percent,
                                          tool_in_mm
                                          )


    return success;
end
```

:CreatePrismCarvingToolpath(

String name,
Tool tool,
PrismCarveParameterData prism_data,
ToolpathPosData pos_data,
GeometrySelector geometry_selector,
bool create_2d_preview,
bool interactive
)

Creates a prism carving toolpath for the currently selected vectors. Returns the UUID for the toolpath created.

name-string - Name for the toolpath to be created
tool -Tool -Tool to use for the toolpath
prism_data -PrismParameterData -Settings for prism carving
pos_data-ToolpathPosData -Settings for home position, safe z etc
geometry_selector-GeometrySelector -Can be used to automatically select vectors on layers etc
create_2d_preview-bool -If true create preview vectors in 2d view
interactive-bool -If true display warnings etc to user

Example Code - from Create_Prism_CarvingToolpath.lua

```
-- VECTRIC LUA SCRIPT
require "strict"
--[[  ------------------- CreatePrismToolpath -------------------
|
|   Create a prism toolpath within the program for the currently selected vectors
|  Parameters:
|     name,                  -- Name for toolpath
|     start_depth       -- Start depth for toolpath below surface of material
|    cut_depth           -- cut depth for drilling toolpath
     vbit_angle           -- angle of vbit to use
|    vbit_dia              -- diameter of VBit  to use
|    vbit_stepdown     -- stepdown for tool
|    tool_stepover_percent - percentage stepover for tool
|    tool_in_mm         -- true if tool size and stepdown are in mm
|
|  Return Values:
|     true if toolpath created OK else false
|
]]

function CreatePrismToolpath( name , start_depth, cut_depth, vbit_angle,
                              vbit_dia, vbit_stepdown, tool_stepover_percent, tool_in_mm)

   -- Create tool we will use to machine vectors
   local tool = Tool(
                    "Lua VBit",
                    Tool.VBIT       -- BALL_NOSE, END_MILL, VBIT
                    )


   tool.InMM = tool_in_mm
   tool.ToolDia = vbit_dia
   tool.Stepdown = vbit_stepdown
   tool.Stepover = vbit_dia * (tool_stepover_percent / 100)
   tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN,
                                 -- INCHES_SEC, INCHES_MIN, FEET_MIN
   tool.FeedRate = 30
   tool.PlungeRate = 10
   tool.SpindleSpeed = 20000
   tool.ToolNumber = 1
   tool.VBitAngle = 90.0               -- used for vbit only
```

```
   tool.ClearStepover = vbit_dia * (tool_stepover_percent / 100) * 2  -- used for vbit only


   -- Create object used to set home position and safez gap above material surface
   local pos_data = ToolpathPosData()

   pos_data:SetHomePosition(0, 0, 5.0)
   pos_data.SafeZGap = 5.0

   -- Create  object used to pass profile options
   local prism_data = PrismCarveParameterData()
      -- start depth for toolpath
      prism_data.StartDepth = start_depth

      -- cut depth for toolpath this is depth below start depth
      prism_data.CutDepth = cut_depth

      -- direction of cut for offet clearance - ProfileParameterData.CLIMB_DIRECTION
      -- or ProfileParameterData.CONVENTIONAL_DIRECTION - NOTE: enum from ProfileParameterData
      prism_data.CutDirection = ProfileParameterData.CLIMB_DIRECTION

   -- calculate the minimum cut depth to fully form the bevel on the current
   -- selection with the current tool
   local min_bevel_depth = prism_data:CalculateMinimumBevelDepth(tool, true)
   if min_bevel_depth > cut_depth then
      DisplayMessageBox(
         "A prism will not be fully formed with a depth of " .. cut_depth .. "\r\n" ..
         "A depth of " .. min_bevel_depth .. " is required to fully form the prism"
          )
   end

   -- Create object which can be used to automatically select geometry
   local geometry_selector = GeometrySelector()

   -- if this is true we create 2d toolpaths previews in 2d view, if false we dont
   local create_2d_previews = true

   -- if this is true we will display errors and warning to the user
   local display_warnings = true

   -- Create our toolpath

   local toolpath_manager = ToolpathManager()

   local toolpath_id = toolpath_manager:CreatePrismCarvingToolpath(
                                          name,
                                          tool,
                                          prism_data,
                                          pos_data,
                                          geometry_selector,
                                          create_2d_previews,
                                          display_warnings
                                          )

   if toolpath_id  == nil  then
      DisplayMessageBox("Error creating toolpath")
      return false
   end

   return true

end


--[[  ------------------- main -------------------
|
|  Entry point for script
|
]]

function main()

    -- Check we have a job loaded
    job = VectricJob()

    if not job.Exists then
       DisplayMessageBox("No job loaded")
```

```
      return false;
    end

   local selection = job.Selection
   if selection.IsEmpty  then
      MessageBox("Please select one or more vectors to bevel carve")
      return false
    end

    local start_depth = 0.0
    local cut_depth = 20.0
    local vbit_angle = 90.0
    local tool_dia = 32.0
    local tool_stepdown = 2.5
    local tool_stepover_percent = 2.0
    local tool_in_mm = true

    local success = CreatePrismToolpath(
                                        "Lua Prism Toolpath",
                                        start_depth,
                                        cut_depth,
                                        vbit_angle,
                                        tool_dia,
                                        tool_stepdown,
                                        tool_stepover_percent,
                                        tool_in_mm
                                        )


    return success;
end
```

:CreateFlutingToolpath(

String name,
Tool tool,
FlutingParameterDatafluting_data,
ToolpathPosData pos_data,
GeometrySelector geometry_selector,
bool create_2d_preview,
bool interactive
)

Creates a fluting toolpath for the currently selected vectors. Returns the UUID for the toolpath created.

name-string - Name for the toolpath to be created
tool -Tool -Tool to use for the toolpath
fluting_data -FlutingParameterData -Settings for fluting
pos_data-ToolpathPosData -Settings for home position, safe z etc
geometry_selector-GeometrySelector -Can be used to automatically select vectors on layers etc
create_2d_preview-bool -If true create preview vectors in 2d view
interactive-bool -If true display warnings etc to user

Example Code - from Create_Fluting_Toolpath.lua

```
-- VECTRIC LUA SCRIPT
require "strict"
--[[  ----------------- CreateFlutingToolpath -----------------
|
|   Create a drilling toolpath within the program for the currently selected vectors
|  Parameters:
|    name,               -- Name for toolpath
|    start_depth         -- Start depth for toolpath below surface of material
|    cut_depth           -- cut depth for toolpath
|    tool_dia            -- diameter of tool to use
|    tool_stepdown       -- stepdown for tool
|    tool_in_mm          -- true if tool size and stepdown are in mm
|
|  Return Values:
|     true if toolpath created OK else false
|
]]

function CreateFlutingToolpath( name , start_depth, cut_depth, tool_dia, tool_stepdown,
tool_in_mm)

   -- Create tool we will use to machine vectors
   local tool = Tool(
                   "Lua Ball Nose",
                   Tool.BALL_NOSE       -- BALL_NOSE, END_MILL, VBIT, THROUGH_DRILL
                   )

   tool.InMM = tool_in_mm
   tool.ToolDia = tool_dia
   tool.Stepdown = tool_stepdown
   tool.Stepover = tool_dia * 0.25
   tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN, INCHES_SEC,
                                 -- INCHES_MIN, FEET_MIN
   tool.FeedRate = 30
   tool.PlungeRate = 10
   tool.SpindleSpeed = 20000
```

```
   tool.ToolNumber = 1
   tool.VBitAngle = 90.0                -- used for vbit only
   tool.ClearStepover = tool_dia * 0.5  -- used for vbit only


   -- Create object used to set home position and safez gap above material surface
   local pos_data = ToolpathPosData()

   pos_data:SetHomePosition(0, 0, 5.0)
   pos_data.SafeZGap = 5.0

   -- Create  object used to pass fluting options
   local fluting_data = FlutingParameterData()
      -- start depth for toolpath
      fluting_data.StartDepth = start_depth

      -- cut depth for toolpath this is depth below start depth
      fluting_data.CutDepth = cut_depth

      -- type of fluting FULL_LENGTH, RAMP_START or RAMP_START_END
      fluting_data.FluteType = FlutingParameterData.RAMP_START_END

      -- type of ramping RAMP_LINEAR, RAMP_SMOOTH
      fluting_data.RampType = FlutingParameterData.RAMP_LINEAR

      -- if true use ratio field for controling ramp length else absolute length value
      fluting_data.UseRampRatio = false

      -- length of ramp as ratio of flute length(range 0 - 1.0)
      -- (for start and end -  ratio is of half length)
      fluting_data.RampRatio = 0.2

      -- length to ramp over - if UseRampRatio == false
      fluting_data.RampLength = 15

      -- if true in Aspire, project toolpath onto composite model
      fluting_data.ProjectToolpath = false

   -- Create object which can be used to automatically select geometry
   local geometry_selector = GeometrySelector()

   -- if this is true we create 2d toolpaths previews in 2d view, if false we dont
   local create_2d_previews = true

   -- if this is true we will display errors and warning to the user
   local display_warnings = true

   -- Create our toolpath

   local toolpath_manager = ToolpathManager()

   local toolpath_id = toolpath_manager:CreateFlutingToolpath(
                                          name,
                                          tool,
                                          fluting_data,
                                          pos_data,
                                          geometry_selector,
                                          create_2d_previews,
                                          display_warnings
                                          )

   if toolpath_id  == nil  then
      DisplayMessageBox("Error creating toolpath")
      return false
   end

   return true

end


--[[  ----------------- main -----------------
|
|  Entry point for script
|
]]
```

```
function main()

    -- Check we have a job loaded
    job = VectricJob()

    if not job.Exists then
       DisplayMessageBox("No job loaded")
       return false;
    end

   local selection = job.Selection
   if selection.IsEmpty  then
       MessageBox("Please select one or more vectors to flute")
       return false
    end

    local start_depth = 0.0
    local cut_depth = 5.0
    local tool_dia = 15.0
    local tool_stepdown = 5.0
    local tool_in_mm = true

    local success = CreateFlutingToolpath(
                                          "Lua Fluting Toolpath",
                                          start_depth,
                                          cut_depth,
                                          tool_dia,
                                          tool_stepdown,
                                          tool_in_mm
                                          )


    return success;
end
```

:CreateRoughingToolpath(- Aspire Only

String name,
Tool tool,
RoughingParameterData roughing_ data,
ToolpathPosData pos_data,
GeometrySelector geometry_selector,
bool interactive
)

Creates a roughing toolpath. Returns the UUID for the toolpath created.

NOTE: This method is only available in Aspire

name-string - Name for the toolpath to be created
tool -Tool -Tool to use for the toolpath
roughing_data -RoughingParameterData -Settings for roughing, strategy etc
pos_data-ToolpathPosData -Settings for home position, safe z etc
geometry_selector-GeometrySelector -Can be used to automatically select vectors on layers etc
interactive-bool -If true display warnings etc to user

Example Code from 05_Roughing_Toolpath.lua

```
function CreateRoughingToolpath()

  -- Toolpath name
  local name = "Lua Roughing Toolpath"

  -- Metric unit parameters
  local tool_dia = 6.35 -- Quarter of an inch
  local tool_stepdown = 5
  local tool_stepover_percent = 40

  -- Create tool we will use to machine vectors
  local tool = Tool("Lua End Mill", Tool.END_MILL)

  tool.InMM = true
  tool.ToolDia = tool_dia
  tool.Stepdown = tool_stepdown
  tool.Stepover = tool_dia * (tool_stepover_percent / 100)
  tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN, INCHES_SEC, INCHES_MIN,
FEET_MIN
  tool.FeedRate = 30
  tool.PlungeRate = 10
  tool.SpindleSpeed = 20000
  tool.ToolNumber = 1
  tool.VBitAngle = 90.0                -- used for vbit only
  tool.ClearStepover = tool_dia * (tool_stepover_percent / 100)  -- used for vbit only

  -- we will set home position and safe z relative to material block size
  local mtl_block = MaterialBlock()
  local mtl_box = mtl_block.MaterialBox
  local mtl_box_blc = mtl_box.BLC

   -- Create object used to set home position and safez gap above material surface
  local pos_data = ToolpathPosData()
  pos_data:SetHomePosition(mtl_box_blc.x, mtl_box_blc.y, mtl_box.TRC.z + (mtl_block.Thickness
* 0.2))
  pos_data.SafeZGap = mtl_block.Thickness * 0.1

  -- Pocketing parameters
  local start_depth = 0
  local cut_depth = mtl_box.ZLength

  -- Create object used to pass roughing options
  local roughing_data = RoughingParameterData()
```

```
  -- start depth for toolpath
  roughing_data.StartDepth = start_depth

  -- cut depth for toolpath this is depth below start depth
  roughing_data.CutDepth = cut_depth

  -- Machining allowance
  roughing_data.MachiningAllowance = 0.0

  -- Allowance to leave on when machining - (this is different from machining allowance)
  roughing_data.Allowance = 0.0

  -- if true use z level roughing
  roughing_data.DoZLevelRoughing = true
  -- z level clearance strategy
  roughing_data.ZLevelStrategy = RoughingParameterData.RASTER_X
  roughing_data.ZLevelProfile = RoughingParameterData.LAST
  -- angle for raster if using raster clearance
  roughing_data.RasterAngle = 0

  -- if true we ramp into pockets (always zig-zag)
  roughing_data.DoRamping = false
  --  if ramping, distance to ramp over
  roughing_data.RampDistance = 10.0

  -- Create object which can used to automatically select geometry on layers etc
  local geometry_selector = GeometrySelector()

  -- if this is true we will display errors and warning to the user
  local display_warnings = true

  -- Create our toolpath
  local toolpath_manager = ToolpathManager()
  local toolpath_id = toolpath_manager:CreateRoughingToolpath(
                                          name,
                                          tool,
                                          roughing_data,
                                          pos_data,
                                          geometry_selector,
                                          display_warnings
                                          )

  if not toolpath_id then
    DisplayMessageBox("Error creating toolpath")
  end

end
```

:CreateFinishingToolpath(- Aspire Only

String name,
Tool tool,
PocketParameterData pocket_data,
ToolpathPosData pos_data,
GeometrySelector geometry_selector,
bool create_2d_preview,
bool interactive
)

Creates a finishing toolpath for the currently selected vectors. Returns the UUID for the toolpath created.

NOTE: This method is only available in Aspire

name-string - Name for the toolpath to be created
tool -Tool -Tool to use for the toolpath
pocket_data -PoketParameterData - Setting for pocketing depth, style etc
pos_data-ToolpathPosData -Settings for home position, safe z etc
geometry_selector-GeometrySelector -Can be used to automatically select vectors on layers etc
create_2d_preview-bool -If true create preview vectors in 2d view
interactive-bool -If true display warnings etc to user

Example Code from 04_Finishing_Toolpath.lua

```
function CreateFinishingToolpath()

  -- Toolpath name
  local name = "Lua Finishing Toolpath"

  -- Metric unit parameters
  local tool_dia = 3.175 -- 8th of an inch
  local tool_stepdown = 5
  local tool_stepover_percent = 10

  -- Create tool we will use to machine vectors
  local tool = Tool("Lua Ball Nose", Tool.BALL_NOSE)

  tool.InMM = true
  tool.ToolDia = tool_dia
  tool.Stepdown = tool_stepdown
  tool.Stepover = tool_dia * (tool_stepover_percent / 100)
  tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN, INCHES_SEC, INCHES_MIN,
FEET_MIN
  tool.FeedRate = 30
  tool.PlungeRate = 10
  tool.SpindleSpeed = 20000
  tool.ToolNumber = 1
  tool.VBitAngle = 90.0                -- used for vbit only
  tool.ClearStepover = tool_dia * (tool_stepover_percent / 100)  -- used for vbit only

  -- we will set home position and safe z relative to material block size
  local mtl_block = MaterialBlock()
  local mtl_box = mtl_block.MaterialBox
  local mtl_box_blc = mtl_box.BLC

   -- Create object used to set home position and safez gap above material surface
  local pos_data = ToolpathPosData()
```

```
  pos_data:SetHomePosition(mtl_box_blc.x, mtl_box_blc.y, mtl_box.TRC.z +
(mtl_block.Thickness * 0.2))
  pos_data.SafeZGap = mtl_block.Thickness * 0.1

  -- Pocketing parameters
  local start_depth = 0
  local cut_depth = mtl_box.ZLength

  -- Create object used to pass pocketing options
  local pocket_data = PocketParameterData()
  -- start depth for toolpath
  pocket_data.StartDepth = start_depth

  -- cut depth for toolpath this is depth below start depth
  pocket_data.CutDepth = cut_depth

  -- direction of cut for offet clearance - ProfileParameterData.CLIMB_DIRECTION or
ProfileParameterData.CONVENTIONAL_DIRECTION - NOTE: enum from ProfileParameterData
  pocket_data.CutDirection = ProfileParameterData.CLIMB_DIRECTION

  -- Allowance to leave on when machining
  pocket_data.Allowance = 0.0

  -- if true use raster clearance strategy , else use offset area clearance
  pocket_data.DoRasterClearance = true
  -- angle for raster if using raster clearance
  pocket_data.RasterAngle = 0
  -- type of profile pass to perform  PocketParameterData.PROFILE_NONE ,
PocketParameterData.PROFILE_FIRST orPocketParameterData.PROFILE_LAST
  pocket_data.ProfilePassType = PocketParameterData.PROFILE_LAST

  -- if true we ramp into pockets (always zig-zag)
  pocket_data.DoRamping = false
  --  if ramping, distance to ramp over
  pocket_data.RampDistance = 10.0

  -- if true in Aspire, project toolpath onto composite model
  pocket_data.ProjectToolpath = false

  -- Create object which can used to automatically select geometry on layers etc
  local geometry_selector = GeometrySelector()

  -- if this is true we create 2d toolpaths previews in 2d view, if false we dont
  local create_2d_previews = true

  -- if this is true we will display errors and warning to the user
  local display_warnings = true

  -- Create our toolpath
  local toolpath_manager = ToolpathManager()
  local toolpath_id = toolpath_manager:CreateFinishingToolpath(
                                          name,
                                          tool,
                                          pocket_data,
                                          pos_data,
                                          geometry_selector,
                                          create_2d_previews,
                                          display_warnings
                                          )

  if not toolpath_id then
    DisplayMessageBox("Error creating toolpath")
  end

end
```

:LoadToolpathTemplate(string template_path)

Load a toolpath template from passed file. Returns true if template loaded OK, else false

template_path - string - Full path to template file to load

Example Code

```
-- VECTRIC LUA SCRIPT

require "strict"


--[[  -------------- main ---------------------
|
|  Entry point for script
|
]]

function main()

    -- Check we have a job loaded
    local job = VectricJob()

    if not job.Exists then
       DisplayMessageBox("No job loaded")
       return false;
    end

    local template_path = "c:\\temp\\TestToolpathTemplate.ToolpathTemplate"

    local toolpath_manager = ToolpathManager()

    if not toolpath_manager:LoadToolpathTemplate(template_path) then
       MessageBox("Failed to load template " .. template_path)
           return false
    end

    MessageBox("Loaded template " .. template_path)

    local calc_result = toolpath_manager:RecalculateAllToolpaths()
    if calc_result == nil then
        MessageBox("Recalculate all toolpaths failed")
    else
       MessageBox("Results from recalculate all\n" .. calc_result)
    end

    return true;
end
```

:CopyToolpathWithId(UUID id, bool insert_at_end, bool rename)

Copy (duplicate) toolpath with passed id - returns copied toolpath and adds to toolpath list

id -UUID - Id of toolpath to copy
insert_at_end -bool -If true, new toolpath is inserted at end of list, else following original to copy
rename -bool -If true, new toolpath is automatically renamed, else keeps same name

:DeleteAllToolpath ()

Delete ALL toolpaths in job.

:DeleteToolpath (Toolpath toolpath)

Delete passed toolpath - make sure you do not refer to toolpath in script after this call!

toolpath -Toolpath - Toolpath to delete

:DeleteToolpathWithId(UUID id)

Delete toolpath with passed id

id - UUID -id of toolpath to delete

:Find(UUID id)

Returns the POSITION in the list of the toolpath with hte passed id - nil if no toolpath with id found

id - UUID -id of toolpath to find position for

:GetAt(pos)

Returns the toolpath at the passed position

pos - POSITION - current position in list

:GetGroupToolpathWithIndex(UUID id, integer index)

Return the 'n'th toolpathof the passed group id. Toolpaths created together such as the two toolpaths from a flat bottom v-carve with two tools share a unique groupid property.

id -UUID - Id of toolpath group
index -integer -Index (inrange 0 to n-1) of toolpath in group to return

:GetNext(pos)

Returns the toolpath at the current position AND a new value for position pointing to the next item in the list (or nil if at end of list)

pos - POSITION - current position in list

Example - note that GetNext(pos) is returning two values ...

```
local pos = toolpath_manager:GetHeadPosition()
local toolpath
while pos ~= nil do
    toolpath, pos = toolpath _manager:GetNext(pos)
    DO SOMETHING WITH TOOLPATH ....
end
```

:GetNumberOfToolpathsInGroup(UUID id)

Return the number of toolpaths which have the passed group id. Toolpaths created together such as the two toolpaths from a flat bottom v-carve with two tools share a unique groupid property.

id -UUID - Id of toolpath group to count members for

:GetPrev(pos)

Returns the toolpath at the current position AND a new value for position, pointing to the previous item in the list (or nil if at start of list)

pos - POSITION - current position in list

:GetSelectedToolpath()

Returns the currently selected toolpath or nil if no toolpath currently selected

:GetHeadPosition()

Returns a POSITION variable to allow access to the head of the list of toolpaths

:GetTailPosition()

Returns a POSITION variable to allow access to the tail of the list of toolpaths

:RecalculateAllToolpaths()

Recalculate all toolpaths in the job. Returns a string containing a list of all the toolpaths recalculated if it succeeds, else nil.

WARNING: Recalculating all toolpaths will destroy the original toolpaths and create new ones with the same ID. Do not try and reuse a toolpath variable after this call. See :RecalculateToolpath() for information on how to retrieve a specific toolpath after it has been recalculated.

Example Code

```
local toolpath_manager = ToolpathManager()

local calc_result = toolpath_manager:RecalculateAllToolpaths()
if calc_result == nil then
   MessageBox("Recalculate all toolpaths failed")
else
   MessageBox("Results from recalculate all\n" .. calc_result)
end
```

:RecalculateToolpath(Toolpath toolpath)

Recalculates passed toolpath. Returns true if toolpath recalculated ok

toolpath - Toolpath - Toolpath to recalculate

WARNING: The passed toolpath is invalid after this call as a new toolpath with the same id is created internally. If you wish to continue accesing the toolpath after recalculating, save its id BEFORE calling :RecalculateToolpath() and then use :Find(id) to find the position of the recalculated toolpath and use :GetAt(position) to return the recalculated toolpath.

Example Code

```
local orig_id = luaUUID()
orig_id:SetId(toolpath.Id)
local recalced_toolpath = nil
if toolpath_manager:RecalculateToolpath(toolpath) then
   local recalced_pos = toolpath_manager:Find(orig_id.RawId)
   if recalced_pos == nil then
      MessageBox("Failed to find toolpath after calculation")
      return false
   else
      recalced_toolpath = toolpath_manager:GetAt(recalced_pos)
   end
end
```

:SaveToolpathAsTemplate(Toolpath toolpath, string template_path)

Save a toolpath as a template to passed file. Returns true if template saved, else false. If passed toolpath is part of a group (e.g flat bottom vcarving) all toolpaths in group are saved to the template

toolpath - Toolpath - Toolpath to save as a template
template_path - string - Full path to template file to save

:SaveVisibleToolpathsAsTemplate(string template_path)

Save all visible toolpaths as a template to passed file. Returns true if template saved, else false.

template_path - string - Full path to template file to save

:ToolpathModified(Toolpath toolpath)

This method must be called if parameters for a toolpath are modified from script. E.g tool parameters are changed. If this method is not called the ui will not show the changes you have made.

toolpath - Toolpath - Toolpath which has been modified

:ToolpathWithNameExists(string name)

Returns true if there is one or more existing toolpaths with passed name

name -string - Toolpath name to check for

id -UUID - Id of toolpath to delete

:UndrawAllToolpath ()

Undraw (set visibility to false) ALL toolpaths in job.

:SetAllToolpathsVisibility (bool visibility)

Set visibility for all toolpaths in job.

visibility -bool - true to draw all toolpaths, false to stop drawing them

Toolpath

The Toolpath object represents a toolpath within the program. Toolpaths are never created directly, they are created using the CreateXXX methods of ToolpathManager.

Properties

.ActiveSheetIndex

R/O- integer- Active sheet index when toolpath was calculated. Older toolpaths will return 0 for this value until recalculated, use HasActiveSheetIndex to check for this case.

.FirstPoint

R/O- Point3D- Position of first point in toolpath - start of first plunge move

.GroupId

R/O-UUID - returns group id of toolpath, if this toolpath is part of a group (e.g from flat bottomed v-carving). All toolpaths in a group will have the same group id

.HasActiveSheetIndex

R/O- bool- true if this toolpath has a record of active sheet index when it was calculated

.Id

R/O-UUID - returns id of toolpath, all toolpaths have a unique id

.InMM

R/O- bool - flag indicating if toolpath is in mm or inches

.LastPoint

R/O- Point3D- Position of last point in toolpath - end of last retract move

.Name

R/W- string - name of toolpath - call ToolpathManager:ToolpathModified() after changing

.Notes

R/W- string - notes for toolpath - call ToolpathManager:ToolpathModified() after changing

.PositionData

R/O- ToolpathPosData- Position data (home pos, safe z etc) for toolpath

.Tool

R/O- Tool- returns a Tool object which references the tool in the toolpath and can be used to update the tool data. See Tool:`UpdateToolParameters()` for more details. Also call ToolpathManager:ToolpathModified() after changing tool values.

Methods

:DeleteActiveSheetIndex()
Delete the record of the active sheet index for this toolpath.

:MachiningTime()
Return estimated machining time for toolpath. This estimate will use the Rapid Feed rate and Scale Factor set within the program on the 'Toolpaths Summary' page.

:ReplaceTool(Tool tool)
Replace the existing tool with the passed tool.

tool -Tool -tool which will replace the current tool in the toolpath

:Statistics()
Return a ToolpathStats object with information about toolpath such as tolal length of feed rate moves etc.

:Transform(Matrix2D xform)
Transform the toolpath using passed xform. Only translation and rotation are supported

xform -Matrix2D -transformation matrix to apply to toolpath

ToolDatabase

The ToolDatabase object gives access to the single Tool database within the program. Currently this object can only be used to select an existing tool from the database.

Constructor

ToolDatabase - constructor

Returns a new object which gives access to the single Tool database for the program.

e.g

```
local tooldb = ToolDatabase()
```

Methods

:SelectTool()

Displays the Tool Database dialog and allows the user to choose a tool which is returned from this method.

e.g

```
local tool_database = ToolDatabase();

local tool = tool_database:SelectTool();
```

Tool

The Tool object represents a tool within the program. All the methods for creating toolpaths take one or more Tool objects as arguments. Any of the supported tool types can be accessed from the ToolDatabase as shown in the section describing the ToolDatabase:SelectTool() method. In addition the most common tool types (BallNose, EndMill, VBit and Drill) can be created programmatically from Lua.

Constructor

Tool(string name, ToolType tool_type) - constructor

Create a new tool. Only a limited number of tool types are supported for creation from script. To access the full range of tools use the ToolDatabase:SelectTool() method to allow users to select tools from the tool database.

name-string - Name for the too to be created
tool_type -integer -Type of tool to create. Valid values are ...
Tool.BALL_NOSE
Tool.END_MILL
Tool.VBIT
Tool.THROUGH_DRILL

NOTE: If you change ANY parameters / properties on a tool retrieved from an **EXISTING** toolpath, you **MUST** call UpdateParameters() after you have finished editing tool properties to flush the changes through to the toolpath.

Properties

.ClearStepover

R/W - double - Clearance pass stepover for tool - VBits only

.InMM

R/W - bool - flag indicating if tool is in mm or inches

.FeedRate

R/W - double - Feedrate for tool - see RateUnits for the units

.Name

R/O - string - returns name of tool

.Notes

R/W - string – Notes field for tool . (Aspire 4.015/VCP 7.015 onwards only)

.PlungeRate

R/W - double - Feedrate for tool - see RateUnits for the units

.RateUnits
R/W - integer - Feed rate units for tool. Valid values are

Tool.MM_SEC
Tool.MM_MIN
Tool.METRES_MIN
Tool.INCHES_SEC
Tool.INCHES_MIN
Tool.FEET_MIN

.RateUnitsText
R/O -string - Feed rate units for tool as a string for display e.g “mm/sec” or “inch/sec”

.SpindleSpeed
R/W - integer - Spindle speed for tool

.Stepdown
R/W - double - Stepdown for tool

.Stepover
R/W - double -Stepover for tool

.ToolDB_Location
R/O - string - returns path to tool in tool database - not including tool name

.ToolType
R/O - integer - Returns an integer indicating the type of the tool – values are …

Tool.BALL_NOSE
Tool.END_MILL
Tool.RADIUSED_END_MILL
Tool.VBIT
Tool.ENGRAVING
Tool.RADIUSED_ENGRAVING
Tool.THROUGH_DRILL
Tool.FORM_TOOL
Tool.DIAMOND_DRAG
Tool.RADIUSED_FLAT_ENGRAVING

.ToolTypeText
R/O - string - Returns a string indicating the type of the tool

.ToolDia
R/W - double - Diameter of tool

.ToolNumber
R/W - integer - Tool number for tool

.VBit_Angle
R/W - double - Included angle for VBit tools

Methods

:ConvertRateUnitsTo(integer new_units)
Convert the feed and plunge rates for the tool into the passed units.

new_units-integer -new units for feed and plunge rate. Valid values are *the same as for* .RateUnits above

:IsCompatibleFeedRates(Tool tool_to_check)
Return true if this tool is has same feed, plunge and spindles speeds as passed tool. Does not check geometry or cut depths (see: IsCompatibleTool and :IsCompatibleCutDepthss)

tool_to_check-Tool -tool we check for compatibility

:IsCompatibleTool(Tool tool_to_check)
Return true if this tool is compatible (same type and geometry) as passed tool. Does not check cut depths or feedrates (see :IsCompatibleCutDepths and :IsCompatibleFeedRates for this data)

tool_to_check-Tool -tool we check for compatibility

:IsCompatibleCutDepths(Tool tool_to_check)
Return true if this tool is has same cut depths as passed tool. Does not check geometry or feedrates (see: IsCompatibleTool and :IsCompatibleFeedRates)

Tool_to_check-Tool -tool we check for compatibility

:FlatRadius(bool in_mm)
Retrieve the flat radius for tools flat tools otherwise returns zero.

in_mm- bool -If true return the tip radius in millimetres

:GetBool(string parameter_name, bool default_value)
Retrieve a Boolean flag (true / false) with the passed name, if no value with passed name returns passed default value. (Aspire 4.015/VCP 7.015 onwards only)

parameter_name - string - the name of the parameter
default_value - bool - the value which will be returned if there is no existing value stored

:GetDouble(string parameter_name, double default_value,)
Retrieve a double with the passed name, if no value with passed name returns passed default value. (Aspire 4.015/VCP 7.015 onwards only)

parameter_name - string - the name of the parameter
default_value - double - the value which will be returned if there is no existing value stored

:GetInt(string parameter_name, integer default_value)

Retrieve an integer with the passed name, if no value with passed name returns passed default value. (Aspire 4.015/VCP 7.015 onwards only)

parameter_name - string - the name of the parameter
value - integer - the value which will be returned if there is no existing value stored

:ParameterExists(string parameter_name, utParameterType type)

Returns true if there is an existing parameter with passed name and type. (Aspire 4.015/VCP 7.015 onwards only)

parameter_name - string - the name of the parameter
type - utParameterType - the type of parameter

:SetBool(string parameter_name, bool value)

Store a Boolean flag (true / false) with the passed name and value. (Aspire 4.015/VCP 7.015 onwards only)

parameter_name - string - the name which will be used to store and retrieve the value
value - bool - the value which will be stored in the parameter list

:SetDouble(string parameter_name, double value)

Store a double with the passed name and value. (Aspire 4.015/VCP 7.015 onwards only)

parameter_name - string - the name which will be used to store and retrieve the value
value - double - the value which will be stored in the parameter list

:SetInt(string parameter_name, integer value)

Store an integer with the passed name and value. (Aspire 4.015/VCP 7.015 onwards only)

parameter_name - string - the name which will be used to store and retrieve the value
value - integer - the value which will be stored in the parameter list

:SetString(string parameter_name, string value)

Store a string with the passed name and value. (Aspire 4.015/VCP 7.015 onwards only)

parameter_name - string - the name which will be used to store and retrieve the value
value - string - the value which will be stored in the parameter list

:TipRadius(bool in_mm)

Retrieve the tip radius for tools of type Tool.RADIUSED_END_MILL or Tool.RADIUSED_ENGRAVING otherwise returns zero.

in_mm- bool -If true return the tip radius in millimetres

:UpdateParameters()

If editing a tool on an existing toolpath update feedrates and other parameters set on the tool to new values .

NOTE: You **MUST** call this method if you have edited any parameters on an existing tool, otherwise your changes will be ignored!

Example Code

```
-- Create tool we will use to machine vectors
local tool = Tool(
                "Lua End Mill",
                Tool.END_MILL       -- BALL_NOSE, END_MILL, VBIT
                )

tool.InMM = tool_in_mm
tool.ToolDia = tool_dia
tool.Stepdown = tool_stepdown
tool.Stepover = tool_dia * 0.25
tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN,
                              -- INCHES_SEC, INCHES_MIN, FEET_MIN
tool.FeedRate = 30
tool.PlungeRate = 10
tool.SpindleSpeed = 20000
tool.ToolNumber = 1
tool.VBitAngle = 90.0                 -- used for vbit only
tool.ClearStepover = tool_dia * 0.5  -- used for vbit only
```

ToolpathPosData

This object is used to pass data to the various toolpath creation function in the ToolpathManager object to specify the home position and safe z etc.

Constructor

ToolpathPosData() - Constructor

Creates a new ToolpathPosData object with default values.

Properties

.HomeX

R/O - double - X value for the home position (where tool starts from)

.HomeY

R/O - double - Y value for the home position (where tool starts from)

.HomeZ

R/O - double - Z value for the home position (where tool starts from)

.InMM

R/O - bool - True if program working in mm, else inches

.SafeZ

R/O - double - Absolute Z value for the safe z moves (rapid moves)

.SafeZGap

R/W - double - Get / Set the SafeZ gap - distance above surface of material for rapid moves

.StartZGap

R/W - double - Get / Set the StartZ gap - distance above surface of material when plunges change from rapid to plunge federate.

Methods

:EnsureHomeZIsSafe()

As Home position is specified as an absolute value, it is possible to set it programmatically to an invalid value within the block. This method will ensure that it is always at least 'SafeZ' above the material surface.

:SetHomePosition(double x, double y, double z)

Set the position from which the tool path starts and usually returns to.

x -double -X value for tool home position
y -double -Y value for tool home position
z -double -Z value for tool home position

GeometrySelector

This object is used to pass data to the various toolpath creation function in the ToolpathManager object to specify automatic selection of geometry for the toopath when it is calculated. The default constructor for the object will leave the GeometrySelector inactive, and the toolpath will calculate using the currently selected vectors.

Constructor

GeometrySelector() - Constructor

Creates a new GeomterySelector object with default values.

Properties

.AllowOpen

R/W - bool – if true selector will allow open vectors in the selection

.AllowToolDia

R/W - bool – if true selector will use *.ToolDia* field when selecting vectors

.CircleDia

R/W - double –Diameter of circles we select if *.CircleMatchCircleDia* and *.SelectCircles* is true

.CircleTolerance

R/W - double –tolerance we use when deciding if a circle matches the ToolDia / CircleDia criteria

.CircleMatchAll

R/W - bool – if true we select all circles if *.SelectCircles* is true

.CircleMatchCircleDia

R/W - bool – if true we select all circles matching *.CircleDia* if *.SelectCircles* is true

.CircleMatchToolDia

R/W - bool – if true we select all circles that match the diameter of the tool used for the toolpath if *.SelectCircles* is true

.GeometryDepthOffset

R/W - double –Offset value to add to depth

.GeometryDepthOffset Formula

R/W - string –Formula for depth offset from geometry

.MixedGroupsOk

R/W - bool – if true groups which only match some criteria are selected

.OnlyOnLayers

R/W - bool – if true only vectors on the layers added with *:AddLayerName* are selected.

.SelectClosed

R/W - bool – if true closed vectors are selected

.SelectCircles

R/W - bool – if true we are only selecting circles

.SelectOpen
R/W - bool – if true open vectors are selected

.SetDepthFromGeometry
R/W - bool – if true depth for toolpath is set from selected geometry (which must have been imported from DXF files with depth set)

.ToolDia
R/W - double –Current diameter of circles we select if *.CircleMatchToolDia* and *.SelectCircles* is true

Layer List Access

.HaveLayerNames
R/O - bool – true if the geometry selector has a list of layer names

.LayerNameCount
R/O - integer – number of layers

:AddLayerhName(string name)
Add passed layer name to list of names selector uses if *.OnlyOnLayers* is true.

name - string – name of layer to add to list

:GetAtLayerName(POSITION pos)
Returns the layer name at the current position

pos - POSITION - current position in list

:GetLayerNameHeadPosition()
Returns a POSITION variable to allow access to the head of the list of layer names

:GetNextLayerName(POSITION pos)
Returns the layer name at the current position AND a new value for position pointing to the next item in the list (or nil if at end of list)

pos - POSITION - current position in list

:GetPrevLayerName(POSITION pos)
Returns the layer name at the current position AND a new value for position, pointing to the previous item in the list (or nil if at start of list)

pos - POSITION - current position in list

:GetLayerNameTailPosition()
Returns a POSITION variable to allow access to the tail (end) of the list of layer names

:FindLayerWithName(string name)

Returns the POSITION of layer with passed name or nil if no layer in list with name found

name - string – name of layer to find position of

Methods

:HasSelectorData(Toolpath toolpath)

Returns true if the passed toolpath has selector data stored with it

toolpath -Toolpath –toolpath to check for data

:LoadSelectorData(Toolpath toolpath)

Loads the selector data from passed toolpath into this object data

toolpath -Toolpath –toolpath to get data from

:SaveSelectorData(Toolpath toolpath)

Saves the selector data to passed toolpath

toolpath -Toolpath –toolpath to set data on

:RemoveAllLayerNames()

Remove all layer names associated with this selector.

ProfileParameterData

This object is used to hold the settings for a profile toolpath.

Constructor

ProfileParameterData() - Constructor

Create a new object ready to have its parameters set

```
e.g
    local profile_data = ProfileParameterData()
```

Properties

.Allowance

R/W - double - allowance to leave on profile when calculating toolpath

.AllowanceFormula

R/W - string -Formula for allowance to leave on profile when calculating toolpath

.CutDepth

R/W - double -Final cutting depth below start depth

.CornerSharpen

R/W - bool - True if want to create 3D 'corner sharpening' moves for internal corners

.CreateSquareCorners

R/W - bool - true if want to create 'square' external corners

.CutDepthFormula

R/W - string –Optional formula for final cutting depth below start depth. The formula is the same as can be entered in the CutDepth field on the normal toolpath page and has access to the same variables (e.g "z * 0.5" to set cut depth to half the material thickness).

.CutDirection

R/W - integer - Cutting direction for toolpath. Valid values are ...

ProfileParameterData.CLIMB_DIRECTION
ProfileParameterData.CONVENTIONAL_DIRECTION

.KeepStartPoints

R/W - bool - If true, start points of vectors are maintained, else start points are optimised

.Name

R/W - string - The name for the toolpath

.ProfileSide

R/W - integer - Side of the vector to cut on. Valid values are ...

ProfileParameterData.PROFILE_OUTSIDE
ProfileParameterData.PROFILE_INSIDE
ProfileParameterData.PROFILE_ON

.ProjectToolpath

R/W - bool - If true and used with Aspire, toolpath is projected onto model surface after calculation

.StartDepth

R/W - double - Start depth for toolpath below material surface

.StartDepthFormula

R/W - string – Optional formula for start depth for toolpath below material surface. The formula is the same as can be entered in the Start Depth field on the normal toolpath page and has access to the same variables (e.g "z * 0.5" to set start depth to half the material thickness).

Example Code

```
-- Create  object used to pass profile options
local profile_data = ProfileParameterData()
   -- start depth for toolpath
   profile_data.StartDepth = start_depth

   -- cut depth for toolpath this is depth below start depth
   profile_data.CutDepth = cut_depth

   -- direction of cut - ProfileParameterData.CLIMB_DIRECTION or
   -- ProfileParameterData.CONVENTIONAL_DIRECTION
   profile_data.CutDirection = ProfileParameterData.CLIMB_DIRECTION

   -- side we machine on - ProfileParameterData.PROFILE_OUTSIDE,
   -- ProfileParameterData.PROFILE_INSIDE or ProfileParameterData.PROFILE_ON
   profile_data.ProfileSide = ProfileParameterData.PROFILE_OUTSIDE

   -- Allowance to leave on when machining
   profile_data.Allowance = 0.0

   -- true to preserve start point positions, false to reorder start points to
   -- minimise toolpath length
   profile_data.KeepStartPoints = false

   -- true if want to create 'square' external corners on toolpath
   profile_data.CreateSquareCorners = false

   -- true to perform corner sharpening on internal corners (only with v-bits)
   profile_data.CornerSharpen = false


   -- if true in Aspire, project toolpath onto composite model
   profile_data.ProjectToolpath = false
```

RampingData

This object is used to hold parameters relating to ramping for toolpaths which support ramping.

Constructor

RampingData() - Constructor

Create a new object ready to have its parameters set

```
e.g
    local ramping_data = RampingData()
```

Properties

.DoRamping

R/W - bool - If true ramping is performed. If false all other fields are ignored and tool plunges straight down.

.RampAngle

R/W - double -angle (in degrees) to ramp at if constrained by angle

.RampConstraint

R/W - integer - How the ramp is constrained, either by angle or distance. Valid values are

RampingData.CONSTRAIN_DISTANCE

RampingData.CONSTRAIN_ANGLE

.RampDistance

R/W - double - distance to ramp over if constrained by distance

.RampMaxAngleDist

R/W - double -max distance to ramp over if constrained by angle

.RampOnLeadIn

R/W - bool - If true ramps are created on the lead in moves

.RampType

R/W - integer - Type of ramping to perform. Valid values are ...

RampingData.RAMP_LINEAR

RampingData.RAMP_ZIG_ZAG

RampingData.RAMP_SPIRAL

Example Code

```
-- Create object used to control ramping
local ramping_data = RampingData()
   -- if true we do ramping into toolpath
   ramping_data.DoRamping = false
   -- type of ramping to perform RampingData.RAMP_LINEAR , RampingData.RAMP_ZIG_ZAG
   -- or RampingData.RAMP_SPIRAL
   ramping_data.RampType = RampingData.RAMP_ZIG_ZAG
   -- how ramp is contrained - either by angle or distance RampingData.CONSTRAIN_DISTANCE
   -- or RampingData.CONSTRAIN_ANGLE
   ramping_data.RampConstraint = RampingData.CONSTRAIN_ANGLE
   -- if we are constraining ramp by distance, distance to ramp over
   ramping_data.RampDistance = 100.0
   -- if we are contraining ramp by angle , angle to ramp in at (in degrees)
   ramping_data.RampAngle = 25.0
   -- if we are contraining ramp by angle, max distance to travel before
   -- 'zig zaging' if zig zaging
      ramping_data.RampMaxAngleDist = 15
      -- if true we restrict our ramping to lead in section of toolpath
      ramping_data.RampOnLeadIn = false
```

LeadInOutData

This object is used to hold parameters relating to lead in / out for toolpaths which support leads.

Constructor

LeadInData() - Constructor

Create a new object ready to have its parameters set

```
e.g
    local lead_data = LeadInOutData()
```

Properties

.CircularLeadRadius

R/W - double - radius for circular leads

.DoLeadIn

R/W - bool - If true we do lead in else all other fields for lead ins are ignored

.DoLeadOut

R/W - bool - If true we do lead out else all other fields for lead outs are ignored

.LeadType

R/W - integer - Type of lead to create. Valid values are ...

LeadInOutData.LINEAR_LEAD
LeadInOutData.CIRCULAR_LEAD

.LeadLength

R/W - double - Length for lead in / out

.LinearLeadAngle

R/W - double - Angle in degrees for linear leads from a perpendicular lead

.OvercutDistance

R/W - double - If greater than 0.0 tool will continue past entry point by this distance at end of profile

Example Code

```
-- Create object used to control lead in/out
   local lead_in_out_data = LeadInOutData()
      -- if true we create lead ins on profiles (not for profile on)
      lead_in_out_data.DoLeadIn = false
      -- if true we create lead outs on profiles (not for profile on)
      lead_in_out_data.DoLeadOut = false
      -- type of leads to create LeadInOutData.LINEAR_LEAD or LeadInOutData.CIRCULAR_LEAD
      lead_in_out_data.LeadType = LeadInOutData.CIRCULAR_LEAD
      -- length of lead to create
      lead_in_out_data.LeadLength = 10.0
      -- Angle for linear leads
      lead_in_out_data.LinearLeadAngle = 45
      -- Radius for circular arc leads
      lead_in_out_data.CirularLeadRadius = 5.0
      -- distance to 'overcut' (travel past start point) when profiling
      lead_in_out_data.OvercutDistance = 0.0
```

PocketParameterData

This object is used to hold the settings for a pocketing toolpath and also the clearance tool section of a flat bottomed v carving toolpath.

Constructor

ProfileParameterData() - Constructor

Create a new object ready to have its parameters set

```
e.g
    local pocket_data = PocketParameterData()
```

Properties

.Allowance

R/W - double - allowance to leave on pocket sides when calculating toolpath

.AllowanceFormula

R/W - string -Formula for allowance to leave on pocket sides when calculating toolpath

.CutDepth

R/W - double -Final cutting depth below start depth

.CutDepthFormula

R/W - string -Formula for cutting depth below start depth

.CutDirection

R/W - integer - Cutting direction for toolpath. Valid values are ...

ProfileParameterData.CLIMB_DIRECTION
ProfileParameterData.CONVENTIONAL_DIRECTION

Note: Direction constants use ProfileParameterData. NOT PocketParameterData.

.DoRamping

R/W - bool - If true ramp entry to pockets (always zig-zag)

.DoRasterClearance

R/W - bool - if true doing raster area clearance, else offset

.Name

R/W - string - The name for the toolpath

.ProfilePassType

R/W - integer - type of profile pass to perform . Valid values are ...

PocketParameterData.PROFILE_NONE
PocketParameterData.PROFILE_FIRST
PocketParameterData.PROFILE_LAST

.ProjectToolpath

R/W - bool - If true and used with Aspire, toolpath is projected onto model surface after calculation

.RampDistance

R/W - double - distance to ramp over if doing ramping

.RasterAllowance
R/W - double - allowance to leave on pocket edge between rasters

.RasterAngle
R/W - double - Angle in degrees to create raster toolpaths at

.StartDepth
R/W - double - Start depth for toolpath below material surface

.StartDepthFormula
R/W - string -Formula for start depth for toolpath below material surface

.UseAreaClearTool
R/W - bool - If true use a larger tool for area clearance

Example Code

```
-- Create  object used to pass pocketing options - used for area clearance only
   local pocket_data = PocketParameterData()
      -- start depth for toolpath
      pocket_data.StartDepth = start_depth

      -- cut depth for toolpath this is depth below start depth
      pocket_data.CutDepth = flat_depth

      -- direction of cut for offet clearance - ProfileParameterData.CLIMB_DIRECTION
      -- or ProfileParameterData.CONVENTIONAL_DIRECTION - NOTE: enum from ProfileParameterData
      pocket_data.CutDirection = ProfileParameterData.CLIMB_DIRECTION

      -- if true use raster clearance strategy , else use offset area clearance
      pocket_data.DoRasterClearance = true

      -- angle for raster if using raster clearance
      pocket_data.RasterAngle = 0

      -- type of profile pass to perform  PocketParameterData.PROFILE_NONE ,
      -- PocketParameterData.PROFILE_FIRST orPocketParameterData.PROFILE_LAST
      pocket_data.ProfilePassType = PocketParameterData.PROFILE_LAST
```

DrillParameterData

This object is used to hold the settings for a drilling toolpath .

Constructor

DrillParameterData() - Constructor

Create a new object ready to have its parameters set

```
e.g
    local drill_data = DrillParameterData()
```

Properties

.CutDepth

R/W - double -Final cutting depth below start depth

.CutDepthFormula

R/W - string -Formula for cut depth below start depth

.DoPeckDrill

R/W - bool - if true we will do peck drilling

.Name

R/W - string - The name for the toolpath

.PeckRetractGap

R/W - double - distance above surface to retract to when peck drilling

.ProjectToolpath

R/W - bool - If true and used with Aspire, toolpath is projected onto model surface after calculation

.StartDepth

R/W - double - Start depth for toolpath below material surface

.StartDepthFormula

R/W - string -Formula for start depth for toolpath below material surface

Example Code

```
-- Create  object used to pass drilling options
   local drill_data = DrillParameterData()
      -- start depth for toolpath
      drill_data.StartDepth = start_depth

      -- cut depth for toolpath this is depth below start depth
      drill_data.CutDepth = cut_depth

      -- if true perform peck drilling
      drill_data.DoPeckDrill = retract_gap > 0.0

      -- distance to retract above surface when peck drilling
      drill_data.PeckRetractGap = retract_gap

      -- if true in Aspire, project toolpath onto composite model
      drill_data.ProjectToolpath = false
```

VCarveParameterData

This object is used to hold the settings for a vcarving toolpath .

Constructor

VCarveParameterData() - Constructor

Create a new object ready to have its parameters set

```
e.g
    local vcarve_data = VCarveParameterData()
```

Properties

.FlatDepth

R/W - double -If DoFlatBottom is true the toolpath will not cut any deeper than this depth below StartDepth.

.FlatDepthFormula

R/W - string -Formula for flat depth - see .FlatDepth for when used

.Name

R/W - string - The name for the toolpath

.ProjectToolpath

R/W - bool - If true and used with Aspire, toolpath is projected onto model surface after calculation

.StartDepth

R/W - double - Start depth for toolpath below material surface

.StartDepthFormula

R/W - string -Formula for start depth for toolpath below material surface

.DoFlatBottom

R/W - bool - If true the toolpath depth will be limited to the value specified in .FlatDepth and the bottom of the carving will be flat.

.UseAreaClearTool

R/W - bool - If true, the area clearance tool will be used to clear the flat areas of the toolpath

Example Code

```
-- Create  object used to pass vcarving options
   local vcarve_data = VCarveParameterData()
      -- start depth for toolpath
      vcarve_data.StartDepth = start_depth

      -- flag indicating if we are creating a flat bottomed toolpath
      vcarve_data.DoFlatBottom = flat_depth > 0.0

      -- cut depth for toolpath this is depth below start depth
      vcarve_data.FlatDepth = flat_depth

      -- if true in Aspire, project toolpath onto composite model
      vcarve_data.ProjectToolpath = false
```

FlutingParameterData

This object is used to hold the settings for a fluting toolpath.

Constructor

FlutingParameterData() - Constructor

Create a new object ready to have its parameters set

```
e.g
    local fluting_data = FlutingParameterData()
```

Properties

.CutDepth

R/W - double -Final cutting depth below start depth

.CutDepthFormula

R/W - string -Formula for final cutting depth below start depth

.FluteType

R/W - integer - Type of flute to create. Valid values are ...

FlutingParameterData.FULL_LENGTH
FlutingParameterData.RAMP_START
FlutingParameterData.RAMP_START_END

.Name

R/W - string - The name for the toolpath

.ProjectToolpath

R/W - bool - If true and used with Aspire, toolpath is projected onto model surface after calculation

.RampLength

R/W - double -Length of ramp. This is only used if *.UseRampRatio* is false.

.RampRatio

R/W - double -Length of ramp as ratio of flute length (range 0-1.0). For ramp at start end ratio is of half length. This is only used if *.UseRampRatio* is true.

.RampType

R/W - integer - Type of ramping to perform. Valid values are ...

FlutingParameterData.RAMP_LINEAR
FlutingParameterData.RAMP_SMOOTH

.StartDepth

R/W - double - Start depth for toolpath below material surface

.StartDepthFormula

R/W - string -Formula for start depth for toolpath below material surface

.UseRampRatio

R/W - bool -If true, use *.RampRatio* field for controling ramp length, else absolute value from *.RampLength*

.UseSelectionOrder

R/W - bool - If true use selection order of vectors as machining order

PrismCarveParameterData

This object is used to hold the settings for a prism carving toolpath.

Constructor

PrismCarveParameterData() - Constructor

Create a new object ready to have its parameters set

```
e.g
    local prism_data = PrismCarveParameterData()
```

Properties

.CutDepth

R/W - double -Final cutting depth below start depth

.CutDepthFormula

R/W - string -Formula for cut depth below start depth

.CutDirection

R/W - integer - Cutting direction for toolpath. Valid values are ...
ProfileParameterData.CLIMB_DIRECTION
ProfileParameterData.CONVENTIONAL_DIRECTION

.Name

R/W - string - The name for the toolpath

.StartDepth

R/W - double - Start depth for toolpath below material surface

.StartDepthFormula

R/W - string -Formula for start depth for toolpath below material surface
Note: Direction constants use ProfileParameterData. NOT PrismCarveParameterData.

Methods

:CalculateMinimumBevelDepth(Tool tool, bool show_warnings)

Return the minimum cut depth required with the passed tool for the currently selected vectors.
Tool -Tool -Tool to use for the toolpath
show_warnings -bool -if true display warning to user if tool not suitable of vectors not selected

Example Code

```
-- Create  object used to pass toolpath options
   local prism_data = PrismCarveParameterData()
      -- start depth for toolpath
      prism_data.StartDepth = start_depth

      -- cut depth for toolpath this is depth below start depth
      prism_data.CutDepth = cut_depth

      -- direction of cut for offet clearance - ProfileParameterData.CLIMB_DIRECTION or
      -- ProfileParameterData.CONVENTIONAL_DIRECTION - NOTE: enum from ProfileParameterData
      prism_data.CutDirection = ProfileParameterData.CLIMB_DIRECTION


-- calculate the minimum cut depth to fully form the bevel on the current selection
-- with the current tool
   local min_bevel_depth = prism_data:CalculateMinimumBevelDepth(tool, true)
```

```
if min_bevel_depth > cut_depth then
   DisplayMessageBox("A prism will not be fully formed with a depth of " .. cut_depth ..
                     "\r\n" ..
                     "A depth of " .. min_bevel_depth ..
                     " is required to fully form the prism"
                     )
```

RoughingParameterData – Aspire Only

This object is used to hold settings for a 3D roughing toolpath.

Constructors

RoughingParameterData() – Constructor

Create a new object ready to have its parameters set

Properties

.Name

R/W – string – The name for the toolpath

.StartDepth

R/W – double – Start depth for the toolpath below the material surface

.StartDepthFormula

R/W – string – Formula for start depth for toolpath below material surface

.CutDepth

R/W – double – Final cutting depth below the start depth

.CutDepthFormula

R/W – string – Formula for final cutting depth below start depth

.Allowance

R/W – double – allowance to leave for a finishing toolpath to clear

.DoZLevelRoughing

R/W – bool – If true do Z level roughing

.ZLevelStrategy

R/W – integer – type of Z level roughing to perform. Valid values are:

RoughingParameterData.RASTER_X
RoughingParameterData.RASTER_Y
RoughingParameterData.OFFSET

.ZLevelProfile

R/W – integer – type of roughing pass to perform. Valid values are:

RoughingParameterData.LAST
RoughingParameterData.FIRST
RoughingParameterData.NONE

.RasterAngle

R/W – double – The angle to raster at

.MachiningAllowance

R/W – double – Allowance to leave on when roughing

ExternalToolpath

This object is used to create a toolpath within the program with the user supplying ALL data for the toolpath including the 3D moves.

WARNING

This is an extremely powerful capability as you can control the complete movement of the tool. However it is extremely important to realize how dangerous this can be, the program makes no checks on the data supplied via this method, so the user must make sure that the toolpath is doing what they expect. Scripts that use this capability should be rigorously tested with different jobs with both the Z origin on the material top and bottom, as well as all variations of origin position and in both mm and inches.

Note: Once the toolpath is created, use the AddExternalToolpath method of ToolpathManager to add it to the program.

Constructor

ExternalToolpath(

string name,
Tool tool,
ToolpathPosData pos_data,
ExternalToolpathOptions options,
ContourGroup contours
)

Create a new toolpath object

Name - string - Name for the toolpath
Tool -Tool -Tool to use for the toolpath
pos_data-ToolpathPosData -Settings for home position, safe z etc
options -ExternalToolpathOptions -Settings for external toolpath
contours - ContourGroup - the contours describing the actual toolpath

Properties

.Error

R/O- bool - true if an error occurred while creating the toolpath

ExternalToolpathOptions

This object is used to hold the settings for an external toolpath.

Constructor

ExternalToolpathOptions() - Constructor

Create a new object ready to have its parameters set

Properties

.CreatePreview

R/W - bool - if true, create preview in 2d view for this toolpath

.StartDepth

R/W - double - Start depth for toolpath below material surface

Example Code

```
function CreateToolpath(name , vectors, start_depth)

   -- Create tool we will use to machine vectors
   local tool = Tool(
                  "Lua End Mill",
                  Tool.END_MILL       -- BALL_NOSE, END_MILL, VBIT
                  )
   tool.InMM = true
   tool.ToolDia = 3.0
   tool.Stepdown = 2.0
   tool.Stepover = 1.0
   tool.RateUnits = Tool.MM_SEC  -- MM_SEC, MM_MIN, METRES_MIN,
                                 -- INCHES_SEC, INCHES_MIN, FEET_MIN
   tool.FeedRate = 30
   tool.PlungeRate = 10
   tool.SpindleSpeed = 20000
   tool.ToolNumber = 1
   tool.VBitAngle = 90.0     -- used for vbit only
   tool.ClearStepover = 1.0  -- used for vbit only


   -- Create object used to set home position and safez gap above material surface
   local pos_data = ToolpathPosData()
   pos_data:SetHomePosition(0, 0, 5.0)
   pos_data.SafeZGap = 5.0

  -- object used to pass data on toolpath settings
   toolpath_options = ExternalToolpathOptions()
   toolpath_options.StartDepth = start_depth
   toolpath_options.CreatePreview = true

   -- Create our toolpath
   local toolpath = ExternalToolpath(
                                    name,
                                    tool,
                                    pos_data,
                                    toolpath_options,
                                    vectors
                                    )

   if toolpath:Error() then
      DisplayMessageBox("Error creating toolpath")
      return
   end

   local toolpath_manager = ToolpathManager()
   success = toolpath_manager:AddExternalToolpath(toolpath)
   return success

end
```

PostPInfo

This object is used access a post processor within the program. A post processor is acquired from the ToolpathSaver objects *:GetPostWithName(), :GetPostWithFilename() or :GetPostAtIndex()* methods.

Properties

.Name

R/O -string – the name of the post processor. This is the name displayed in the dropdown list within the program

.FileName

R/O -string – the filen name of the post processor. This does not include the path to the PostP directory.

.Extension

R/O -string – the file extension to use for toolpaths saved using this post processor. Does not include the '.' E.g "txt" NOT ".txt"

.SupportsArcs

R/O -bool – true if the post processor supports arcs. If arcs aren't supported, any arc moves in the toolpath are output as a series of straight lines approximating the arc within tolerance.

.SupportsToolchange

R/O -bool – true if the post processor supports tool changing.

.Wrap_X_Axis

R/O -bool – true if the post processor wraps X moves onto a rotary axis

.Wrap_Y_Axis

R/O -bool – true if the post processor wraps Y moves onto a rotary axis

ToolpathSaver

This object is used to save toolpaths entirely from script. It gives access to the set of PostProcessors available to the program and holds a list of toolpaths to be saved.

IMPORTANT: It is the script writer's responsibility to ensure that all the toolpaths added to the ToolpathSaver object are suitable for saving to one file using the selected post-processor. The ToolpathSaver gives low-level access to the toolpath output from the program and will write what ever toolpaths you specify to a single file irrespective of the tool geometry or the post processors support for tool changing. It is the responsibility of the script writer to ensure that a tool changing post is selected if multiple toolpaths using different tools are output. The *:IsCompatible* method of the *Tool* object can be used to check if tools used for different toolpaths are suitable for outputting to a single file if a tool changer is not available.

Example Code

```
--[[  ------------------------ DoAllToolpathsUseSameTool ---------------------------------
|
| Check if all the toolpaths in the passed list use the same tool. If they do, we can
| save all the toolpaths to the same file, regardles of whether the post supports
| toolchanging. If the toolpaths use different tools and the post does not support tool
| changing, the toolpaths will need to be saved individually
|
]]
function DoAllToolpathsUseSameTool(toolpath_ids, toolpath_manager)

   local tool = nil

        for i,toolpath_id in ipairs(toolpath_ids) do
               local pos = toolpath_manager:Find(toolpath_id)
               if pos == nil then
                  MessageBox("Failed to find toolpath " .. i)
                  return false
               end
               local toolpath = toolpath_manager:GetAt(pos)
               -- do we have an existing tool to check against?
               if tool == nil then
                  -- no this is first toolpath - save a copy of tool to check against
                  tool = toolpath.Tool
               else
                  -- check if this tool is compatible with previous tools
                  if not tool:IsCompatibleTool(toolpath.Tool) then
                     return false  -- we have a mixture of tools
                  end
               end
        end

   return true
end
```

Constructor

ToolpathSaver() - Constructor

Create a new object which can be used to save toolpaths

Properties

.DefaultPost

R/O -PostPInfo – the currently selected post processor within the program

.NumberOfToolpaths
R/O -integer – the number of toolpaths currently held by this object ready for saving. Toolpaths are added to the object using the :AddToolpaths() method.

Methods

:GetNumPosts()
Return number of post processers the program knows about. This is usually used to iterate through all the posts using :GetPostAtIndex

:AddToolpath(Toolpath toolpath)
Add the passed toolpath to the list of toolpaths to save managed by this object. Retunrs true if toolpath added to list ok.

toolpath -Toolpath –toolpath to be saved

:ClearToolpathList()
Clear the list of toolpaths to save managed by this object.

:GetNumPosts()
Return number of post processers the program knows about. This is usually used to iterate through all the posts using :GetPostAtIndex

:GetPostAtIndex(integer post_index)
Return the post processor (PostPInfo) for the specified index.

post_index - integer –index of post to return. The index go from 0 to num_posts - 1

:GetPostWithName(string post_name)
Return the post processor (PostPInfo) with the specified name. This is the name displayed in the drop down list in the program.

post_name - string –name of post to return.

:GetPostWithFilename(string post_file_name)
Return the post processor (PostPInfo) with the specified file name. This is the name of the .pp file on disk without the path.

post_file_name - string –filename of post to return.

:SaveToolpaths(PostPInfo postp, string filename, bool output_to_mc)
Save all the toolpaths which have been added with :AddToolpath using passed post processor to passed filename, Returns true if file saved OK, false if an error occurs.

postp -PostPInfo – Post processor to use to save file
filename -string – Full pathname to file to be saved. This MUST include the extension you want for the file as well.
output_to_mc -bool –if true **AND** the post processor supports direct output to the machine the file will be sent to the machine after saving.

Example Code

```
local toolpath_manager = ToolpathManager()
if toolpath_manager.IsEmpty then
   MessageBox("There are no toolpaths to save")
   return false
end
local output_path = "c:\\temp\\test_toolpath_output.txt"
-- get post processor to use for saving ...
local toolpath_saver = ToolpathSaver();
local post = toolpath_saver:GetPostWithName("Text Output Arcs (mm) (*.txt)")
if post == nil then
   MessageBox("Failed to load Post Processor with name " .. post_name)
   return false
end

-- save toolpath to file ...
local toolpath_names = ""  -- we keep a list of toolpaths we save
if selected_only then
   local toolpath = toolpath_manager:GetSelectedToolpath()
   if toolpath == nil then
      MessageBox("There is no currently selected toolpath")
      return false
   end
   toolpath_saver:AddToolpath(toolpath)
   toolpath_names = toolpath.Name .."\n"
else
    -- we are saving all toolpaths ...
   local pos = toolpath_manager:GetHeadPosition()
   while pos ~= nil do
      local toolpath
      toolpath, pos = toolpath_manager:GetNext(pos)
      toolpath_saver:AddToolpath(toolpath)
      toolpath_names = toolpath_names .. toolpath.Name .."\n"
   end
end

local success = toolpath_saver:SaveToolpaths(post, output_path, false)
if not success then
   MessageBox("Failed to save toolpath(s)\n\n" ..  toolpath_names ..
              "\nto file\n" .. output_path )
   return false
end

MessageBox("Saved toolpath(s)\n\n" .. toolpath_names .. "\nto file\n" ..
           output_path .. "\nUsing post\n" .. post.Name)
```

ToolpathStats

This object is used to hold information about a calculated toolpath. This object is returned by the Toolpath:Statistics() method

Properties

.IsValid

R/O -bool – true if the statistics in this object are valid. If toolpaht has not yeet been calculated this would return fals

.FeedLength

R/O -double –total length of feed rate moves in the toolpath in either mm or inches depending on units of job.

.PlungeLength
R/O -double –total length of plunge moves in the toolpath in either mm or inches depending on units of job.

.RetractLength
R/O -double –total length of retract moves in the toolpath in either mm or inches depending on units of job.

.RapidLength
R/O -double –total length of rapid moves in the toolpath in either mm or inches depending on units of job.

.MinimumZ
R/O -double – the z value of the deepest move in the toolpath in either mm or inches depending on units of job.

ToolpathSaveInfo

This object is used only within the GetNameForToolpathFile.lua configuration handler. This file can be used to customise the behavior of the file save mechanism for toolpath. Just before the program displays the File Save dialog when saving a toolpaht, it will check if a file called GetNameForToolpathFile.lua was present when the program started. The path to the file varies depending on the operating system (the equivalent location of “My Documents”) and the program name and version. For Aspire V4.0 on Windows 7, this path is …

C:\Users\USER_NAME\Documents\Vectric Files\Config\Aspire 4.0\ GetNameForToolpathFile.lua

For VCarve Pro V7.0 it would be ..

C:\Users\USER_NAME\Documents\Vectric Files\Config\VCarve Pro 7.0\ GetNameForToolpathFile.lua

An example of a Lua file which just displays some information abou the file being saved is shown later.

Properties

.BaseName

R/W -string – base name for toolpath to save without extension. Before saving the default extension from *.PostP* will be added

.PathName

R/W -string – The full pathname to the file to be saved. On calling the lua script this field will be empty. If the Lua script sets this value it will be used as the default path for the file open dialog.

.PostP

R/O -PostPInfo – The post processor the user has chosen to save the toolpath(s) with.

.ShowFileDialog

R/W -bool – If the Lua script sets this to true AND *.PathName* is not empty, The File Save As dialog will not be displayed by the program and the toolpath will be written to the file specified in *.PathName* . If the file in *.PathName* does NOT have an extension, the default extension from the PostP will be added automatically.

.ToolpathList

R/O -ToolpathList – The list of toolpaths being saved.

Example Code – GetNameForToolpathFile.lua

```
-- VECTRIC LUA SCRIPT

require "strict"


--[[  --- GetNameForToolpathFile -------------------------------------------------
|
|  Entry point for script
|
]]
function GetNameForToolpathFile(save_info)

   local tc_support;
   if save_info.PostP.SupportsToolchange then
       tc_support = "Toolchanges Are Supported"
   else
       tc_support = "Toolchanges Not Supported"
   end

   MessageBox(
              "GetNameForToolpathFile:" ..
              "\nNumber of Toolpaths = " .. save_info.ToolpathList.Count ..
              "\nDefault Name = " .. save_info.BaseName ..
              "\nPost Processor Name = " .. save_info.PostP.Name ..
              "\nPost Processor Extension = " .. save_info.PostP.Extension ..
              "\nPost Processor " .. tc_support
             )


   -- save_info.BaseName = "Lua Toolpath Name"
   -- save_info.PathName = "d:\\temp\\Lua Toolpath Name"
   -- save_info.ShowFileDialog = false
   return true
end


--[[  -------------- main ---------------------------------------------------
|
|  Entry point for script
|
]]

function main(script_path)


    return true

end
```

ToolpathList

This object holds a list of toolpaths for another object.

Properties

.Count

R/O -integer – return the number of toolpaths in the list

.IsEmpty

R/O -bool – return true if the toolpath list is empty

Methods

:GetAt(POSITION pos)

Returns the toolpath at the current position

pos – *POSITION* – current position in list

:GetHeadPosition()

Returns a POSITION variable to allow access to the head of the list of toolpaths

:GetNext(POSITION pos)

Returns the toolpath at the current position AND a new value for position pointing to the next item in the list (or nil if at end of list)

pos – *POSITION* – current position in list

Example - note that GetNext(pos) is returning two values ...

```
local pos = save_info.ToolpathList:GetHeadPosition()
local toolpath
while pos ~= nil do
    toolpath, pos = save_info.ToolpathList:GetNext(pos)
    DO SOMETHING WITH TOOLPATH ....
end
```

:GetPrev(POSITION pos)

Returns the toolpath at the current position AND a new value for position, pointing to the previous item in the list (or nil if at start of list)

pos – *POSITION* – current position in list

:GetTailPosition()

Returns a POSITION variable to allow access to the tail (end) of the list toolpaths

User Interface

HTML_Dialog

This object is used display a dialog to the user. The dialog is defined using a HTML web page and can collect information from the user as well as display it. The system works by associating variables within the script with HTML elements within the dialog. When the dialog is closed, the current values of edit boxes, check boxes etc can be examined. As well as passively collecting all the data once the dialog is closed, 'callbacks' into the lua code can be made when the user clicks on a button, selects a file or tool or chooses from a list.

Button Callbacks

If a button on the dialog has class="LuaButton", when the user clicks on the button, HTML_Dialog will search the calling script for a method with the signature

function OnLuaButton_**BUTTON_ID**(dialog)

where **BUTTON_ID** is the id of the button pressed and 'dialog' is the HTML_Dialog object the button was pressed on.

e.g HTML

```
<BUTTON CLASS="LuaButton" ID="TestButton1" type=button>Press Me!</BUTTON></td>
```

e.g Lua

```
function OnLuaButton_TestButton1(dialog)

   MessageBox("User pressed TestButton1")
   local cur_num_machines = dialog:GetIntegerField("MachineCount")

   return true
end
```

NOTE: The callback functions MUST have "return true" at the end or you will get script errors!

If a specific handler is not found for the button an attempt will be made to call a 'generic' handler with the signature

```
function OnLuaButton_XXXX(element_id, dialog)

    MessageBox(Button with id = " .. element .. " was pressed!")
    local cur_num_machines = dialog:GetIntegerField("MachineCount")
    return true

end
```

e.g HTML

```
<BUTTON CLASS="LuaButton" ID="TestButton1" type=button>Press Me!</BUTTON></td>
```

e.g Lua

```
function OnLuaButton_TestButton1(dialog)
   MessageBox("User pressed TestButton1")
   local cur_num_machines = dialog:GetIntegerField("MachineCount")
   return true
end
```

DirectoryPicker Callbacks

When a DirectoryPicker is used to pick a directory, HTML_Dialog will will search the calling script for a method with the signature

function OnDirectoryPicker_**PICKER_ID**(dialog)

where **PICKER_ID** is the id of the directory picker button pressed and 'dialog' is the HTML_Dialog object the button was pressed on.

FilePicker Callbacks

When a FilePicker is used to pick a file, HTML_Dialog will will search the calling script for a method with the signature

function OnFilePicker_**PICKER_ID**(dialog)

where **PICKER_ID** is the id of the file picker button pressed and 'dialog' is the HTML_Dialog object the button was pressed on.

ToolPicker Callbacks

When a ToolPicker is used to pick a tool, HTML_Dialog will will search the calling script for a method with the signature

function OnToolPicker_**PICKER_ID**(dialog)

where **PICKER_ID** is the id of the tool picker button pressed and 'dialog' is the HTML_Dialog object the button was pressed on.

Selector / DropDownList Callbacks

When a Selector / DropDownList is used to pick a value, HTML_Dialog will will search the calling script for a method with the signature

function OnLuaSelector_**ELEMENT_ID**(dialog)

where **ELEMENT_ID** is the id of the selector / dropdown list button pressed and 'dialog' is the HTML_Dialog object the list was seelcted on.

If a specific handler is not found for the list an attempt will be made to call a 'generic' handler with the signature

```
function OnLuaSelector_XXXX(element_id, dialog)

    local selected_value = dialog:GetDropDownListValue(element_id)

    MessageBox("Value " .. selected_value ..
               " was selected from List with id = " .. element_id)

    return true

end
```

Constructor

HTML_Dialog(bool local_html, string html, integer width, integer height, string dialog_name) - Constructor

Create a new dialog object , specifying the source for the HTML and the dialog size and title.

local_html -bool –true if the next argument (html) contains all the html code for the dialog in the string. If false the html string is the path to the html file.

html -string – local_html was true, contains all the html code for the dialog in the string. If local_html was false the html string is the path to the html file. The path must be fully qualified and is usually built up from the scripts location as shown in the example below.

width -integer – width of the dialog in pixels

height -integer – height of the dialog in pixels

dialog_name -string – name for dialog displayed in title bar

```
e.g
    function main(script_path)
      local html_path = "file:" .. script_path .. about.htm"
      local dialog = HTML_Dialog(false, html_path, 800, 600, "About Gadgets")
      dialog:ShowDialog()
    end
```

Properties

.WindowWidth

R/O-integer –Width of dialog window in pixels when user closed the dialog.

.WindowHeight

R/O-integer –Height of dialog window in pixels when user closed the dialog.

Methods

:AddCheckBox(string element_ id, bool value)

Sets a check box element (input box displaying a check box) in the HTML with id=”element_id” to passed value.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – bool – true to ‘check box, false to leave unchecked

e.g – HTML
<INPUT class="LuaButton" type="checkbox" checked ID="CreatePreviewCheck" value='active'>Create a preview

e.g – Lua
dialog:AddCheckBox("CreatePreviewCheck”, true)

:AddDirectoryPicker(string element_ id, string buddy_name, bool buddy_is_edit)

Sets a button on the dialog to act as a ‘directory’ chooser. When the user presses the button a dialog is displayed to allow them to select a directory / folder . When the selection dialog is closed the ‘buddy’ field (which can either be an edit box or a label) is updated with the path to the directory/folder chosen.

NOTE: The <Button > element MUST have a class = “DirectoryPicker” set

element_id -string –Id of the button element in HTML to use (id=”element_id” in HTML). The button MUST have class = “DirectoryPicker”

buddy_name – string – id of associated label or edit field which will hold the value the user picks

buddy_is_edit – bool – true if the ‘buddy’ field is an edit field (added with AddTextField), false if ‘buddy’ is a label field (added with AddLabelField).

e.g HTML for picker associated with an edit field …

```
 <input name="textfield" type="text" size="40" maxlength="128" ID="DirNameEdit">
<BUTTON CLASS="DirectoryPicker" ID="ChooseDirButton1" type=button>Choose...</BUTTON>
```

e.g Lua for picker associated with an editable text field …

```
dialog:AddTextField("DirNameEdit", "c:\\temp");
dialog:AddDirectoryPicker("ChooseDirButton1", "DirNameEdit", true);
```

e.g HTML for picker associated with a label field …

```
 <span id="DirNameLabel"></span>
<BUTTON CLASS="DirectoryPicker" ID="ChooseDirButton2" type=button>Choose...</BUTTON>
```

e.g Lua for picker associated with a label field …

```
dialog:AddLabelField("DirNameLabel", "c:\\");
dialog:AddDirectoryPicker("ChooseDirButton2", "DirNameLabel", false);
```

:AddDoubleField(string element_ id, double value)

Sets a text box element (text input box displaying a double value) in the HTML with id=”element_id” to passed value.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – double – value to set for element in HTML

e.g HTML
<input name="textfield" type="text" size="8" maxlength="8" ID="MachineCost">

e.g Lua
dialog:AddDoubleField("MachineCost", 6495.12)

:AddDropDownList(string element_ id, string default_value)

Sets a drop downlistin the HTML (type=”radio”) with id=”element_id” to passed value.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – string – initial value to set for element in HTML

e.g HTML

```
<select id="ProductList" >
        <option value="1">first</option>
        <option value="2" selected>second</option>
        <option value="3" >third</option>
</select>
```

e.g Lua

```
dialog:AddDropDownList("ProductList ","Aspire")
dialog:AddDropDownListValue("ProductList ", "Cut2D")
dialog:AddDropDownListValue("ProductList ", "VCarve Pro")
dialog:AddDropDownListValue("ProductList ", "Aspire")
dialog:AddDropDownListValue("ProductList ", "Cut3D")
dialog:AddDropDownListValue("ProductList ", "PhotoVCarve")
```

:AddDropDownListValue(string element_ id, string value)

Add a value to a dropdown list n the HTML (type=”radio”) with id=”element_id” to passed value.

element_id -string –Id of the select list in HTML to add value to (id=”element_id” in HTML)
value – string –value to add to list of values

:AddFilePicker(string element_ id, string buddy_name, bool buddy_is_edit)

Sets a button on the dialog to act as a ‘file’ chooser. When the user presses the button a dialog is displayed to allow them to select a file. When the selection dialog is closed the ‘buddy’ field (which can either be an edit box or a label) is updated with the path to file chosen.

NOTE: The <Button > element MUST have a class = “FilePicker” set

element_id -string –Id of the button element in HTML to use (id=”element_id” in HTML). The button MUST have class = “FilePicker”

buddy_name – string – id of associated label or edit field which will hold the value the user picks

buddy_is_edit – bool – true if the ‘buddy’ field is an edit field (added with AddTextField), false if ‘buddy’ is a label field (added with AddLabelField).

e.g HTML for picker associated with an edit field …

```
 <input name="textfield" type="text" size="40" maxlength="128" ID="FileNameEdit">
<BUTTON CLASS="FilePicker" ID="ChooseFileButton1" type=button>Choose...</BUTTON>
```

e.g Lua for picker associated with an editable text field …

```
dialog:AddTextField("FileNameEdit", "c:\\temp");
dialog:AddDirectoryPicker("ChooseFileButton1", "FileNameEdit", true);
```

e.g HTML for picker associated with a label field …

```
 <span id="FileNameLabel"></span>
<BUTTON CLASS="FilePicker" ID="ChooseFileButton2" type=button>Choose...</BUTTON>
```

e.g Lua for picker associated with a label field …

```
dialog:AddLabelField("FileNameLabel", "c:\\");
dialog:AddDirectoryPicker("ChooseFileButton2", "FileNameLabel", false);
```

:AddIntegerField(string element_ id, integer value)

Sets a text box element (text input box displaying an integer value) in the HTML with id=”element_id” to passed value.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – integer – value to set for element in HTML

e.g HTML

```
<input name="textfield" type="text" size="4" maxlength="8" ID="MachineCount">
```

e.g Lua

```
dialog:AddIntegerField("MachineCount", 1)
```

:AddLabelField(string element_ id,string value)

Sets a text element in the HTML with id=”element_id” to passed value

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – string – Value to set for element in HTML

e.g – HTML

```
<span id="WelcomeText">Welcome to our first test HTML dialog <br>created via Lua </spe.g – Lua
```

e.g Lua

```
dialog:AddLabelField("WelcomeText", "Welcome to my <b>Lua</b> HTML dialog")
```

:AddRadioGroup(string element_ id, int default_index)

Sets a radio group in the HTML (type=”radio”) with id=”element_id” to passed value. Note that the radio index is one based, so the first item in the group is 1.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
default_index – integer – index of radio button to check in HTML – 1 based

e.g – HTML

```
<br> Type of Machine<br><br>
 <input class="LuaButton" type="radio" name="MachineType" ID="MachineRadio1"> Engraver
 <input class="LuaButton" type="radio" name="MachineType" ID="MachineRadio2"> Router
 <input class="LuaButton" type="radio" name="MachineType" ID="MachineRadio3"> CNC Mill
  <br>
```

e.g – Lua

```
dialog:AddRadioGroup("MachineType", 2)
```

:AddTextField(string element_ id, string value)

Sets a text element (text input box) in the HTML with id=”element_id” to passed value.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – string – value to set for element in HTML

e.g HTML

```
<input name="textfield" type="text" size="20" maxlength="8" ID="MachineName">
```

e.g Lua

```
dialog:AddTextField("MachineName", "Rosy")
```

:AddToolEditor(string element_ id, string buddy_name)

Sets a button on the dialog to act as a ‘tool’ editor. When the user presses the button, the tool editor dialog is displayed to allow them to edit the tool chosen by the associated ‘buddy’ ToolPicker. The ‘buddy’ field which MUST be a ToolPicker added with AddToolPickers has the path to the tool chosen by the tool picker.

element_id -string –Id of the button element in HTML to use (id=”element_id” in HTML). The button MUST have class = “ToolPicker”

buddy_name – string – id of associated tool picker which will hold the path to the tool the user picks.

:AddToolPicker(string element_ id, string buddy_name, string default_tool_name)

Sets a button on the dialog to act as a 'tool' chooser. When the user presses the button the tool database dialog is displayed to allow them to select a tool. When the selection dialog is closed the 'buddy' field which MUST be a label added with AddLabelField is updated with the path to the tool chosen.

element_id -string –Id of the button element in HTML to use (id="element_id" in HTML). The button MUST have class = "ToolPicker"

buddy_name – string – id of associated label field which will hold the path to the tool the user picks.

default_tool_name – string – default tool name and path in the tool database– usually saved from a previous selection by the tool picker.

e.g HTML for picker associated with a label field …

```
<span id="ToolNameLabel>Tool Name</span>
<input id="ToolChooseButton" class="ToolPicker" name="ToolChooseButton" type="button" value="Tool...">
```

e.g Lua for picker associated with a label field …

```
 dialog:AddLabelField("ToolNameLabel ", "");
dialog:AddToolPicker("ToolChooseButton", "ToolNameLabel", "")
dialog:AddToolPickerValidToolType("ToolChooseButton",    Tool.END_MILL)
dialog:AddToolPickerValidToolType("ToolChooseButton",    BALL_NOSE)
```

:AddToolPickerValidToolType(string element_ id, integer tool_type)

Used to add valid tool types for the tool databse. See below for a list of valid values

element_id -string –Id of the toolpicker element in HTML to add valid type to (id="element_id" in HTML). The button MUST have class = "ToolPicker"
tool_type – integer – Integer indicating a type of tool which will be valid for selection – choose from ..

- *Tool*.BALL_NOSE
- *Tool*.END_MILL
- *Tool*.RADIUSED_END_MILL
- *Tool*.VBIT
- *Tool*.ENGRAVING
- *Tool*.RADIUSED_ENGRAVING
- *Tool*.THROUGH_DRILL
- *Tool*.FORM_TOOL
- *Tool*.DIAMOND_DRAG
- *Tool*.RADIUSED_FLAT_ENGRAVING

:ClearToolPickerValidToolType(string element_ id)

Used to clear list of valid tool types for the tool databse.

element_id -string –Id of the toolpicker element in HTML to clear (id=”element_id” in HTML). The button MUST have class = “ToolPicker”

:GetCheckBox(string id)

Returns a bool containing current value of a check box in the HTML with id=”element_id” . The field must have been created with a previous call to AddCheckBox before the dialog was displayed.

element_id -string –Id of the element in HTML to get to value for (id=” element_id” in HTML)

:GetDropDownListValue(string id)

Returns a string containing current value of a dropdown list (select element) in the HTML with id=”element_id” . The field must have been created with a previous call to AddDropDownList before the dialog was displayed.

element_id -string –Id of the element in HTML to get value for (id=” element_id” in HTML)

:GetDoubleField(string id)

Returns a double containing current value of a text input box displaying a double value in the HTML with id=”element_id” . The field must have been created with a previous call to AddDoubleField before the dialog was displayed.

element_id -string –Id of the element in HTML to get value for (id=” element_id” in HTML)

:GetIntegerField(string id)

Returns an integer containing current value of a text input box displaying an integer value in the HTML with id=”element_id” . The field must have been created with a previous call to AddIntegerField before the dialog was displayed.

element_id -string –Id of the element in HTML to get value for (id=” element_id” in HTML)

:GetLabelField(string id)

Returns a string containing current value of an element in the HTML with id=”element_id” . The field must have been created with a previous call to AddLabelField before the dialog was displayed.

element_id -string –Id of the element in HTML to get value for (id=” element_id” in HTML)

:GetRadioIndex(string id)

Returns an integer containing current value of a radio group in the HTML with id=”element_id” . The field must have been created with a previous call to AddRadioGroup before the dialog was displayed.

element_id -string –Id of the element in HTML to get value for (id=” element_id” in HTML)

:GetTextField(string id)

Returns a string containing current value of a text input box element in the HTML with id=”element_id” . The field must have been created with a previous call to AddTextField before the dialog was displayed.

element_id -string –Id of the element in HTML to get value for (id=” element_id” in HTML)

:GetTool (string element_ id)

Returns the Tool selected by the ToolPicker with the passed id.

element_id -string –Id of the button element in HTML to use (id=”element_id” in HTML). The button MUST have class = “ToolPicker”

:SetInnerHtml(string element_ id, string html)

Sets the ‘inner html’ for an element on the dialog.

element_id -string –Id of the element in HTML to set inner html for (id=” element_id” in HTML)
html – string – new ‘inner’ html for the element

:SetOuterHtml(string element_ id, string html)

Sets the ‘outer html’ for an element on the dialog. This can replace the complete element definition

element_id -string –Id of the element in HTML to set outer html for (id=” element_id” in HTML)
html – string – new ‘outer’ html for the element

:UpdateCheckBox(string element_ id, bool value)

Update check box in the HTML with id=”element_id” to passed value. The field must have been created with a previous call to AddCheckBox before the dialog was displayed. This method is used within ‘callbacks’ from the dialog, while the dialog is still visible.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – bool – new value to set for element in HTML

:UpdateDropDownListValue(string element_ id, string value)

Update dropdown list in the HTML with id=”element_id” to passed value. The field must have been created with a previous call to AddDropDownList before the dialog was displayed and the value being set must be a value that was added to the list using AddDropDownListValue. This method is used within ‘callbacks’ from the dialog, while the dialog is still visible.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – string – new value to set for dropdown list

:UpdateDoubleField(string element_ id, double value)

Update text input box displaying a double value in the HTML with id=”element_id” to passed value. The field must have been created with a previous call to AddDoubleField before the dialog was displayed. This method is used within ‘callbacks’ from the dialog, while the dialog is still visible.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – double – new value to set for element in HTML

:UpdateIntegerField(string element_ id, integer value)

Update text input box displaying an integer value in the HTML with id=”element_id” to passed value. The field must have been created with a previous call to AddIntegerField before the dialog was displayed. This method is used within ‘callbacks’ from the dialog, while the dialog is still visible.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – double – new value to set for element in HTML

:UpdateLabelField(string element_ id, string value)

Update a text element in the HTML with id=”element_id” to passed value. The field must have been created with a previous call to AddLabelField before the dialog was displayed. This method is used within ‘callbacks’ from the dialog, while the dialog is still visible.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – string – new value to set for element in HTML

:UpdateRadioIndex(string element_ id, integer index)

Update radio group in the HTML with id=”element_id” to passed value. The field must have been created with a previous call to AddRadioGroup before the dialog was displayed. This method is used within ‘callbacks’ from the dialog, while the dialog is still visible.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – integer – new one based index to set for element in HTML

:UpdateTextField(string element_ id, string value)

Update a text element (text input box) in the HTML with id=”element_id” to passed value. The field must have been created with a previous call to AddTextField before the dialog was displayed. This method is used within ‘callbacks’ from the dialog, while the dialog is still visible.

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
value – string – new value to set for element in HTML

:UpdateToolPickerField(string element_ id, Tool tool)

Update a tool picker element in the HTML with id=”element_id” with values from passed tool. The field must have been created with a previous call to AddToolPickerField before the dialog was displayed. This method is used within ‘callbacks’ from the dialog, while the dialog is still visible if you change a tool value in the handler. (Aspire 4.015/ VCP 4.015 and later)

element_id -string –Id of the element in HTML to set to value (id=”element_id” in HTML)
tool – Tool –tool with new values to set for element in HTML

FileDialog

This object is used to display either a File Open or a File Save dialog box to the user.

Constructor

FileDialog() - Constructor

Create a new object used for displaying a File Open / Save dialog

```
local file_dialog = FileDialog()
```

Methods

:FileOpen(string default_ext,string file_name, string filter)

Display a File Open dialog – this is used to select an Existing file.

default_ext – string – default extension of files to display e.g "txt"
file_name – string –path to default file top open. Can just be path or include filename
filter – string –filter string to display for dialog file type.*The string is build up of pairs of 'file description' followed by 'file extension' separated by '|'. The filter list is terminated by '||'* . E.g "Toolpaths (*.tap) | *.tap||".

:FileSave(string default_ext,string file_name, string filter)

Display a File Save dialog – this is usually used to select a new file for saving to but user can choose to overwite an existing file.

default_ext – string – default extension of files to display e.g "txt"
file_name – string –path to default file to save. Can just be path or include filename
filter – string –filter string to display for dialog file type

Example Code

```
local post_name = dialog:GetDropDownListValue("PostNameSelector");
local toolpath_saver = ToolpathSaver();
local post = toolpath_saver:GetPostWithName(post_name)
if post == nil then
   MessageBox("Failed to load Post Processor with name " .. post_name)
   return false
end

-- get name for output file
local file_dialog = FileDialog()
local dir = dialog:GetTextField("PostOutputFolderEdit")

-- MessageBox("Ext = " .. post.Extension .. "\nFolder = " .. dir)
local file_filter = "Toolpaths (*." .. post.Extension .. ")|" ..
                    "*." .. post.Extension .. "||"

if not file_dialog:FileSave( post.Extension, dir .. "\\*" , file_filter ) then
   MessageBox("No file to save to selected")
   return false
end


local output_path = file_dialog.PathName
```

DirectoryReader

This object is used to build a list of files and directories matching various criteria.

Constructor

DirectoryReader() - Constructor

Create a new object used for building lists of files

```
local dir_reader = DirectoryReader()
```

Methods

:BuildDirectoryList(string dir_path, bool recurse)

Build a list of directories we will search for files

dir_path – string – path to build list of files from
recurse – bool – if true include sub directories as well

:ClearDirs()

Clear list of directories

:ClearFiles()

Clear list of files

:DirAtIndex(integer index)

Return *DirectoryEntry* for directory at passed index

index – integer – index of directory entry to get. Entries are indexed from 1 to *:NumberOfDirs*()

:FileAtIndex(integer index)

Return *FileEntry* for file at passed index

index – integer – index of file entry to get. Entries are indexed from 1 to *:NumberOfFiles*()

:GetFiles(string filter, bool include_files_in_list, bool include_dirs_in_list)

Add files / directories matching passed filter to list held by this object. Note this method can be called multiple times to add different sets of files

filter – string – Filter (eg "*.dxf") to use for finding matching files
include_files_in_list – bool - if true we collect names of matching files
include_dirs_in_list – bool - if true we collect names of matching directories

:SortDirs()

Sort list of directories into alphabetical order

:SortFiles(integer sort_order, bool reverse)

Sort list of files into order

sort_order – integer – Sorting algorithm to use, options are

DirectoryReader.SORT_NONE
DirectoryReader.SORT_ALPHA
DirectoryReader.SORT_WRITE_DATE
DirectoryReader.SORT_SIZE

reverse – bool – if true sort in reverse order

:NumberOfDirs()

Return the number of directories found

:NumberOfFiles()

Return the number of files found

DirectoryEntry

This object is returned from the :GetDirAtIndex() method of DirectoryReader and holds information about a sub directory found while building the list of files and directories.

Properties

.Name

R/O – string – Name of the directory

FileEntry

This object is returned from the :GetFileAtIndex() method of DirectoryReader and holds information about a file found while building the list of files and directories.

Properties

.Name

R/O – string – Name of the file – this is the full pathname to the file

.Size

R/O – integer – Size of the file in bytes

Example Code – from DXF_Batch_Processor.Lua

```
--[[ ----------- DirectoryProcessor ---------------------------------------------
--
|
| Given a root directory and file filter, call the passed function with
| every file which matches the filter
|
| The do_sub_dirs flag indicates if sub-directories should be processed as
| well as the root directory
|
]]
function DirectoryProcessor(job, dir_name, filter, do_sub_dirs, function_ptr)

    local num_files_processed = 0;

    local directory_reader = DirectoryReader()
    local cur_dir_reader = DirectoryReader()

    directory_reader:BuildDirectoryList(dir_name, do_sub_dirs)
    directory_reader:SortDirs()

    local number_of_directories = directory_reader:NumberOfDirs()

    for i = 1, number_of_directories do

        local cur_directory = directory_reader:DirAtIndex(i)

        -- get contents of current directory
        -- dont include sub-dirs, use passed filter
        cur_dir_reader:BuildDirectoryList(cur_directory.Name, false)
        cur_dir_reader:GetFiles(filter, true, false)

        -- call passed method for each file ...
        local num_files_in_dir = cur_dir_reader:NumberOfFiles()
        for j=1, num_files_in_dir  do
           local file_info = cur_dir_reader:FileAtIndex(j)
           if not function_ptr(job, file_info.Name) then
              return -1
           end
           num_files_processed = num_files_processed + 1
        end

        -- empty out our directory object ready for next go
        cur_dir_reader:ClearDirs()
        cur_dir_reader:ClearFiles()

    end

    return num_files_processed
end
```

Registry

This object is used to save and load values from the registry. This allows gadgets to persist settings across different runs of the gadget. All the values are stored under a single 'section' in the registry, so it is important that each gadget uses a unique name for its section. The sections are unique to each product, so the same gadget run under both Aspire and VCarve Pro will store their settings in different areas.

Constructor

Registry(string section_name) - Constructor

Create a new object used for reading and writing values

Section_name –string – Name of section in registry to store all values under

```
local dir_reader = DirectoryReader("MyGadget")
```

Methods

:BoolExists(string parameter_name)

Return true if there is an existing bool parameter with passed name

:DoubleExists(string parameter_name)

Return true if there is an existing double parameter with passed name

:IntExists(string parameter_name)

Return true if there is an existing int parameter with passed name

:GetDouble(string parameter_name, double default_value)

Retrieve a double with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
default_value - double - the value which will be returned if there is no existing value stored

:GetBool(string parameter_name, bool default_value)

Retrieve a bool with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
value - integer - the value which will be returned if there is no existing value stored

:GetInt(string parameter_name, integer default_value)

Retrieve an integer with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
value - integer - the value which will be returned if there is no existing value stored

:GetString(string parameter_name, string value)

Retrieve a string with the passed name , if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
value - string - the value which will be returned if there is no existing value stored

:SetBool(string parameter_name, bool default_value)

Retrieve a Boolean flag (true / false) with the passed name, if no value with passed name returns passed default value

parameter_name - string - the name of the parameter
default_value - bool - the value which will be returned if there is no existing value stored

:SetBool(string parameter_name, bool value)

Store a Boolean flag (true / false) with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - bool - the value which will be stored in the parameter list

:SetDouble(string parameter_name, double value)

Store a double with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - double - the value which will be stored in the parameter list

:SetInt(string parameter_name, integer value)

Store an integer with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - integer - the value which will be stored in the parameter list

:SetString(string parameter_name, string value)

Store a string with the passed name and value

parameter_name - string - the name which will be used to store and retrieve the value
value - string - the value which will be stored in the parameter list

:StringExists(string parameter_name)

Return true if there is an existing string parameter with passed name

FileDialog

This object is used to display either a File Open or a File Save dialog box to the user.

Constructor

FileDialog() - Constructor

Create a new object used for displaying a File Open / Save dialog

```
local file_dialog = FileDialog()
```

Methods

:FileOpen(string default_ext,string file_name, string filter)

Display a File Open dialog – this is used to select an Existing file. Returns true if user selects a file else false if they cancel the dialog.

default_ext – string – The default filename extension. If the user does not include an extension in the Filename edit box, the extension specified here is automatically appended to the filename. If this parameter is empty (“”), no file extension is appended.

file_name – string –path to default file to open. Can just be path or include filename or just be left empty (“”), usualy use “*.ext” at end of string to preselect all files of a particular extension.

filter – string –filter string to display for dialog file type.The string is build up of pairs of ‘file description’ followed by ‘file extension’ separated by ‘|’. The filter list is terminated by ‘||’ . E.g “Toolpaths (*.tap) | *.tap||”.

:FileSave(string default_ext,string file_name, string filter)

Display a File Save dialog – this is usually used to select a new file for saving to but user can choose to overwite an existing file. Returns true if user selects a file else false if they cancel the dialog.

default_ext – string – The default filename extension. If the user does not include an extension in the Filename edit box, the extension specified here is automatically appended to the filename. If this parameter is empty (“”), no file extension is appended.

file_name – string –path to default file to open. Can just be path or include filename or just be left empty (“”), usualy use “*.ext” at end of string to preselect all files of a particular extension.

filter – string –filter string to display for dialog file type.The string is build up of pairs of ‘file description’ followed by ‘file extension’ separated by ‘|’. The filter list is terminated by ‘||’ . E.g “Toolpaths (*.tap) | *.tap||”.

Example Code

```
local file_dialog = FileDialog()
if not file_dialog:FileSave(
                          "tap",
                          "d:\\Temp\\*.tap" ,
                           "Toolpaths (*.tap)|*.tap|"
                           ) then
   MessageBox("No file to save to selected")
   return false
end

MessageBox("User chose file ...\n" .. file_dialog.PathName)
```

Properties

.InitialDirectory
R/W – string – folder/directory the dialog should display when first opened

The properties below are used to access the data on the file chosen and are only valid after the dialog has returned from :FileOpen() or FileSave()

.Directory
R/O – string – directory/folder (and drive) of the file the user chose (same as .Folder)

.FileExt
R/O – string – extension of the file chosen

.FileName
R/O – string – filename without the path but including the extension

.FileTitle
R/O – string – name of file without path or extension

.PathName
R/O – string – The full pathname to the file the user

.Folder
R/O – string – folder/directory (and drive) of the file the user chose (same as .Directory)

ProgressBar

This object is used to display a progress bar in the host program while doing time consuming tasks.

Constructor

ProgressBar(string text, ProgressBarMode progress_bar_mode) – Constructor

Creates a new progress bar and displays it in the host program.

text – string – The text displayed to the left of the progress bar.
progress_bar_mode-integer-Type of progresss bar to create. Valid values are...
ProgressBar.LINEAR – Use this when you can compute your percentage progress while processing.
ProgressBar.PINGPONG – Use this when you can't compute your percentage progress.

Methods

:SetPercentProgress(integer percent)

Call this moderately frequently to update the progress bar with your percentage progress. Should be used in conjunction with ProgressBar.LINEAR

percent-integer-The percentage task completion between 0 and 100, does not have to be monotonically increasing in value. Progress bars can be reappropriated for different tasks by using SetText().

:StepProgress()

Call this moderately frequently to update the progress bar when you can't complete your percentage progress. Should be used in conjunction with ProgressBar.PINGPONG

:SetText(string text)

Sets the text displayed to the left of the progress bar.

text-string-The text to be displayed.

:Finished()

This should always be called when you're finished with a progress bar.

Components – Aspire Only

Components form the basis of 3D modeling within Aspire. A Component is created when a Relief is added to the ComponentManager. In order to reduce memory consumption, the Components use a Relief referencing scheme coordinated by the ComponentManager. In short several Components can reference a single Relief.

Example Code from 01_Creation.lua

```
-- Check for the existence of a job
local job = VectricJob()
if not job.Exists then
  return false
end

-- Try and create a relief
local pixel_width = 1000
local pixel_height = 1000
local width = 5.0
local height = 5.0
local relief = Relief(pixel_width, pixel_height, width, height)

-- Set its data
local max_r = 0.5 * math.sqrt(pixel_width * pixel_width + pixel_height * pixel_height)
for y = 0, pixel_height - 1 do
  -- We want our waves to eminate from the centre
  local y_r = (2 * y - pixel_height) / 2
  for x = 0, pixel_width - 1 do
    -- We want our waves to eminate from the centre
    local x_r = (2 * x - pixel_width) / 2
    -- Compute our normalized radial position from the model centre
    local r = math.sqrt(x_r * x_r + y_r * y_r) / max_r
    -- Our amplitude is a gaussian function of our radial position
    local amp = math.exp(-10.0 * (r - 0.3) * (r - 0.3))
    --Set the height from a cos function
    local z = 0.5 + 0.2 * amp * math.cos(2 * math.pi * 5 * r)
    relief:Set(x, y, z)
  end
end


-- Add the relief to the manager
local component_manager = job.ComponentManager
component_manager:AddRelief(relief, CombineMode.Add, "01 Creation")
```

CombineMode – Aspire Only

This is an ‘enum’ which represents the values used to specify a combine mode for many operations. Combine modes instruct Aspire how overlapping components should interact with each other.

Properties

.Add

R/O – When a component overlaps with another at a pixel combine their heights additively

.Subtract

R/O – When a component overlaps with another at a pixel combine their heights subtractively

.MergeHeighest

R/O – When a component overlaps with another at a pixel combine their heights by taking the highest value

.MergeLowest

R/O – When a component overlaps with another at a pixel combine their heights by taking the lowest value

.Multiply

R/O – When a component overlaps with another at a pixel combine them by multiplying their heights together

.Replace

R/O – When a component overlaps with another at a pixel combine replacing pixels

Relief – Aspire Only

Reliefs are two dimensional grids of pixels with a height value associated at each pixel. They are used to represent 2D height fields that can have transparent regions. Reliefs can be created programmatically and converted into Components by adding a relief to the ComponentManager or obtained by querying a Component for its relief. The position of a relief is set using the Transform methond on the associated component once it is created.

Constructors

Relief(integer pixel_width, integer pixel_height, double real_width, double_real_height) – Constructor

pixel_width-integer–The width of the relief in pixels
pixel_height-integer –The height of the relief in pixels
real_width-double –The width of the relief in real world units
real_height-double –The height of the relief in real world units

Relief(Box2D bounds, float pixel_size) – Constructor

bounds-Box2D –The real world dimensions of the relief
pixel_size-float –The side length of a pixel in real world units

Properties

.Error

R/O – bool – true if there was an error during relief construction

.PixelWidth

R/O – integer – the width in pixels of the relief

.PixelHeight

R/O – integer – the height in pixels of the relief

.RealWidth

R/O – float – the real width of the relief

.RealHeight

R/O – float – the real height of the relief

.XPixelSize

R/O – integer – the width of a pixel in real units

.YPixelSize

R/O – integer – the height of a pixel in real units

.Thickness

R/O – float – the thickness of the block in real units - 0.0 for reliefs created by Lua

.SurfaceZ

R/O – float – returns the surface z in real units – 0.0 for reliefs created by Lua

.Volume
R/O – float – returns the volume of the relief

.IsTransparent
R/O – bool – returns true if the relief consists entirely of transparent pixels

Methods

:Reset(float value, bool free_memory)
Sets all heights in the relief

value- float –The height to set all pixels in the relief
free_memory-bool –Has no effect, ignored

:MakeValuesTransparent(float value)
Makes all parts of the relief at the given height transparent

value-float –The height of the pixels to be made transparent

:ReplaceTransparentValues(float value)
Replaces all transparent pixels with the given height

value-float-The height that all transparent pixels will become

:PointIsOnModel(integer x, integer y)
Returns true if x is between zero and the pixel width and y is between zero and the pixel height

x-integer -The x coordinate of the pixel to test
y-integer-The y coordinate of the pixel to test

:Set(integer x, integer y, float value)
Sets the height for the relief at the point to the specified value

x-integer -The x coordinate of the pixel to set
y-integer-The y coordinate of the pixel to set
value-float-The height value to assign to the pixel

:SetLowest(integer x, integer y, float value)
Sets the height for the relief at the point to whichever is lower, the current value or the passed value. Returns true if the height was changed

x-integer-The x coordinate of the pixel to set
y-integer-The y coordinate of the pixel to set
value-float-The height value to try and assign to the pixel

:Get(integer x, integer y)
Returns the height for the relief at the point in pixel coordinates

x-integer -The x coordinate of the pixel to get
y-integer-The y coordinate of the pixel to get

:GetInterpolated(double x, double y)
Returns the interpolated height for the relief for the specified coordinate

x-double -The x coordinate in pixel units
y-double -The y coordinate in pixel units

:GetMinMaxZ()
Returns the maximum and minimum z values of the relief taking into account the thickness and a boolean flag indicating if the model is transparent

```
local transparent, min_z, max_z = relief:GetMinMaxZ()
```

:GetTrueMinMaxZ()
Returns the maximum and minimum z values of the relief ignoring the thickness and a boolean flag indicating if the model is transparent

```
local transparent, min_z, max_z = relief:GetTrueMinMaxZ()
```

:Add(float value)
Adds onto all non transparent pixels

value-float -The height to add onto each non transparent pixel

:Subtract(float value)
Subtracts from all non transparent pixels

value-float -The height to subtract from each non transparent pixel

:Multiply(float value)
Multiply all non transparent pixels

value-float -The height to multiply each non transparent pixel

:MergeHeighest(float value, bool is_abs_value, bool preserve_transparent)
Merge heighest over the entire relief

value-float -The height value to merge into the relief
is_abs_value-bool -If true the relief's Z offset is taken into account
preserve_transparent-bool - If true transparent pixels remain transparent

:MergeLowest(float value, bool is_abs_value, bool preserve_transparent)
Merge lowest of the entire relief

value-float -The height value to merge into the relief
is_abs_value-bool -If true the relief's Z offset is taken into account
preserve_transparent-bool - If true transparent pixels remain transparent

:CombineReliefs(Relief relief, CombineMode combine_mode)
Combines the relief

relief-Relief -The relief to combine into this
combine_mode- CombineMode *-How the relief should be combined*

:BlendBetweenReliefs(Relief first_relief, Relief second_relief, double blend_factor)
Returns a new relief that is a blended version of the two passed reliefs

first_relief-Relief -The first relief to blend
second_relief-Relief -The second relief to blend, must have the same dimensions as the first
blend_factor-double –0.0 all first, 1.0 all second

:FlipY()
Flips the relief mirroring it vertically

:FlipX()
Flips the relief mirroring it horizontally

:TransposeXY()
Transposes the relief's pixels

:RotateClockwise90()
Rotates the relief 90 degrees clockwise

:RotateCounterClockwise90()
Rotates the relief 90 degrees counter clockwise

:RenderTriangle(Point3D p1, Point3D p2, Point3D p3, bool merge_high)
Renders a triangle into the relief

p1-Point3D -The first point of the triangle
p2-Point3D -The second point of the triangle
p3-Point3D -The third point of the triangle
merge_high-bool –If true merge the triangle high

:CreateSlice(float min_z, float max_z, bool make_below_min_z_transparent)
Returns a relief sliced between the specified Z range

min_z-float-The minimum Z value to include
max_z-float-The maximum Z value to include
make_below_min_z_transparent-bool-If true heights below min_z become transparent

:ColumnIsTransparent(integer x, float transparent_value)
Returns true if the column consists entirely of pixels that are of the transparent value

x-integer-The column pixel coordinate
transparent_value-float-The Z value to consider transparent

:RowIsTransparent(integer y, float transparent_value)
Returns true if the row consists entirely of pixels that are of the transparent value

x-integer-The row pixel coordinate
transparent_value-float-The Z value to consider transparent

:CreateSectionCopy(integer min_x, integer min_y, integer max_x, integer max_y)

Returns a subset of the relief between the coordinate ranges

min_x-integer-The minimum x pixel coordinate to include
min_y-integer- The minimum y pixel coordinate to include
max_x-integer- The maximum x pixel coordinate to include
max_y-integer- The maximum y pixel coordinate to include

:TiltRelief(Point2D anchor_pt, Point2D direction_pt, double angle)

Tilts the relief

anchor_pt-Point2D –Start of vector determining tilt direction
direction_pt-Point2D –End of vector determining tilt direction
angle-double –The the angle to tilt in degrees

ComponentManager – Aspire Only

This object is responsible for managing all the components within the application. A reference to the ComponentManager is obtained via the ComponentManager of the VectricJob. The ComponentManager manage all the Components in the job.

```
local component_manager = job.ComponentManager
```

Methods

:AddRelief(Relief relief, CombineMode combine_mode, string name)

Creates a Component from a Relief and returns the UUID of the component created

relief-Relief- The relief to add as a component
combineMode- CombineMode -The combine mode to assign to the component
name -string-The name to give to the component created from the relief

:FindObjectIdWithName(string name)

Returns the UUID of the component with the specified name

name-string - Name for the component to find

:FindObjectWithId(UUID uuid)

Returns the Component with the given id

uuid- UUID-The unique id of the component to find

:GetSelectedObjectIds()

Returns a UUID_List containing the UUIDs of the currently selected components

:GetTopLevelObjectIds()

Returns a UUID_List containing the UUIDs of the top level components in the component tree

:DeleteObjectWithId(UUID uuid)

Returns true if the Component with the given is UUID deleted

uuid- UUID-The unique id of the component to delete

:DeleteObjectsWithIds(UUID_List uuids)

Returns true if the Components with the UUIDs in the list are successfully deleted
uuids- UUID_List -The unique ids of the components to delete

:CreateBakedCopyOfObjects(UUID_List uuids, bool delete_originals)

Returns the UUID of the created component

uuids-UUID_List- The list of component ids to bake

delete_orignal-bool –If true delete the original components

:CreateBakedCopyOfObject(UUID uuid, bool delete_originals)

Returns the UUID of the created Component

uuid-UUID - The id of the component to bake
delete_orignal-bool -If true delete the original components

:GroupSelectedObjects()

Group the currently selected components. Returns the UUID of the grouped component

:GroupObjectsWithIds(UUID_List uuids)

Group the Components with the ids in the list. Returns the UUID of the grouped component

uuids- UUID_List -The unique ids of the components to group

:UnGroupSelectedObjects()

Ungroup the currently selected components. Returns true if the ungroup was successful

:UnGroupObjectsWithId(UUID uuid)

Ungroup the Component with the given UUID in the list. Returns true if the component was ungrouped

uuid- UUID -The unique id of the component to ungroup

:SelectionCanBeGrouped()

Returns true if the selection can be grouped

:SelectionCanBeUngrouped()

Returns true if the selection can be ungrouped

:CloneObjectWithId(UUID uuid, bool make_unique)

Returns the cloned version of the Component

uuid- UUID - The id of the component to clone
make_unique-bool –If true make a deep copy of the component

:CreateReliefFromObjectWithId(UUID uuid)

Returns a Relief cloned from the Component with the given id

uuid- UUID - The id of the component to clone a relief from

:ReplaceComponentRelief(UUID target_uuid, UUID source_uuid)

Returns true if the ComponentManager was able to replace the target component's Relief with that of the source

target_uuid-UUID - The id of the component whose relief we are to replace
source_uuid-UUID - The id of the component whose relief we are to use as the replacement

:UpdateCompositeModel()

Updates the Composite Model. This method should always be called when you have completed a set of Component operations to ensure that the Composite Model displayed to the user is up to date.

:CreateReliefFromCompositeModel()

Creates a Relief from the Compsite Model

:UpdatePreviews()

Updates the Component Previews in the 2D view. This method should always be called when you have completed a set of Component operations to ensure that the Component Previews are correctly positioned

Component – Aspire Only

This object is the base class for all the different types of components managed by the ComponentManager. There are two classes of Component available to Lua; Component itself and ComponentGroup. Components reference the Reliefs that were used to create them. It is possible to have multiple Components referring to the same original Relief and this should be kept in mind while manipulating Component data.

Properties

.Id

R/O - UUID - Returns the id of the component

.CombineMode

R/W - CombineMode - The combine mode of the component

.IsTilted

R/O – bool -Returns true if the component is tilted

.UseTilt

R/W – bool – Is the component using tilt

.IsFaded

R/O – bool – Returns true if the component is faded

.UseFade

R/W – bool – Is the component using fade

Methods

:Transform(Matrix2D xform, bool update_all)

Tranforms the component

xform- Matrix2D -The transform
update_all -bool-Update the entire bounds of the component tree

:SetZOffset(float z_offset)

Set the Z offset of the component

z_offset-float-The amount by which to offset the component

:SetZScale(float scale)

Scale the Z heights of the component

scale-float-The amount by which to scale the heights of the component

:TiltWouldChange(Point2D anchor_pt, Point2D direction_pt, double tilt_angle, bool do_tilt)

Returns true if the parameters would change the current tilt

anchor_pt-Point2D -Start of vector determining tilt direction
direction_pt- Point2D -End of vector determining tilt direction
tilt_angle-double -The angle to tilt at in degrees
do_tilt -bool -If true we want to apply a tilt

:SetTilt(Point2D anchor_pt, Point2D direction_pt, double tilt_angle)

Returns true if the component was tilted successfully

anchor_pt-Point2D -Start of vector determining tilt direction
direction_pt- Point2D-End of vector determining tilt direction
tilt_angle-double -The angle to tilt at in degrees

:GetTiltData(Point2D anchor_pt, Point2D direction_pt)

Returns the tilt angle and the passed points are updated to contain the component's tilt anchor and directions points

anchor_pt-Point2D -Start of vector determining tilt direction
direction_pt- Point2D -End of vector determining tilt direction

```
local anchor_pt = Point2D()
local direction_pt = Point2D()
local tilt_angle = component:GetTiltData(anchor_pt, direction_pt)
```

:FadeWouldChange(Point2D anchor_pt, Point2D direction_pt, double end_fade_val, bool do_fade)

Returns true if the parameters would change the current fade

anchor_pt-Point2D -Start of vector determining fade direction
direction_pt- Point2D -End of vector determining fade direction
end_fade_val-double -The final value of the fade
do_fade-bool -If true we want to apply a fade

:SetFade(Point2D anchor_pt, Point2D direction_pt, double tilt_angle)

Returns true if the component was faded successfully

anchor_pt-Point2D -Start of vector determining fade direction
direction_pt- Point2D -End of vector determining fade direction
end_fade_val-double -The final value of the fade

:GetFadeData(Point2D anchor_pt, Point2D direction_pt)

Returns the end fade value and the passed points are updated to contain the component's fade anchor and directions points

anchor_pt-Point2D -Start of vector determining fade direction
direction_pt- Point2D -End of vector determining fade direction

```
local anchor_pt = Point2D()
local direction_pt = Point2D()
local tilt_angle = component:GetTiltData(anchor_pt, direction_pt)
```

:GetRelief()

Returns the Relief for the component. If the component is a ComponentGroup then this will return nil. Please note any changes made to this relief will affect the appearance of any other components that reference this Relief

ComponentGroup – Aspire Only

This object represents a group of Components. A ComponentGroup may contain ComponentGroups.

Properties

.GetCount

R/O – integer – returns the number of components in the group

.Count

R/O – integer – returns the number of components in the group

.IsEmpty

R/O – bool – returns true if the group is empty

Methods

:GetHead()

Return the component at the head of the group, the component remains in the group

:GetHeadPosition()

Returns a POSITION variable to allow access to the head of the list of components in the group

:GetTailPosition()

Returns a POSITION variable to allow access to the tail of the list of components in the group

:GetNext(POSITION pos)

Returns the component at the current position AND a new value for position pointing to the next item in the group (or nil if at end of group)

pos - POSITION - current position in group

:GetPrev(POSITION pos)

Returns the component at the current position AND a new value for position pointing to the previous item in the group (or nil if at end of group)

pos - POSITION - current position in group

:GetAt(POSITION pos)

Returns the component at the passed position

pos - POSITION - position in group