abs

ABS returns the absolute value of x. Syntax ABS (x) Parameter x: if x is an integer, the return value is also an integer. Otherwise the return value is decimal.

acos

ACOS returns the arc cosine of x in degrees. Syntax ACOS (x) Parameters x (decimal): argument for the arc cosine parameter restrictions: -1,0 <= x <= 1,0 Remarks Possible return values: 0 degrees <= ACOS (x) <= 180 degrees

all

ALL can be used in the following commands: RESTORE ALL resets all current transformations. PLACE2 ALL inserts all objects; the current transformations are considered. PLACE ALL inserts all objects; the current transformations are considered. Syntax RESTORE ALL PLACE2 ALL, use_current_attribute_flag

arc

ARC generates a circular arc in the XY plane with its center situated in the coordinate system's origin; it is defined by its radius and a central angle from alpha to beta. Syntax ARC r, alpha, beta Parameters r (decimal): radius alpha (decimal): angle (in degrees) beta (decimal): angle (in degrees) Remarks The angular offset is based on the (local) X axis; positive direction is counterclockwise. Alpha and beta are in degrees.

arc2

ARC2 creates a circular arc based on center point, radius and central angle from alpha to beta. Syntax ARC2 x, y, r, alpha, beta Parameters x (decimal): X coordinate of center point y (decimal): Y coordinate of center point r (decimal): radius length alpha (decimal): angle (in degrees) beta (decimal): angle (in degrees)   Remarks The angular offset is based on the (local) X axis; positive direction is counterclockwise. Alpha and beta are in degrees.

armc

ARMC creates a cylinder originating from another cylinder. One end of the new cylinder is straight, the other end is smoothed to the cylinder it originates from. Syntax ARMC r1, r2, l, h, d, alpha Parameters r1 (decimal): radius of the given cylinder from which the new cylinder originates r2 (decimal): radius of the new cylinder l (decimal): length of new cylinder h (decimal): distance in Z direction for new cylinder d (decimal): distance in Y direction for new cylinder alpha (decimal): angle of inclination to Z axis, in degrees Restrictions of parameters: r1 <= r2 + d r1 >= l*sin(alpha) - r2*cos(alpha) Remarks The new cylinder is not a solid, therefore no penetration will be calculated. The curves, however, are calculated and drawn.

arme

ARME creates a right cylinder originating from an ellipsoid in the YZ plane. Its upper end is straight and parallel to the XY plane. Syntax ARME l, r1, r2, h, d Parameters l (decimal): overall length of new cylinder (based on YZ plane) r1 (decimal): radius of the given ellipsoid from which the new cylinder originates r2 (decimal): radius of the new cylinder h (decimal): height of ellipsoid d (decimal): distance from Z axis in Y direction for new cylinders center point Restrictions of parameters: r1 <= r2 + d l <= h*sqr(1-(r2-d)2/r12) Remarks The new cylinder is not a solid, therefore no penetration will be calculated. The curves, however, are calculated and drawn.

asin

ASIN returns the arc sine of x in degrees. Syntax ASIN (x) Parameters x (decimal): argument for the arc sine parameter restrictions: -1,0 <= x <= 1,0 Remarks Possible return values: -90 degrees <= ASIN (x) <= 90 degrees

atan

ATAN returns the arc tangent of x in degrees. Syntax ATAN (x) Parameters x (decimal): argument for the arc tangent Remarks Possible return values: -90 degrees <= ATAN (x) <= 90 degrees

base

BASE resets the counters for the low-level geometric elements VERT, EDGE and PGON. The command is implicitly issued after every compound element definition, therefore it is not necessary after the command BODY. Syntax BASE

bitmap

DEFINE BITMAP generates a reference to a bitmap, for later use as filling of 2D polylines (bitmap areas). Bitmap definitions prior to the first reference to that bitmap may be included in any script; however, bitmaps defined that way can be used solely in those scripts, in which they are defined, and their subsequent second generation-scripts. Syntax DEFINE BITMAP name, image [, angle, scale_x, scale_y, ref_x, ref_y, offset_x, offset_y, metric, tile, mask, red, green, blue, toler] Parameters name (string): name of the bitmap image (string): expression containing the base name of the bitmap file without extension, e.g. "name" for "name.png" angle (decimal): rotation angle (default: 0.0) scale_x, scale_y (decimal): scaling of the bitmap in X resp. Y direction (default: 1.0) ref_x, ref_y (decimal): X resp. Y coordinate of the reference point (default: (0.0;0.0)) offset_x, offset_y (decimal): X resp. Y coordinate of the offset point (default: (0.0;0.0)) metric (integer): adjustment of bitmap size with metric may be 0 or 1 0: fit size 1: one pixel equals 1 cm tile (integer): arrangement of the bitmap with tile may be 0 or 1 0: fit size 1: tile bitmap mask (integer): masking of transparent bitmap areas with mask may be 0 or 1 0: no transparent areas 1: mask black pixels with defined color red, green, blue (integer): color palette for masking/background color (default: white) with 0 <= red, green resp. blue <= 256 toler (integer): tolerance of the transparent color with 0 <= toler <= 256 (default: 0) Corresponding commands [set] BITMAP name_string [set] BITMAP index Index is a constant referring to a bitmap stack in the internal data structure; this stack is modified during script analysis and can also be modified from within the program. Any use of the index instead of the bitmap's name is only recommended with the prior use of the INDEX function.

bitset

BITSET sets the b bit of x to 1 or 0 depending on the value of the specified expression, and returns the result.   Syntax BITSET (x, b [, expr]) Parameters x (integer): value b (integer, >= 0): place expr (binary): 0 or 1 Remarks Parameter value should be integer, returned value is integer.

bittest

BITTEST returns 1 if the b bit of x is set, 0 otherwise. Syntax BITTEST (x, b) Parameters x (integer): value b (integer, >= 0): place

body

BODY generates a body consisting of the above defined primitives. Syntax BODY status Parameters status (integer): form and behavior of the body with status may be 1: closed body 4: surface model (in case of cutting, cut surface shows no surface) -1: automatic (status of the body will be calculated by the engine) 128: wire frame 256: invert orientation body_b

BODY_B creates a NURBS cube of Bezier type according to the passed list of control points. Syntax BODY_B m, n, subdiv_n, subdiv_m, rise, x11, y11, ... z11, x12, y12, ... z12, ... x1m, y1m, ... z1m, x21, y21, ... z21, ... zn1, zn2, ... znm Parameters m (integer, >= 2): count of points at each edge curve n (integer, 12): count of curves n must be 12 - but other values produce no errors subdiv_n, subdiv_m (integer): the desired subdivision of each face in two directions rise (decimal): Sets smoothness. The error of the arc approximation (i.e., the greatest distance between the theoretical curve and the generated chord) will be smaller than rise. xnm, ynm, znm (decimal): coordinates of points defining the Bezier curved edges of the cube Remarks The order of edges must be arranged in sequence as depicted in the diagram.

box

BOX creates a rectangular solid. Syntax BOX a, b, c Parameters a (decimal): length along x axis, a >= 0 b (decimal): length along y axis, b >= 0 c (decimal): length along z axis, c >= 0 Remarks The first vertex of the rectangular solid is in the local origin. The edges with the lengths a, b, and c follow the X, Y and Z axes. To create a rectangle or a line, set a, b and/or c to zero.

break

BREAK terminates the execution of the current loop. The script continues with the next command after the loop. BREAK may only be used inside a loop, such as a FOR ... NEXT statement. Syntax BREAK

breakpoint

BREAKPOINT halts the execution of the current script in SmartParts editor. In normal execution mode all breakpoints are ignored. Syntax BREAKPOINT expression Parameter expression (integer): the debugger will stop at this command, if the value of the parameter (a numeric expression) is true (#0).

call

CALL calls another SmartParts object (subscript). The called subscript must be located in \\STD\\SmartParts folder and is called only by its name without extension. The name of the subscript may be a constant or a variable. You can pass values to the called subscript. Syntax CALL subscript_name [,parameter_list] CALL subscript_name PARAMETERS [name1=value1 , ... namen=valuen] CALL subscript_name PARAMETERS ALL Parameter subscript_name: name of the subscript without extension. The name must not contain blanks. Subscript names can be string constants, string variables or parameters. String operations cannot be used with a subscript call as a subscript name. The subscript name must be put between quotation marks ("), unless it matches the definition of identifiers, i.e., it begins with a letter. Otherwise, the quotation marks used in the CALL statement must be the same at the beginning and at the end, and should be different from any character of the subscript name. PARAMETERS [name1=value1 , ... namen=valuen]: here you can list the parameters of the subscript in any order and assign values to them. After the PARAMETERS keyword you need to list the parameter names of the called subscript in any sequence, with both an "=" sign and a value for each. You can use string type expressions here, but only give a string value to string type parameters of the called subscript. Array parameters have to be given full array values. Parameters of the called subscript that are not listed in the subscript call will be given their original default values as defined in the library part called as a subscript. For a parameter which cannot be found in the called subscript, the default value will be used. A subscript has its own environment which depends on its calling order. The current values of the MODEL, RADIUS, RESOL, RISE, PEN, STROKE, MATERIAL, FILL, STYLE, COLOR ... options and the current transformation are all valid in the subscript. You can use or modify them, but the modifications will only have an effect locally. They do not take effect on the level the subscript was called from. Giving parameters to a subscript call means an implicit value assignment on the subscript's level. Remarks Subscript names must not be localized.

ceil

CEIL returns the smallest following integer. Syntax CEIL (x) Parameter x: If x is an integer, the return value is x. If x is a decimal, the return value is the smallest integer following x. Negative values of x are rounded up. CEIL (2) = 2 CEIL (2,34) = 3 CEIL (-1,95) = -1

circle
CIRCLE generates a full circle in the XY plane with its center situated in the origin of the coordinate system; it is defined solely by its radius. Syntax CIRCLE r Parameters r (decimal): radius

circle2
CIRCLE2 creates a full circle based on its center point and radius. Syntax CIRCLE2 x, y, r Parameters x (decimal): X coordinate of center point y (decimal): Y coordinate of center point r (decimal): radius length

color
COLOR defines the color of an element. Syntax [SET] COLOR id Parameters id (integer, -2 < id <= 255): identification number of the color id = -1 (default): the current color is used id = -2: the color is set "by Layer" You can also use constants instead: CURR = -1 BY_LAYER = -2

cone
CONE creates a cone or a frustum of a cone. Syntax CONE h, r1, r2, alpha1, alpha2 Parameters h (decimal): height along z axis, h >= 0 r1 (decimal): radius of lower circle, r1 >= 0 r2 (decimal): radius of upper circle, r2 >= 0 alpha1 (decimal): angle of lower surface to z axis, 0 < alpha1 < 180 alpha2 (decimal): angle of upper surface to z axis, 0 < alpha1 < 180

continue
CONTINUE terminates the current iteration of a loop. The script proceeds with the next iteration of the loop. CONTINUE may only be used inside a loop, such as a FOR ... NEXT statement. Syntax CONTINUE

cos
COS returns the cosine of x. Syntax COS (x) Parameters x (decimal): argument for the cosine (in degrees!) Remarks Possible return values: -1 <= COS (x) <= 1

curve_b
CURVE_B creates a Bezier curve with n control points in 3D. Syntax CURVE_B n, status, x1, y1, z1, t_next_x1, t_next_y1, t_next_z1, ... t_prev_xn, t_prev_yn, t_prev_zn, xn, yn, zn, t_next_xn, t_next_yn, t_next_zn Parameters n (integer, <= 2): number of control points status: defines visibility of the curve, 0 - standard xi, yi, zi (real): control point coordinates t_prev_xi, t_prev_yi, t_prev_zi (real): previous tangent point coordinates t_next_xi, t_next_yi, t_next_zi (real): next tangent point coordinates First point has only next tangent point coordinates!

curve_b2
CURVE_B2 creates a Bezier curve with n control points in 2D. Syntax CURVE_B2 n, status, x1, y1, t_next_x1, t_next_y1, ... t_prev_xn, t_prev_yn, xn, yn, t_next_xn, t_next_yn   Parameters n (integer, <= 2): number of control points status (integer, 0 or 1): defines the visibility of the curve, 0 - standard status= j1 + 2*j2 where j1, j2 can be 0 or 1. j1 (1): contour only j2 (2): fill only xi, yi (real): control point coordinates t_prev_xi, t_prev_yi (real): previous tangent point coordinates t_next_xi, t_next_yi (real): next tangent point coordinates First point has only next tangent point coordinates!

curve_n
CURVE_N creates a NURBS curve with n control points in 3D. Syntax CURVE_N n, status, x1, y1, z1, ... xn, yn, zn Parameters n (integer, <= 2): number of control points status (integer, 0 or 1): defines curve's appearance 0 - standard 1 - the last and first nodes of the curve will be connected, thus closing the curve xi, yi, zi (real): control point coordinates

curve_n2
CURVE_N2 creates a NURBS curve with n control points in 2D. Syntax CURVE_N2 n, status, x1, y1, ... xn, yn   Parameters n (integer, <= 2): number of control points status (integer, 0 or 1): defines curve's appearance status= j1 + 2*j2 + 4*j3 where j1, j2, j3 can be 0 or 1. j1 (1): contour only j2 (2): fill only j3 (4): the last and first nodes of the curve will be connected, thus closing the curve. xi, yi, (decimal): control point coordinates

cutend
CUTEND is to be inserted at the end of any cutting operation in order to finish it. Syntax CUTEND Remarks If CUTEND is missing, cutting operations affect all elements until the end of the script. If CUTEND is missing in subscripts, the cutting operation ends with the end of the script.

cutform
CUTFORM is similar to CUTPOLYA. In addition, you can define form and extent of the cutting body. Syntax CUTFORM n, method, status, rx, ry, rz, d, x1, y1, mask1, ... xn, yn, maskn CUTEND Parameters n (integer, >3): number of points method (integer, 0 < method < 4): defines the shape of the cutting body 1: prism-shaped 2: pyramid-shaped 3: wedge-shaped The wedge's top edge is parallel to the Y axis, a status (integer): defines the extent of the cutting body status = 8*j4 + 16*j5 + 32*j6 j4, j5: define the limit of the cut: j4 = 0 and j5 = 0:finite cut j4 = 0 and j5 = 1:semi-infinite cut j4 = 1 and j5 = 1:infinite cut j6: generate a boolean intersection with the cutting body rather than a boolean difference (only for CUTFORM) rx, ry, rz (decimal): coordinates of the directional vector for cutting form (prism-shaped) or the height (pyramid-shaped) d (decimal): distance to the end of the cut along directional vector. For infinite cuts: parameter d has no effect For finite cuts: cutting body begins in the local coordinate system and ends in distance d along the directional vector For semi-infinite cuts: cutting body begins in distance d along the directional vector, and the cut will be in the opposite direction of the directional vector. xi, yi (decimal): coordinates of cutting form in XY plane. maski (integer): defines the visibility of the edges of the cutting body (not supported)

cutplane
CUTPLANE creates a cutting plane in 3-dimensional space. All 3D elements in its area are cut, the cut parts are removed. A different number of parameters is possible. CUTPLANE must be finished with CUTEND. Syntax CUTPLANE [x, y, z [, side]] [statement1 ... statementn] CUTEND or CUTPLANE angle [statement1 ... statementn] CUTEND Parameters x, y, z (decimal): coordinates defining the plane, max. one coordinate = 0 side (boolean): cutting side statement (integer, 0 <= statement <= 3): define behavior of cutting edges and areas Number of statements and their effects: 0: cutting plane is XY plane 1: cutting plane across X axis, angle between cutting plane and XY plane 2: cutting plane parallel to Z axis, intersects X and Y axis at given values 3: cutting plane intersects X, Y and Z axis at given values 4: side value is added: side = 0: parts above cutting plane are cut off (default); side = 1: parts below cutting plane are cut off; in case of XY, XZ, YZ planes, the parts in the negative direction of the axis are cut off. angle (decimal): rotation angle of cutting plane Remarks Without side parameter, the elements are cut above the cutting plane. If CUTEND is missing, CUTPLANE affects all elements until the end of the script. CUTPLANE commands can be included in subscripts. Only the shapes in the subscript are affected. CUTPLANE parameters refer to the current coordinate system. Transformations must be stated before in order to be effective. CUTPLANE - CUTEND pairs can be nested. If there are current material, pen and fill settings, they will be used for the cut surfaces.

cutplane_2

CUTPLANE_2 is similar to CUTPLANE. In addition, you can define a status parameter for the cutting edges and faces, when the angle parameter is used. Syntax CUTPLANE_2 angle [, status] [...] CUTEND Parameters angle (decimal): rotation angle of cutting plane Remarks Without side parameter, the elements are cut above the cutting plane. If CUTEND is missing, CUTPLANE_2 affects all elements until the end of the script. CUTPLANE_2 commands can be included in subscripts. Only the shapes in the subscript are affected. CUTPLANE_2 parameters refer to the current coordinate system. Transformations must be stated before in order to be effective. CUTPLANE_2 - CUTEND pairs can be nested. If there are current material, pen and fill settings, they will be used for the cut surfaces.

cutplane_3

CUTPLANE_3 is similar to CUTPLANE and creates a cutting plane in 3-dimensional space. All 3D elements in its area are cut, the cut parts are removed. A different number of parameters is possible. CUTPLANE_3 must be finished with CUTEND.   Syntax CUTPLANE_3 [x [, y,[z [, side [, status]]]]] [statement1 ... statementn] CUTEND or CUTPLANE angle [statement1 ... statementn] CUTEND Parameters x, y, z (decimal): coordinates defining the plane, max. one coordinate = 0 side (boolean): cutting side statement (integer, 0 <= statement <= 3): define behavior of cutting edges and areas Number of statements and their effects: 0: cutting plane is XY plane 1: cutting plane across X axis, angle between Cutting plane and XY plane 2: cutting plane parallel to Z axis, intersects X and Y axis at given values 3: cutting plane intersects X, Y and Z axis at given values 4: side value is added: side = 0: parts above cutting plane are cut off (default); side = 1: parts below cutting plane are cut off; in case of XY, XZ, YZ planes, the parts in the negative direction of the axis are cut off. angle (decimal): rotation angle of cutting plane Remarks It is advisable to use CUTPLANE_3 instead of CUTPLANE, because if you use CUTPLANE with more than 4 parameters, the 4th parameter will be omitted.

cutpoly

CUTPOLY is similar to CUTPLANE and defines a cutting polygon. All elements within an infinite "tube" above the cutting polygon are cut. The CUTPOLY parameters refer to the current coordinate system. Syntax CUTPOLY n, x1, y1, ... xn, yn [, x, y, z] [statement1 statement2 ... statementn] CUTEND Parameters n (decimal): number of points xi, yi (decimal): coordinates in XY plane x, y, z (decimal): coordinates of optional vector statement (integer, 0 <= statement <= 3): define behavior of cutting edges and areas Number of statements and their effects: 0: cutting plane is XY plane 1: cutting plane across X axis, angle between Cutting plane and XY plane 2: cutting plane parallel to Z axis, intersects X and Y axis at given values 3: cutting plane intersects X, Y and Z axis at given values 4: side value is added: side = 0: parts above cutting plane are cut off (default); side = 1: parts below cutting plane are cut off; in case of XY, XZ, YZ planes, the parts in the negative direction of the axis are cut off. Remarks Polygon must be convex; its sides must not intersect. Cutting is along Z axis rsp. along the optional vector.

cutpolya

CUTPOLYA is similar to CUTPOLY and creates a cutting tube, too. The tube needs not to be infinite, it can be closed on one side.   Syntax CUTPOLYA n, status, d, x1, y1, mask1, ... xn, yn, maskn [, x, y, z] [statement1 statement2 ... statementn] CUTEND Parameters n (decimal): number of points status (integer): defines how the cut polygons are handled (not supported) d (decimal): end of cutting tube xi, yi (decimal): coordinates in XY plane maski (integer): defines how the cutting tube's edges affect the cut elements x, y, z (decimal): coordinates of optional vector statement (integer, 0 <= statement <= 3): define behavior of cutting edges and areas Number of statements and their effects: 0: cutting plane is XY plane 1: cutting plane across X axis, angle between cutting plane and XY plane 2: cutting plane parallel to Z axis, intersects X and Y axis at given values 3: cutting plane intersects X, Y and Z axis at given values 4: side value is added: side = 0: parts above cutting plane are cut off (default); side = 1: parts below cutting plane are cut off; in case of XY, XZ, YZ planes, the parts in the negative direction of the axis are cut off. Remarks If the cutting tube intersects a solid or body partially or completely, the corresponding part is cut out.

cutshape

CUTSHAPE defines a cutting shape and is similar to CUTPLANE. Syntax CUTSHAPE d [statement1 statement2 ... statementn] CUTEND Parameters d (decimal): defines the form of the cutting shape d=0: cutting shape is the XY plane; parts above XY plane are cut off. d>0: L-shaped cut; parts above XY plane with X <= 0 are cut off. d<0: U-shaped cut; parts above XY plane with 0 >= X >= 0 are cut off. statement (integer, 0 <= statement <= 3): define behavior of cutting edges and areas Number of statements and their effects: 0: cutting plane is XY plane 1: cutting plane across X axis, angle between cutting plane and XY plane 2: cutting plane parallel to Z axis, intersects X and Y axis at given values 3: cutting plane intersects X, Y and Z axis at given values 4: side value is added: side = 0: parts above cutting plane are cut off (default); side = 1: parts below cutting plane are cut off; in case of XY, XZ, YZ planes, the parts in the negative direction of the axis are cut off.

cylind

CYLIND creates a right cylinder centered in Z axis. The center of the circle is in the point of origin. Syntax CYLIND h, r Parameters h (decimal): height along z axis r (decimal): radius Remarks To create a line along Z axis, set r=0 To create a circle in the XY plane, set h=0

dim

DIM declares one or more variables as 1- or 2-dimensional array. If no dimensions are specified, the size of the array is dynamic and corresponds to the largest referenced index within the script. You can specify any numbers of array variables within one DIM statement. Syntax DIM var1[dim_1], var2[dim_1][dim_2], var3[ ], var4[ ][ ], var5[dim_1][ ], var5[ ][dim_2] Parameter var1, var2: name of the array. dim1, dim2: dimensions of the array. If they are not specified, the array is specified as dynamic (one or both dimensions). Variables cannot be used as dimensions. Referencing the elements of the array in a script Parameter arrays are dynamic by default, and therefore it is not necessary to declared them in the script. When the parameter array is defined, you can reference the elements of the array anywhere in the script but if they are variables, only after the declaration. var1[num_expr] or var1 var2[num_expr1][num_expr2] or var2[num_expr1] or var2 When referencing the array name without an index you reference the whole array (or a line of a two-dimensional array) which is accepted in some cases (CALL, PRINT, LET, PUT, REQUEST, INPUT, OUTPUT, SPLIT statements). For dynamic arrays there is no limitation for the actual index value. During the interpretation, when a non-existing dynamic array element is given a value, the necessary quantity of memory is allocated and the missing elements are all set to 0 (numerical). Warning: With dynamical arrays, the size of the array during the execution of the script is not limited and may cause an out of memory error in some cases. Each index -regardless of its value - is considered valid. A non-existing dynamic array element is 0 (numerical). Arrays, which have a fixed dimension, are checked for the validity of the actual index on the fixed dimension. Fixed arrays cannot be given whole dynamic array values. However, dynamic arrays that are given whole array values will take on those values. This also applies to some statements where whole array references can be used as return parameters (REQUEST, INPUT, SPLIT). Array elements can be used in any numerical or string expression. They can be given string or numerical values. Indices start with 1, and any numerical expression can be used as an index. Array elements can be of different simple types (numerical, string). The type of the whole array (main type) is the type of its first element ([1] or [1][1]). Parameter and global variable arrays cannot be of mixed type.

do

DO starts a DO ... WHILE loop. Syntax DO ... WHILE condition Parameters condition: Remarks Commands between DO ... WHILE are executed, as long as condition = true. condition is checked after the loop, so the commands between DO ... WHILE are executed at least one time, even if condition <> true.

draw_seq

DRAW_SEQ sets the order, in which elements defined within the 2D-Scripts are drawn. Elements with the lowest numbers will be the first to be drawn. Syntax DRAW_SEQ number Parameters number (integer): drawing number of the element parameter restrictions: 0 < number <= 50 default: - nIndex = -1 (use current) Remarks If no drawing sequence is set, the drawing order is as follows: 25 text 15 normal elements 12 hatching, patterns 11 face styles 10 bitmaps 8 fillings

edge

EDGE generates an edge referring to prior to this via the command VERT defined nodes. Syntax EDGE vert1, vert2 Parameters vert1, vert2: (integer): indexes of the starting point and the end point, referenced to previously defined nodes; starting point and end point must be different.

elbow

ELBOW creates a segmented, elbow-shaped tube in the XZ plane. Syntax ELBOW r1, alpha, r2 Parameters r1 (decimal): radius of arc alpha (decimal): angle of arc r2 (decimal): radius of tube segment, in degrees Restrictions of parameters: r1 < r2

ellips

ELLIPS creates a half ellipsoid.   Syntax ELLIPS h, r Parameters h (decimal): half axis along Z axis r (decimal): radius in origin   Remarks The ellipsoid's section in the XY plane is a circle centered in the origin.

else

ELSE is part of an IF...THEN...ELSE...ENDIF construction. ELSE extends an IF statement to execute a statement in case the expression in the IF statement evaluates to FALSE.   Syntax IF condition THEN command IF condition THEN command1 command2 ELSE ENDIF Parameters condition: if condition=true (<>0) all commands after THEN are executed. Otherwise the commands after ELSE are executed. If ELSE is absent, the commands after ENDIF are executed. command: any SmartPart Script command Remarks If the command after THEN is GOTO or GOSUB, you can omit the THEN statement and shorten to IF condition GOTO label. If there is only 1 command after THEN or ELSE in the same row, you can omit ENDIF.

end

END determines the end of the current script, even if there are commands after it. The program either terminates or returns 1 level above. Syntax END [expression1, ..., expressionn] Parameters expression (String or Decimal, depending on the variable in the calling script): passes return values to the calling script. Remarks You can have more than 1 END statements in a file. Never use an END or EXIT statement in Master-Script, because then the other parts will not be executed.

endif

ENDIF ends a IF ... ELSE ... ENDIF construction. Syntax IF condition THEN command IF condition THEN command1 command2 ELSE ENDIF Parameters condition: if condition=true (<>0) all commands after THEN are executed. Otherwise the commands after ELSE are executed. If ELSE is absent, the commands after ENDIF are executed. command: any SmartPart Script command Remarks If the command after THEN is GOTO or GOSUB, you can omit the THEN statement and shorten to IF condition GOTO label. If there is only 1 command after THEN or ELSE in the same row, you can omit ENDIF.

endwhile

END ends a WHILE ... DO ... ENDWHILE loop. Syntax WHILE condition DO ... ENDWHILE Parameters condition: Remarks Commands between WHILE ... ENDWHILE are executed, as long as condition = true. condition is checked before the loop, so the commands between WHILE ... ENDWHILE are executed only, if condition = true at the beginning of the loop.

exit

EXIT determines the end of the current script, even if there are commands after it. The program either terminates or returns 1 level above. Syntax EXIT [expression1, ..., expressionn] Parameters expression (String or Decimal, depending on the variable in the calling script): passes return values to the calling script. Remarks You can have more than 1 EXIT statements in a file. Never use an END or EXIT statement in Master-Script, because then the other parts will not be executed.

exp

EXP returns x th power of e (e = 2.7182818). Syntax EXP (x) Parameters x (decimal): exponent to e

extrude

EXTRUDE generates a general prism with its base defined by a polyline in the XY plane; the displacement vector between the bases is (dx, dy, dz). This command is a generalization of the commands PRISM and SOLID_SLAB; the base polyline is not necessarily closed, as the lateral edges are not always perpendicular to the XY plane. Arcs and segments can be defined within the polyline by using additional status code values. Tipp: The base polyline may include holes, just like PRISM_. Syntax EXTRUDE n, dx, dy, dz, mask, x1, y1, s1, ... , xn, yn, sn Parameters n (integer): number of polyline nodes dx, dy, dz (decimal): displacement vector in X, Y and Z direction mask: controls the existence of the bottom, top and - in the case of an open polyline - side surfaces (see below) xi, yi (decimal): X and Y coordinates of the polyline nodes si (boolean): marks the end of the polygon or hole parameter restrictions: n > 2 Masking mask = j1 + 2*j2 + 4*j3 with j1, j2, j3 may be 0 or 1 j1 (1): bottom surface is present j2 (2): top surface is present j3 (4): side (closing) surface is present Status Values -1: marks the end of the enclosing polygon or a hole, and means that the next node will be the first vertex of another hole.

face_style

DEFINE FACE_STYLE defines a facestyle, which can be referred to by name further on in other scripts. Facestyle definitions prior to the definition of this facestyle may be included in any script; however, facestyles defined that way can be used solely in those scripts, in which they are defined, and their subsequent second generation-scripts. Syntax DEFINE FACE_STYLE name, id [, angle, ref_x, ref_y] Parameters name (string): name of the facestyle, by which can be referred to this style id (integer): number of the facestyle angle (decimal): rotation angle (default: 0.0) ref_x, ref_y (decimal): X resp. Y coordinate of the reference point (default: (0.0;0.0)) Corresponding commands [SET] FACE_STYLE name_string [SET] FACE_STYLE index Index is a constant referring to a facestyle stack in the internal data structure; this stack is modified during script analysis and can also be modified from within the program. Any use of the index instead of the facestyle's name is only recommended with the prior use of the INDEX function. INDEX (FACE_STYLE, "Name")

fill

DEFINE FILL defines a fill color, which can be referred to by name further on in other scripts. Fill color definitions prior to the definition of this fill may be included in any script; however, patterns defined that way can be used solely in those scripts, in which they are defined, and their subsequent second generation-scripts. Syntax DEFINE FILL name, 0, color_id [, alpha1] DEFINE FILL name, 1, red1, green1, blue1, alpha1 [, shading ,variant, angle] DEFINE FILL name, 2, red1, green1, blue1, alpha1, shading ,variant, angle, red2, green2, blue2, alpha2 Parameters name (string): name of the fill fill type (integer, 0, 1, 2): defines the fill type. The number of possible parameters depends on the fill type. 0: plain fill color 1: single color with shading and angle 2: two colors with shading, gradient and angle color_id (integer): number of first color with 0 <= color_id <= 255 red1, green1, blue1, alpha1 (integer): color parameters of first color with 0 <= red1, green1, blue1 <= 255 0 <= red2, green2, blue2 <= 255 0 <= alpha1 <= 100 0 <= alpha2 <=

100 shading (integer): type of shading shading=0: Linear transition shading=1: transition from corner shading=2: transition from center shading=3: round transition variant (integer): type of gradient variant=0: gradient from color 1 to color 2 variant=0: gradient from color 2 to color 1 variant=0: gradient from color 1 to color 2 and back variant=0: gradient from color 2 to color 1 and back angle (integer): tilt angle of shading or gradient Corresponding commands [set] FILL name_string [set] FILL index All 2D polygons generated after a fill setting will obtain this fill color, until the next fill is set. Index is a constant referring to a fill stack in the internal data structure; this stack is modified during script analysis and can also be modified from within the program. Any use of the index instead of the fill's name is only recommended with the prior use of the INDEX function. Remarks If there is no fill set in a script, an empty fill rsp. no fill is used as default: SET FILL -1

foil

DEFINE FOIL defines a foil (corresponding to a foil of an <Product> smart symbol). Syntax DEFINE FOIL name visible2D, visible3D, from_scale, to_scale, refPnt1X, refPnt1y, refPnt1z, refPnt2X, refPnt2y, refPnt2z, layerA, layerB, layerC, scaleX, scaleY, scaleZ Parameter name (string): name of the foil, must be enclosed in quotes visible2D, visible3D (boolean): 0 (invisiblor 1 (visible) in 2D or 3D view from_scale: content of foil is only visible, if current scale is greater than from_scale to_scale: content of foil is visible, if current scale is less than or equal to to_scale refPnt1x, refPnt1y, refPnt1z (decimal): X, Y, Z of Refpoint1 refPnt2x, refPnt2y, refPnt2z (decimal): X, Y, Z of Refpoint2 layerA, layerB, layerC (boolean): 0 (invisiblor 1 (visible) for layer scaleX, scaleY, scaleZ: scale dependency for direction X, Y, Z 0=X , 1=Y, 2=Z Remarks The foil can be used only in the script in which it was defined and its subsequent second generation scripts. After the definition of the foil you can activate the foil with SET FOIL, all objects created afterwards in the script will be placed on this foil. The foil can be specified either by its name or by its index number, with the following syntax: [SET] FOIL name_of_foil: name of the foil enclosed in quotes [SET] FOIL index: The index is a constant referring to a foil stack in the internal data structure. This stack is modified during script analysis and can also be modified from within the program. The use of the index instead of the foil name is only recommended with the prior use of the INDEX function. With SET FOIL 0 you can reset the script to the automatic foil creation, where 2D objects and 3D objects are placed on different foils and are visible in the scale range from 1:1 to 1:1000.

for

FOR starts a FOR...NEXT loop. Syntax FOR variable=start_value TO end_value [STEP step_value] ... NEXT Parameters variable: name of the variable. Only local variables are allowed, no global variables. start_value (decimal): initial value of the variable end_value (decimal): value, till which the loop is executed. Before a loop is started, it is checked, whether end_value is reached. step_value (decimal): value by which start_value is increased with every cycle of the loop. If not specified, start_value is increased by 1. step_value can be negative also. A change of step_value inside the loop is ignored.

fra

FRA returns the difference of x to the next smaller integer. Syntax FRA (x) Parameter x: if x is an integer, the return value is also an integer. Otherwise the return value is decimal. FRA (3.45) = 0.45 FRA (- 2.34) = 0.66

get

GET reads values from the parameter buffer and then deletes them. The parameters are read from the buffer in the order they have been added with the PUT command. Syntax GET (n) Parameter n: number of parameters to read from the parameter buffer Remarks GET is a function and therefore can not be used standalone. If n=1, you can assign the returned value to a variable. If n>1, GET has to be used as an argument for another command.

glob_color

GLOB_COLOR is a global variable and returns the current color ID of the SmartParts object.

glob_id
GLOB_ID is a global variable and returns the ID of the SmartParts object.

glob_intguid
GLOB_INTGUID is a global variable and returns the GUID of the SmartParts object.

glob_layer
GLOB_LAYER is a global variable and returns the number of the current layer, which is assigned to the SmartParts object.

glob_layer_name
GLOB_LAYER_NAME is a global variable and returns the short name of the layer, which is assigned to the SmartParts object.

glob_modpar_name
GLOB_MODPAR_NAME is a global variable and returns the name of the last modified parameter.

glob_pen
GLOB_PEN is a global variable and returns the number of the current pen, which is assigned to the SmartParts object.

glob_scale
GLOB_SCALE is a global variable and returns the current reference scale of the drawing file. E.g if the reference scale is 1:100, GLOB_SCALE returns 100.

glob_script_type
GLOB_SCRIPT_TYPE is a global variable and returns a number, which reflects the current part of the SmartPart Script according to the following assignment: 2D Script 2 3D Script 3 Dialog Script 4 Parameter Script 5 Master Script 6

glob_stroke
GLOB_STROKE is a global variable and returns the number of the current line type, which is assigned to the SmartParts object.

glob_ui_button_id
GLOB_UI_BUTTON_ID is a global variable and returns the ID of the last clicked button on the I_PAGE.

gosub
GOSUB calls an internal subroutine. Syntax GOSUB label Parameters label: the entry point of the subroutine. label can be any numerical or string expression, and even a variable (but this may slow down the script). The subroutine must end with a RETURN statement.

goto
GOTO jumps to an internal subroutine. Syntax GOTO label Parameters label: the entry point of the subroutine. label can be any numerical or string expression, and even a variable (but this may slow down the script). GOTO does not return to the calling line of the script (in contrary to GOSUB)

group
GROUP starts a group definition. A group definition must be finished with GROUP_END. All elements within the script between GROUP and GROUP_END belong to a group called "name". The group is defined, but not yet used. Therefore, you have to "activate" a group using GROUP_PLACE.   Syntax GROUP "name" Parameters name (string): group name   Remarks Group names must be unique within a current script. All transformations, cutplanes etc. must be performed within the group. Nested group definitions are not possible, but subscripts can contain groups. Attribute DEFINEs and SETs (pens, materials, fills) before group definitions as well as within group definitions are all effective.

group_del
GROUP_DEL is a group operation and removes the named group bodies from memory. Use GROUP_DEL when you don't need the group anymore.   Syntax GROUP_DEL g_expr Parameters g_expr (string): group name or variable or result of group operation Remarks At the end of a script or subscript, all groups will be removed from the memory automatically.

group_diff
GROUP_DIFF is a group operation and subtracts one group from another; the result is a new group. The new group can be used with GROUP_PLACE, as variable or in another group definition.   Syntax GROUP_DIFF (g_expr1, g_expr2) Parameters g_expr1, g_expr2 (string): group name or variable or result of group operation g_expr1 is the minuend, g_expr2 the subtrahend.   Remarks For group operations, the groups must have been defined already. They also can be results of other group operations. The minuend's surface attributes are used for the new group.

group_end
GROUP_END finishes a group definition started with GROUP.   Syntax GROUP_END

group_isect
GROUP_ISECT is a group operation and intersects two groups; the result is the groups' intersected volume in a new group. The new group can be used with GROUP_PLACE, as variable or in another group definition.   Syntax GROUP_ISECT (g_expr1, g_expr2) Parameters g_expr1, g_expr2 (string): group name or variable or result of group operation Remarks For group operations, the groups must have been defined already. They also can be results of other group operations.

group_place
GROUP_PLACE is a group operation and actually generates a previously defined group and places it in the current position. All transformations, cutplanes etc. are performed. Syntax GROUP_PLACE g_expr Parameters g_expr (string): group name or variable or result of group operation Remarks The resulting bodies are stored in the 3D data structure.

group_sweep
GROUP_SWEEP is a group operation and creates a new group by sweeping the existing one along a directional vector. The covered area is added to the resulting group's volume. Syntax GROUP_SWEEP (g_expr, x, y, z) Parameters g_expr (string): group name or variable or result of group operation x, y, z (real): directional vector for sweeping   Remarks The difference between GROUP_SWEEP and

GROUP_SWEEP_2 is how transformations affect the directional vector: With GROUP_SWEEP, the directional vector for the sweeping refers to the local coordinate system of the transformed group. With GROUP_SWEEP_2, the transformation is applied to the local origin of the object (not the project origin). Only for solid models.

group_sweep_2
GROUP_SWEEP_2 is a group operation and similar to GROUP_SWEEP. It creates a new group by sweeping the existing one along a directional vector. The covered area is added to the resulting group's volume. Syntax GROUP_SWEEP_2 (g_expr, x, y, z) Parameters g_expr (string): group name or variable or result of group operation x, y, z (real): directional vector for sweeping Remarks The difference between GROUP_SWEEP and GROUP_SWEEP_2 is how transformations affect the directional vector: With GROUP_SWEEP, the directional vector for the sweeping refers to the local coordinate system of the transformed group. With GROUP_SWEEP_2, the transformation is applied to the local origin of the object (not the project origin). Only for solid models.

group_union
GROUP_UNION is a group operation and adds one group to another; the result is a new group. The new group can be used with GROUP_PLACE, as variable or in another group definition. Syntax GROUP_UNION (g_expr1, g_expr2) Parameters g_expr1, g_expr2 (string): group name or variable or result of group operation Remarks For group operations, the groups must have been defined already. They also can be results of other group operations.

handle
HANDLE creates a 3D handle in the point (X, Y, Z). With handles, you can move or modify an object. HANDLE is similar to HANDLE2. Syntax HANDLE x, y, z [, unID [, paramReference, flags][, displayParam]] Parameters x (decimal): X coordinate of handle y (decimal): Y coordinate of handle z (decimal): Z coordinate of handle unID (integer, > 0): handle's unique identifying number paramReference: parameter to be edited by current handle, using the method of graphical parameter editing using handles. Valid arguments: D, Arr[5], Arr[2*I+3][D+1], etc. flags: handle type + handle attribute handle type: 1: length type editing, base handle (always hidden) 2: length type editing, moving handle 3: length type editing, reference handle (always hidden) 4: angle type editing, base handle (always hidden) 5: angle type editing, moving handle 6: angle type editing, center of angle (always hidden) 7: angle type editing, reference handle (always hidden) 8: switch type 9: input type handle attribute: it can be the following value or zero: 512: reverse the angle in 2D (for handle type 6) Remarks Length type parameters: You have to define three handles, one of type 1 (base handle), one of type 2 (moving handle) and one of type 3 (reference handle). The vector from the reference handle to the base handle gives the positive direction of the editing line. You have to place the moving handle along the editing line at a distance given by the associated parameter's value measured from the base handle. Angle type parameters: You have to define four handles, one of type 4 (base handle), one of type 5 (moving handle), one of type 6 (center of angle) and one of type 7 (reference handle). The angle's plane is perpendicular to the vector going from center handle to reference handle. Positive direction for measuring an angle is counter-clockwise (default for 2D). With the attribute 512 for the center handle (type 6) you can change the positive direction to clockwise. To be consistent, the vectors from the center handle to the moving and the base handles must be perpendicular to the vector from the center to the reference handle. The moving handle must be placed at an angle determined by the associated parameter measured from the base handle around the center handle. If you define several handle sets to modify the same parameter, the handles are grouped together in the order in which the handle commands are executed. You can combine two or three length type handles in order to change several parameters with only one drag. If two handles are combined, you can drag the handles within the plane defined by the two lines belonging to each length editing handle. The two lines must not be parallel to each other. If you combine three sets of length type handles in 3D, then you can position the handle anywhere in space. The three lines must not be on the same plane. If there are two or three editable handles referring to different parameters at the place where you click a point, you start a combined editing

operation. If parameters are designed for combined editing, the base and reference handles are not fixed in the object's coordinate frame, but must move according to the changes of the other parameters' value.

handle_

HANDLE_ is a command in SmartParts converted from GDL. It is similar to HANDLE, but it does not support HANDLEFACTOR and the handle types switch (8) and input (9).

handle2

HANDLE2 creates a 2D handle in the point (X, Y). With handles, you can move or modify an object. HANDLE2 is similar to HANDLE. Syntax HANDLE2 x, y [, unID [, paramReference, flags][, displayParam]] Parameters x (decimal): X coordinate of handle y (decimal): Y coordinate of handle unID (integer, > 0): handle's unique identifying number paramReference: parameter to be edited by current handle, using the method of graphical parameter editing using handles. Valid arguments: D, Arr[5], Arr[2*I+3][D+1], etc. flags: handle type + handle attribute handle type: 1: length type editing, base handle (always hidden) 2: length type editing, moving handle 3: length type editing, reference handle (always hidden) 4: angle type editing, base handle (always hidden) 5: angle type editing, moving handle 6: angle type editing, center of angle (always hidden) 7: angle type editing, reference handle (always hidden) 8: switch type 9: input type handle attribute: it can be the following value or zero: 512: reverse the angle in 2D (for handle type 6) Remarks Length type parameters: You have to define three handles, one of type 1 (base handle), one of type 2 (moving handle) and one of type 3 (reference handle). The vector from the reference handle to the base handle gives the positive direction of the editing line. You have to place the moving handle along the editing line at a distance given by the associated parameter's value measured from the base handle. Angle type parameters: You have to define four handles, one of type 4 (base handle), one of type 5 (moving handle), one of type 6 (center of angle) and one of type 7 (reference handle). The angle's plane is perpendicular to the vector going from center handle to reference handle. Positive direction for measuring an angle is counter-clockwise, when looking from the reference handle on the plane. With the attribute 512 for the center handle (type 6) you can change the positive direction to clockwise. To be consistent, the vectors from the center handle to the moving and the base handles must be perpendicular to the vector from the center to the reference handle. The moving handle must be placed at an angle determined by the associated parameter measured from the base handle around the center handle. If you define several handle sets to modify the same parameter, the handles are grouped together in the order in which the handle commands are executed. You can combine two length type handles in order to change several parameters with only one drag. If two handles are combined, you can drag the handles within the plane defined by the two lines belonging to each length editing handle. The two lines must not be parallel to each other. If there are two or three editable handles referring to different parameters at the place where you click a point, you start a combined editing operation. If parameters are designed for combined editing, the base and reference handles are not fixed in the object's coordinate frame, but must move according to the changes of the other parameters' value.

handle2_

HANDLE2_ is a command in SmartParts converted from GDL. It is similar to HANDLE2, but it does not support HANDLEFACTOR and the handle types switch (8) and input (9).

handlefactor

HANDLEFACTOR sets a factor by which a measured handle distance is multiplied in order to get a parameter value. HANDLEFACTOR is a directive and is valid until it is changed to another value. Syntax HANDLEFACTOR expr Parameter expr (decimal, default = 1): factor to multiply a measured handle distance Remarks You can use HANDLEFACTOR for example for a circle with a handle on the circular arc: the measured distance between handle and center point is the radius, the parameter represents the diameter. Therefore, before the handle definition you insert HANDLEFACTOR 2.0 Then the measured value (radius) is multiplied by 2, and the parameter gets the value 2*radius = diameter Don't forget to reset HANDLEFACTOR to 1.0 after the affected handle.

hatching
DEFINE HATCHING defines a hatching, which can be referred to by name further on in other scripts. Hatching definitions prior to the definition of this hatching may be included in any script; however, hatchings defined that way can be used solely in those scripts, in which they are defined, and their subsequent second generation-scripts. Syntax DEFINE HATCHING name, id [, scale_dependent, angle, ref_x, ref_y, red, green, blue, alpha] Parameters name (string): name of the hatching, by which can be referred to this hatching id (integer): number of the hatching scale_dependent (integer): appearance of the hatching depends on scale or not with scale_dependent may be 0 or 1 0: not scale dependant 1: scale dependant angle (decimal): rotation angle (default: 0.0) ref_x, ref_y (decimal): X resp. Y coordinate of the reference point (default: (0.0;0.0)) red, green, blue (integer): color palette for masking/background color (default: white) with 0 <= red, green resp. blue <= 256 alpha (integer): tilt angle of the hatching with 0 <= alpha <= 100 Corresponding commands [set] HATCHING name_string [set] HATCHING index Index is a constant referring to a hatch stack in the internal data structure; this stack is modified during script analysis and can also be modified from within the program. Any use of the index instead of the hatching's name is only recommended with the prior use of the INDEX function.

hide
HIDEPARAMETER or HIDE hides a named parameter (and its deriving parameters) in the settings dialog box.   Syntax HIDE name1 [, name2, ..., namen] Parameters namei (text): parameter name

hideparameter
HIDEPARAMETER or HIDE hides a named parameter (and its deriving parameters) in the settings dialog box.   Syntax HIDEPARAMETER name1 [, name2, ..., namen] Parameters namei (text): parameter name

i_button
I_BUTTON creates a button on the current page. Buttons can be used to move from page to page or to open a website.   Syntax I_BUTTON type, text, x, y, width, height [, id] [, url] [I_TOOLTIP tooltiptext] Parameters type: defines the type of the button: I_PREV: displays the previous page I_NEXT: displays the next page I_FUNCTION: sets the global variable GLOB_UI_BUTTON_ID to the value specified in id I_LINK: opens the web site which is defined in url in the default web browser text (string): defines the text on the button x, y (integer): defines the position of the button (in pixel). Negative values can be used to make the button invisible. width, height (integer, >= 0): defines width and height of the button in pixels. id (integer, >= 0): unique identifier url (string): a URL in double quotes (only if type = I_LINK) I_TOOLTIP: defines a tooltip for the interface element (optional), for description see I_TOOLTIP.   Remarks Each button has to be defined for every single page. The pages itself have to be defined by I_PAGE.

i_current_page
I_CURRENT_PAGE defines the tab of the dialog, which should be displayed. Syntax I_CURRENT_PAGE index Parameter index (integer, > 0): index of I_PAGE to display

i_dialog
I_DIALOG defines the title and size of a dialog box. I_DIALOG is mandatory for defining a dialog. Syntax I_DIALOG title [, size_x, size_y] Parameters title (string): the title of the dialog, in double quotes. size_x, size_y (integer, >= 0): optional, defines size of dialog in pixels.   Remarks Each Dialog Script can contain only one I_DIALOG command.

i_groupbox

I_GROUPBOX creates a rectangle in the dialog box. You can use it to group parameters, which are related. Syntax I_GROUPBOX text, x, y, width, height Parameter text (string): defines the title of the group box. The title is displayed in the upper left corner of the rectangle. Can be left empty, in this case the group box is drawn as a closed rectangle without text. x, y (integer): defines the position of upper left corner in pixels. width, height (integer): defines width and height in pixels.

i_infield

I_INFIELD creates a control (e.g. text entry field or popup) in which the user of the SmartParts object can enter or select parameters. Syntax I_INFIELD "name", x, y, width, height [, version_flag, picture_name, images_number, rows_number, cell_x, cell_y, image_x, image_y, expression_image1, text1, ..., expression_imagen, textn] [I_TOOLTIP tooltiptext] Parameter name (string): name of parameter x,y (integer, >=0): position of element in pixels width, height (integer, >=0): width and height of element in pixels version_flag: reserved, always 1 method (integer, 1 <= method <= 7): defines the type of control: 1: list view control 2: popup menu control 3: popup icon radio control 4: push icon radio control 5: pushbutton with text 6: pushbutton with bitmap 7: ckeckbox with text picture_name (string): name of the bitmap file, which contains the matrix of the concatenated images. If left empty, the images are specified by expression_image(n). images_number: number of images in the matrix rows_number: number of rows in the matrix cell_x, cell_y: width and height of a cell, including image and text image_x, image_y: width and height of image expression_image(n): index of image (n) in the matrix. If picture_name is empty, you can specify the file name here. A mixture of file names and indices is not allowed. text(n): text in cell (n) I_TOOLTIP: defines a tooltip for the interface element (optional), for description see I_TOOLTIP.

i_infield_2

I_INFIELD_2 creates a control (e.g. text entry field or popup) in which the user of the SmartParts object can enter or select parameters. Other than with I_INFIELD the name of the parameter is not entered as a string but as name of the parameter. Syntax I_INFIELD_2 name, x, y, width, height [, version_flag, picture_name, images_number, rows_number, cell_x, cell_y, image_x, image_y, expression_image1, text1, ..., expression_imagen, textn] [I_TOOLTIP tooltiptext] Parameter name: name of parameter x,y (integer, >=0): position of element in pixels width, height (integer, >=0): width and height of element in pixels version_flag: reserved, always 1 method (integer, 1 <= method <= 7): defines the type of control: 1: list view control 2: popup menu control 3: popup icon radio control 4: push icon radio control 5: pushbutton with text 6: pushbutton with bitmap 7: ckeckbox with text picture_name (string): name of the bitmap file, which contains the matrix of the concatenated images. If left empty, the images are specified by expression_image(n). images_number: number of images in the matrix rows_number: number of rows in the matrix cell_x, cell_y: width and height of a cell, including image and text image_x, image_y: width and height of image expression_image(n): index of image (n) in the matrix. If picture_name is empty, you can specify the file name here. A mixture of file names and indices is not allowed. text(n): text in cell (n) I_TOOLTIP: defines a tooltip for the interface element (optional), for description see I_TOOLTIP.

i_infield_3

I_INFIELD_3 creates a control (e.g. text entry field or popup) in which the user of the SmartParts object can enter or select parameters. Other than with I_INFIELD the name of the parameter is not entered as a string but as name of the parameter Syntax I_INFIELD_3 name, x, y, width, height [, version_flag, picture_name, images_number, rows_number, cell_x, cell_y, image_x, image_y, expression_image1, text1, ..., expression_imagen, textn] [I_TOOLTIP tooltiptext] Parameter name: name of parameter x,y (integer, >=0): position of element in pixels width, height (integer, >=0): width and height of element in pixels version_flag: reserved, always 1 method (integer, 1 <= method <= 7): defines the type of control: 1: list view control 2: popup menu control 3: popup icon radio control 4: push icon radio control 5: pushbutton with text 6: pushbutton with bitmap 7: checkbox with text picture_name (string): name of the bitmap file, which contains the matrix of the concatenated images. If left empty, the images are specified by expression_image(n). images_number: number of images in the matrix rows_number: number of rows in the matrix cell_x, cell_y: width and height of a cell, including image and text image_x, image_y: width and height of image expression_image(n): index of image (n) in the matrix. If picture_name is empty, you can specify the file name here. A mixture of file names and indices is not allowed. text(n): text in cell (n) I_TOOLTIP: defines a tooltip for the interface element (optional), for description see I_TOOLTIP.

i_outfield

I_OUTFIELD generates a static text block, in which you can enter some text, e.g. the description of a parameter. The style of the text is defined by I_STYLE. Syntax I_OUTFIELD expression,x,y,width,height [, alignment] [I_TOOLTIP tooltiptext] Parameter expression (string): any string (text or numerical), must be enclosed between double quotes. x, y (integer, >=0): upper left corner of the text field. width, height (integer, >0): width and height of the text field in pixels. alignment (integer): defines the alignment of the text within the text field. If not specified, the text is left aligned. 0: left aligned (default) 1: right aligned 2: centered I_TOOLTIP: defines a tooltip for the interface element, for description see I_TOOLTIP.

i_page

I_PAGE defines the page, on which the following interface elements are placed on. Everything after this command is placed on this page, until the next I_PAGE command. Syntax I_PAGE page_number [, name] I_PAGE name Parameters page_number (integer, > 0): defines the page number name (string): defines the name of the page Remarks Page numbering must be started with 1 and has to be continuous. If the numbering is not continuous, new empty pages are inserted until the desired number is reached.

i_pict

I_PICT creates a picture element in the dialog box. The picture file must be located in STD\\SmartParts\\bitmaps folder or stored as a resource image in the SmartPart object. Syntax I_PICT expression, x, y [,width, height [,mask]] [I_TOOLTIP tooltiptext] Parameter expression (string): file name of the picture. The file name must be enclosed in double quotes. x, y (integer): top left corner of the picture in pixels. width, height (integer): width and height in pixels. If not specified, the original width and height of the picture is used. mask (always 0): distortion 0: picture is adjusted to rectangle, the original size and aspect ratio is neglected I_TOOLTIP: defines a tooltip for the interface element (optional), for description see I_TOOLTIP.

i_separator

I_SEPARATOR defines a separator as a rectangle or a line. Syntax I_SEPARATOR x1, y1, x2, y2 Parameter x1, y1 (decimal): coordinates of the upper left node (=starting point of the line). x2, y2 (decimal): coordinates of the lower right node (=endpoint of the line). Remarks The rectangle becomes a vertical or horizontal line if x1 = x2 or y1 = y2.

i_style

I_STYLE defines the font size and appearance of all I_OUTFIELD and I_INFIELD elements generated after this keyword, till the next I_STYLE command. Syntax I_STYLE fontsize, face_code Parameter fontsize (integer): one of the following font size values: 0: small 1: extra small 2: large face_code (integer): the values cannot be used in combination 0: normal 1: bold 2: italic 4: underline

i_tooltip

I_TOOLTIP creates a tooltip for interface elements. I_TOOLTIP is not an independent command but only used in conjunction with commands, which create interface elements. Syntax I_TOOLTIP tooltiptext Parameter tooltiptext (string): text which should be displayed as tooltip for the interface element. Remarks I_TOOLTIP is available for I_BUTTON, I_INFIELD, I_INFIELD_2, I_INFIELD_3, I_OUTFIELD, I_PICT

if

IF executes the following SmartPart Script commands depending on a condition. Syntax IF condition THEN command IF condition THEN command1 command2 ELSE ENDIF Parameters condition: if condition=true (<>0) all commands after THEN are executed. Otherwise the commands after ELSE are executed. If ELSE is absent, the commands after ENDIF are executed. command: any SmartPart Script command Remarks If the command after THEN is GOTO or GOSUB, you can omit the THEN statement and shorten to IF condition GOTO label. If there is only 1 command after THEN or ELSE in the same row, you can omit ENDIF.

index

INDEX returns the integer value of the corresponding ressource. Syntax INDEX (resource_type, resource_name) Parameters resource_type: the type of the resource. resource_type can be one of the foolowing types MATERIAL FILL STYLE TEXTURE FOIL HATCHING PATTERN BITMAP PATTERN_LINE LAYER FACE_STYLE resource_name: the name of the resource in double quotes. resource_name is case sensitive. Remarks The ressource has to be defined previously with DEFINE..., except with layers, where INDEX returns the layer number of a layer short name. e.g. INDEX (LAYER, "DE_GEN01") will return 3700 (with an english Allplan version, in other language versions you will need the localized layer short name).

int

INT returns the integral part of a number, rounded down to the next integer. Syntax INT (expression) Parameter expression (decimal): any number

layer

LAYER defines the layer for the following objects in the script till the next LAYER command. You can define the layer either by its ID or by its short name. Syntax [SET] LAYER id [SET] LAYER short_name Parameter id: sets the layer for the following objects to the layer with id. The use of the id instead of the layer short name is only recommended with the prior use of the INDEX function. short_name: sets the layer for the following objects to the layer with "short_name". INDEX (LAYER,"short_name") Remarks LAYER CURR resets the layer to the current layer. LAYER STANDARD resets the layer to the STANDARD Layer.

let

LET assigns a value to a variable. The calculated value of n is stored by the variable.   Syntax [LET] varnam = n Parameters varnam (string): name of variable n (string, numeric): expression, corresponding to variable type Remarks LET is not necessary for assigning a value to a variable, you normally use varnam = n.

line

LINE generates a line in the three-dimensional space between the two points P1 (x1,y1,z1) and P2 (x2,y2,z2). Syntax LINE x1, y1, z1, x2, y2, z2 Parameters x1, y1, z1 (real): X, Y and Z coordinate of the starting point x2, y2, z2 (real): X, Y and Z coordinate of the end point

line2

LINE2 creates a line between two points. Syntax LINE2 x1, y1, x2, y2

ln

LN returns the natural logarithm of x. Syntax LN (x)   Parameter x (decimal, >0) Remarks All returned values are decimal.

lock

LOCK deactivates a named parameter in the settings dialog box. The related entry box is displayed gray. The user is not allowed to enter or change the parameter's values. Syntax LOCK name1 [, name2, ..., namen] Parameters namei (text): parameter name Remarks Locked parameters can be changed in the script.

log

LOG returns the base 10 logarithm of x. Syntax LOG (x) Parameter x (decimal, >0)

material

DEFINE MATERIAL defines a material, which can be referred to by name further on in other scripts. Material definitions prior to the definition of this material may be included in any script; however, materials defined that way can be used solely in those scripts, in which they are defined, and their subsequent second generation-scripts. Syntax DEFINE MATERIAL name type, parameter1, parameter2, ... parametern Parameters name (string): name of the material, by which can be referred to this material type (string): type of material. parametern (decimal): The actual number (n) of parameters possible depends on the material type. For further information, see the parameter descriptions. 0: general definition, n=16 1: simple definition, n=9 (additional parameters are constants or calculated from given values) 2-7: predefined material types, n=3 (RGB components of the surface color) Other parameters deriving from the color. 2: matte 3: metal 4: plastic 5: glass 6: glowing 7: constant 10: general definition with fill parameter, n=17 11: simple definition with fill parameter, n=10 12-17: predefined material types with fill parameter, n=4 20: general definition with fill, color index of fill and index of texture parameters, n=19 21: simple definition with fill, color index of fill and index of texture parameters, n=12 22-27: predefined material types with fill, color index of fill and index of texture parameters, n=6 Special meanings for types 20-27: If the pen number is zero, vectorial hatches will be generated with the active pen. (not supported) Zero value for a texture index allows you to define materials without a vectorial hatch or texture. Corresponding commands [set] MATERIAL name_string [set] MATERIAL index All surfaces generated after a material setting will obtain this material, until the next material is stated. Exceptions are surfaces in the PRISM_B, PRISM_C, PRISM_F, PRISM_H, PRISM_S, SOLID_SLAB_C, SOLID_WALL_, SOLID_WALL_B, SOLID_WALL_X, CROOF_, MESH_P elements. Index is a constant referring to a material stack in the internal data structure; this stack is modified during script analysis and can also be modified from within the program. Any use of the index instead of the material's name is only recommended with the prior use of the INDEX function. INDEX 0 means that the material used for the surfaces is assigned to the line color. Default: MATERIAL 0 if the script does not contain a MATERIAL statement.

max

MAX returns the largest value of x1, ..., xn. The number of arguments is unlimited. Syntax MAX (x1,x2, ... xn) Parameter x (decimal): unlimited number of values

mesh

MESH generates a mesh based on a grid with an equidistant grid-length within a rectangular contour with userdefined length of the rectangle sides. Syntax MESH a, b, m, n, mask, z11, z12, ... , z1m, z21, z22, ... , z2m, zn1, zn2, ... , znm Parameters a, b (decimal): length of the rectangle sides m, n (integer): number of nodes along the X resp. Y axis mask: controls the existence of the bottom and side surfaces (see below) zij (decimal): height of the corresponding node in Z direction parameter restrictions: m >= 2, n >= 2 Masking mask = j1 + 4*j3 with j1, j3 may be 0 or 1 j1 (1): bottom surface is present j3 (4): side surfaces are present

mesh_b

MESH_B generates a mesh which approximates the Bezier surface defined by the specified control points. The surface evaluation may happen in two different modes, static mode and adaptive mode. Adaptive mode is being chosen, if subdiv_n or subdiv_m is less than or equal to 2. Otherwise, the static mode will be chosen and parameter value rise will be ignored. Syntax MESH_B m, n, subdiv_n, subdiv_m, rise, x11, y11, z11, x12, y12, z12, ... , x1m, y1m, z1m, x21, y21, z21, x22, y22, z22, ... , x2m, y2m, z2m, ... , xn1, yn1, zn1, xn2, yn2, zn2, ... , xnm, ynm, znm Parameters m, n (integer): number of nodes along the X resp. Y axis subdiv_n, subdiv_m (integer): number of subdivisions in the corresponding direction rise (integer, >0): valid values must be greater than 0 xij, yij, zij (decimal): X, X and Z coordinates of the nodes Remarks Modes for surface evaluation: > In the static mode, the surface is being evaluated at subdiv_n and subdiv_m parameter values. > In the adaptive mode, the surface is being evaluated at as many parameter values as needed, to guarantee the following: Distance from surface to the mid of each resulting mesh face is less than or equal to the parameter value rise.

mesh_c

MESH_C generates a coon's patch defined by four boundary curves forming a chain. Syntax MESH_C n, m, x11, y11, z11, ... , x1n, y1n, z1n, x21, y21, z21, ... , x2n, y2n, z2n, x31, y31, z31, ... , x3m, y3m, z3m, x41, y41, z41, ... x4m, y4m, z4m Parameters n, m (integer): number of nodes on the sides along the x- resp. y-axis x1n, y1n, z1n (decimal): X, Y, Z coordinates of the nodes on the first curve in X direction x2n, y2n, z2n (decimal): X, Y, Z coordinates of the nodes on the second curve in X direction x3m, y3m, z3m (decimal): X, Y, Z coordinates of the nodes on the first curve in Y direction x4m, y4m, z4m (decimal): X, Y, Z coordinates of the nodes on the second curve in Y direction parameter restrictions: n, m > 1 Masking not supported

mesh_p

MESH_P generates a free-form surface based on a coon's patch with a plinth; materials can be defined for the top-, bottom- and side surfaces. Syntax MESH_P top_material, bottom_material, side_material, n, m, mask, h, x1, y1, z1, s1, ... , xn, yn, zn, sn, xn+1, yn+1, zn+1, sn+1, ... , xn+m, yn+m, zn+m, sn+m Parameters top_material, bottom_material, side_material (integer resp. string): index or name of the top, bottom and side material n (integer): number of nodes in the polygon m (integer): number of nodes on the ridges h (decimal): base height (can be negative) xi, yi, zi (decimal): X, Y and Z coordinates of the node si (boolean): controls the appearance of the line segment; if si = 0 the line segment starting from xi, yi, zi will not be drawn, if si = 1 this line segment will be drawn; with si = -1 holes are defined directly parameter restrictions: n >= 3, m >= 0 Masking mask = j1 + 4*j3 with j1 and j3 may be 0 or 1 j1 (1): bottom surface is present j3 (4): side surfaces are present

min

MIN returns the smallest value of x1, ..., xn. The number of arguments is unlimited. Syntax MIN (x1,x2, ... xn) Parameter x (decimal): unlimited number of values

model

MODEL defines the type of a body: wire frame model (transparent), surface model (= hollow body; opaque) or solid body (opaque). The difference between a surface model and a solid model can only be seen after cutting the body by a plane: the surface model is hollow, the solid model can show another surface on the cutting plane. Syntax MODEL type Parameters type (string): type of the model with type may be wire: defines a wire frame model without surfaces and/or volume surface: defines a hollow body solid: defines a solid body

next

NEXT ends a FOR...NEXT loop. Syntax FOR variable=start_value TO end_value [STEP step_value] ... NEXT Parameters variable: name of the variable. Only local variables are allowed, no global variables. start_value (decimal): initial value of the variable end_value (decimal): value, till which the loop is executed. Before a loop is started, it is checked, whether end_value is reached.   step_value (decimal): value by which start_value is increased with every cycle of the loop. If not specified, start_value is increased by 1. step_value can be negative also. A change of step_value inside the loop is ignored.

not

NOT (x) returns the logical negation of x. It returns false (=0 integer) if x is true (><0), and true (=1 integer) if x is false (=0). Syntax NOT (x) Parameter x (decimal): expression which should be checked.

num_sp

NUM_SP is a local parameter which returns the number of stored parameters in the parameter buffer. Syntax NUM_SP

num_tr

NUM_TR returns the current number of transformations on the transformation stack. Syntax NUM_TR ()

parameters

PARAMETERS can be used in a Parameter Script and changes the value of the named parameter from the following call on. Commands in a subscript use the settings of the calling script. For the parameter being a value list, the parameter either will use an exisrting value or the custom value or the list's first value. Note that GLOB_MODPAR_NAME returns the name of the last modified parameter only, if it is edited by the user (dragging a handle or values input in dialog). Syntax PARAMETERS name1 = expression1 [,name2 = expression2, ...,namen = expressionn] Parameter namei: the name of the parameter expressioni: the new value of the parameter

pattern

DEFINE PATTERN defines a pattern, which can be referred to by name further on in other scripts. Pattern definitions prior to the definition of this pattern may be included in any script; however, patterns defined that way can be used solely in those scripts, in which they are defined, and their subsequent second generation-scripts. Syntax DEFINE PATTERN name, id [, scale_dependent, angle, scale_x, scale_y, ref_x, ref_y, fitting_mode, red, green, blue, alpha] Parameters name (string): name of the pattern, by which can be referred to this pattern id (integer): number of the pattern scale_dependent (integer): appearance of the pattern depends on scale or not with scale_dependent may be 0 or 1 0: not scale dependant 1: scale dependant angle (decimal): rotation angle (default: 0.0) scale_x, scale_y (decimal): scaling of the pattern in X resp. Y direction (default: 1.0) with scale_x, scale_y <> 0 ref_x, ref_y (decimal): X resp. Y coordinate of the reference point (default: (0.0;0.0)) fitting_mode (integer): fitting of the pattern with fitting_mode may be 1: outside fitting 2: fitting 3: inside fitting red, green, blue (integer): color palette for masking/background color (default: no color) with 0 <= red, green resp. blue <= 255 alpha (integer): tilt angle of the pattern with 0 <= alpha <= 100 Corresponding commands [set] PATTERN name_string [set] PATTERN index Index is a constant referring to a pattern stack in the internal data structure; this stack is modified during script analysis and can also be modified from within the program. Any use of the index instead of the pattern's name is only recommended with the prior use of the INDEX function.

pattern_line

DEFINE PATTERN_LINE defines a pattern line, which can be referred to by name further on in other scripts. Pattern line definitions prior to the definition of this pattern line may be included in any script; however, pattern lines defined that way can be used solely in those scripts, in which they are defined, and their subsequent second generation-scripts. Syntax DEFINE PATTERN_LINE name, id [, width, height, align, type, ref_line, mirror, lock_corner, scale_dependent] Parameters name (string): name of the pattern line, by which can be referred to this pattern line id (integer): number of the pattern line width (decimal): width of the pattern (default 0.1) height (decimal): height of the pattern (default 0.1) align (integer):

alignment of the pattern with align may be 0: left 1: center 2: right type (integer): string together of the pattern with type may be 0: none 1: miter 2: join 3: seamless ref_line (integer): appearance of the reference line with ref_line may be 0: reference line doesn't appear (default) 1: reference line is shown mirror (integer): mirroring of the pattern with mirror may be 0: pattern is not mirrored (default) 1: pattern is mirrored lock_corner (integer): corner locking of the pattern with lock_corner may be 0: corner of the pattern is not locked (default) 1: corner of the pattern is locked scale_dependent (integer): appearance of the pattern depends on scale or not with scale_dependent may be 0 or 1 0: not scale dependant 1: scale dependant Corresponding commands [set] PATTERN_LINE name_string [set] PATTERN_LINE index Index is a constant referring to a pattern line stack in the internal data structure; this stack is modified during script analysis and can also be modified from within the program. Any use of the index instead of the pattern line's name is only recommended with the prior use of the INDEX function.

pen

PEN sets the pen for an element, to define its line width. Syntax PEN n Parameters n (integer): identification number of the pen with 0 < n <= 255 n = -1 (default): the current pen is used You can also use constants instead: CURR = -1 BY_LAYER = -2

pgon

PGON generates a polygon referring to prior to this defined edges. Syntax PGON n, edge1, edge2, ... , edgen Parameters n (integer): number of edges edge1, ... , edgen: reference to previously defined edges; the direction of a prior to this defined edge can be changed temporarily by a negative index (i.e. the original edge does not change direction); a beginning or an end of a hole can be defined by setting a zero value

pi

PI returns the value of the constant p (p = 3.1415926...). Syntax PI

place

PLACE inserts one or all resource objects into the current 3D script, transformations are considered. Syntax PLACE index,use_current_attributes_flag Parameter Index (boolean): -1: all resource objects are inserted 0 ... n : resource objects with that specific index are inserted use_current_attributes_flag (boolean): defines whether or not the current attributes will be used 0: the fragment appears with the color, line type and fill type defined for it. 1: the current settings of the script are used instead of the color, line type and fill type of the fragment.

place2

PLACE2 inserts a resource object with the stated index number. The current transformations are considered. Syntax PLACE2 fragment_index, use_current_attributes_flag Parameters fragment_index (integer, <=16): inserts object with the given number use_current_attributes_flag (boolean): defines usage of the current attributes 0: object is displayed with its defined color, line type and fill type 1: object is displayed with the current settings of the script Remarks You can use ALL instead of the fragment_index parameter in order to insert all objects. PLACE2 ALL, use_current_attributes_flag

poly

POLY generates a polyline in the XY plane. Syntax POLY n, x1, y1, ..., xn, yn Parameters n (integer): number of polyline nodes xi, yi (decimal): X and Y coordinates of the polyline nodes parameter restrictions: n >= 3

poly_

POLY_ generates a polyline in the XY plane, but in contrast to POLY single line segments can be blanked out. Arcs and segments can be defined within the polyline by using additional status code values. Syntax POLY_ n, x1, y1, s1, ..., xn, yn, sn Parameters n (integer): number of polyline nodes xi, yi (decimal): X and Y coordinates of the polyline nodes si (integer): with si = -1 holes are defined directly parameter restrictions: n >= 3

poly2
POLY2 creates an open or closed polygon with n nodes. Syntax POLY2 n, frame_fill, x1, y1, ... xn, yn Parameters n (integer, >= 2): number of nodes frame_fill (integer): defines type of contour and/or fill, j1 + 2*j2 + 4*j3 j1: contour only, 0=invisible, 1=visible j2: fill only, 0=invisible, 1=visible j3: polygon open/closed, 0=do not close polygon, 1=close polygon x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n Remarks To display a fill, you have to define the fill beforehand, e.g. with the FILL attribute.

poly2_
POLY2_ is similar to POLY2 and creates an open or closed polygon with n nodes. Edges can be omitted: si = 0 defines that the edge beginning in (xi, yi) will not be drawn. Holes can be defined directly by using si = -1. Arcs and segments can be defined within the polyline by using additional status code values. Syntax POLY2_ n, frame_fill, x1, y1, s1, ... xn, yn, sn Parameters n (integer, >= 2): number of nodes frame_fill (integer): defines type of contour and/or fill, j1 + 2*j2 + 4*j3 + 8*j4 + 16*j5 + 32*j6 + 64*j7 j1: contour only, 0=invisible, 1=visible j2: fill only, 0=invisible, 1=visible j3: polygon open/closed, 0=do not close polygon, 1=close polygon j5: arcs/polylines, 0=real arcs, 1=polylines x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n s (n) (decimal): status value -1: end of a contour Remarks Real arcs are not generated automatically; you have to set the flag status j5 (1) because they have some disadvantages: Resizing real arcs with different factors for dX and dY moves end points. Polylines with pattern lines assigned to are drawn with gaps, because the polyline doesn't consist of combined line sections anymore, but behaves like a sequence of lines and arcs.

poly2_a
POLY2_A is similar to POLY2 and creates an open or closed polygon with n nodes.   Syntax POLY2 n, frame_fill, fill_pen, x1, y1, ... xn, yn

poly2_b
POLY2_B is similar to POLY2 and creates an open or closed polygon with n nodes.   Syntax POLY2_B n, frame_fill, fill_pen, fill_background_pen, x1, y1, s1, ... xn, yn, sn

poly2_b_2
POLY2_B_2 is similar to POLY2_B with extended options and creates an open or closed polygon with n nodes.   Syntax POLY2_B_2 n, frame_fill, fill_pen, fill_background_pen, fillOrigoX, fillOrigoY, fillAngle, x1, y1, s1, ..., xn, yn, sn Parameters n (integer, >= 2): number of nodes frame_fill (integer): defines type of contour and/or fill, j1 + 2*j2 + 4*j3 j1: contour only, 0=invisible, 1=visible j2: fill only, 0=invisible, 1=visible j3: polygon open/closed, 0=do not close polygon, 1=close polygon x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n

polyplane
POLYPLANE generates a polyline within an arbitrary plane. The polygon must be planar in order to get a correct result, but the interpreter does not check this condition. Syntax POLYPLANE n, x1, y1, z1, ... , xn, yn, zn Parameters n (integer): number of polyline nodes xi, yi, zi (decimal): X, Y and Z coordinates of the polyline nodes parameter restrictions: n >= 3

polyplane_
POLYPLANE_ generates a polyline within an arbitrary plane; the command is similar to POLYPLANE. Arcs and segments can be defined within the polyline by using additional status code values. Syntax POLYPLANE_ n, x1, y1, z1, s1, ..., xn, yn, zn, sn Parameters n (integer): number of polyline nodes xi, yi, zi (decimal): X, Y and Z coordinates of the polyline nodes si (integer): with si = -1 holes are defined directly parameter restrictions: n >= 3 Remarks The polygon must be planar in order to get a correct result, but the interpreter does not check this condition.

prev_arc2
PREV_ARC2 defines a circular preview arc based on center point, radius and central angle from startangle to endangle. PREV_ARC2 is similar to ARC2.   Syntax PREV_ARC2 x, y, r, startangle, endangle Parameters x (decimal): X coordinate of center point y (decimal): Y coordinate of center point r (decimal): radius length startangle (decimal): startangle (in degrees) endangle (decimal): endangle (in degrees) Remarks The angular offset is based on the (local) X axis; positive direction is counterclockwise. Startangle and endangle are in degrees.

prev_line
PREV_LINE creates a 3D preview line between two points. It is similar to LINE.   Syntax PREV_LINE x1, y1, z1, x2, y2 ,z2 Parameters x1, y1, z1 (decimal): X, Y and Z coordinate of the starting point x2, y2, z2 (decimal): X, Y and Z coordinate of the end point

prev_line2
PREV_LINE2 creates a 2D preview line between two points. It is similar to LINE2.   Syntax PREV_LINE x1, y1, x2, y2   Parameters x1, y1, (decimal): X and Y coordinate of the starting point x2, y2, (decimal): X and Y coordinate of the end point

prev_mode
PREV_MODE returns 0, if current script execution is during object generation and 1, if current script execution is during preview generation (e.g. if a SmartPart is modified by dragging of handles). With this parameter you can simplify complex preview calculations (e.g reducing the resolution of curved geometries with RESOL/RISE/RADIUS) and thus improve the performance: IF PREV_MODE THEN ...(code for preview construction) ELSE ...(code for final construction) ENDIF

print
PRINT creates a dialog box and displays all arguments in it.   Syntax PRINT expression [, expression, ...] Parameters expression (string, real): argument to be displayed in dialog box Remarks The string can consist of a maximum of 256 characters. The dialog box can contain as many arguments as you like. You must separate the expressions by commas. PRINT is widely used for checking results or parameter values while developing scripts.

prism
PRISM creates a right prism. Its base polygon is in the XY plane.   Syntax PRISM n, h, x1, y1, ... xn, yn Parameters n (integer, >= 3): number of nodes h (decimal): height along Z axis x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n Remarks With negative h value, the prism extrudes below the XY plane.

prism_

PRISM_ creates a right prism. It is similar to PRISM. In addition, you can define holes and create segments and arcs in the polyline. Syntax PRISM_ n, h, x1, y1, s1, ... xn, yn, sn Parameters n (integer, >= 3): number of nodes h (decimal): height along z axis x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n s(n) (integer): status code of node n

prism_b

PRISM_B is similar to PRISM_C and creates a smooth curved prism. Syntax PRISM_B top_material, bottom_material, side_material, n, h, radius, x1, y1, s1, ... xn, yn, sn Parameters top_material (string/integer): name/index of top surface bottom_material (string/integer): name/index of bottom surface side_material (string/integer): name/index of side surfaces n (integer, >= 3): number of nodes h (decimal): height along z axis radius (integer): bends xy-plane onto a cylinder tangential to it x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n s(n) (integer): status code of node n Remarks Only top_material is used for the whole prism. Edges along the X axis are transformed to circular arcs; edges along the Y axis remain horizontal; edges along the Z axis will be radial in direction. All parameters except radius correspond with PRISM_C.

prism_c

PRISM_C is similar to PRISM_. In addition, you can define a material for its surfaces. Syntax PRISM_C top_material, bottom_material, side_material, n, h, x1, y1, s1, ... xn, yn, sn Parameters top_material (string/integer): name/index of top surface bottom_material (string/integer): name/index of bottom surface side_material (string/integer): name/index of side surfaces n (integer, >= 3): number of nodes h (decimal): height along Z axis x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n s(n) (integer): status code of node n Remarks Only top_material is used for the whole solid.

prism_f

PRISM_F is similar to PRISM_. In addition, you can chamfer or round out the upper edges of the prism and define the prism's overall thickness and the material, height and angle for the chamfered rsp. rounded part. Syntax PRISM_F top_material, bottom_material, side_material, hill_material, n, thickness, angle, hill_height, x1, y1, s1, ... xn, yn, sn Parameters top_material (string/integer): name/index of top surface bottom_material (string/integer): name/index of bottom surface side_material (string/integer): name/index of side surfaces hill_material (string/integer): name/index of the side material for the chamfered part n (integer, >= 3): number of nodes thickness (decimal): overall height of prism and chamfered parts along Z axis angle (decimal, 0 <= angle < 90): inclination of chamfered edges, in degrees If angle = 0, the chamfered edges seen from an orthogonal view form a quarter circle with the current resolution. hill_height (decimal, < thickness): height of chamfered edges along Z axis x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n s(n) (integer): status code of node n. si allows you to define holes and create segments and arcs in the polyline using special constraints. Remarks Only top_material is used for the whole solid. Note that the thickness parameter represents the whole height of the PRISM_F; hill_height defines the height of the chamfered rsp. rounded part only.

prism_h

PRISM_H is similar to PRISM_F. Syntax PRISM_H top_mat, bottom_mat, side_mat, hill_mat, n, thickness, angle, hill_height, status, x1, y1, s1, ..., xn, yn, sn Parameters top_material (string/integer): name/index of top surface bottom_material (string/integer): name/index of bottom surface side_material (string/integer): name/index of side surfaces hill_material (string/integer): name/index of the side material for the chamfered part n (integer, >= 3): number of nodes thickness (decimal): overall height of prism and chamfered parts along Z axis angle (decimal, 0 <= angle < 90): inclination of chamfered edges, in degrees If angle = 0, the chamfered edges seen from an orthogonal view form a quarter circle with the current resolution. hill_height (decimal, < thickness): height of chamfered edges along z axis x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n s(n) (integer): status code of node n. si allows you to define holes and create segments and arcs in the polyline using special constraints. Remarks The status parameter is currently not supported. Only top_material is used for the whole solid.

prism_s
PRISM_S is similar to PRISM_C. In addition, you can define an upper polygon which is not parallel to the XY plane. Syntax PRISM_S top_material, bottom_material, side_material, n, xb, yb, xe, ye, h, angle, x1, y1, s1, ..., xn, yn, sn Parameters top_material (string/integer): name/index of top surface bottom_material (string/integer): name/index of bottom surface side_material (string/integer): name/index of side surfaces n (integer, >= 3): number of nodes xb, yb, xe, ye (decimal): coordinates of reference line for inclined upper polygon h (decimal): height along Z axis angle (decimal): upper polygon's rotation angle around reference line in degrees x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n s(n) (integer): status code of node n Remarks Only top_material is used for the whole solid. The height of the upper polygon is defined at the reference line. Upper and lower polygon must not intersect.

put
PUT stores one or more values in the parameter buffer. Syntax PUT expression [, expression, ...] Parameter expression (decimal): value(s) to store in the parameter buffer

pyramid
PYRAMID generates a pyramid with its base defined by a polyline in the XY plane; its peak is situated in P (0,0,h). Arcs and segments can be defined within the polyline by using additional status code values. Syntax PYRAMID n, h, mask, x1, y1, s1, ..., xn, yn, sn Parameters n (integer): number of polyline nodes h (decimal): height of the pyramid; defines the Z coordinate of the peak mask: controls the existence of the bottom and, in the case of an open polyline, side surfaces (see below) xi, yi (decimal): X and Y coordinates of the polyline nodes si: with si = -1 holes are defined directly parameter restrictions: h > 0 and n > 2 Masking mask = j1 + 4*j3 with j1, j3 may be 0 or 1 j1 (1): bottom surface is present j3 (4): side surfaces are present

radius
RADIUS controls the appearance of arcs, circles and correlated forms or bodies depending on their radius. A circle with a radius of r is represented by: - a hexagon, if r < radius_min - a n-edged polygon, if radius_min < r < radius_max (with n = (6+30*(r - radius_min)/(radius_max - radius_min)) - a 36-edged polygon, if r > radius_max The conversion of arcs is performed proportional. As soon as the command RADIUS is used, the settings of prior used commands RESOL and/or RISE are no longer valid. Syntax RADIUS radius_min, radius_max Parameters radius_min (decimal): lower limiting value for the radius radius_max (decimal): upper limiting value for the radius parameter restrictions: radius_max > radius_min

rect
RECT generates a rectangle in the x-y-plane with its first edge placed in the origin of the coordinate system. Syntax RECT a, b Parameters a (decimal): length in X direction b (decimal): length in Y direction parameter restrictions: a, b >=0

rect_pict
RECT_PICT inserts a bitmap into the XY plane. Syntax RECT_PICT expression, a, b, mask Parameters expression (string or integer): bitmap file name as string rsp. bitmap index number as integer for pictures from the library. a (decimal, >0): picture width in meters b (decimal, >0): picture height in meters mask alpha+distortion (integer, >=0): defines usage of alpha channel and picture size alpha: controls the usage of the alpha channel. 0: alpha channel is not used; picture is rectangular 1: alpha channel is used; picture may be partly transparent. distortion: controls size and scaling of the picture 0: picture fits to a rectangle a x b 2: picture is placed with metric status, and it is repeated. Remarks Picture is displayed only in photorealistic rsp. animation view.

rect_pict2
RECT_PICT2 inserts a bitmap into the plan view and is similar to RECT_PICT for 3D. Syntax RECT_PICT2 expression, a, b, mask Parameters expression (string or integer): bitmap file name as string rsp. bitmap index number as integer for pictures from the library. a (decimal, >0): picture width in meters b (decimal, >0): picture height in meters mask (binary 0,1): switches transparency off/on Picture is always fit to rectangle. Remarks With 2D pictures, the settings for mask are not considered.

rect_pict2_2
RECT_PICT2_2 inserts a bitmap into the plan view and is similar to RECT_PICT2 and RECT_PICT for 3D. Syntax RECT_PICT2_2 expression, a, b, mask Parameters expression (string or integer): bitmap file name as string rsp. bitmap index number as integer for pictures from the library. a (decimal, >0): picture width in meters b (decimal, >0): picture height in meters mask (binary 0,1): switches transparency off/on Remarks Only pixels in exact white color appear transparent.

rect2
RECT2 creates a rectangle based on two nodes (diagonal). Syntax RECT2 x1, y1, x2, y2

ref_handles
REF_HANDLES creates handles at each corner of the reference box. Syntax REF_HANDLES

ref_handles2
REF_HANDLES2 creates handles at each corner of the bottom rectangle of the reference box. Syntax REF_HANDLES2

repeat
REPEAT starts a REPEAT ... UNTIL loop. Syntax REPEAT ... UNTIL condition Parameters condition: Remarks Commands between REPEAT ... UNTIL are executed, as long as condition = true. condition is checked after the loop, so the commands between REPEAT ... UNTIL are executed at least one time, even if condition <> true at the beginning of the loop.

req
REQ returns the version number of the SmartPart interpreter. Syntax REQ (question) Parameter question: "SMT_Version"

request
REQUEST queries values or the status of the program or the SmartPart and stores the answers in variables. The return value of REQUEST is the number of answers found. Syntax REQUEST (question_text, name | index, variable1 [, variable2,...]) Parameter question_text (string): represents the question, the valid questions are predefined (see below) question_object (string or numeric): the object which is queried variable1,...,variablen: name of variables for storing the answers.   List of valid questions: Name_of_program: queries the name of the program and stores its value in a variable. Example: REQUEST ("name_of_program" , "" , name) stores the name of the program ("Allplan") in the variable 'name'. Name_of_macro: queries the name of the SmartPart and stores its value in a variable. Name_of_main: queries the name of the main subscript and stores its value in a variable (empty, if main subscript doesn't exist). Height_of_style: queries the total height of the style measured in millimeters (height in meters is height / 1000 * GLOB_SCALE); the descent (the distance in millimeters from the text base line to the descent line) and the leading (the distance in millimeters from the descent line to the ascent line). Example: REQUEST ("height_of_style", name, height [, descent, leading]) Fontnames_list: queries

the names of the available fonts (including character coding). The list or part of thereof can be used together with VALUES for a selection of fonts. The syntax is REQUEST ("FONTNAMES_LIST", type, fontnames), with 4 possible values for 'type': type="Standard": only Allplan fonts type="System": only Windows fonts type="Predefined": a subset of Windows fonts type="All": all fonts (Windows and Allplan) DateTime: queries the current date and time. Available formats are: %a short name of weekday (e.g. Tue) %A full name of weekday (e.g. Tuesday) %b short name of month (e.g. Oct) %B full name of month (e.g. October) %c date and time formatted like: 01:35:56 PM Tuesday, October 04, 2011 %d day of month as integer number (e.g 04) %H hour (24) as integer number (e.g. 14) %I hour (12) as integer numb(e.g. 02) %j day of the year as integer numb(e.g. 277) %m month of the year as integer numb(e.g. 10) %M minute as integer numb(e.g. 59) %S second as integer number (e.g. 59) %U number of the week as integer number, with Sunday as first day of the week (00-53) %w weekday as integer number (Sunday=0, Saturday=6) %W number of the week as integer number with Monday as first day of the week (00-53) %x date formatted as: (e.g. 10/04/2011) %X time formatted as: 01:35:56 %y year without century as integer number (e.g. 11) %Y year with century as integer number (e.g. 2011) %Z time zone as string %% the % character

resol

RESOL controls the appearance of arcs, circles and correlated forms or bodies by specification of a certain number of chords; the conversion of arcs is performed proportional. As soon as the command RESOL is used, the settings of prior used commands RADIUS and/or RISE are no longer valid. Syntax RESOL n Parameters n (integer): number of chords for the full circle parameter restrictions: n >= 3 default: n = 36

restore

RESTORE resets transformations of the coordinate system. Syntax RESTORE n [, begin_with] Parameters n (integer): number of transformations to reset. If begin_with is not set, the previous n entries are reset. If begin_with is set, n entries starting from begin_with are reset. If n is negative, the transformations are reset backwards. begin_with (integer, optional): sets the number, from which n entries are reset.

return

RETURN exits an internal subroutine, which was called with GOSUB, and continues the script. Syntax RETURN

revolve

REVOLVE generates a surface of revolution, which is described by a polyline defined in the XY plane rotating about the X axis. Arcs and segments can be defined within the polyline by using additional status code values. Syntax REVOLVE n, alpha, mask, x1, y1, s1, ... , xn, yn, sn Parameters n (integer): number of polyline nodes alpha (decimal, in degrees): rotation angle mask: controls the existence of the bottom, top and - in case of alpha is < 360 degrees - cut surfaces (see below) xi, yi (decimal): X and Y coordinates of the polyline nodes si (boolean): with si = -1 holes are defined directly parameter restrictions: n >= 2; yi >= 0; yi and y(i + 1) (i.e., the Y coordinate of two adjacent nodes) cannot be zero at the same time Masking mask = j1 + 2*j2 + 4*j3 + 8*j4 with j1, j2, j3, j4 may be 0 or 1 j1 (1): bottom surface is present j2 (2): top surface is present j3 (4): cut surface at initial angle is present j4 (8): cut surface at final angle is present

rise

RISE controls the appearance of arcs, circles and correlated forms or bodies by limiting the distance between the ideal arc and the generated chord to the value of d. As soon as the command RISE is used, the settings of prior used commands RADIUS and/or RESOL are no longer valid. Syntax RISE d Parameters d (decimal): maximum distance between the ideal arc and the generated chord

rnd

RND returns a random value between 0.0 and x. The value is recalculated with every run of the script. Syntax RND (x) Parameter x (decimal, >0)

rot

ROT rotates the local coordinate system around a vector, which is specified as a line from point 0,0,0 to point X,Y,Z. Syntax ROT x, y, z, alpha Parameters x, y, z (decimal): coordinates of the end point of the vector, around which the rotation is executed. alpha (decimal): angle in degrees. Positive values rotate counterclockwise. Remarks ROT has only one entry in the transformation stack, thus it can be deleted with RESTORE 1.

rot2

ROT2 rotates the local coordinate system around the point of origin. Only for 2D script. Syntax ROT2 alpha Parameters alpha (decimal): angle in degrees. Positive values rotate counterclockwise.

rotx

ROTX rotates the local coordinate system around the X axis. Syntax ROTX alpha Parameters alpha (decimal): angle in degrees. Positive values rotate counterclockwise.

roty

ROTY rotates the local coordinate system around the Y axis. Syntax ROTY alpha Parameters alpha (decimal): angle in degrees. Positive values rotate counterclockwise.

rotz

ROTZ rotates the local coordinate system around the Z axis. Syntax ROTZ alpha Parameters alpha (decimal): angle in degrees. Positive values rotate counterclockwise.

round_int

ROUND_INT returns the rounded integer part of x. The values are rounded half up. Syntax ROUND_INT (x) Parameters x (decimal): any number. The return value is an integer. The 'i = ROUND_INT (x)' expression is equivalent with the following script: IF x > 0.0 THEN i = INT (x - 0.5) ELSE i = INT (x + 0.5)

ruled

RULED generates a curved surface between a planar and a spatial polyline, both with identical numbers of nodes; the corresponding edges are connected by plane surfaces (each either a planar quadrangle or two triangles). RULED is together with RULED_2 the only SmartPart which allows the overlapping of adjacent nodes. Syntax RULED n, mask, u1, v1, s1, ... , un, vn, sn, x1, y1, z1, ... , xn, yn, zn Parameters n (integer): number of nodes in each polyline mask: controls the existence of the bottom, top and side surfaces (see below); the side surface connects the first and last nodes of the polylines, if any of them is not closed ui, vi (decimal): X and Y coordinates of the planar polyline nodes xi, yi, zi (decimal): X, Y and Z coordinates of the spatial polyline nodes parameter restrictions: n > 1 Masking mask = j1 + 2*j2 + 4*j3 with j1, j2, j3 may be 0 or 1 j1 (1): bottom surface is present j2 (2): top surface is present (not effective if the top surface is not planar) j3 (4): side surfaces are present (each either a planar quadrangle or two triangles)

ruled_2

RULED_2 generates a curved surface between a planar and a spatial polyline, both with identical numbers of nodes; the corresponding edges are connected by plane surfaces (each either a planar quadrangle or two triangles). The command is similar to RULED, but in contrast to RULED this command checks the definition order of the nodes in each polyline (clockwise or counterclockwise); if necessary the order is autoreversed for one of the polylines. RULED_2 is together with RULED the only SmartPart which allows the overlapping of adjacent nodes. Syntax RULED_2 n, mask, u1, v1, s1, ... , un, vn, sn, x1, y1, z1, ... , xn, yn, zn Parameters n (integer): number of nodes in each polyline mask: controls the existence of the bottom, top and side surfaces (see below); the side surface connects the first and last nodes of the polylines, if any of them is not closed ui, vi (decimal): X and Y coordinates of the planar polyline nodes xi, yi, zi (decimal): X, Y and Z coordinates of the spatial polyline nodes parameter restrictions: n > 1 Masking mask = j1 + 2*j2 + 4*j3 with j1, j2, j3 may be 0 or 1 j1 (1): bottom surface is present j2 (2): top surface is present (not effective if the top surface is not planar) j3 (4): side surfaces are present (each either a planar quadrangle or two triangles)

scale
SCALE scales the local coordinate system. Syntax SCALE x, y, z Parameters x, y, z (decimal): defines the value for scaling in X, Y and Z direction. Values < 1 compress, values > 1 stretch the coordinate system. Entering negative values mirrors the coordinate system. Remarks SCALE has only one entry in the transformation stack, thus it can be deleted with RESTORE 1.

scale2
SCALE2 scales the local coordinate system. Syntax SCALE2 x, y Parameters x, y (decimal): defines the value for scaling in x and y direction. Values < 1 compress, values > 1 stretch the coordinate system. Entering negative values mirrors the coordinate system.

scalex
SCALEX scales the local coordinate system in X direction. Syntax SCALEX x Parameters x (decimal): defines the value for scaling in X direction. Values < 1 compress, values > 1 stretch the coordinate system. Entering negative values mirrors the coordinate system.

scaley
SCALEY scales the local coordinate system in Y direction. Syntax SCALEX y Parameters y (decimal): defines the value for scaling in Y direction. Values < 1 compress, values > 1 stretch the coordinate system. Entering negative values mirrors the coordinate system.

scalez
SCALEZ scales the local coordinate system in Z direction. Syntax SCALEZ z Parameters z (decimal): defines the value for scaling in Z direction. Values < 1 compress, values > 1 stretch the coordinate system. Entering negative values mirrors the coordinate system.

set
SET is an optional command which may precede the commands for setting attributes. Syntax SET command Parameters command: MATERIAL, STYLE, FILL Remarks DEFINE STYLE "My_Style" Verdana, 3.0,8,0 SET STYLE "My_Style" is equal to DEFINE STYLE "My_Style" Verdana, 3.0,8,0 STYLE "My_Style"

sgn
SGN returns +1 if x is positive, or -1 if ix is negative, or 0 if x = 0 Syntax SGN (x) Parameters x (decimal): any number

sin

SIN returns the sine of x. Syntax SIN (x) Parameters x (decimal): argument for the sine (in degrees!) Remarks Possible return values: -1 <= SIN (x) <= 1

solid_beam
SOLID_BEAM is similar to SOLID_WALL_X and creates a beam. Syntax SOLID_BEAM left_material, right_material, vertical_material, top_material, bottom_material, height, x1, x2, x3, x4, y1, y2, y3, y4, t, mask1, mask2, mask3, mask4 Parameters left_material (string/integer): name/index of left surface right_material (string/integer): name/index of right surface vertical_material (string/integer): name/index of vertical surfaces top_material (string/integer): name/index of top surface bottom_material (string/integer): name/index of bottom surface height (decimal): height of beam (in Z direction) x1, x2, x3, x4 (decimal): X coordinates of beam's delimiting points y1, y2, y3, y4 (decimal): Y coordinates of beam's projected delimiting points (XY plane) t (decimal): width of beam (in Y direction) Remarks left_material is used for the whole solid.

solid_slab
SOLID_SLAB creates a polygonal prism; it is similar to PRISM. The side faces are perpendicular to XY plane. The bottom and top face can be inclined to XY plane. Syntax SOLID_SLAB n, h, x1, y1, z1, ... xn, yn, zn Parameters n (integer, >= 3): number of nodes defining the base polygon h (decimal): distance between bottom and top surface, always along z axis x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n z(n) (decimal): Z coordinate of node n Restrictions of parameters: n >= 3 Remarks Inclination is defined by differing z values, not by an angle. A negative h value can be used. Then the first polygon is above the second one, and the slab extrudes in negative Z direction. There is no check whether all nodes are on one plane. If they are not, this may cause strange effects with shading and rendering.

solid_slab_
SOLID_SLAB_ is similar to SOLID_SLAB and is an analogy to PRISM_. It creates a polygonal prism, but using status codes. Openings can be inserted, and the polylines can get segments and arcs using special constraints.   Syntax SOLID_SLAB_ n, h, x1, y1, z1, s1, ... xn, yn, zn, sn Parameters n (integer, >= 3): number of nodes defining the base polygon h (decimal): distance between bottom and top surface, always along z axis x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n z(n) (decimal): Z coordinate of node n s(n) (integer): not supported

solid_slab_c
SOLID_SLAB_C is similar to SOLID_SLAB_. In addition, you can define materials for top, bottom and side faces. Using status codes one can hide edges and side faces. Openings can be inserted, and the polylines can get segments and arcs using special constraints.   Syntax SOLID_SLAB_C top_material, bottom_material, side_material, n, h, x1, y1, z1, s1, ... xn, yn, zn, sn Parameters top_material (string/integer): name/index of top surface bottom_material (string/integer): name/index of bottom surface side_material (string/integer): name/index of side surfaces n (integer, >= 3): number of nodes defining the base polygon h (decimal): distance between bottom and top surface, always along z axis x(n) (decimal): X coordinate of node n y(n) (decimal): Y coordinate of node n z(n) (decimal): Z coordinate of node n Remarks top_material is used for the whole slab.

solid_wall_
SOLID_WALL_ creates perpendicular walls with materials and openings. Syntax SOLID_WALL_ left_material, right_material, side_material, height, x1, x2, x3, x4, t, mask1, mask2, mask3, mask4, n, x_start1, y_low1, x_end1, y_high1, frame_shown1, ... x_startn, y_lown, x_endn, y_highn, frame_shownn, m, a1, b1, c1, d1, ... am, bm, cm, dm Parameters left_material (string/integer): name/index of left surface right_material (string/integer): name/index of right surface side_material (string/integer): name/index of side surfaces height (real): height of wall from its bottom (in Z direction) x1, x2, x3, x4 (real): wall's end points projected on XY plane t (real): width of wall (in Y direction) With positive values the wall extrudes

to the right of the X axis, with negative values the wall extrudes to the left of the X axis. If t=0, the wall is shown as a polygon, and frames are drawn around the openings. n (integer, >= 0): number of openings in the wall x_starti, y_lowi, x_endi, y_highi (real): coordinates of wall's opening m (integer, >= 0): number of cutting planes ai, bi, ci, di (real): coefficients of the equation defining the cutting plane [ai*x + bi*y + ci*z = di] Parts on the positive side of the cutting plane will be cut and removed. Remarks left_material is used for the whole wall.

solid_wall_b

SOLID_WALL_B creates curved walls and uses the same data structure as SOLID_WALL_. In addition, you can define a radius. Syntax SOLID_WALL_B left_material, right_material, side_material, height, x1, x2, x3, x4, t, radius mask1, mask2, mask3, mask4, n, x_start1, y_low1, x_end1, y_high1, frame_shown1, ... x_startn, y_lown, x_endn, y_highn, frame_shownn, m, a1, b1, c1, d1, ... am, bm, cm, dm Parameters left_material (string/integer): name/index of left surface right_material (string/integer): name/index of right surface side_material (string/integer): name/index of side surfaces height (decimal): height of wall from it's bottom (in Z direction) x1, x2, x3, x4 (decimal): wall's end points projected on XY plane t (decimal): width of wall (in Y direction) With positive values the wall extrudes to the right of the X axis, with negative values the wall extrudes to the left of the X axis. If t=0, the wall is shown as a polygon, and frames are drawn around the openings. radius (decimal): radius for curve (cylinder tangential to XZ plane) Edges in X direction become arcs, edges in Y direction are radial, vertical edges are vertical. n (integer, >= 0): number of openings in the wall x_starti, y_lowi, x_endi, y_highi (decimal): coordinates of wall's opening m (integer, >= 0): number of cutting planes ai, bi, ci, di (decimal): coefficients of the equation defining the cutting plane [ai*x + bi*y + ci*z = di] Parts on the positive side of the cutting plane will be cut and removed. Remarks left_material is used for the whole wall. The wall's curve consists of segments. Their number is defined by the actual resolution (cf. RADIUS, RESOL and RISE).

solid_wall_x

SOLID_WALL_X uses the same data structure as SOLID_WALL_B. In addition, you can define more parameters, e.g. for log walls. Syntax SOLID_WALL_X left_material, right_material, vertical_material, horizontal_material, height, x1, x2, x3, x4, y1, y2, y3, y4, t, radius log_height, log_offset, mask1, mask2, mask3, mask4, n, x_start1, y_low1, x_end1, y_high1, frame_shown1, ... x_startn, y_lown, x_endn, y_highn, frame_shownn, m, a1, b1, c1, d1, ... am, bm, cm, dm status Parameters left_material (string/integer): name/index of left surface right_material (string/integer): name/index of right surface vertical_material (string/integer): name/index of vertical surfaces (not supported) horizontal_material (string/integer): name/index of horizontal surfaces (not supported) height (decimal): height of wall from its bottom (in Z direction) x1, x2, x3, x4 (decimal): wall's end points projected on XY plane y1, y2, y3, y4 (decimal): wall's end points projected on XY plane t (decimal): width of wall (in Y direction) With positive values the wall extrudes to the right of the X axis, with negative values the wall extrudes to the left of the X axis. If t=0, the wall is shown as a polygon, and frames are drawn around the openings. n (integer, >= 0): number of openings in the wall x_starti, y_lowi, x_endi, y_highi (decimal): coordinates of wall's opening m (integer, >= 0): number of cutting planes ai, bi, ci, di (decimal): coefficients of the equation defining the cutting plane [ai*x + bi*y + ci*z = di] Parts on the positive side of the cutting plane will be cut and removed. Remarks left_material is used for the whole wall. The wall's curve consists of segments. Their number is defined by the actual resolution (cf. RADIUS, RESOL and RISE).

solid_wall_x2

SOLID_WALL_X2 uses the same data structure as SOLID_WALL_B. In addition, you can define more parameters, e.g. for materials. Syntax SOLID_WALL_X2 left_material, right_material, vertical_material,horizontal_material, height, x1, x2, x3, x4, y1, y2, y3, y4, t, radius, log_height, log_offset, mask1, mask2, mask3, mask4, n, x_start1, y_low1, x_end1, y_high1, sill_depth1, frame_shown1, ... x_startn, y_lown, x_endn, y_highn, sill_depthn, frame_shownn, m, a1, b1, c1, d1, ... am, bm, cm, dm, status Parameters left_material (string/integer): name/index of left surface right_material (string/integer): name/index of right surface vertical_material (string/integer): name/index of vertical surfaces (not supported) horizontal_material (string/integer): name/index of horizontal surfaces (not supported) height (decimal): height of wall from its bottom (in Z direction) x1, x2, x3, x4 (decimal): wall's end points projected on XY plane y1, y2, y3, y4 (decimal): wall's end points projected on XY plane t (decimal): width of wall (in Y direction) With positive values the wall extrudes to the right of the X axis, with negative values the wall extrudes to the left of the X axis. If t=0, the wall is shown as a polygon, and frames are drawn around the openings. n (integer, >= 0): number of openings in the wall x_starti, y_lowi, x_endi, y_highi (decimal): coordinates of wall's opening m (integer, >= 0): number of cutting planes ai, bi, ci, di (decimal): coefficients of the equation defining the cutting plane [ai*x + bi*y + ci*z = di] Parts on the positive side of the cutting plane will be cut and removed. Remarks left_material is used for the whole wall.

sphere

SPHERE defines a ball.   Syntax SPHERE r Parameters r (decimal, >0): radius of ball   Remarks The ball's center point is in the origin.

spline2

SPLINE2 creates a spline based on n control points. Angles of tangents in spline control points must be stated. Syntax SPLINE2 n, status, x1, y1, angle1, ..., xn, yn, anglen Parameters n (integer, >= 2): number of nodes status (integer): defines the spline type 0: default 1: closed spline, with connected first and last control point 2: spline is smoothed automatically (angle parameters are not used) x(n) (decimal): X coordinate of control point n y(n) (decimal): Y coordinate of control point n angle (n) (decimal): angle of tangent in control point (in degrees)   Remarks The angular offset is based on the (local) X axis; positive direction is counterclockwise.

spline2_a

SPLINE2_A is an extension of SPLINE2 and creates a Bezier spline based on n control points. Tangent's angle direction in and tangent's length before and after the spline control points must be stated. Syntax SPLINE2 n, status, x1, y1, angle1, length_previous1, length_next1, ..., xn, yn, anglen, length_previousn Parameters n (integer, >= 2): number of nodes status (integer): defines the spline type 0: default 1: closed spline, with connected first and last control point 2: spline is smoothed automatically (angle, length_previousi and length_nexti parameter values of the nodes between the first and the last node are not used) x(n) (decimal): X coordinate of control point n y(n) (decimal): Y coordinate of control point n angle (n) (decimal): angle of tangent in control point (in degrees) length_previous (n) (decimal): length of tangent before control point length_next (n) (decimal): length of tangent before control point   Remarks The angular offset is based on the (local) X axis; positive direction is counterclockwise. Autosmoothing algorithm is used, angle, length_previousi and length_nexti parameter values of the nodes between the first and the last node are omitted.

split

SPLIT splits a text according to the format into several parts (numeric or string) and assigns these parts to one or more variables. The execution of SPLIT is halted, when the first non-matching part is encountered. SPLIT returns the number of successfully read values as integer. Syntax SPLIT (string, format, variable1 [, variable2, ..., variablen]) Parameter string (string): the string which should be split. format: any combination of constant strings, %s and %n -s. Parts in the string must fit the constant strings, %s identifies any string value, which is delimited by spaces or tabs, while %n identifies any numeric value. variablei (string or decimal, depends on format): the names of the variable, into which the split strings are stored. Remarks Decimal numbers must be separated by ".", otherwise only the places before the decimal point are considered.

sqr

SQR returns the square root of x (decimal). Syntax SQR (x) Parameter x (decimal): any number

str

STR converts a numeric value to a string and returns the string value. There are 2 variations of the STR command: with the first one you can define the total number of characters and the number of characters after the floating point. With the second variation you can define the format in detail and also convert the result to another unit (e.g. m to inch). Syntax STR (numeric_expression, length, fractions) or: STR (format_string, numeric_expression) Parameter numeric_expression: any numeric expression. length (integer, <> 0): total number of characters. fractions (integer, 0 <= fractions < length): number of characters after the floating point. format_string: with format_string the result can be formatted in detail. The syntax is: %[0 or more flags][length][.decimal_place] conv_spec The parameters are as below: %: obligatory flags (for m, mm, cm, e, df, di, sqm, sqcm, sqf, sqi, dd, gr, rad, cum, l, cucm, cumm, cuf, cui, cuy, gal): (none): right aligned (default) -: left aligned +: explicit plus sign for values > 0 space: blanks in place of a + sign for values > 0 flags (for m, mm, cm, df, di, sqm, sqcm, sqf, sqi, dd, fr, rad, cum, l, cucm, cumm, cuf, cui, cuy, gal): #don't display 0 wholes flags (for ffi, fdi, fi): 0display 0 inches flags (for m, mm, cm, fdi, df, di, sqm, sqcm, sqf, sqi, dd, fr, rad, cum, l, cucm, cumm, cuf, cui, cuy, gal): ~hide 0 decimals (effective only if the '#' flag is not specified) ^: do not change decimal separator and digit grouping characters (if not specified, these characters will be replaced as set in the current system) length: unsigned decimal integer, the minimum number of characters to generate decimal_place: the number of decimal places conv_spec (conversion specifier): e: exponential format (meter) m: meters mm: millimeters cm: centimeters ffi: feet and fractional inches fdi: feet and decimal inches df: decimal feet fi: fractional inches di: decimal inches for areas: sqm: square meters sqcm: square centimeters sqmm: square millimeters sqf: square feet sqi: square inches for angles: dd: decimal degrees dms: degrees, minutes, seconds gr: grads rad: radians surv: surveyors unit for volumes: cum: cubic meters l: liters cucm: cubic centimeters cumm: cubic millimeters cuf: cubic feet cui: cubic inches cuy: cubic yards gal: gallons

str_2

STR_2 converts a numeric value to a string and returns the string value. STR_2 is identical to STR with the additional possibility tp specify the accuracy. There are 2 variations of the STR_2 command: with the first one you can define the total number of characters and the number of characters after the floating point. With the second variation you can define the format in detail and also convert the result to another unit (e.g. m to inch). Syntax STR_2 (numeric_expression, length, fractions) or: STR_2 (format_string, numeric_expression) Parameter numeric_expression: any numeric expression. length (integer, <> 0): total number of characters. fractions (integer, 0 <= fractions < length): number of chracters after the floating point. format_string: with format_string the result can be formatted in detail. The syntax is: %[0 or more flags][accuracy][length][.decimal_place] conv_spec The parameters are as below: %: obligatory flags (for m, mm, cm, e, df, di, sqm, sqcm, sqf, sqi, dd, gr, rad, cum, l, cucm, cumm, cuf, cui, cuy, gal): (none): right aligned (default) -: left aligned +: explicit plus sign for values > 0 space: blanks in place of a + sign for values > 0 * 0extra accuracy Off (default) * 1extra accuracy .5 * 2extra accuracy .25 * 3extra accuracy .1 * 4extra accuracy .01 * 5extra accuracy .5 * 6 extra accuracy .25 flags (for m, mm, cm, df, di, sqm, sqcm, sqf, sqi, dd, fr, rad, cum, l, cucm, cumm, cuf, cui, cuy, gal): #don't display 0 wholes flags (for ffi, fdi, fi): 0display 0 inches flags (for m, mm, cm, fdi, df, di, sqm, sqcm, sqf, sqi, dd, fr, rad, cum, l, cucm, cumm, cuf, cui, cuy, gal): ~hide 0 decimals (effective only if the '#' flag is not specified) ^: do not change decimal separator and digit grouping characters (if not specified, these characters will be replaced as set in the current system) length: unsigned decimal integer, the minimum number of characters to generate conv_spec (conversion specifier): e: exponential format (meter) m: meters mm: millimeters cm: centimeters ffi: feet and fractional inches fdi: feet and decimal inches df: decimal feet fi: fractional inches di: decimal inches for areas: sqm: square meters sqcm: square centimeters sqmm: square millimeters sqf: square feet sqi: square inches for angles: dd: decimal degrees dms: degrees, minutes, seconds gr: grads rad: radians surv: surveyors unit for volumes: cum: cubic meters l: liters cucm: cubic centimeters cumm: cubic millimeters cuf: cubic feet cui: cubic inches cuy: cubic yards gal: gallons decimal_place: the number of decimal places

strlen

STRLEN returns the number of characters (including blanks) in string_expression as an integer number. Syntax STRLEN (string_expression) Parameter string_expression: the string which characters should be counted.

stroke
STROKE defines the type of the stroke for an element. Syntax STROKE id Parameters id (integer): identification number of the stroke with -2 < id <= 255 id = -1 (default): the current stroke is used id = -2: the stroke is set "by Layer" You can also use constants instead: CURR = -1 BY_LAYER = -2

strstr
STRSTR returns the position of the first appearance of search_text in source_text. If search_text is not found, 0 is returned. The search is case sensitive. Syntax STRSTR (source_text, search_text) Parameter source_text, search_text (string)

strsub
STRSUB returns a part of a string (a substring). The starting point and the number of characters of the substring can be defined. Syntax STRSUB (string, start, number) Parameter string (string): string which should be evaluated start (integer): starting point, where the substring starts number (integer): number of characters for the substring

stw
STW returns the approximate width of a string in meter according to the current style (which has been set with SET STYLE). Syntax STW (string) Parameter string: any string expression Remarks You receive the width at current scale, with STW (string_expression) / 1000 * GLOB_SCALE.

style
DEFINE STYLE defines styles for text fonts. It should be used with TEXT2 and TEXT. Style definitions prior to the definition of this style may be included in any script; however, styles defined that way can be used solely in those scripts, in which they are defined, and their subsequent second generation-scripts. Syntax DEFINE STYLE name font_family, size, anchor, face_code Parameters name (string): name of the style font_family (string): name of the font family used, e.g. Arial size (decimal): height of the "l" character in mm with TEXT2 and TEXT, size means character height in mm anchor (integer, 0< anchor < 10): code for text position face_code (integer): not supported, setting is ignored Corresponding commands [set] STYLE name_string [set] STYLE index All texts generated after a style setting will obtain this style, until the next style is set. Index is a constant referring to a style stack in the internal data structure; this stack is modified during script analysis and can also be modified from within the program. Any use of the index instead of the style's name is only recommended with the prior use of the INDEX function. Remarks If there is no style set in a script, the following default is used: SET STYLE 0 (application font, size 5 mm, anchor = 1, normal face)

sweep
SWEEP generates a tubular surface by displacement of a polylined contour along a spatial polyline (= path curve); in order to get a twisted tube with a tapering or dilating cross-section, the polylined contour may be scaled and/or rotated during displacement. The plane of the polylined contour follows the spatial polyline determining the path: at node (xi, yi, zi) the corresponding plane is perpendicular to the spatial polyline segment between the nodes (xi-1, yi-1, zi-1) and (xi, yi, zi). The path determining polyline must have its beginning in the XY plane. If this condition is not fulfilled, the polyline will be moved along the Z axis automatically, until the condition is fulfilled. Syntax SWEEP n, m, alpha, scale, mask, u1, v1, s1, ... , un, vn, sn, x1, y1, z1, ... ,xm, ym, zm Parameters n (integer): number of polyline nodes m (integer): number of path curve nodes alpha: incremental rotation of the polylined contour from one node to the next one in its

own plane scale: incremental scale factor of the polylined contour from one node to the next one mask: controls the existence of the bottom, top and side surfaces (see below) ui, vi (decimal): X and Y coordinates of the polyline nodes xi, yi, zi: X, Y and Z coordinates of the path curve nodes parameter restrictions: n > 1, m > 1, z1 < z2 Masking mask = j1 + 2*j2 + 4*j3 with j1, j2, j3 may be 0 or 1 j1 (1): bottom surface is present j2 (2): top surface is present j3 (4): side surfaces are present

symb_mirrored
SYMB_MIRRORED returns the current mirror status of the SmartParts object (0 if not mirrored, 1 if mirrored).

symb_pos_x
SYMB_POS_X returns the current x-coordinate of the reference box (calculated from global origin).

symb_pos_y
SYMB_POS_Y returns the current y-coordinate of the reference box (calculated from global origin).

symb_pos_z
SYMB_POS_Z returns the current z-coordinate of the reference box (calculated from global origin).

symb_rotangle
SYMB_ROTANGLE returns the current rotation of the SmartParts object.

tan
TAN returns the tangent of x. Syntax TAN (x) Parameters x (decimal): argument for the tangent (in degrees!)

text
TEXT is similar to TEXT2 and creates 3-dimensional texts in the set style. Texts start from the local origin. Syntax TEXT d, flag, expression Parameters d (decimal): text thickness in Z direction flag = 0: text height from STYLE is interpreted in mm flag = 1: text height is similar to TEXT2 absolute height (in m) = scale*height/1000) expression (decimal, string): text or value or the result of a mathematical formula. Remarks Use [SET] STYLE rsp. DEFINE STYLE for text parameters as font name, size, anchor point etc.

text2
TEXT2 creates text in a defined style at a defined starting point. Syntax TEXT2 x, y, expression Parameters x (decimal): X coordinate of starting point y (decimal): Y coordinate of starting point expression (decimal, string): text or value or the result of a mathematical formula. Maximum string length is 255 characters. Remarks Use [SET] STYLE rsp. DEFINE STYLE for text parameters as font name, size, anchor point etc.

texture
DEFINE TEXTURE defines a texture, which can be referred to by name further on in other scripts. Texture definitions prior to the definition of this texture may be included in any script; however, textures defined that way can be used solely in those scripts, in which they are defined, and their subsequent second generation-scripts. Syntax DEFINE TEXTURE name expression, x, y, mask, angle Parameters name (string): name of the texture, expression (string): bitmap used for the texture, e.g. a file name or an index number of a bitmap stored in the SmartPart. A 0 index value refers to the preview picture of the library

part. x (decimal, > 0): texture width in meter y (decimal, > 0): texture height in meter mask (integer, 0 <= mask <= 511): appearance of texture Alpha channel controls (j1... j6): j1 can be 0 or 1 j1: alpha channel changes the transparency of texture Corresponding commands [set] TEXTURE name_string [set] TEXTURE index All 2D polygons generated after a texture setting will obtain this texture, until the next texture is set. Index is a constant referring to a fill stack in the internal data structure; this stack is modified during script analysis and can also be modified from within the program. Any use of the index instead of the texture's name is only recommended with the prior use of the INDEX function. Remarks If there is no texture set in a script, an empty texture rsp. no texture is used as default: SET TEXTURE -1

then

THEN is part of an IF...THEN...ELSE...ENDIF construction. IF executes the following SmartPart Script commands depending on a condition. Syntax IF condition THEN command IF condition THEN command1 command2 ELSE ENDIF Parameters condition: if condition=true (<>0) all commands after THEN are executed. Otherwise the commands after ELSE are executed. If ELSE is absent, the commands after ENDIF are executed. command: any SmartPart Script command Remarks If the command after THEN is GOTO or GOSUB, you can omit the THEN statement and shorten to IF condition GOTO label. If there is only 1 command after THEN or ELSE in the same row, you can omit ENDIF.

to

TO is part of a FOR...NEXT loop. Syntax FOR variable=start_value TO end_value [STEP step_value] ... NEXT Parameters variable: name of the variable. Only local variables are allowed, no global variables. start_value (decimal): initial value of the variable end_value (decimal): value, till which the loop is executed. Before a loop is started, it is checked, wether end_value is reached. step_value (decimal): value by which start_value is increased with every cycle of the loop. If not specified, start_value is increased by 1. step_value can be negative also. A change of step_value inside the loop is ignored.

trans

TRANS moves the local coordinate system along the X, Y and Z axis. Syntax TRANS x, y, z Parameters x, y, z (decimal) Remarks TRANS has only one entry in the stack, therefore you can delete it with RESTORE 1.

trans2

TRANS2 moves the local coordinate system along the X and Y axis. Syntax TRANS2 x, y Parameters x, y (decimal)

transx

TRANSX moves the local coordinate system along the X axis. Syntax TRANSX x Parameters x (decimal)

transy

TRANSY moves the local coordinate system along the Y axis. Syntax TRANSY y Parameters y (decimal)

transz

TRANSZ moves the local coordinate system along the Z axis. Syntax TRANSX z Parameters z (decimal)

tube

TUBE generates a tubular surface by displacement of a closed polylined contour along a spatial polyline (= path curve). In each joint of the path curve the polylined contour is situated in the bisector plane of the joint segments. At the middle of the path segments the polylined contour is always identical with the polylined contour at the beginning of the displacement (the coordinates u1, w1, ... , un, wn remain unchanged). Arcs and segments can be defined within the polyline by using additional status code values. (See Remarks section for details.)   Syntax TUBE n, m, mask, u1, w1, s1, ... , un, wn, sn, x1, y1, z1, angle1, ... , xm, ym, zm, anglem Parameters n (integer): number of polyline nodes m (integer): number of path curve nodes mask: controls the existence of the surfaces at the beginning and the end of the tube (see below) ui, wi: U and W coordinates of the base polyline nodes in the local U-V-W coordinate system si (boolean): controls the appearance of the lateral edges; if si = 0 the line segment between ni and ji will not be drawn, if si = 1 this line segment will be drawn xi, yi, zi: X, Y and Z coordinates of the path curve nodes Important: The path curve comprises two nodes more than the number of generated sections. The first and the last node determine the position in space of the first and the last surface belonging to the tube. These nodes are relevant solely for determining the normal of the surfaces, they are no actual nodes of the path curve. The orientation of the surfaces is the same as that of the surfaces that would be generated at the nodes nearest to the two endpoints, if the tube was continued in the directions indicated by these. anglei: rotation angle of the cross-section parameter restrictions: n > 2, m > 3 Masking mask = j1 + 2*j2 with j1, j2 may be 0 or 1 j1 (1): surface at the beginning of the tube is present j2 (2): surface at the end of the tube is present Remarks The polylined contour must be closed. The generating cross-section is not distorted or resized. However, the internal connection surfaces are rotatable about the V axis of the local UVW coordinate system. The local UVW coordinate system has its origin in the current point with its V axis is congruent with the tangent of the spatial polyline at the current point (= tangential point) W axis is perpendicular to the V axis and points upward with respect to the local Z axis (If the V axis is vertical, then the W direction is not correctly defined) U axis is perpendicular to the VW plane; together they form a right-hand sided Cartesian coordinate system. For determining a horizontal direction, the W axis in the previous path node is used.

tube_a

TUBE_A generates a tubular surface by displacement of a polylined contour along a spatial polyline (= path curve). In each joint of the path curve the polylined contour is situated in the bisector plane of the projections of the joint segments to the local XY plane; the cross-section is identical with the polylined contour at the beginning of the displacement (the coordinates u1, w1, ... , un, wn remain unchanged). Arcs and segments can be defined within the polyline by using additional status code values.   Syntax TUBE_A n, m, mask, u1, w1, s1, ... , un, wn, sn, x1, y1, z1, ... , xm, ym, zm Parameters n (integer): number of polyline nodes m (integer): number of path curve nodes mask: controls the existence of the surfaces at the beginning and the end of the tube (see below) ui, wi: U and W coordinates of the base polyline nodes in the UW coordinate system si (boolean): controls the appearance of the lateral edges; if si = 0 the line segment between ni and ji will not be drawn, if si = 1 this line segment will be drawn xi, yi, zi: X, Y and Z coordinates of the path curve nodes Important: The path curve comprises two nodes more than the number of generated sections. The first and the last node determine the position in space of the first and the last surface belonging to the tube. These nodes are relevant solely for determining the normal of the surfaces, they are no actual nodes of the path curve. The orientation of the surfaces is the same as that of the surfaces that would be generated at the nodes nearest to the two endpoints, if the tube was continued in the directions indicated by these. parameter restrictions: n > 2, m > 3 Masking mask = j1 + 2*j2 with j1, j2 may be 0 or 1 j1 (1): surface at the beginning of the tube is present j2 (2): surface at the end of the tube is present Remarks In contrast to TUBE the polylined contour in the middle of each path segment may be different from the polylined contour at the beginning of the displacement; furthermore the polylined contour must not be closed: in this case the section polygons will be generated to reach the local XY plane as in the case of REVOLVE surfaces.

until

UNTIL ends a REPEAT ... UNTIL loop. Syntax REPEAT ... UNTIL condition Parameters condition: Remarks Commands between REPEAT ... UNTIL are executed, as long as condition = true. condition is checked after the loop, so the commands between REPEAT ... UNTIL are executed at least one time, even if condition <> true at the beginning of the loop.

use

USE reads values from the parameter buffer without deleting them. The parameters are read from the buffer in the order they have been added with the PUT command. Syntax USE (n) Parameter n: number of parameters to read from the parameter buffer Remarks USE is a function and therefore can not be used standalone. If n=1, you can assign the returned value to a variable. If n>1, USE has to be used as an argument for another command.

values

VALUES provides values or value ranges for the parameters of an object. Syntax VALUES "name" value_definition1 [, value_definition2, ...] Parameters name: the name of the parameter, for which you define the values value_definitioni: value definition, can be: expressioni: numerical or string expression CUSTOM: keyword, meaning that any custom value can be entered RANGE left_delimiter [expression1], [expression2] right_delimiter [STEP step_start_value, step_value] range definition, with optional step left_delimiter: there are 2 possibilities for the left delimiter: [(meaning "<=") ((meaning "<") expression1: lower limit expression expression2: upper limit expression right_delimiter: there are 2 possibilities for the right delimiter:] (meaning ">=")) (meaning ">") step_start_value: starting value step_value: step value

vardim1

VARDIM1 returns the value for the current 1. dimension of an array (lines). Syntax VARDIM1 (var) Parameter var: name of a valid variable. Remarks With dynamical arrays the value depends on the last used index for this dimension. If the variable has not yet been accessed, VARDIM1 returns 0.

vardim2

VARDIM2 returns the value for the current 2. dimension of an array (rows). Syntax VARDIM2 (var) Parameter var: name of a valid variable. Remarks With dynamical arrays the value depends on the last used index for this dimension. If the variable has not yet been accessed, or when the array is one-dimensional, VARDIM2 returns 0.

vartype

VARTYPE returns the type (string or numerical) of a variable. Returns 1 if the variable has a numerical value, 2 if it is a string. This command can be useful to check the type of variables after splitting up a text with SPLIT. Syntax VARTYPE (expression) Parameters expression: name of a variable

vert

VERT generates a node in the three-dimensional space, defined by its X, Y and Z coordinate. Syntax VERT x, y, z Parameters x (decimal): X coordinate of the node y (decimal): Y coordinate of the node z (decimal): Z coordinate of the node

while

WHILE ends a DO ... WHILE loop or starts a WHILE ... DO ... ENDWHILE loop. Syntax DO ... WHILE condition WHILE condition DO ... ENDWHILE Parameters condition: Remarks Commands between DO ... WHILE are executed, as long as condition = true. condition is checked after the loop, so the commands between DO ... WHILE are executed at least one time, even if condition <> true. Commands between WHILE ... ENDWHILE are executed, as long as condition = true. condition is checked before the loop, so the commands between WHILE ... ENDWHILE are executed only, if condition = true at the beginning of the loop.

while ... do ... endwhile

WHILE ... DO ... ENDWHILE loop is executed as long as condition = true. Syntax WHILE condition DO ... ENDWHILE Parameters condition: Remarks Commands between WHILE ... ENDWHILE are executed, as long as condition = true. condition is checked before the loop, so the commands between WHILE ... ENDWHILE are executed only, if condition = true at the beginning of the loop.

xform

XFORM defines a complete transformations matrix. You can use it to generate code automatically. Syntax XFORM a11, a12, a13, a14, a21, a22, a23, a24, a31, a32, a33, a34 Representation rules: x' = a11 * x + a12 * y + a13 * z + a14 y' = a21 * x + a22 * y + a23 * z + a24 z' = a31 * x + a32 * y + a33 * z + a34 Remarks XFORM has only one entry in the stack.