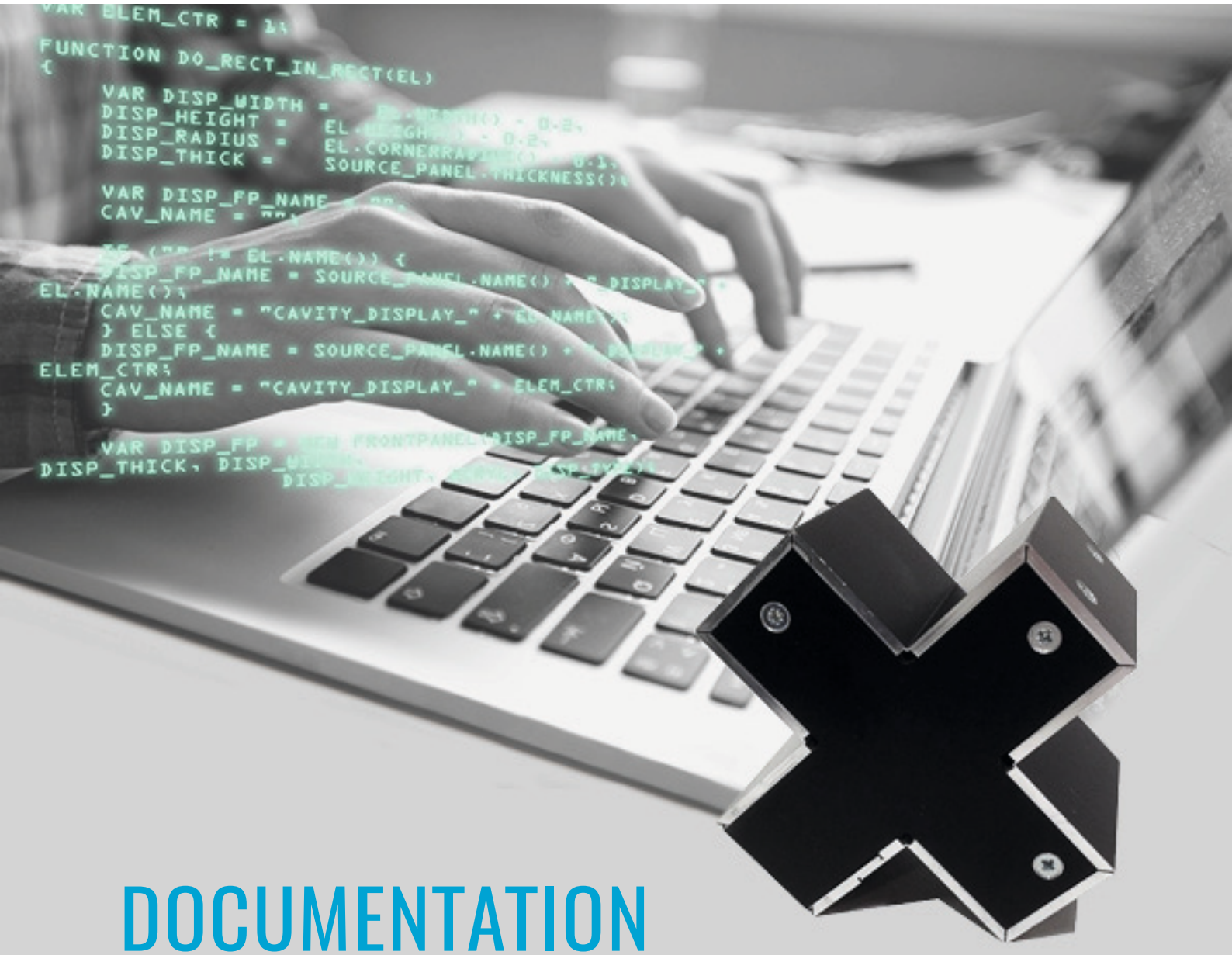
Schaeffer AG

FRONT PANELS ▪ ENCLOSURES ▪ MILLING PARTS



DOCUMENTATION
FRONTPANEL DESIGNER SCRIPTING INTERFACE

www.schaeffer-ag.de

FRONTPANEL DESIGNER SCRIPTING INTERFACE

The Front Panel Designer offers a programming interface (API), which allows users to formulate action sequences using milling and engraving objects on the current or on a newly created front panel as an executable script written in JavaScript. All actions will be performed in the scripting window, which can be opened by clicking Edit > Scripts .

Description of the scripting window in ONLINE HELP

This document describes the Frontpanel Designer's scripting programming interface. The scripting language employed is JavaScript. We assume basic knowledge of programming in JavaScript. The Frontpanel Designer bundles some example scripts that provide an overview of a script's structure.

CONTENTS

Imprint: Schaeffer AG, Nahmitzer Damm 32, 12277 Berlin, Germany, 030 / 805 86 95-0, info@schaeffer-ag.de
Board of Directors: Jörg Schaeffer und Kai Schaeffer, HRB Berlin: HRB 93611 B, USt-IdNr.: DE 136 500 268

1 Introduction

In principle, there are two methods of operating: one is to work on the frontpanel currently focussed in the Frontpanel Designer; this frontpanel will be available as the global object *frontpanel*. The second way is to create one or more frontpanels from the script and add them to the Frontpanel Designer. In both cases, frontpanel engraving objects, herein called "elements", may be added to the frontpanel, modified, deleted, etc. In addition, it is possible to query values from the user by means of a freely configurable dialog box, and to use these values, for example, for creating elements.

Example:

```
var fp = new Frontpanel("My Frontpanel", thick_4mm, 300, 300,
                        alu_elox, elox_natural);

var cv1 = new Cavity("cv1", 10);
cv1.MakeCircular(25, 2);
fp.AddElement(cv1, 50, 50);

AddFrontpanel(fp);
```

This example shows different objects: first, we create a new frontpanel object and assign it to the variable *fp*. Then we create a new cavity in a similar manner and assign it to the variable *cv1*. To change the cavity into a circular cavity, we call the method *MakeCircular* on the object *cv1*. In a similar way, we call the method *AddElement* on the frontpanel object *fp* to place the cavity *cv1* on the frontpanel. Finally, we call the global function *AddFrontpanel* to display the frontpanel *fp* in the Frontpanel Designer.

Elements are named and an element can be found by its name without having to keep a variable reference. This permits working with frontpanels loaded from disk that contain elements generated by script. The element's name should be unique per frontpanel and is passed to the element as first parameter of its constructor. In the same way, each generated frontpanel has a name, which is also used as the frontpanel's filename.

The currently focussed frontpanel is available as *frontpanel*. For example, we could have added *cv1* to the current frontpanel by using `frontpanel.AddElement(cv1, 50, 50);` .

The function *AddFrontpanel* sets the given frontpanel as *frontpanel*; *frontpanel* also changes if the user focusses a different frontpanel. If no frontpanel is displayed in the Frontpanel Designer, then *frontpanel* is unbound. If used, an error message is displayed. This case can be avoided by using the global function *HasFrontpanel*.

Unless specified differently, all length specifications and positions are in *mm* (millimeter). The frontpanel's origin is at the lower left.

Below we will list the global functions and the specific element objects together with their methods. Each entry corresponds to the pattern

MethodName (*Parameter1*, *Parameter2*, ...) → *ResultValue*

Multiple values are returned as a JavaScript array. A result specified as *true* | *false* means *true* or *false*.

Parameters written in square brackets ([]) are optional and are kept at default values if they are not given.

Global functions work as methods on an implicit, *global* object that needn't be specified. Any other function needs a corresponding object, on which it is called by `object.Method(...);` . Many methods that are called just because of their side effects, and which don't return a meaningful value, return this object instead. Therefore, we can chain these method calls in a convenient way, for example, if *el* is an ellipsis engraving: `el.SetWidth(50).MirrorX().MirrorY();` .

Some functions require predefined constants as parameters (e. g. for colors, material or threadings.) Those are listed in the last section under "Overview of Constants".

Note: The scripting interface is currently marked as experimental. Some functions will probably change in the future. It is likely that not all functions work correctly and even crashes can be expected. It is recommended to save changed frontpanels before using scripts. We appreciate bug reports, change suggestions and feature requests.

2 Global Functions

Global functions work on an implicit global object which is not specified. They are called e. g. as following: `if (HasFrontpanel()) ...;` .

HasFrontpanel () → *true* | *false*

Whether a focussed frontpanel is available.

AddFrontpanel (*frontpanel*) → *unspec*

Adds *frontpanel* to the frontpanels displayed by Frontpanel Designer.

RemoveFrontpanel (*frontpanel*) → *unspec*

Removes the panel *frontpanel*. After calling this function, the *frontpanel* object is invalid and must not be used anymore. *[NB: Not available in this release.]*

EvalFile (*filename*) → *unspec*

Runs script file *filename*.

Print (*message*) → *unspec*

Displays *message* in the output window.

Error (*message*) → *stop*

Displays *message* in the output window and terminates script.

Halt () → *stop*

Ends script execution without error.

_ (*text*) → *translated_text*

Function name underscore (_): translate *text* using a Gettext catalog. If there is no entry in the catalog, *text* is returned unchanged.

__ ("'*text_de*|*text_en*|*text_fr*"') → *translated_text*

Function name double underscore (__): select one of the three possible translations according to language settings of the Frontpanel Designer. The three strings are separated by the vertical line character (|). If a translation is missing, the next one from the order English, German, French will be chosen.

LoadFrontpanel (*filename*, *name*) → *Frontpanel*

Loads a frontpanel from the FPD file *filename*. The frontpanel's name is set to *name*.

SaveFrontpanel (*frontpanel*, *filename*, *overwrite*) → *[undef]*

Writes the frontpanel *frontpanel* into a file *filename*. The boolean value *overwrite* specifies whether an existing file may be overwritten.

3 Frontpanels

Frontpanel objects are of type *Frontpanel* and represent a frontpanel. Frontpanel functions need a frontpanel object on which they act as methods. Such an object can be created by *new Frontpanel* and made visible in the Frontpanel Designer by *AddFrontpanel*. Additionally, the frontpanel currently focussed in Frontpanel Designer is bound as global object *frontpanel*.

Constructor

new Frontpanel (*name*, *thickness*, *width*, *height*, *material*, *matcolor*) → *Frontpanel*

Creates a new frontpanel with parameters passed. The new frontpanel can be added to Frontpanel Designer with the global function *AddFrontpanel*.

Methods

AddElement (*e*, [*x*, *y*]) → *e*

Adds Element *e* to the frontpanel. If a position *x*,*y* is provided, the reference point of *e* will be adjusted.

FindElement (*name*) → *element* | *false*

Finds Element with name *name*.

RemoveElement (*e*) → *e*

Removes Element *e* from the frontpanel.

MoveElement (*e*, *x*, *y*) → *e*

Moves Element to new position *x*,*y* (sets reference point).

Count () → *nelem*

Returns number of elements on the frontpanel.

Elements () → *list of elements*

Returns all elements on the frontpanel.

ElementsInBoundingBox (*x1*, *y1*, *x2*, *y2*) → *list of elements*

Returns elements that are contained entirely within the bounding box.

Thickness () → *mm*
Height () → *mm*
Width () → *mm*

Returns the frontpanel's thickness, height or width in mm.

SetHeight (*mm*) → *Frontpanel*
SetWidth (*mm*) → *Frontpanel*

Sets the frontpanel's height or width in mm.

Material () → *mattype*

Returns material type (cf. Overview).

SetMaterial (mattype) → *Frontpanel*

Sets material type.

MaterialColor () → *matcolor*

Returns material color (cf. Overview).

SetMaterialColor (*matcolor*) → *Frontpanel*

Sets material color.

Origin () → *x,y*

Returns the origin as 2-element array.

SetOrigin (*x*, *y*) → *Frontpanel*

Sets the origin to (*x*,*y*). The panel must be visible in the Frontpanel Designer (cf. *AddFrontpanel*).

IsModified () → *true* | *false*

Whether the frontpanel has unsaved modifications.

SetModified () → *Frontpanel*

UnsetModified () → *Frontpanel*

Sets/unsets modified-flag (whether panel has unsaved modifications).

Ungroup (*g*) → *Frontpanel*

Dissolve group of elements *g*.

IsFilledIn () → *true* | *false*

Whether elements are filled-in with color.

SetFilledIn () → *Element*

Fills-in elements with color. Cf. *Element.SetColor*.

UnsetFilledIn () → *Element*

Does not fill-in elements with color.

IsPrinted () → *true* | *false*

Whether elements are printed.

SetPrinted () → *Element*

Prints elements. Cf. *Element.SetColor*.

UnsetPrinted () → *Element*

Does not print elements.

SetRectangular () → *Frontpanel*

SetElliptical () → *Frontpanel*

Makes the frontpanel rectangular or elliptical in shape.

SetBorderContour (*dxfcontour*) → *Frontpanel*

Specifies the frontpanel's frame contour by the DXF contour *dxfcontour*.

SetCornerRadii (*r1*, *r2*, *r3*, *r4*) → *Frontpanel*

Sets the frontpanel's corner radii in mm.

Name () → *name*

Returns the frontpanel's name as string.

SelectElement (*element*) → *Frontpanel*

Selects the element.

Remark () → *string*

SetRemark (*remark*) → *frontpanel*

Obtains or sets the frontpanel's manufacturing remarks.

IsCustomerMat () → *true* | *false*

Whether the frontpanel material is flagged as customer material.

SetCustomerMat () → *Frontpanel*

UnsetCustomerMat () → *Frontpanel*

Flags (unflags) the frontpanel as using customer material.

GridOn () → *Frontpanel*

GridOff () → *Frontpanel*

Switches the grid display for this frontpanel on or off.

Selection () → *array of Element*

Returns an array containing the selected elements on the panel.

SetEdgeMachining (*front_shape*, *front_depth*, *rev_shape*, *rev_depth*) → *Frontpanel*

Sets edge machining for the frontpanel (bevel/chamfer or radius). Setting depth is only meaningful for a bevel, set to 0 for radius. For constants cf. Overview.

4 Elements

4.1 Common Functions

All elements have a set of common functions. The following functions are available for all elements and won't be listed there specifically.

NB: Not every function makes sense for every element type; for example, not every element can be filled-in with a color.

Methods

Is<ElementTy> () → *true* | *false*

Whether element is of type <*ElementTy*>.

Example: `if (elem.IsCavity()) Print("It's a Cavity!\n");`

Resize (*width*, *height*) → *Element*

Sets size of element in mm.

Width () → *mm*

Height () → *mm*

Returns width or height of the element's bounding box.

SetWidth (*width*) → *Element*

SetHeight (*height*) → *Element*

Sets the element's width or height in mm.

Rescale (*factor*) → *Element*

Rescales the element by a factor.

X () → *mm*

Y () → *mm*

Returns the X (resp. Y) position of the element's reference point.

MoveTo (*x*, *y*) → *Element*

Moves the element (sets reference point) to position *x*,*y*.

CanMirror () → *true* | *false*

Whether the element can be mirrored.

MirrorX () → *Element*

Mirrors the element horizontally.

MirrorY () → *Element*

Mirrors the element vertically.

CanReverseSide () → *true* | *false*

Whether the element can be placed on the frontpanel's reverse side.

IsReverseSide () → *true* | *false*

Whether the element is placed on the frontpanel's reverse side.

PutOnReverseSide () → *Element*

Places the element on the frontpanel's reverse side.

PutOnFrontSide () → *Element*

Places the element on the frontpanel's obverse side.

Color () → *color*

Returns the element's color.

SetColor (color) → *Element*

Sets the element's fill-in color (engravings colors, cf. Overview).

Angle () → *deg*

Returns the rotation angle in degrees.

SetAngle (angle) → *Element*

Sets the rotation angle in degrees.

Rotate (angle) → *Element*

Rotates the element counter-clockwise by *angle* degrees.

Tool () → *tool*

Returns the tool used for this element (cf. Overview).

SetTool (tool) → *Element*

Sets the tool used for this element. Set *auto* for automatic tool selection. The following elements support tool setting: *DHole*, *DxfContour*, *Line*, *Rectangle*, *Ellipse*, *Cavity*, *RectHole*, *CurvedSlot*, *TextEngraving*.

IsAutoTool () → *true* | *false*

Whether the currently used tool is to be selected automatically.

Duplicate (*name*) → *Element*

Returns a copy of the element with new name *name*.

Name () → *name*

SetName (*name*) → *Element*

Gets or sets the element's name as a string.

CanEdgeMachining () → *true* | *false*

Whether edge machining (bevel/chamfer or radius) is possible for this element.

SetEdgeMachining (*front_shape*, *front_depth*, *rev_shape*, *rev_depth* [, *outer_contour_only*]) → *Element*

Sets edge machining for this element (bevel/chamfer or radius). Setting depth is only meaningful for a bevel, set to 0 for radius. For constants cf. Overview. For elements with inner and outer contours, per default both contours will be machined. This can be disabled by setting *outer_contour_only* to *true*.

BoundingBox () → *x1, y1, x2, y2*

Returns the axis-parallel bounding box of the element as 4-element array.

ExtInfo () → *string*

SetExtInfo (*extinfo*) → *Element*

Returns or sets the element's extinfo string.

SetHelpElement (*boolean*) → *Element*

Marks (unmarks) Element as Help Element. (ignored in Production)

IsHelpElement () → *true* | *false*

Whether the element is help element. (ignored in Production)

4.2 Drill Hole

Constructor

new DrillHole (*name*, *diameter*) → *Element*

New drill hole of diameter *diameter* in mm.

Methods

Diameter () → *mm*

Returns inner diameter.

SetDiameter (*diameter*) → *Element*

Sets inner diameter in mm.

SetBlindHole (*depth*) → *Element*

Changes the drill hole into a blind hole of depth *depth* mm. Specifying *depth* = 0 changes a blind hole into a drill hole.

Depth () → *mm*

Returns the blind hole's depth, or 0 if not a blind hole.

SetDepth (*depth*) → *Element*

Equivalent to *MakeBlindHole(depth)*.

IsBlindHole () → *true* | *false*

Equivalent to *Depth()* = 0.

SetCountersink (*type*) → *Element*

Adds a countersink of type *type* (cf. Overview).

SetCountersinkWithParameters (*cone_diameter*, *drill_hole_diameter*, *sink_depth*, *cone_angle*) → *Element*

Adds to the drill hole a countersink of specified parameters in mm.

HasCountersink () → *true* | *false*

Whether the drill hole has a countersink.

ConeDiameter () → *mm*
DrillHoleDiameter () → *mm*
SinkDepth () → *mm*
ConeAngle () → *mm*

Returns countersink parameter in mm.

SetConeDiameter (*cone_diameter*) → *Element*
SetDrillHoleDiameter (*drill_hole_diameter*) → *Element*
SetSinkDepth (*sink_depth*) → *Element*
SetConeAngle (*cone_angle*) → *Element*

Sets countersink parameter in mm.

SetThreading (*type*) → *Element*

Adds a threading of type *type* (cf. Overview).

Diameter () → *mm*
Pitch () → *mm*

Returns respective threading parameters.

SetDiameter (*diameter*) → *Element*
SetPitch (*pitch*) → *Element*

Sets respective threading parameter in mm.

HasThreading () → *true* | *false*

Whether the drill hole has a countersink.

4.3 Rectangular Hole

Constructor

new RectHole (*name*, *width*, *height* [,*corner_radius*]) → *Element*

New rectangular hole. Default value for *corner_radius* is 1.5mm.

Methods

CornerRadius () → *radius*

Returns corner radius.

SetCornerRadius (*radius*) → *Element*

Sets corner radius.

SetTrapezoid (*angle_left* [, *angle_right*]) → *Element*

Creates the hole as a trapezoid with respective angles in degrees. If only *angle_left* is given, the trapezoid will be realized as a triangle.

AngleLeft () → *deg*

AngleRight () → *deg*

Returns respective angle in degrees.

4.4 D-Hole

Constructor

new DHole (*name*, *shape*, *diameter*, *width* [, *height*]) → *Element*

New D-Hole, *shape* types cf. Overview. Parameter *height* is used when *shape* = *dhole_quadruple*.

Methods

Shape () → *shape*

Returns the *shape* parameter.

SetShape (*shape*) → *Element*

Sets the *shape* parameter.

4.5 Curved Slot

Constructor

new CurvedSlot (*name*, *inner_radius*, *width*, *arc_length* [, *corner_radius*]) → *Element*

New curved slot with inner radius and width in mm, arc length in degrees. Default value for *corner_radius* is $\frac{1}{2}$*width*.

Methods

InnerRadius () → *mm*

ArcLength () → *deg*

CornerRadius () → *mm*

Returns respective parameter.

SetInnerRadius (*inner_radius*) → *Element*

SetArcLength (*arc_length*) → *Element*

SetCornerRadius (*corner_radius*) → *Element*

Sets respective parameter.

4.6 Cavity

Constructor

new Cavity (*name*) → *Element*

Create new cavity. Cavity type is defined by using one of the *Make*-methods below.

Methods

MakeCircular (*diameter*, *depth*) → *Element*

MakeRectangular (*width*, *height*, *depth*, [, *corner_radius*]) → *Element*

MakeRectInRect (*width*, *height*, *depth*, *width_inner*, *height_inner*,
corner_radius, *radius_inner* [, *offset_x*, *offset_y*]) → *Element*

MakeCircleInCircle (*diameter*, *depth*, *diameter_inner* [, *offset_x*, *offset_y*]) → *Element*

MakeCircleInRect (*width*, *height*, *depth*, *diameter_inner*,
corner_radius [, *offset_x*, *offset_y*]) → *Element*

MakeRectInCircle (*diameter*, *depth*, *width_inner*,
height_inner, *radius_inner* [, *offset_x*, *offset_y*]) → *Element*

Changes cavity type into the one specified.

Example: `var c = new Cavity("cav1"); c.MakeCircular(12.5, 5.3);`

Type () → *type*

Returns type of cavity (cf. Overview).

Depth () → *mm*

Diameter () → *mm*

CornerRadius () → *mm*

WidthInner () → *mm*

HeightInner () → *mm*

OffsetX () → *mm*

OffsetY () → *mm*

DiameterInner () → *mm*

RadiusInner () → *mm*

Returns respective parameter.

SetDepth (*depth*) → *Element*

SetDiameter (*diameter*) → *Element*

SetCornerRadius (*corner_radius*) → *Element*

SetWidthInner (*width_inner*) → *Element*

SetHeightInner (*height_inner*) → *Element*

SetOffsetX (*offset_x*) → *Element*

SetOffsetY (*offset_y*) → *Element*

SetDiameterInner (*diameter_inner*) → *Element*

SetRadiusInner (*radius_inner*) → *Element*

Sets respective parameter. Error if parameter is invalid for cavity type.

4.7 DXF Contour

Constructor

new DxfContour (*name*, *filename*, *refpoint*, *scale_percentage* [, *as_cavity*]) → *Element*

Reads contour from DXF file *filename*. Sets reference point according to specified gravity (cf. Overview). If *filename* is empty, an empty element for drawing a contour into it will be created.

Methods

RefPoint () → *refpoint*

Scale () → *scale_percentage*

AsCavity () → *true* | *false*

Depth () → *depth*

Returns respective parameter.

SetRefPoint (*refpoint*) → *Element*

SetScale (*scale_percentage*) → *Element*

SetAsCavity (*boolean*) → *Element*

SetDepth (*depth*) → *Element*

Sets respective parameter.

The element *DxfContour* permits relatively free contouring, provided the paths do not cross and are closed. A path is created by the methods *LineTo*, *ArcToMP*, *Bezier2To* and *Bezier3To*. Start and end of a path must be marked by methods *Start* and *Finish*. A contour may be comprised of multiple contour paths, which are used as the element's outer contours.

Start (*x*, *y*) → *Element*

Starts contour path at given starting point.

LineTo (*x*, *y*) → *Element*

Draws a line from the current position to *x,y*.

ArcToMP (*x1*, *y1*, *x2*, *y2*, *dir*) → *Element*

Draws circle arc segment to *x1,y1*. The circle's center is specified by *x2,y2*.

Bezier2To (*x1*, *y1*, *x2*, *y2*) → *Element*

Draws a quadratic Bézier curve from the current position via control point *x2,y2* to end point *x1,y2*.

Bezier3To (*x1*, *y1*, *x2*, *y2*, *x3*, *y3*) → *Element*

Draws a cubic Bézier curve from the current position via control points *x3,y3* and *x2,y2* to end point *x1,y2*.

Finish () → *Element*

Ends a path. If the end point is not identical to the starting point, a line will be inserted to connect the two.

4.8 Text Engraving

Fonts are specified by a string ("fontspec") of the form *font*:*size*, e. g., "century-outline:8mm". For the values of *font*, see Overview, *size* is a height specification like, e. g., "8mm".

Constructor

new TextEngraving (*name*, *string*) → *Element*

New text engraving of text *string*.

Methods

Text () → *string*

TextHeight () → *mm* DEPRECATED

TextSize () → *mm*

ScalingX () → *scaling_x*

LineSpacing () → *mm*

Incline () → *incline_percentage*

Font () → *font_spec*

Alignment () → *align_left* | *align_center* | *align_right*

Returns respective parameter.

SetText (*string*) → *Element*

SetTextHeight (*height*) → *Element* DEPRECATED

SetTextSize (*height*) → *Element*

SetScalingX (*scaling_x*) → *Element*

SetLineSpacing (*line_spacing*) → *Element*

SetIncline (*incline*) → *Element*

SetFont (*font*) → *Element*

SetAlignment (*alignment*) → *Element*

SetVAlignment (*valignment*) → *Element*

Sets respective parameter.

4.9 HPGL Engraving

Constructor

new HpglEngraving (*name*, *filename*, *refpoint*, *scale_percentage*
[, *cross_formed_by*, *engrave_cross*]) → *Element*

New HPGL engraving. *Refpoint* analogous to *DXFContour()*. *Cross_formed_by* specifies the number of the pen used for drawing the reference point. *Engrave_cross* specifies whether the reference point cross is to be engraved aswell. Pens are enumerated $1 \ldots n$ and are specified as a number. If *filename* is empty, an empty element will be created for drawing an engraving path (contour) into it.

Methods

RefPoint () → *refpoint*

Scale () → *scale_percentage*

Returns respective parameter.

SetRefPoint (*refpoint*) → *Element*

Sets the element's reference point. For valid values, see Overview of Constants.

SetScale (*scale_percentage*) → *Element*

Sets respective parameter.

NumberOfPens () → *num_pens*

Returns number of pens defined.

PenColor (*pen*) → *color*

PenTool (*pen*) → *tool*

Returns respective pen attribute for pen number *pen*.

DefinePen (*color*, *tool*) → *pen*

Defines a new pen with color *color* and tool *tool*. Returns the new pen's number.

ChangePen (*pen*) → *Element*

Changes to pen number *pen*.

The element *DxfContour* permits relatively free engraving contours. A contour is created by the methods *LineTo*, *ArcToMP*, *Bezier2To* and *Bezier3To*. Start and end of a contour must be marked by methods *Start* and *Finish*.

Start (*x*, *y*) → *Element*

Starts contour at given starting point.

PenMoveTo (*x*, *y*) → *Element*

Moves the pen to position *x,y*.

LineTo (*x*, *y*) → *Element*

Draws a line from the current position to *x,y*.

ArcToMP (*x1*, *y1*, *x2*, *y2*, *dir*) → *Element*

Draws circle arc segment to *x1,y1*. The circle's center is specified by *x2,y2*.

Bezier2To (*x1*, *y1*, *x2*, *y2*) → *Element*

Draws a quadratic Bézier curve from the current position via control point *x2,y2* to end point *x1,y2*.

Bezier3To (*x1*, *y1*, *x2*, *y2*, *x3*, *y3*) → *Element*

Draws a cubic Bézier curve from the current position via control points *x3,y3* and *x2,y2* to end point *x1,y2*.

Finish () → *Element*

Ends a contour.

4.10 Engraving Elements

4.10.1 Line Engraving

Constructor

new Line (*name*, *length* [, *line_width*]) → *Element*

New line engraving of length *length* and width *line_width* mm. If *line_width* is not specified, the tool's default value will be used.

Methods

Length () → *mm*

LineWidth () → *mm*

Returns the line's length resp. width.

SetLength (*length*) → *Element*

SetLineWidth (*width*) → *Element*

Sets line length resp. width in mm.

4.10.2 Rectangle Engraving

Constructor

new Rectangle (*name*, *width*, *height*, [, *fill_area*, *line_width*, *corner_radius*]) → *Element*

New rectangle engraving. If *line_width* is not specified, then the tool's default value will be used. If *fill_area* is specified, the area will be filled with color (see also the *Frontpanel* methods *IsFilledIn()*, *SetFilledIn()*, *UnsetFilledIn()*.)

Methods

LineWidth () → *mm*

CornerRadius () → *mm*

Returns respective parameters.

SetLineWidth (*line_width*) → *Element*

SetCornerRadius (*radius*) → *Element*

Sets respective values.

4.10.3 Elliptic Engraving

Constructor

new Ellipse (*name*, *width*, *height* [, *fill_area*, *line_width*]) → *Element*

New elliptic engraving. If *fill_area* is specified, the area will be filled in with color (see also *FilledIn()*, *SetFilledIn()*, *UnsetFilledIn()*.)

Methods

LineWidth () → *mm*

Returns line width.

SetLineWidth (*line_width*) → *Element*

Sets the line's width.

4.11 Studs and Standoffs

Constructor

new Bolt (*name*, *bolt_type*, *length*) → *Element*

New bolt (stud or standoff). Parameter *bolt_type* specifies the type of bolt as a string. Currently there are only two: "GU30", a stud with 3mm thread, and "GO30", a standoff with 3mm inner thread. For available length, cf. Overview. The bolt is placed on the panel's reverse side. You can move it to the front by using *PutOnFrontSide()*.

Methods

Length () → *mm*

Type () → *bolt_type*

Returns respective parameter.

SetLength (length) → *Element*

SetType (bolt_type) → *Element*

Sets respective parameter.

4.12 Macros

Macros are specified by a hierarchical pathname as string, corresponding to the view "Macro objects" in the Frontpanel Designer. E. g., "Standard/BNC flange connector/Hole distance: 12,7mm, Drilling 2,8mm", or "User/Foo" for a user-created macro "Foo". The macro pathname must be exactly like the text in the list, including space characters. Language settings are not relevant.

LoadMacro (*path*) → *Element*

Load predefined or user macro element *path*.

4.13 Element Group

A group bundles elements as a unit. The group as a whole can then be treated like a primitive element.

Constructor

new Group (*name*) → *Element*

Create empty group of elements.

Methods

AddElement (*element*) → *Element*

Adds element *element* to group. Returns group.

FindElement (*name*) → *Element* | *false*

Finds element *name* in group.

Count () → *n*

Returns number of elements in group.

Elements () → *array of elements*

Returns an array with the group's elements.

5 Dialogs

The Frontpanel Designer provides the script programmer with dialog functions which allow the creation of simple dialog windows, where values can be queried from the user via graphical user interface elements. There are two methods for graphical elements: a simpler grid layout of values and their corresponding widgets, and a relatively free layout of elements in the dialog window. The latter is somewhat more complicated to program.

[Note: the simpler grid layout and the Grid element are not yet available.]

A dialog window is created by the *Dialog()* constructor. Then elements can be added. Finally, the dialog window will be displayed modally by the *Show()* function, that is, keyboard input is only permitted in this window. If a dialog button is pressed, the window is closed and *Show()* returns the value of that button. Additionally, if the window is closed externally, for example by pushing the titlebar close button, the value 0 is returned.

Initial values for graphical elements are passed using parameter objects, created by the function *Param()*. These parameters will be updated to the values changed by the user when the window is closed. The following example illustrates this. The dialog functions will be explained further below.

```
// For the TextEntry, we need a text parameter. This passes in
// the initial value and will receive the changed text.
var label_text = new Param("Hello, World!");

var d = new Dialog("Text Engraving");       // create the dialog

// Newlines and spacers arrange the layout...
d.NewlineC().VSpacer(10).NewlineC().d.Spacer(5)

// We have a label (Text) followed by a TextEntry editor...
d.Text("Engraving text:").VSpacer(5).NewlineC();
d.Spacer(5).TextEntry(1, label_text).Spacer(5);
d.Newline().VSpacer(0).NewlineC().Divider().NewlineC();

// And an "Ok" button, which, when pressed, will close
// the dialog, returning 1 from Show().
d.Spacer(0).DlgButton(1, "Ok!").Spacer(0);

// Show dialog, pressed button is returned but not used
// in this example.
var b = d.Show();

// Use the text as engraving string.
var te1 = new TextEngraving("textengraving1", label_text.get());
frontpanel.AddElement(te1, 10, 55);
```

This example shows creation of a string parameter, how it is used as second parameter of *TextEntry()* to pass in the initial string and how we obtain the result string from the parameter via *get()* after the dialog has been closed. Generally, each method's only got one *Param()* parameter: the one which also receives the return value. All other parameters are given directly.

As mentioned above, there exist two variants for querying values from the user. In the simplified variant, the programmer can create a *Grid* element that contains rows of a short description ("label"), the actual control element and an optional unit label. In the more complicated variant, the programmer may define the layout relatively freely. Layout of graphical elements is then controlled by a combination of vertical and horizontal containers. Vertical containers are "Boxes", horizontal ones are "Lines". A Box is ordering its elements from top to bottom, a Line from left to right. By nesting these two types of container, and with the use of spacers, a relatively flexible layout is possible.

Constructor

Dialog (*title* [, *width*, *height*]) → *Dialog*

Creates a new dialog window. After adding elements, the window is displayed by the *Show()* function. The optional parameters *width* and *height* specify the window's size. If they are omitted, default values will be used.

Methods

Box () → *Dialog*

BoxC () → *Dialog*

EndBox () → *Dialog*

Creates a new vertical box, a layout element containing lines. In every newly opened box, a *Newline()* or *NewlineC()* must open a new line, before other elements may be added to the box. *EndBox()* closes a box. *Box()* creates a box which is horizontally expanded in the surrounding line. *BoxC()* creates a box that only uses as much horizontal space as is necessary for its elements.

Newline () → *Dialog*

NewlineC () → *Dialog*

Ends the current line and opens a new one. *NewlineC()* creates a compact line, which only uses as much space vertically as is needed for its elements. Lines are terminated by the next *Newline()*/*NewlineC()* or by closing the surrounding box by *EndBox()*.

Text (*text*) → *Dialog*

Inserts static text at this position.

TextEntry (*#lines*, *text_param*) → *Dialog*

Creates a text entry field. If *#lines* = 1, a single-line entry field will be created, otherwise it is multi-line. The text parameter *text_param* passes in the initial text and receives the possibly changed text when the dialog is closed.

Slider (*value*, *min*, *max*, *fmt*) → *Dialog*

Creates a slider control. The parameter *value* is a numerical *Param* and passes in the initial value and receives the return value. The parameters *min* and *max* specify minimum and maximum values. The parameter *fmt* specifies a format string for formatting of the value

label. For example, in the format string "%f deg", the symbol %f would be replaced with the (floating-point-) slider value.

Selection (*strings*, *value*) → *Dialog*

Creates a selection list of strings. The strings are passed in via *strings*, separated by newlines (\n). The numerical parameter *value* is an index into the list and specifies the initial selection, and receives as return value the selected line.

SpinValue (*value*, *min*, *max*, *incr*) → *Dialog*

Creates a spin value control. The parameter *value* specified the initial value and receives the return value. The parameters *min* and *max* specify minimum and maximum values. The parameter *incr* specifies the step size.

DlgButton (*number*, *title*) → *Dialog*

Creates a push-button with title *title*. Pressing the button ends the dialog. The Dialog function *Show()* will then return the pressed button's *number*. DlgButtons are usually arranged in one line at the dialog window's bottom.

CheckBox (*value*, *label*) → *Dialog*

Creates a checkbox with labelling *label*. Parameter *value* specifies whether the checkbox is checked (1) or unchecked (0), and receives the selected value as return value.

RadioBox (*value*, *labels*) → *Dialog*

Creates a field of checkbox buttons. Only one of these buttons may be pressed. Each button is labelled, these labels are passed in via *labels*, seperated by newlines (\n). The first label is used as a title for the whole radiobox. The parameter *value* specifies the initial selection as numeric index, and receives the index of the checked button as return value.

Divider () → *Dialog*

Inserts a horizontal line spanning the current box width.

Spacer ([*size*]) → *Dialog*

VSpacer ([*size*]) → *Dialog*

Inserts an invisible horizontal (*Spacer*) or vertical (*VSpacer*) spacer of size *size*. If *size* = 0 or is not specified, maximum size will be used.

Show () → *button#*

Displays the dialog modally. Returns the number of the pushed DlgButton, or 0, if the window was closed by some other way.

6 Ordering Frontpanels

The following functions for communicating with the ordering program are available.

OrderAddFrontpanel (*frontpanel* [, *cnt*]) → *[undefined]*

OrderAddFrontpanelFile (*fpd-filename* [, *cnt*]) → *[undefined]*

Adds the *Frontpanel* object *frontpanel* or the frontpanel file *fpd-filename* *cnt* times (or 1) to the basket.

OrderAddItem (*id*, *cnt*) → *[undefined]*

Adds *cnt* items of article id *id* (string) to the basket.

OrderAddItemWithLength (*id*, *cnt*, *len*) → *[undefined]*

Adds *cnt* items of article id *id* and length *len* in mm to the basket.

OrderSetRemark (*remark*) → *[undefined]*

Sets the user remarks text in the ordering program (not in the frontpanel; for frontpanel manufacturing remarks, see *Frontpanel.SetRemark()*.)

7 Overview of Constants

Types of Material	
alu_elox	Aluminum anodized
alu_elox_chrom	Aluminum anodized, chromated
alu_raw	Aluminum raw
acryl	Acrylic
alu_powder_coated	Aluminum powder-coated

Thickness	
thick_1_5mm	*thick_5mm*
thick_2mm	*thick_6mm*
thick_2_5mm	*thick_8mm*
thick_3mm	*thick_10mm*

Anodizing Colors
elox_natural
elox_gold
elox_red
elox_blue
elox_green
elox_dark_bronze
elox_black
elox_medium_bronze

Acrylic Colors
acryl_clear
acryl_red

Powder Coating
pcoat_black
pcoat_traffic_white
pcoat_grey_white
pcoat_light_grey
pcoat_clay_brown
pcoat_signal_green
pcoat_turquoise_blue
pcoat_signal_blue
pcoat_signal_red
pcoat_sulfur_yellow
pcoat_signal_yellow
pcoat_alu_grey

Engraving Colors
engrave_no_color
engrave_jet_black
engrave_pure_white
engrave_signal_grey
engrave_signal_red
engrave_mint_green
engrave_sky_blue
engrave_gentian_blue
engrave_light_blue
engrave_luminous_yellow
engrave_pastel_orange
engrave_ochre_brown

Tools	
cutter_3_0mm	Cutter, 3mm
cutter_2_4mm	Cutter, 2.4mm
cutter_2_0mm	Cutter, 2mm
cutter_1_5mm	Cutter, 1.5mm
cutter_1_0mm	Cutter, 1mm
cutter_0_8mm	Cutter, 0.8mm, identical to *engraver_0_8mm*
cutter_0_6mm	Cutter, 0.6mm, identical to *engraver_0_6mm*
engraver_0_8mm	Engraver, 0.8mm
engraver_0_6mm	Engraver, 0.6mm
engraver_0_4mm	Engraver, 0.4mm
engraver_0_2mm	Engraver, 0.2mm
auto	Automatic selection

Text Alignment
align_left
align_center
align_right

Text vertical Alignment
text_center
text_baseline

D-Hole Forms	
dhole_single	(Single) D-Shape
dhole_double	Double D-Shape
dhole_quadruple	Quadruple D-Shape

Cavity Variants	
cavity_circular	Circular cavity
cavity_rectangular	Rectangular cavity
cavity_rect_in_rect	Rectangle in rectangle
cavity_circle_in_circle	Circle in circle
cavity_circle_in_rect	Circle in rectangle
cavity_rect_in_circle	Rectangle in circle

Alignment (Gravity)		
grav_northwest	*grav_north*	*grav_northeast*
grav_west	*grav_center*	*grav_east*
grav_southwest	*grav_south*	*grav_southeast*

Values for HPGL reference point	
hpgl_leftbottom	Lower left corner
hpgl_lefttop	Upper left corner
hpgl_center	Center point
hpgl_rightbottom	Lower right corner
hpgl_righttop	Upper right corner
hpgl_file	Take from file
hpgl_pen	From pen
hpgl_element	From element

Edge Machining	
bevel_none	No machining
bevel_30	30 deg. bevel
bevel_45	45 deg. bevel
bevel_60	60 deg. bevel
radius_0_5mm	0.5mm edge radius
radius_1_0mm	1.0mm edge radius
radius_1_5mm	1.5mm edge radius
radius_2_0mm	2.0mm edge radius
radius_2_5mm	2.5mm edge radius
radius_3_0mm	3.0mm edge radius

Threadings			
thread_M1_6	Metric-M1.6	*thread_M1_8*	Metric-M1.8
thread_M2	Metric-M2	*thread_M2_2*	Metric-M2.2
thread_M2_5	Metric-M2.5	*thread_M3*	Metric-M3
thread_M3_5	Metric-M3.5	*thread_M4*	Metric-M4
thread_M4_5	Metric-M4.5	*thread_M5*	Metric-M5
thread_M6	Metric-M6	*thread_M7*	Metric-M7
thread_M8	Metric-M8	*thread_M9*	Metric-M9
thread_M10	Metric-M10	*thread_MF2*	MetricFine-M2 × 0.2
thread_MF2_5	MetricFine-M2.5 × 0.25	*thread_MF3*	MetricFine-M3 × 0.35
thread_MF4	MetricFine-M4 × 0.5	*thread_MF5*	MetricFine-M5 × 0.5
thread_MF6	MetricFine-M6 × 0.75	*thread_MF8*	MetricFine-M8 × 0.75
thread_MF10	MetricFine-M10 × 1.0	*thread_UNC1*	UNC-#1-64
thread_UNC2	UNC-#2-56	*thread_UNC3*	UNC-#3-48
thread_UNC4	UNC-#4-40	*thread_UNC5*	UNC-#5-40
thread_UNC6	UNC-#6-32	*thread_UNC8*	UNC-#8-32
thread_UNC10	UNC-#10-24	*thread_UNC12*	UNC-#12-24
thread_UNC1_4	UNC-1/4”-20	*thread_UNC5_16*	UNC-5/16”-18
thread_UNC3_8	UNC-3/8”-16	*thread_UNF1*	UNF-#1-72
thread_UNF2	UNF-#2-64	*thread_UNF3*	UNF-#3-56
thread_UNF4	UNF-#4-48	*thread_UNF5*	UNF-#5-44
thread_UNF6	UNF-#6-40	*thread_UNF8*	UNF-#8-36
thread_UNF10	UNF-#10-32	*thread_UNF12*	UNF-#12-28
thread_UNF1_4	UNF-1/4”-28	*thread_UNF5_16*	UNF-5/16”-24
thread_UNF3_8	UNF-3/8”-24	*thread_UNF7_16*	UNF-7/16”-20
thread_UNF1_2	UNF-1/2”-20	*thread_UNF9_16*	UNF-9/16”-18
thread_UNF5_8	UNF-5/8”-18	*thread_UNF3_4*	UNF-3/4”-18
thread_UNEF12	UNEF-#12	*thread_UNEF1_4*	UNEF-1/4”
thread_UNEF5_16	UNEF-5/16”	*thread_UNEF3_8*	UNEF-3/8”
thread_UNEF7_16	UNEF-7/16”	*thread_UNEF1_2*	UNEF-1/2”
thread_UNEF9_16	UNEF-9/16”	*thread_UNEF5_8*	UNEF-5/8”
thread_UNEF11_16	UNEF-11/16”	*thread_UNEF3_4*	UNEF-3/4”
thread_UNEF13_16	UNEF-13/16”	*thread_UNEF7_8*	UNEF-7/8”
thread_UNEF15_16	UNEF-15/16”	*thread_UNEF1*	UNEF-1”
thread_UNEF1_1_16	UNEF-1-1/16”	*thread_UNEF1_1_8*	UNEF-1-1/8”
thread_UNEF1_3_16	UNEF-1-3/16”	*thread_UNEF1_1_4*	UNEF-1-1/4”
thread_UNEF1_5_16	UNEF-1-5/16”	*thread_UNEF1_3_8*	UNEF-1-3/8”
thread_UNEF1_7_16	UNEF-1-7/16”	*thread_UNEF1_1_2*	UNEF-1-1/2”
thread_UNEF_1_9_16	UNEF-1-9/16”	*thread_UNEF1_5_8*	UNEF-1-5/8”
thread_UNEF1_11_16	UNEF-1-11/16”		

Countersink			
sink_74A_M1	DIN74A-M1	*sink_74A_M1_2*	DIN74A-M1.2
sink_74A_M1_4	DIN74A-M1.4	*sink_74A_M1_6*	DIN74A-M1.6
sink_74A_M1_8	DIN74A-M1.8	*sink_74A_M2*	DIN74A-M2
sink_74A_M2_5	DIN74A-M2.5	*sink_74A_M3*	DIN74A-M3
sink_74A_M3_5	DIN74A-M3.5	*sink_74A_M4*	DIN74A-M4
sink_74A_M4_5	DIN74A-M4.5	*sink_74A_M5*	DIN74A-M5
sink_74A_M5_5	DIN74A-M5.5	*sink_74A_M6*	DIN74A-M6
sink_74A_M7	DIN74A-M7	*sink_74A_M8*	DIN74A-M8
sink_74A_M10	DIN74A-M10	*sink_74A_M12*	DIN74A-M12
sink_74A_M14	DIN74A-M14	*sink_74A_M16*	DIN74A-M16
sink_74A_M18	DIN74A-M18	*sink_74A_M20*	DIN74A-M20
sink_74B_M3	DIN74B-M3	*sink_74B_M4*	DIN74B-M4
sink_74B_M5	DIN74B-M5	*sink_74B_M6*	DIN74B-M6
sink_74B_M8	DIN74B-M8	*sink_74B_M10*	DIN74B-M10
sink_74B_M12	DIN74B-M12	*sink_74B_M14*	DIN74B-M14
sink_74B_M16	DIN74B-M16	*sink_74B_M18*	DIN74B-M18
sink_74B_M20	DIN74B-M20	*sink_74C_M2_2*	DIN74C-M2.2
sink_74C_M2_9	DIN74C-M2.9	*sink_74C_M3_5*	DIN74C-M3.5
sink_74C_M3_9	DIN74C-M3.9	*sink_74C_M4_2*	DIN74C-M4.2
sink_74C_M4_8	DIN74C-M4.8	*sink_74C_M5_5*	DIN74C-M5.5
sink_74C_M6_3	DIN74C-M6.3	*sink_661_M1*	DIN661-M1
sink_661_M1_2	DIN661-M1.2	*sink_661_M1_6*	DIN661-M1.6
sink_661_M2	DIN661-M2	*sink_661_M2_5*	DIN661-M2.5
sink_661_M3	DIN661-M3	*sink_661_M4*	DIN661-M4
sink_661_M5	DIN661-M5	*sink_661_M6*	DIN661-M6
sink_661_M8	DIN661-M8	*sink_A80_000*	ANSI80-#000
sink_A80_00	ANSI80-#00	*sink_A80_0*	ANSI80-#0
sink_A80_1	ANSI80-#1	*sink_A80_2*	ANSI80-#2
sink_A80_3	ANSI80-#3	*sink_A80_4*	ANSI80-#4
sink_A80_5	ANSI80-#5	*sink_A80_6*	ANSI80-#6
sink_A80_8	ANSI80-#8	*sink_A80_10*	ANSI80-#10
sink_A80_12	ANSI80-#12	*sink_A80_1_4*	ANSI80-1/4
sink_A80_5_16	ANSI80-5/16	*sink_A80_3_8*	ANSI80-3/8
sink_A80_7_16	ANSI80-7/16	*sink_A100_000*	ANSI100-#000
sink_A100_00	ANSI100-#00	*sink_A100_0*	ANSI100-#0
sink_A100_1	ANSI100-#1	*sink_A100_2*	ANSI100-#2
sink_A100_3	ANSI100-#3	*sink_A100_4*	ANSI100-#4
sink_A100_5	ANSI100-#5	*sink_A100_6*	ANSI100-#6
sink_A100_8	ANSI100-#8	*sink_A100_10*	ANSI100-#10
sink_A100_12	ANSI100-#12	*sink_A100_1_4*	ANSI100-1/4
sink_A100_5_16	ANSI100-5/16	*sink_A100_3_8*	ANSI100-3/8
sink_A100_7_16	ANSI100-7/16		

Fonts for fontspec	
arch-1stroke	Arch, 1-stroke
block-outline	Block, outline
bold-7stroke	Bold, 7-stroke
century-3stroke	Century, 3-stroke
century-outline	Century, outline
din1451-1stroke	DIN1451, 1-stroke
din17-1stroke	DIN17, 1-stroke
euro-5stroke	Euro, 5-stroke
euro-outline	Euro, outline
futura-outline	Futura, outline
gothic-3stroke	Gothic, 3-stroke
greek-1stroke	Greek, 1-stroke
helvetica-light-1stroke	Helvetica Light, 1-stroke
helvetica-bold-4stroke	Helvetica Bold, 4-stroke
helvetica-medium-outline	Helvetica Medium, outline
cyrillic-2stroke	Cyrillic, 2-stroke
script1-1stroke	Script 1, 1-stroke
script2-outline	Script 2, outline
script4-4stroke	Script 4, 4-stroke
script-3stroke	Script, 3-stroke
stencil-1stroke	Stencil, 1-stroke
univ-1stroke	Univ, 1-stroke
univ-outline	Univ, outline

Studs	
GU25	Glued-in stud, 2.5mm thread *Lengths (mm):* 6, 7, 8, 10, 12, 16, 20
GU30	Glued-in stud, 3mm thread *Lengths (mm):* 6, 8, 10, 12, 14, 16, 18, 20
GUC4	Glued-in stud, #4 thread *Lengths (mm):* 6.35, 7.938, 9.525, 11.113, 12.7, 15.875, 19.05
GUC6	Glued-in stud, #6 thread *Lengths (mm):* 6.35, 7.938, 9.525, 11.113, 12.7, 15.875, 19.05

Standoff	
GO25	Glued-in standoff, 2.5mm thread *Lengths (mm):* 4, 5, 6, 8, 10, 12, 16, 20
GO30	Glued-in standoff, 3mm thread *Lengths (mm):* 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20
GOC4	Glued-in standoff, #4 thread *Lengths (mm):* 6.35, 7.938, 9.525, 11.113, 12.7, 15.875, 19.05
GO30	Glued-in standoff, #6 thread *Lengths (mm):* 6.35, 7.938, 9.525, 11.113, 12.7, 15.875, 19.05