

GDL Reference Guide

Graphisoft

Visit the Graphisoft website at <http://www.graphisoft.com> for local distributor and product availability information.

GDL Reference Guide

Copyright © 2004 by Graphisoft, all rights reserved. Reproduction, paraphrasing or translation without express prior written permission is strictly prohibited.

Trademarks

ArchiCAD and ArchiFM are registered trademarks and PlotMaker, Virtual Building, StairMaker and GDL are trademarks of Graphisoft. All other trademarks are the property of their respective holders.

Introduction

This manual is a complete reference for Graphisoft's proprietary scripting language, GDL (Geometric Description Language). The manual is recommended for those users who wish to expand on the possibilities presented by the construction tools and object libraries available in Graphisoft software. It gives a detailed description of GDL, including syntax definition, commands, variables, etc.

CONTENTS

General Overview	13	SPHERE.....	34
Starting Out	13	ELLIPS.....	34
Scripting	13	CONE.....	35
Library Part Structure.....	13	PRISM.....	35
Analyze, Deconstruct and Simplify.....	14	PRISM_.....	36
Elaboration.....	15	CPRISM_.....	39
Entry Level.....	15	BPRISM_.....	40
Intermediate Level.....	16	FPRISM_.....	41
Advanced Level.....	18	HPRISM_.....	43
Expert Level.....	19	SPRISM_.....	43
3D Generation	19	SLAB.....	45
The 3D Space.....	20	SLAB_.....	45
Coordinate Transformations.....	20	CSLAB.....	46
The GDL Interpreter.....	20	CWALL_.....	46
The GDL Script Analysis.....	21	BWALL_.....	48
GDL Syntax	23	XWALL_.....	50
Statements	23	XWALL_{2}.....	52
Line	23	BEAM.....	53
Label	23	CROOF_.....	54
Characters	23	MESH.....	57
Strings	24	ARMC.....	58
Identifiers	24	ARME.....	59
Variables	24	ELBOW.....	59
Parameters	25	Planar Shapes in 3D	60
Simple Types	25	HOTSPOT.....	60
Derived Types	25	LIN_.....	60
Coordinate Transformations	27	RECT.....	61
2D Transformations	27	POLY.....	61
3D Transformations	28	POLY_.....	61
Managing the Transformation Stack	31	PLANE.....	62
3D Shapes	33	PLANE_.....	62
Basic Shapes	33	CIRCLE.....	62
BLOCK.....	33	ARC.....	63
BRICK.....	33	Shapes Generated from Polylines	63
CYLIND.....	33	EXTRUDE.....	65
		PYRAMID.....	68

REVOLVE	70	SWEEPGROUP	115
RULED	73	Binary 3D	115
RULED{2}	74	2D Shapes	117
SWEEP	76	Drawing Elements	117
TUBE	79	HOTSPOT2	117
TUBEA	83	LINE2	117
COONS	85	RECT2	118
MASS	88	POLY2	118
Elements for Visualization	90	POLY2_	119
LIGHT	90	POLY2_A	119
PICTURE	92	POLY2_B	120
3D Text Elements	93	POLY2_B{2}	120
TEXT	93	ARC2	121
RIGHTTEXT	94	CIRCLE2	121
Primitive Elements	94	SPLINE2	122
VERT	95	SPLINE2A	123
TEVE	95	PICTURE2	124
VECT	95	PICTURE2{2}	125
EDGE	96	Text Element	125
PGON	96	TEXT2	125
PIPG	97	RIGHTTEXT2	125
COOR	97	Binary 2D	126
BODY	99	FRAGMENT2	126
BASE	101	FRAGMENT2	126
Cutting in 3D	101	3D Projections in 2D	126
CUTPLANE	101	PROJECT2	126
CUTPOLY	103	PROJECT2{2}	127
CUTPOLYA	105	Drawings in the List	128
CUTSHAPE	107	DRAWING2	128
CUTFORM	107	DRAWING3	128
Solid Geometry Commands	109	DRAWING3{2}	129
GROUP	112	Graphical Editing	131
ENDGROUP	112	Hotspot-based Editing Commands	131
ADDGROUP	112	HOTSPOT	131
SUBGROUP	112	HOTLINE2	135
ISECTGROUP	113	HOTARC2	136
ISECTLINES	113	Status Codes	137
PLACEGROUP	113	Status Code Syntax	137
KILLGROUP	113		

Additional Status Codes 139

Previous part of the polyline: current position and tangent is defined
139

Segment by absolute endpoint..... 139

Segment by relative endpoint..... 140

Segment by length and direction..... 140

Tangential segment by length..... 141

Set start point..... 141

Close polyline..... 142

Set tangent..... 142

Set centerpoint..... 143

Tangential arc to endpoint..... 143

Tangential arc by radius and angle..... 144

Arc using centerpoint and point on the final radius..... 144

Arc using centerpoint and angle..... 145

Full circle using centerpoint and radius..... 145

Attributes 149

Directives 149

Directives for 3D and 2D Scripts..... 149

 [LET]..... 149

 RADIUS..... 150

 RESOL..... 151

 TOLER..... 152

 PEN..... 152

 LINE_PROPERTY..... 153

 [SET] STYLE..... 153

 SET STYLE 0..... 153

Directives Used in 3D Scripts Only..... 153

 MODEL..... 153

 [SET] MATERIAL..... 154

 SECT_FILL..... 155

 SHADOW..... 156

Directives Used in 2D Scripts Only..... 157

 DRAWINDEX..... 157

 [SET] FILL..... 157

 [SET] LINE_TYPE..... 158

Inline Attribute Definition..... 158

Materials..... 158

 DEFINE MATERIAL..... 158

 DEFINE MATERIAL BASED_ON..... 161

 DEFINE TEXTURE..... 162

Fills..... 164

 DEFINE FILL..... 164

 DEFINE FILLA..... 167

 DEFINE SYMBOL_FILL..... 169

 DEFINE SOLID_FILL..... 170

 DEFINE EMPTY_FILL..... 170

Line Types..... 170

 DEFINE LINE_TYPE..... 170

 DEFINE SYMBOL_LINE..... 171

Styles..... 171

 DEFINE STYLE..... 171

 DEFINE STYLE {2}..... 172

Paragraph..... 173

Textblock..... 174

Additional Data..... 175

 External file dependence..... 176

Non-Geometric Scripts 177

The Properties Script..... 177

DATABASE_SET..... 177

DESCRIPTOR..... 178

REF DESCRIPTOR..... 178

COMPONENT..... 178

REF COMPONENT..... 179

BINARYPROP..... 179

SURFACE3D ()..... 179

VOLUME3D ()..... 179

POSITION..... 179

DRAWING..... 180

The Parameter Script..... 180

VALUES..... 181

PARAMETERS..... 182

LOCK..... 183

HIDEPARAMETER..... 183

The User Interface Script..... 183

UI_DIALOG..... 183

UI_PAGE..... 183

UI_BUTTON..... 184

UI_SEPARATOR	184	Statistical Functions	195
UI_GROUPBOX	184	MIN	195
UI_PICT	184	MAX	195
UI_STYLE	185	RND	195
UI_OUTFIELD	185	Bit functions	195
UI_INFIELD	185	BITTEST	195
UI_INFIELD {2}	186	BITSET	195
Expressions and Functions	189	Special Functions	196
Expressions	189	String Functions	196
DIM	189	STR	196
VARDIM1(expr)	190	STR	196
VARDIM2(expr)	190	STR{2}	196
Operators	192	SPLIT	199
Arithmetical Operators	192	STW	200
Relational Operators	192	STRLEN	200
Boolean Operators	192	STRSTR	200
Functions	193	STRSUB	201
Arithmetical Functions	193	Control Statements	203
ABS	193	Flow Control Statements	203
CEIL	193	FOR	203
INT	193	NEXT	203
FRA	193	DO	204
ROUND_INT	193	IF	206
SGN	193	GOTO	207
SQR	193	GOSUB	207
Circular Functions	194	RETURN	207
ACS	194	END / EXIT	208
ASN	194	BREAKPOINT	208
ATN	194	Parameter Buffer Manipulation	208
COS	194	Macro Objects	212
SIN	194	The Output Statement	213
TAN	194	File Operations	213
PI	194	OPEN	214
Transcendental Functions	194	INPUT	214
EXP	194	VARTYPE	214
LGT	194	OUTPUT	214
LOG	195	CLOSE	214
Boolean Functions	195	Miscellaneous	215
NOT	195		

Global Variables	215	3D Use Only	257
General environment information	215	2D Use Only	258
Story information	215	2D and 3D Use	258
Fly-through information	216	Non-Geometric Scripts	258
General element parameters	217	<i>Property Script</i>	258
Object, Lamp, Door, Window parameters	217	<i>Parameter Script</i>	259
Object, Lamp parameters	218	<i>Interface Script</i>	259
Object, Lamp, Door, Window parameters - available for listing and labels only	218	Alphabetical List of Current GDL Keywords	260
Object, Lamp parameters - available for listing and labels only	218	A	260
Window, Door and Wall End parameters	219	B	260
Window, Door parameters - available for listing and labels only	220	C	261
Lamp parameters - available for listing and labels only	220	D	264
Label parameters	221	E	266
Wall parameters - available for Doors/Windows	222	F	266
Wall parameters - available for listing and labels only	223	G	267
Column parameters - available for listing and labels only	224	H	267
Beam parameters - available for listing and labels only	226	I	267
Slab parameters - available for listing and labels only	227	K	268
Roof parameters - available for listing and labels only	228	L	268
Fill parameters - available for listing and labels only	229	M	268
Mesh parameters - available for listing and labels only	229	N	269
Free users' globals	230	O	269
Old Global Variables	232	P	270
Requests	233	R	272
REQ	233	S	274
REQUEST	234	T	276
Doors and Windows	242	U	277
General Guidelines	242	V	277
Creation of Door/Window Library Parts	244	W	278
Rectangular Doors/Windows in Straight Walls	244	X	279
Non-Rectangular Doors/Windows in Straight Walls	246	Parameter Naming Convention	280
WALLHOLE	246	GDL Data I/O Add-On	280
WALLNICHE	250	Description of Database	280
Rectangular Doors/Windows in Curved Walls	251	Opening a Database	280
Non-Rectangular Doors/Windows in Curved Walls	253	Reading Values from Database	282
GDL Created from the Floor Plan	255	Writing Values into Database	283
Keywords	256	Closing Database	283
Common Keywords	256	GDL DateTime Add-On	284
Reserved Keywords	256	Opening Channel	284
		Reading Information	286

Closing Channel	286	Property GDL Add-On	292
GDL File Manager I/O Add-On	286	OPEN292
Specifying Folder	286	CLOSE292
Getting File/Folder Name	287	INPUT293
Finishing Folder Scanning	287	OUTPUT296
GDL Text I/O Add-On	288	GDL XML Extension	296
Opening File	288	Opening XML Document297
Reading Values	289	Reading XML Document298
Writing Values	290	Modifying XML Document301
Closing File	291	Index	305

GENERAL OVERVIEW

GDL is a parametric **programming language**, similar to BASIC. It describes 3D solid objects like doors, windows, furniture, structural elements, stairs, and the 2D symbols representing them on the floor plan. These objects are called **library parts**.

STARTING OUT

The needs of your design, your background in programming and your knowledge of descriptive geometry will all probably influence where you start in GDL.

Do not start practicing GDL with complicated objectives in mind. Rather, try to learn GDL through experimenting step by step with all of its features to best utilize them to your advantage. Follow the expertise level recommendations below.

If you are familiar with a programming language like BASIC, you can get acquainted with GDL by observing existing scripts. You can also learn a lot by opening the library parts shipped with your software and taking a look at the 2D and 3D GDL scripts. Additionally, you can save floor plan elements in GDL format and see the resulting script.

If you are not familiar with BASIC, but have played with construction blocks, you can still find your way in GDL through practice. We advise trying the simplest commands right away and then checking their effect in the 3D window of the library part.

For details about the library part editing environment, see “Parametric Objects” on page 199.

Graphisoft has published several books on GDL programming and object library development. Object Making With ArchiCAD is the perfect guide for beginners. David Nicholson Cole’s GDL Cookbook is the most popular course book for entry level and advanced GDL programmers. GDL Technical Standards contains Graphisoft’s official standards for professional library developers; this document can be downloaded free of charge from Graphisoft’s website.

SCRIPTING

Library Part Structure

Every library part described with GDL has **scripts**, which are lists of the actual GDL commands that construct the 3D shape and the 2D symbol. Library parts also have a description for quantity calculations in ArchiCAD.

Master script commands will be executed before each script.

The **2D script** contains parametric 2D drawing description. The **binary 2D data** of the library part (content of the 2D symbol window) can be referenced using the FRAGMENT2 command. If the 2D script is empty, the binary 2D data will be used to display the library part on the floor plan.

The **3D script** contains a parametric 3D model description. The **binary 3D data** (which is generated during an import or export operation) can be referenced using the BINARY command.

The **properties script** contains components and descriptors used in element, component and zone lists. The **binary properties data** described in the **components** and **descriptors** section of the library part can be referenced using the BINARYPROP command. If the properties script and the master script are empty, the binary properties data will be used during the list process.

The **User Interface** script allows the user to define input pages that can be used to edit the parameter values in place of the normal parameter list.

In the **parameter script**, sets of possible values can be defined for the library part parameters.

The parameter set in the **parameters** section are used as defaults in the library part settings when placing the library part on the plan.

The **preview picture** is displayed in the library part settings dialog box when browsing the active library. It can be referenced by the PICTURE and PICTURE2 commands from the 3D and 2D script.

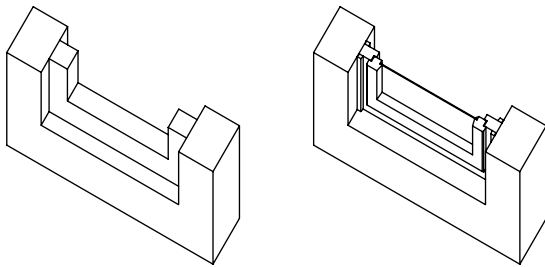
Textual information related to the library part is stored in the **comment** section.

ArchiCAD and ArchiFM provide a helpful environment to write GDL scripts, with on-the-fly visualization, syntax and error checking.

Analyze, Deconstruct and Simplify

No matter how complex, most objects you wish to create can be broken down into “building blocks” of simple geometric shapes. Always start with a simple analysis of the desired object and define all the geometric units that compose it. These building blocks can then be translated into the vocabulary of the GDL scripting language. If your analysis was accurate, the combination of these entities will form the desired object.

To make the analysis, you need to have a good perception of space and at least a basic knowledge of descriptive geometry.



Window representations with different levels of sophistication

To avoid getting discouraged early on in the learning process, start with objects of defined dimensions and take them to their simplest but still recognizable form. As you become familiar with basic modeling, you can increase the level of sophistication and get closer to the ideal form.

“Ideal” does not necessarily mean “complicated.” Depending on the nature of the architectural project, the ideal library part could vary from basic to refined. The window on the left in the above illustration fits the style of a design visualization perfectly. The window on the right gives a touch of realism and detail which can be used later in the construction documents phase of the project.

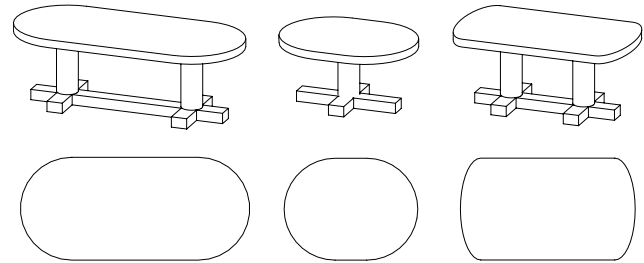
Elaboration

Depending on your purpose, your custom parametric objects may vary in elaboration. Custom objects for internal studio use may be less refined than the ones for general use or for commercial distribution.

If your symbols have little significance on the floor plan, or if parametric changes do not need to appear in 2D, then you can omit parametric 2D scripts.

Even if parametric changes are intended to be present in 2D, it is not absolutely necessary to write a parametric 2D script. You can perform parametric modifications in the 3D Script window or use the 3D top view of the modified object as a new symbol and save the modified object under a new name. Parametric changes to the default values will result in several similar objects derived from the original.

The most complex and sophisticated library parts consist of parametric 3D descriptions with corresponding parametric 2D scripts. Any changes in the settings will affect not only the 3D image of the object, but also its floor plan appearance.



Entry Level

These commands are easy to understand and use. They require no programming knowledge, yet you can create very effective new objects using only these commands.

Simple Shapes

Shapes are basic geometric units that add up to a complex library part. They are the construction blocks of GDL. You place a shape in the 3D space by writing a command in the GDL script.

A shape command consists of a keyword that defines the shape type and some numeric values or alphabetic parameters that define its dimensions.

The number of values varies by shape.

In the beginning, you can omit using parameters and work with fixed values only.

You can start with the following shape commands:

In 3D:

BLOCK CYLIND SPHERE PRISM

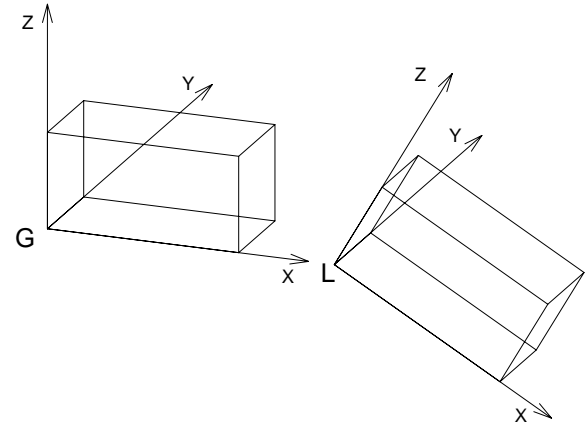
In 2D:

LINE2 RECT2 POLY2 CIRCLE2 ARC2

Coordinate Transformations

Coordinate transformations are like moving your hand to a certain place before placing a construction block. They prepare the position, orientation and scale of the next shape.

```
BLOCK 1, 0.5, 0.5
ADDX 1.5
ROTY 30
BLOCK 1, 0.5, 0.5
```



The 3D window of the library part will optionally show you the home (G = global) and the current (L = local) position of the coordinate axis triad for any object present.

The simplest coordinate transformations are as follows:

In 3D:

```
ADDX ADDY ADDZ
ROTX ROTY ROTZ
```

In 2D:

```
ADD2 ROT2
```

The commands starting with ADD will move the next shape, while the ROT commands will turn it around any of its axes.

Intermediate Level

These commands are a bit more complex, not because they expect you to know programming, but simply because they describe more complex shapes or more abstract transformations.

In 3D:

```
ELLIPS CONE
POLY_ LIN_ PLANE PLANE_
PRISM_ CPRISM_ SLAB SLAB_ CSLAB_
TEXT
```

In 2D:

```
HOTSPOT2 POLY2_ TEXT2 FRAGMENT2
```

These commands usually require more values to be defined than the simple ones. Some of them require status values to control the visibility of edges and surfaces.

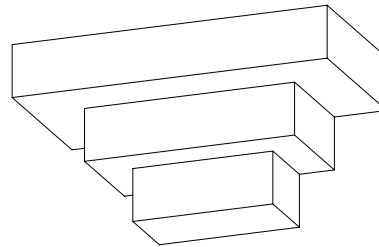
Coordinate Transformations

In 3D:

```
MULX MULY MULZ
ADD MUL ROT
```

In 2D:

```
MUL2
PRISM 4, 1, 3, 0,
      3, 3, -3, 3,
      -3, 0
ADDZ -1
MUL 0.666667, 0.666667, 1
PRISM 4, 1, 3, 0,
      3, 3, -3, 3,
      -3, 0
ADDZ -1
MUL 0.666667, 0.666667, 1
PRISM 4, 1, 3, 0,
      3, 3, -3, 3,
      -3, 0
```



The transformations starting with MUL will rescale the subsequent shapes by distorting circles into ellipses or spheres into ellipsoids. If used with negative values, they can be used for mirroring. The commands in the second row affect all three dimensions of space at the same time.

Advanced Level

These commands add a new level of complexity either because of their geometric shape, or because they represent GDL as a programming language.

In 3D:

BPRISM_	BWALL_	CWALL_	XWALL_
CROOF_	FPRISM_	SPRISM_	
EXTRUDE	PYRAMID	REVOLVE	RULED
SWEEP	TUBE	TUBEA	COONS
MESH	MASS		
LIGHT	PICTURE		

There are shape commands in this group which let you trace a spatial polygon with a base polygon to make smooth curved surfaces. Some shapes require material references in their parameter list.

By using cutting planes, polygons and shapes, you can generate complex arbitrary shapes out of simple shapes. The corresponding commands are CUTPLANE, CUTPOLY, CUTPOLYA, CUTSHAPE and CUTEND.

In 2D:

PICTURE2	POLY2_A
SPLINE2	SPLINE2_A

Flow Control and Conditional Statements

```
FOR NEXT
DO WHILE ENDWHILE
REPEAT UNTIL
IF THEN ELSE ENDIF GOTO GOSUB
RETURN END EXIT
```

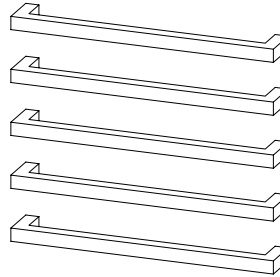
These commands should be familiar to anyone who has ever programmed a computer, but they are basic enough that you can understand them without prior programming experience.

They let you make repetitive script parts to place several shapes with little scripting, or let you make decisions based on prior calculations.


```

FOR I = 1 TO 5
PRISM_ 8, 0.05,
    -0.5, 0, 15,
    -0.5, -0.15, 15,
    0.5, -0.15, 15,
    0.5, 0, 15,
    0.45, 0, 15,
    0.45, -0.1, 15,
    -0.45, -0.1, 15,
    -0.45, 0, 15
ADDZ 0.2
NEXT I

```



Parameters

At this stage of your expertise, you can replace fixed numeric values with variable names. This makes the object more flexible. These variables are accessible from the library part's Settings dialog box while working on the project.

Macro Calls

You are not limited to the standard GDL shapes. Any existing library part may become a GDL shape in its entirety. To place it, you simply “call” (refer to) its name and transfer the required parameters to it, just as with standard shape commands.

Expert Level

By the time you have a good understanding of the features and commands outlined above, you will be able to pick up the few remaining commands that you may need from time to time.

Note: The memory capacity of your computer may limit the file length of your GDL scripts, the depth of macro calls and the number of transformations.

You will find additional information on the above GDL commands throughout the manual. HTML format help files are also available with your software, giving a quick overview of the available commands and their parameter structure.

3D GENERATION

3D modeling is based on floating point arithmetics, meaning that there is no limit imposed on the geometric size of the model. Whatever size it is, it retains the same accuracy down to the smallest details.

The 3D model that you finally see on the screen is composed of **geometric primitives**. These primitives are stored in the memory of your computer in binary format, and the 3D engine generates them according to the floor plan you created. The metamorphosis between the architectural floor plan elements and the binary 3D data is called 3D conversion.

The primitives are the following:

- all the **vertices** of your building components

- all the **edges** linking the vertices
- all the surface **polygons** within the edges

Groups of these primitives are kept together as **bodies**. The bodies make up the 3D model. All of the features of 3D visualization - smooth surfaces, cast shadows, glossy or transparent materials - are based on this data structure.

The 3D Space

The 3D model is created in three-dimensional space measured by the x, y and z axes of a master coordinate system whose origin is called the **global origin**.

In Floor Plan view, you can see the global origin in the lower left corner of the worksheet if you open the program without reading a specific document. In addition, the global origin defines the zero level of all the stories referred to in a floor plan document.

When you place an object into the design, the floor plan position will define its location along the x and y axes of this master coordinate system. The location along the z axis can be set in the Object Settings dialog box or directly adjusted when placed in 3D. This location will be the base and the default position of the **local coordinate system** of the object. The shapes described in the script will be positioned with reference to this local coordinate system.

Coordinate Transformations

Every GDL shape is linked to the current position of the local coordinate system. For example, BLOCKs are linked to the origin. The length, width and height of the block are always measured in a positive direction along the three axes. Thus, the BLOCK command requires only three parameters defining its dimensions along the axes.

How can you generate a shifted and rotated block? With the parameter structure of the BLOCK there is no way to do this. It does not have parameters for shift and rotation.

The answer is to move the coordinate system to the correct position before issuing the BLOCK command. With the coordinate transformation commands, you can pre-define its position and rotation around the axes. These transformations are not applied to the shapes already generated and are only effective on subsequent shapes.

The GDL Interpreter

When a GDL script is executed, the GDL interpreter engine will detect the location, size, rotation angle, user defined parameters and the mirrored state of the library part. It will then move the local coordinate system to the right position, ready to receive the GDL commands from the script of the library parts. Every time a command for a basic shape is read by the interpreter, it will generate the geometric primitives that make up that particular shape.

When the interpreter has finished, the complete binary 3D model will be stored in the memory, and you can perform 3D projections, fly-through renderings or sun studies on it.

ArchiCAD and ArchiFM contain a pre-compiler and an interpreter for GDL. Interpretation of a GDL script uses the pre-compiled code. This feature increases speed of the analysis. If the GDL script is modified, a new code is generated.

Data structures converted from other file formats (e.g., DXF, Zoom, Alias Wavefront) are stored in a binary 3D section of the library parts. This section is referenced by the BINARY statement from the GDL script.

The GDL Script Analysis

Users have no control over the order in which library parts placed on the floor plan are analyzed. The order of GDL script analysis is based on the internal data structure; moreover, Undo and Redo operations as well as modifications may influence that order. The only exceptions to this rule are special GDL scripts of the active library, whose names begin with “**MASTER_GDL**” or “**MASTEREND_GDL**”.

Scripts whose name begins with “**MASTER_GDL**” are executed before a 3D conversion, before creating a Section/Elevation, before starting a list process and after loading the active library.

Scripts whose name begins with “**MASTEREND_GDL**” are executed after a 3D conversion sequence, after creating a Section/Elevation, when finishing a list process and when the active library is to be changed (Load Libraries, Open a project, New project, Quit).

These scripts are not executed when you edit library parts. If your library contains one or more such scripts they will all be executed in an order that is not defined.

MASTER_GDL and MASTEREND_GDL scripts can include attribute definitions, initializations of GDL user global variables, 3D commands (effective only in the 3D model), value list definitions (see the “*VALUES*” on page 181) and GDL extension-specific commands. The attributes defined in these scripts will be merged into the current attribute set (attributes with same names are not replaced, while attributes originated from GDL and not edited in the program are always replaced).

GDL SYNTAX

This chapter presents the basic elements of GDL syntax, including statements, labels, identifiers, variables and parameters. Typographic rules are also explained in detail.

Rules of GDL Syntax

GDL is not case sensitive; uppercase and lowercase letters are not distinguished, except in strings placed between quotation marks. The logical end of a GDL script is denoted by an END or EXIT statement or the physical end of the script.

STATEMENTS

A GDL program consists of statements. A statement can start with a keyword (defining a GDL shape, coordinate transformations or program control flow), with a macro name, or with a variable name followed by an '=' sign and an expression.

LINE

The statements are in lines separated by line-separators (end_of_line characters).

A comma (,) in the last position indicates that the statement continues on the next line. A colon (:) is used for separating GDL statements in a line. After an exclamation mark (!) you can write any comment in the line. Blank lines can be inserted into a GDL script with no effect at all. Any number of spaces or tabs can be used between the operands and operators. The use of a space or tab is obligatory after statement keywords and macro calls.

LABEL

Any line can start with a label. A label is an integer number followed by a colon (:). The label is used as a reference for a subsequent statement. Labels are checked for single occurrence. The execution of the program can be continued from any label by using a GOTO or GOSUB statement.

CHARACTERS

The GDL text is composed of the lower and uppercase letters of the English alphabet, any number and the following characters:

<space> _ (underline) ~ ! : , ; . + - * / ^ = < > <= >= # () [] { } \ @ & | (vertical bar) " ' ` ^ ` ´ ˆ ˜ ` ´ ˆ ˜ <end_of_line>

STRINGS

Any string of characters that is placed between quotation marks (“, ’, `, `), or any string of characters without quotation marks that does not figure in the script as an identifier with a given value (macro call, attribute name, file name). Strings without quotation marks will be converted to all caps, so using quotation marks is recommended. The maximum length allowed in a string is 255 characters.

The ‘\’ character has special control values. Its meaning depends on the next character.

```
\\      ‘\’ char itself
\n      new line
\t      tabulator
\new line continue string in next line without a new line
\others not correct, results in warning
```

Examples:

```
“This is a string”
“washbasin 1'-6"*1'-2”
'Do not use different delimiters'
```

IDENTIFIERS

Identifiers are special character strings:

- they are not longer than 255 characters;
- they begin with a letter of the alphabet or a ‘_’ or ‘~’ character;
- they consist of letters, numbers and ‘_’ or ‘~’ characters;
- upper- and lowercase letters are considered identical.

Identifiers can be GDL keywords, global or local variables or strings (names). Keywords and global variable names are determined by the program you’re using GDL in; all other identifiers can be used as variable names.

VARIABLES

GDL programs can handle numeric and string variables (defined by their identifiers), numbers and character strings.

There are two sets of variables: local and global.

All identifiers that are not keywords, global variables, attribute names, macro names or file names are considered local variables. If left uninitialized (undefined), their value will be 0 (integer). Local variables are stacked with macro calls. When returning from a macro call, the interpreter restores their values.

Global variables have reserved names (the list of global variables available is given in the *“Miscellaneous” on page 215*). They are not stacked during macro calls, enabling the user to store special values of the modeling and to simulate return codes from macros. The user global variables can be

set in any script but they will only be effective in subsequent scripts. If you want to make sure that the desired script is analyzed first, set these variables in the MASTER_GDL library part. The other global variables can be used in your scripts to communicate with the program. By using the “=” command, you can assign a numeric or string value to local and global variables.

PARAMETERS

Identifiers listed in a library part’s parameter list are called parameters. Parameter identifiers must not exceed 32 characters. Within a script, the same rules apply to parameters as to local variables.

Parameters of text-only GDL files are identified by letters A to Z.

SIMPLE TYPES

Variables, parameters and expressions can be of two simple types: numeric or string.

Numeric expressions are constant numbers, numeric variables or parameters, functions that return numeric values, and any combination of these in operations. Numeric expressions can be integer or real. Integer expressions are integer constants, variables or parameters, functions that return integer values, and any combination of these in operations which results in integers. Real expressions are real constants, variables or parameters, functions that return real values, and any combination of these (or integer expressions) in operations which results in reals. A numeric expression being an integer or a real is determined during the compilation process and depends only on the constants, variables, parameters and the operations used to combine them. Real and integer expressions can be used the same way at any place where a numeric expression is required, however, in cases where a combination of these may result in precision problems, a compiler warning appears (comparison of reals or reals and integers using relational operators '=' or '<>', or boolean operators AND, OR, EXOR; IF or GOTO statements with real label expressions).

String expressions are constant strings, string variables or parameters, functions that return strings, and any combination of these in operations which result in strings.

DERIVED TYPES

Variables and parameters can also be arrays, and parameters can be value lists of a simple type.

Arrays are one- or two-dimensional tables of numeric and/or string values, which can be accessed directly by indexes.

Value lists are sets of possible numeric or string values. They can be assigned to the parameters in the value list script of the library part or in the MASTER_GDL script, and will appear in the parameter list as a pop-up menu.

[aaa]

Square brackets mean that the enclosed elements are optional (if they are bold, they must be entered as shown).

{n}

command version number

...

Previous element may be repeated

variable

Any GDL variable name

prompt

Any character string (must not contain quote character)

BOLD_STRING

UPPERCASE_STRING

special characters

Must be entered as shown

other_lowercase_string_in_parameter_list

Any GDL expression

COORDINATE TRANSFORMATIONS

This chapter tells you about the types of transformations available in GDL (moving, scaling, rotating the coordinate system) and the way they are interpreted and managed.

About Transformations

In GDL, all the geometric elements are linked strictly to the local coordinate system. GDL uses a right-handed coordinate system. For example, one corner of a block is in the origin and its sides are in the x-y, x-z and y-z planes.

Placing a geometric element in the desired position requires two steps. First, move the coordinate system to the desired position. Second, generate the element. Every movement, rotation or stretching of the coordinate system along or around an axis is called a transformation.

Transformations are stored in a stack; interpretation starts from the last one backwards. Scripts inherit this stack; they can insert new elements onto it but can only delete the locally defined ones. It is possible to delete one, more or all of the transformations defined in the current script. After returning from a script, the locally defined transformations are removed from the stack.

2D TRANSFORMATIONS

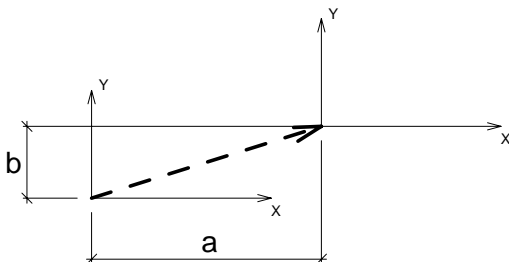
These are the equivalents in the 2D space of the ADD, MUL and ROTZ 3D transformations.

ADD2

ADD2 x, y

Example:

ADD2 a, b



MUL2

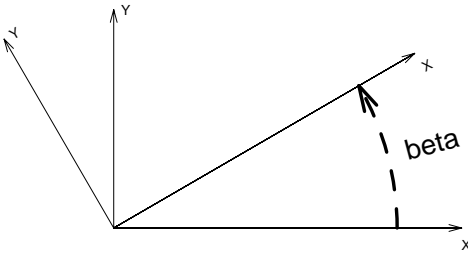
MUL2 x, y

ROT2

ROT2 alpha

Example:

ROT2 beta



3D TRANSFORMATIONS

ADDX

ADDX dx

ADDY

ADDY dy

ADDZ

ADDZ dz

Moves the local coordinate system along the given axis by dx, dy or dz respectively.

ADD

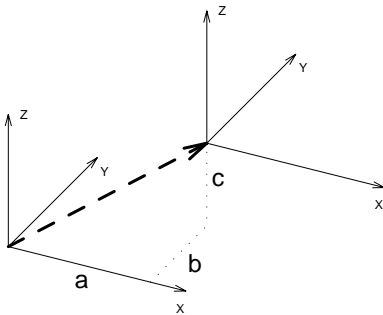
ADD dx, dy, dz

Replaces the sequence ADDX dx: ADDY dy: ADDZ dz.

It has only one entry in the stack, thus it can be deleted with DEL 1.

Example:

ADD a, b, c



MULX

MULX mx

MULY

MULY my

MULZ

MULZ mz

Scales the local coordinate system along the given axis. Negative mx, my, mz means simultaneous mirroring.

MUL

MUL mx, my, mz

Replaces the sequence MULX mx: MULY my: MULZ mz. It has only one entry in the stack, thus it can be deleted with DEL 1.

ROTX

ROTX alphax

ROTY

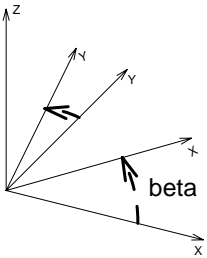
ROTY alphay

ROTZ

ROTZ alphaz

Rotates the local coordinate system around the given axis by alphax, alphay, alphaz degrees respectively, counterclockwise.

Example:



ROTZ beta

ROT

ROT x, y, z, alpha

Rotates the local coordinate system around the axis defined by the vector (x, y, z) by alpha degrees, counterclockwise.

It has only one entry in the stack, thus it can be deleted with DEL 1.

XFORM

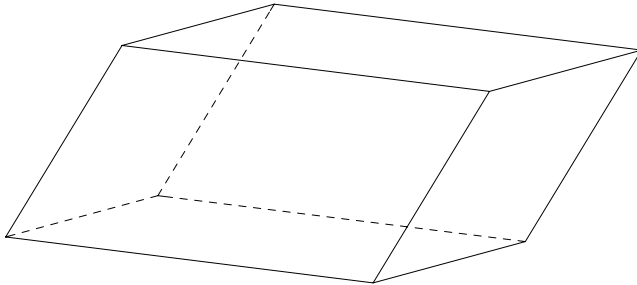
XFORM a11, a12, a13, a14,
a21, a22, a23, a24,
a31, a32, a33, a34

Defines a complete transformations matrix. It is mainly used in automatic GDL code generation. It has only one entry in the stack.

$$\begin{aligned}x' &= a_{11} * x + a_{12} * y + a_{13} * z + a_{14} \\y' &= a_{21} * x + a_{22} * y + a_{23} * z + a_{24} \\z' &= a_{31} * x + a_{32} * y + a_{33} * z + a_{34}\end{aligned}$$

Example:

```
A=60
B=30
XFORM 2, COS(A), COS(B)*0.6, 0,
      0, SIN(A), SIN(B)*0.6, 0,
      0, 0, 1, 0
BLOCK 1, 1, 1
```



MANAGING THE TRANSFORMATION STACK

DEL n

DEL n [, begin_with]

Deletes n entries from the transformation stack.

If the begin_with parameter is not specified, deletes the previous n entries in the transformation stack. The local coordinate system moves back to a previous position.

If the begin_with transformation is specified, deletes n entries forward, beginning with the one denoted by begin_with. Numbering starts with 1. If the begin_with parameter is specified and n is negative, deletes backward.

If fewer transformations were issued in the current script than denoted by the given n number argument, then only the issued transformations are deleted.

DEL TOP

DEL TOP

Deletes all current transformations in the current script.

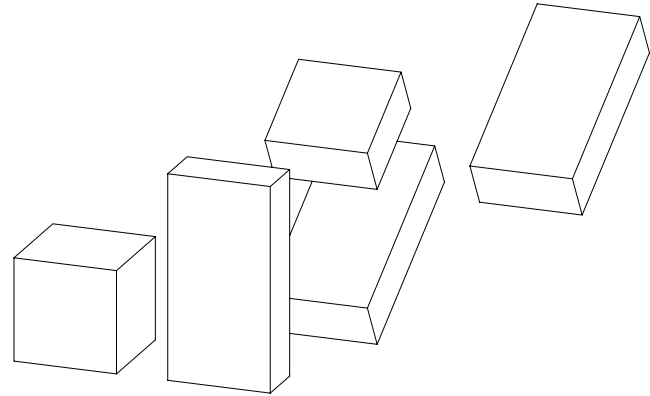
NTR

NTR ()

Returns the actual number of transformations.

Example:

```
BLOCK 1, 1, 1
ADDX 2
ADDY 2.5
ADDZ 1.5
ROTX -60
ADDX 1.5
BLOCK 1, 0.5, 2
DEL 1, 1           !Deletes the ADDX 2
transformation
BLOCK 1, 0.5, 1
DEL 1, NTR() -2   !Deletes the ADDZ 1.5
transformation
BLOCK 1, 0.5, 2
DEL -2, 3         !Deletes the ROTX -60 and ADDY
2.5 transformations
BLOCK 1, 0.5, 2
```



3D SHAPES

This chapter covers all the 3D shape creation commands available in GDL, from the most basic ones to the generation of complex shapes from polylines. Elements for visualization (light sources, pictures) are also presented here, as well as the definition of text to be displayed in 3D. Furthermore, the primitives of the internal 3D data structure consisting of nodes, vectors, edges and bodies are discussed in detail, followed by the interpretation of binary data and guidelines for using cutting planes.

BASIC SHAPES

BLOCK

BLOCK a, b, c

BRICK

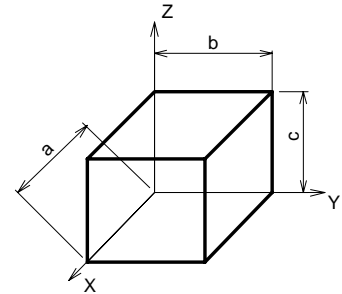
BRICK a, b, c

The first corner of the block is in the local origin and the edges with lengths a, b and c are along the x-, y- and z-axes, respectively.

Zero values create degenerated blocks (rectangle or line).

Restriction of parameters:

a, b, c ≥ 0



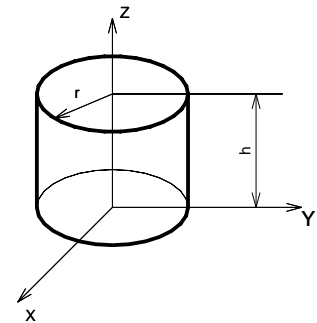
CYLIND

CYLIND h, r

Right cylinder, coaxial with the z-axis with a height of h and a radius of r.

If h=0, a circle is generated in the x-y plane.

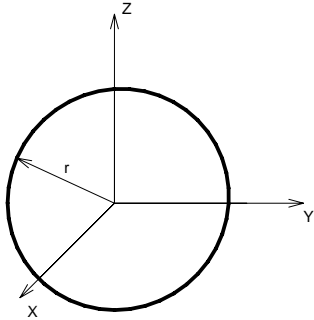
If r=0, a line is generated along the z axis.



SPHERE

SPHERE r

A sphere with its center at the origin and with a radius of r .



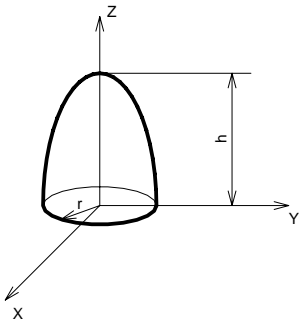
ELLIPS

ELLIPS h, r

Half ellipsoid. Its cross-section in the x-y plane is a circle with a radius of r centered at the origin. The length of the half axis along the z-axis is h .

Example:

ELLIPS r, r ! hemisphere



CONE

CONE $h, r1, r2, \alpha1, \alpha2$

Frustum of a cone where $\alpha1$ and $\alpha2$ are the angles of inclination of the end surfaces to the z axis, $r1$ and $r2$ are the radii of the end-circles and h is the height along the z axis.

If $h=0$, the values of $\alpha1$ and $\alpha2$ are disregarded and an annulus is generated in the x-y plane.

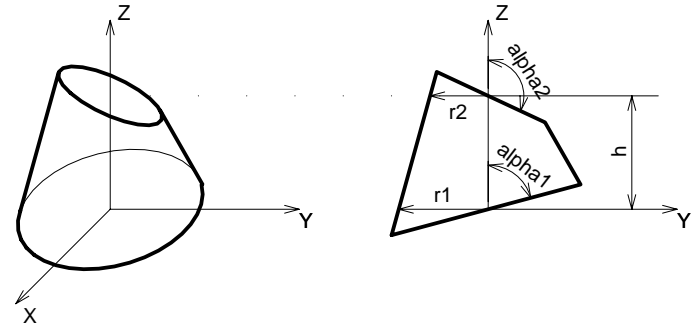
$\alpha1$ and $\alpha2$ are in degrees.

Restriction of parameters:

$0 < \alpha1 < 180^\circ$ and $0 < \alpha2 < 180^\circ$

Example:

CONE $h, r, 0, 90, 90$! a regular cone



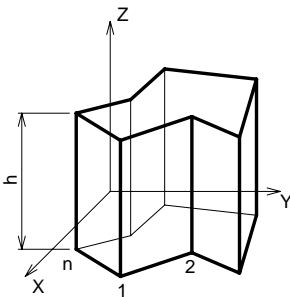
PRISM

PRISM $n, h, x1, y1, \dots, xn, yn$

Right prism with its base polygon in the x-y plane (see the parameters of ["POLY"](#) on page 61 and ["POLY_"](#) on page 61). The height along the z-axis is $\text{abs}(h)$. Negative h values can also be used. In that case the second base polygon is below the x-y plane.

Restriction of parameters:

$n \geq 3$



PRISM_

PRISM_ *n*, *h*, *x1*, *y1*, *s1*, ... *xn*, *yn*, *sn*

Similar to the CPRISM_ statement, but any of the horizontal edges and sides can be omitted.

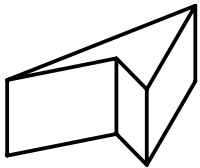
Restriction of parameters:

n >= 3

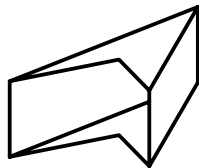
si: status code that allows you to control the visibility of polygon edges and side surfaces. You can also define holes and create segments and arcs in the polyline using special constraints.

See "[Status Codes](#)" on page 137 for details.

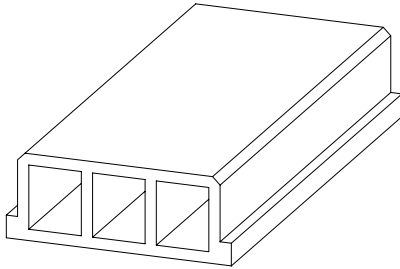
Examples:



```
PRISM_ 4,1,
0,0,15,
1,1,15,
2,0,15,
1,3,15
```



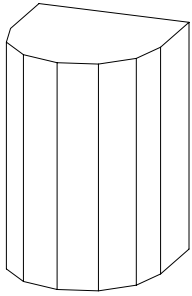
```
PRISM_ 4,1,
0,0,7,
1,1,5,
2,0,15,
1,3,15
```



```

ROTX 90
PRISM_ 26, 1.2,
    0.3, 0, 15,
    0.3, 0.06, 15,
    0.27, 0.06, 15,
    0.27, 0.21, 15,
    0.25, 0.23, 15,
    -0.25, 0.23, 15,
    -0.27, 0.21, 15,
    -0.27, 0.06, 15,
    -0.3, 0.06, 15,
    -0.3, 0, 15,
    0.3, 0, -1,
    0.10, 0.03, 15,
    0.24, 0.03, 15,
    0.24, 0.2, 15,
    0.10, 0.2, 15,
    0.10, 0.03, -1,
    0.07, 0.03, 15,
    0.07, 0.2, 15,
    -0.07, 0.2, 15,
    -0.07, 0.03, 15,
    0.07, 0.03, -1,
    -0.24, 0.03, 15,
    -0.24, 0.2, 15,
    -0.1, 0.2, 15,
    -0.1, 0.03, 15,
    -0.24, 0.03, -1
!End of contour
!End of first hole
!End of second hole
!End of third hole

```



j7 = 0

R=1

H=3

```

PRISM      9,          H,
  -R,      R,          15,
  COS(180)*R, SIN(180)*R, 15,
  COS(210)*R, SIN(210)*R, 15,
  COS(240)*R, SIN(240)*R, 15,
  COS(270)*R, SIN(270)*R, 15,
  COS(300)*R, SIN(300)*R, 15,
  COS(330)*R, SIN(330)*R, 15,
  COS(360)*R, SIN(360)*R, 15,
  R,       R,          15

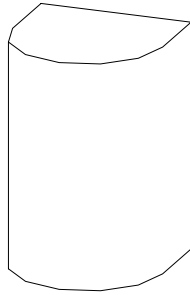
```

ADDX 5

```

PRISM      9,          H,
  -R,      R,          15,
  COS(180)*R, SIN(180)*R, 64+15,
  COS(210)*R, SIN(210)*R, 64+15,
  COS(240)*R, SIN(240)*R, 64+15,
  COS(270)*R, SIN(270)*R, 64+15,
  COS(300)*R, SIN(300)*R, 64+15,
  COS(330)*R, SIN(330)*R, 64+15,
  COS(360)*R, SIN(360)*R, 64+15,
  R,       R,          15

```



j7 = 1

CPRISM_

CPRISM_ top_material, bottom_material, side_material,
n, h, x1, y1, s1, ... xn, yn, sn

Extension of the PRISM_ statement. The first three parameters are used for the material name/index of the top, bottom and side surfaces. The other parameters are the same as above in the PRISM_ statement.

Restriction of parameters:

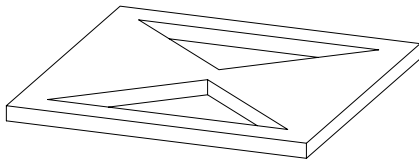
n >= 3

See also *"Materials" on page 158.*

si: status code that allows you to control the visibility of polygon edges and side surfaces. You can also define holes and create segments and arcs in the polyline using special constraints.

See *"Status Codes" on page 137 for details.*

Example:



```
CPRISM_ "Iron", 0, T_,           !"Iron" is a predefined material.
                                !0 is a general material.
                                !T_ is a global variable
                                !(a material index)

13, 0.2,
0, 0, 15,
2, 0, 15,
2, 2, 15,
0, 2, 15,
0, 0, -1,           !end of the contour
0.2, 0.2, 15,
1.8, 0.2, 15,
1.0, 0.9, 15,
0.2, 0.2, -1,       !end of first hole
0.2, 1.8, 15,
1.8, 1.8, 15,
1.0, 1.1, 15,
0.2, 1.8, -1       !end of second hole
```

BPRISM_

BPRISM_ *top_material*, *bottom_material*, *side_material*,
n, *h*, *radius*, *x1*, *y1*, *s1*, ... *xn*, *yn*, *sn*

A smooth curved prism, based on the same data structure as the straight CPRISM_ element. The only additional parameter is radius.

Derived from the corresponding CPRISM_ by bending the x-y plane onto a cylinder tangential to that plane. 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.

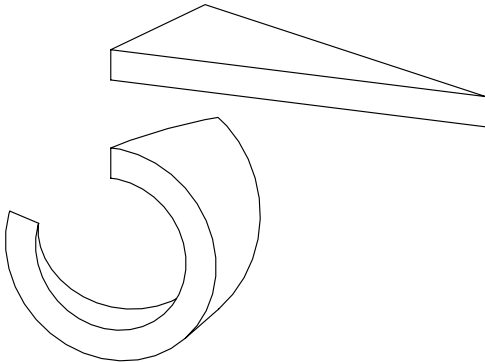
See "*BWALL_*" on page 48 for details.

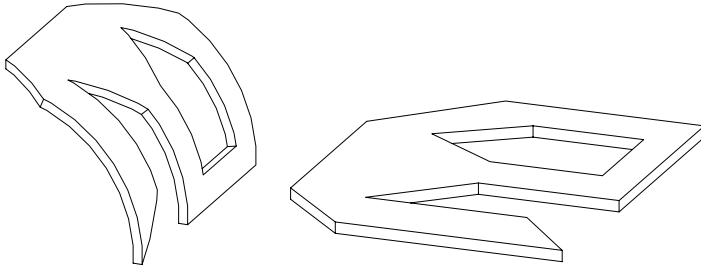
si: status code that allows you to control the visibility of polygon edges and side surfaces. You can also define holes and create segments and arcs in the polyline using special constraints.

See "*Status Codes*" on page 137 for details.

Examples (with the corresponding CPRISM_-s):

```
BPRISM_ "Glass", "Glass", "Glass",
        3, 0.4, 1,           ! radius = 1
        0, 0, 15,
        5, 0, 15,
        1.3, 2, 15
```





```

BPRISM_ "Concrete", "Concrete", "Concrete",
  17, 0.3, 5,
  0, 7.35, 15,
  0, 2, 15,
  1.95, 0, 15,
  8, 0, 15,
  6.3, 2, 15,
  2, 2, 15,
  4.25, 4, 15,
  8, 4, 15,
  8, 10, 15,
  2.7, 10, 15,
  0, 7.35, -1,
  4, 8.5, 15,
  1.85, 7.05, 15,
  3.95, 5.6, 15,
  6.95, 5.6, 15,
  6.95, 8.5, 15,
  4, 8.5, -1

```

FPRISM_

```

FPRISM_ top_material, bottom_material, side_material, hill_material,
  n, thickness, angle, hill_height,
  x1, y1, s1,
  ...
  xn, yn, sn

```

Similar to the PRISM_ statement, with the additional hill_material, angle and hill_height parameters. A hill part is added to the top of the right prism.

hill_material: the side material of the hill part

angle: the inclination angle of the hill side edges. Restriction: $0 \leq \text{angle} < 90$. If $\text{angle} = 0$, the hill side edges seen from an orthogonal view form a quarter circle with the current resolution (see the commands: *"RADIUS" on page 150*, *"RESOL" on page 151* and *"TOLER" on page 152*).

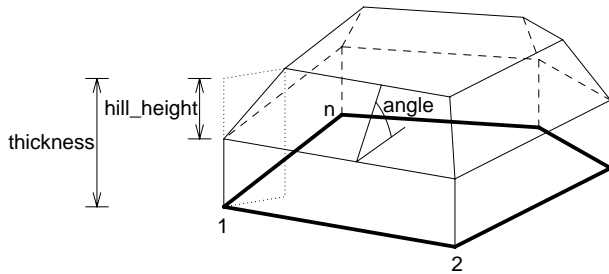
hill_height: the height of the hill. Note that the thickness parameter represents the whole height of the FPRISM_.

Restriction of parameters:

$n \geq 3$, $\text{hill_height} < \text{thickness}$

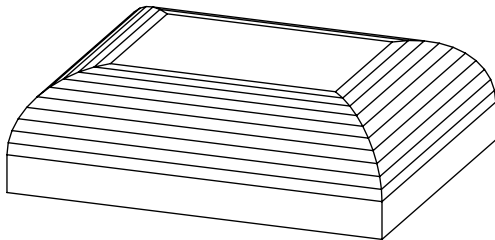
si: status code that allows you to control the visibility of polygon edges and side surfaces. You can also define holes and create segments and arcs in the polyline using special constraints.

See *"Status Codes" on page 137* for details.



Examples:

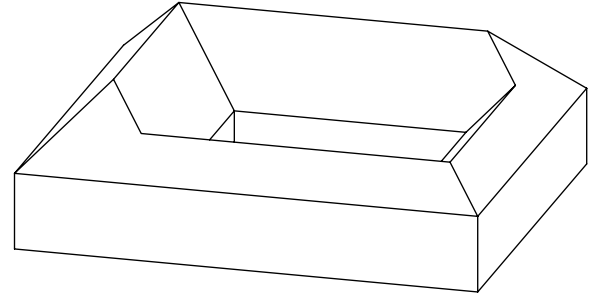
```
RESOL 10
FPRISM_ "Roof Tile", "Red Brick", "Face brick", "Roof Tile",
  4, 1.5, 0, 1.0,           !angle = 0
  0, 0, 15,
  5, 0, 15,
  5, 4, 15,
  0, 4, 15
```




```

FPRISM_ "Roof Tile", "Red Brick", "Face brick",
  "Roof Tile",
  10, 2, 45, 1,
  0, 0, 15,
  6, 0, 15,
  6, 5, 15,
  0, 5, 15,
  0, 0, -1,
  1, 2, 15,
  4, 2, 15,
  4, 4, 15,
  1, 4, 15,
  1, 2, -1

```



HPRISM_

```

HPRISM_ top_mat, bottom_mat, side_mat,
  hill_mat,
  n, thickness, angle, hill_height, status,
  x1, y1, s1,
  ...,
  xn, yn, sn

```

Similar to FPRISM_, with an additional parameter controlling the visibility of the hill edges.

status: controls the visibility of the hill edges:

0: hill edges are all visible (FPRISM_)

1: hill edges are invisible

SPRISM_

```

SPRISM_ top_material, bottom_material, side_material,
  n, xb, yb, xe, ye, h, angle,
  x1, y1, s1, ... xn, yn, sn

```

Extension of the CPRISM_ statement, with the possibility of having an upper polygon non-parallel with the x-y plane. The upper plane definition is similar to the plane definition of the CROOF_ statement. The height of the prism is defined at the reference line. Upper and lower polygon intersection is forbidden.

Additional parameters:

x_b , y_b , x_e , y_e : reference line (vector) starting and end coordinates,

angle: rotation angle of the upper polygon around the given oriented reference line in degrees (CCW),

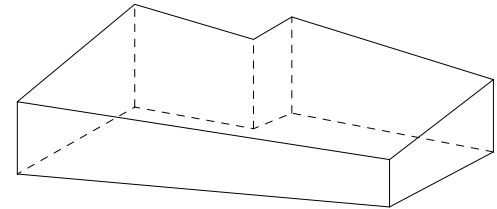
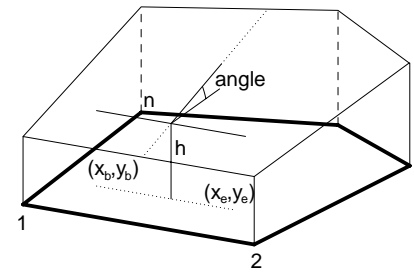
s_i : status code that allows you to control the visibility of polygon edges and side surfaces. You can also define holes and create segments and arcs in the polyline using special constraints.

See *"Status Codes" on page 137* for details.

Note: All calculated z coordinates of the upper polygon nodes must be positive or 0.

Example:

```
SPRISM_ 'Grass', 'Earth', 'Earth',
  6,
  0, 0, 11, 6, 2, -10.0,
  0, 0, 15,
  10, 1, 15,
  11, 6, 15,
  5, 7, 15,
  4.5, 5.5, 15,
  1, 6, 15
```



SLAB

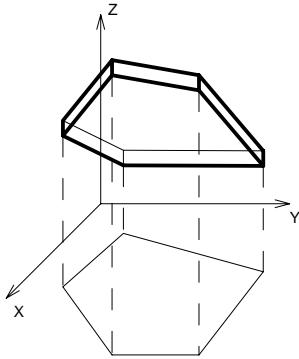
SLAB $n, h, x1, y1, z1, \dots, xn, yn, zn$

Oblique prism. The lateral faces are always perpendicular to the x-y plane. Its bases are flat polygons rotated about an axis parallel with the x-y plane. Negative h values can also be used. In that case the second base polygon is below the given one.

No check is made as to whether the points are really on a plane. Apices not lying on a plane will result in strange shadings/ renderings.

Restriction of parameters:

$n \geq 3$



SLAB_

SLAB_ $n, h, x1, y1, z1, s1, \dots, xn, yn, zn, sn$

Similar to the SLAB statement, but any of the edges and faces of the side polygons can be omitted. This statement is an analogy of the PRISM_ statement.

si: status code that allows you to control the visibility of polygon edges and side surfaces. You can also define holes and create segments and arcs in the polyline using special constraints.

See "*Status Codes*" on page 137 for details.

CSLAB_

CSLAB_ top_material, bottom_material, side_material,
n, h, x1, y1, z1, s1, ... xn, yn, zn, sn

Extension of the SLAB_ statement; the first three parameters are used for the material name/index of the top, bottom and side surfaces. The other parameters are the same as above in the SLAB_ statement.

See also *“Materials” on page 158* and the *“Requests” on page 233* > IND function description *“Miscellaneous” on page 215*.

si: status code that allows you to control the visibility of polygon edges and side surfaces. You can also define holes and create segments and arcs in the polyline using special constraints.

See *“Status Codes” on page 137* for details.

CWALL_

CWALL_ 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

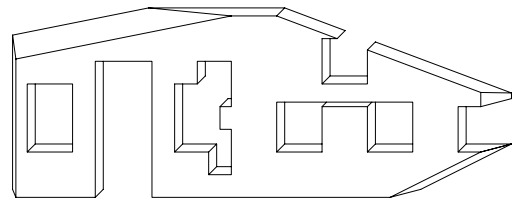
left_material, right_material, side_material:

Material names/indices for the left, right and side surfaces. (The left and right sides of the wall follow the x axis.)

See also *“Materials” on page 158* and the *“Requests” on page 233* > IND function description.

The reference line of the wall is always transformed to coincide with the x axis. The sides of the wall are in the x-z plane.

height: The height of the wall relative to its base.



x_1, x_2, x_3, x_4 : The projected endpoints of the wall lying on the x-y plane as seen below. If the wall stands on its own, then
 $x_1 = x_4 = 0, x_2 = x_3 = \text{the length of the wall.}$

t : the thickness of the wall.

$t < 0$ if the body of the wall is to the right of the x axis,

$t > 0$ if the body of the wall is to the left of the x axis,

$t = 0$ if the wall is represented by a polygon and frames are generated around the holes.

$\text{mask}_1, \text{mask}_2, \text{mask}_3, \text{mask}_4$: Control the visibility of edges and side polygons.

$$\text{mask}_i = j_1 + 2*j_2 + 4*j_3 + 8*j_4$$

where j_1, j_2, j_3, j_4 can be 0 or 1.

The j_1, j_2, j_3, j_4 numbers represent whether the vertices and the side are present (1) or omitted (0).

n : the number of openings in the wall.

$x_start_i, y_low_i, x_end_i, y_high_i$: coordinates of the openings as shown below.

frame_shown_i values:

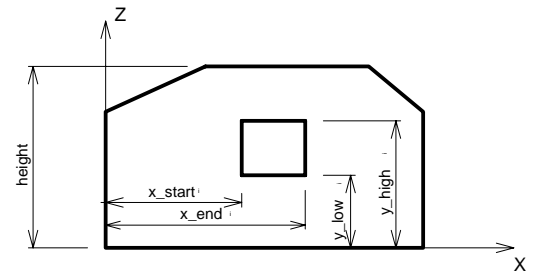
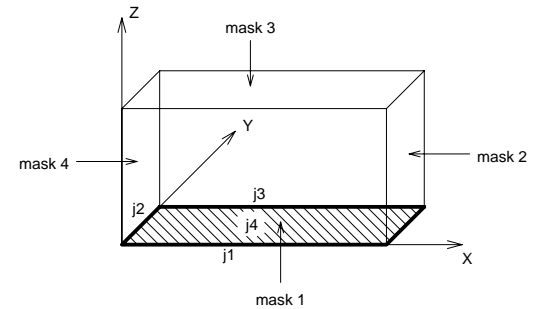
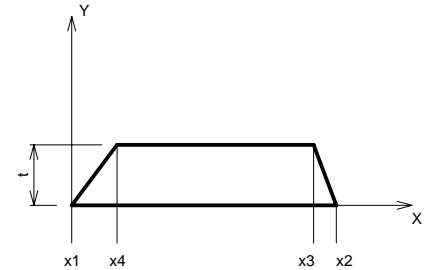
1, if the edges of the hole are visible;

0, if the edges of the hole are invisible.

Negative values control the visibility of each of the opening's edges separately.

$$\text{frame_shown}_i = -(1*j_1 + 2*j_2 + 4*j_3 + 8*j_4 + 16*j_5 + 32*j_6 + 64*j_7 + 128*j_8),$$

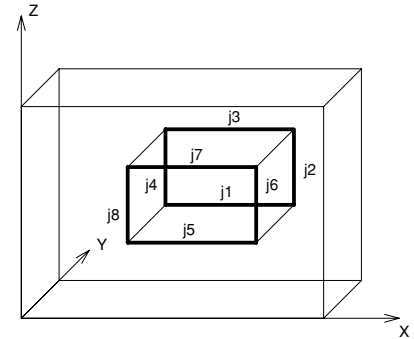
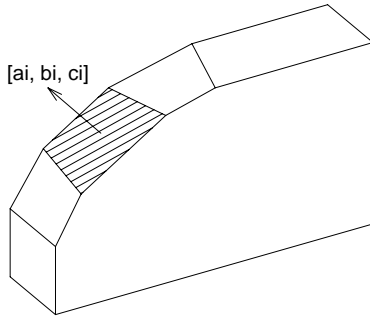
where j_1, j_2, \dots, j_8 can be either 0 or 1. The numbers j_1 to j_4 control the visibility of the edges of the hole on the left-hand side of the wall surface, while j_5 to j_8 affect the edges on the right-hand side, as shown on the illustration below.



An edge that is perpendicular to the surface of the wall is visible if there are visible edges drawn from both of its endpoints.

m: the number of cutting planes.

a_i, b_i, c_i, d_i : coefficients of the equation defining the cutting plane [$a_i*x + b_i*y + c_i*z = d_i$].
Parts on the positive side of the cutting plane (i.e., $a_i*x + b_i*y + c_i*z > d_i$) will be cut and removed.



BWALL_

```

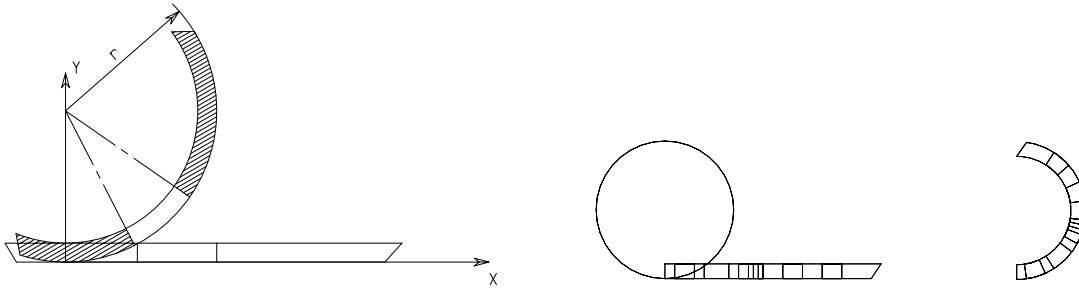
BWALL_ left_material, right_material, side_material,
height, x1, x2, x3, x4, t, radius,
mask1, mask2, mask3, mask4,
n,
x_start1, y_low1, x_end1, y_high1, frame_shown1,
...
x_startn, y_lown, x_endn, y_highn, frame_shownn,
m,
a1, b1, c1, d1,
...
am, bm, cm, dm

```

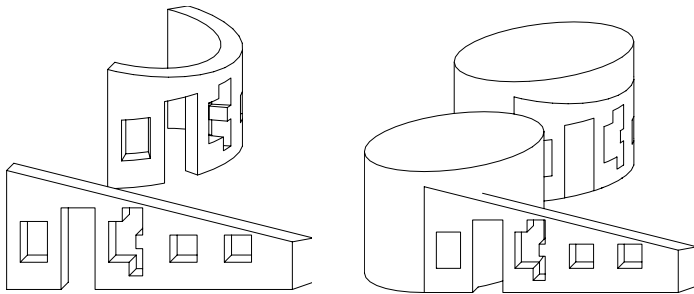
A smooth curved wall based on the same data structure as the straight wall CWALL_ element. The only additional parameter is radius.

Derived from the corresponding CWALL_ by bending the x-z plane onto a cylinder tangential to that plane. Edges along the x axis are transformed to circular arcs, edges along the y axis will be radial in direction, and vertical edges remain vertical. The curvature is approximated by a number of segments set by the current resolution (see the commands: "[RADIUS](#)" on page 150, "[RESOL](#)" on page 151 and "[TOLER](#)" on page 152).

See also "[CWALL_](#)" on page 46 for details.



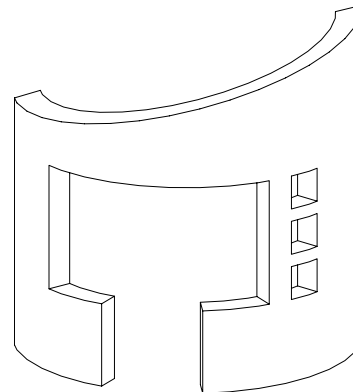
Examples: a `BWALL_` and the corresponding `CWALL_`.



```

ROTZ -60
BWALL_ 1, 1, 1,
        4, 0, 6, 6, 0,
        0.3, 2,
        15, 15, 15, 15,
        5,
        1, 1, 3.8, 2.5, -255,
        1.8, 0, 3, 2.5, -255,
        4.1, 1, 4.5, 1.4, -255,
        4.1, 1.55, 4.5, 1.95, -255,
        4.1, 2.1, 4.5, 2.5, -255,
        1, 0, -0.25, 1, 3

```



XWALL_

```

XWALL_ 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

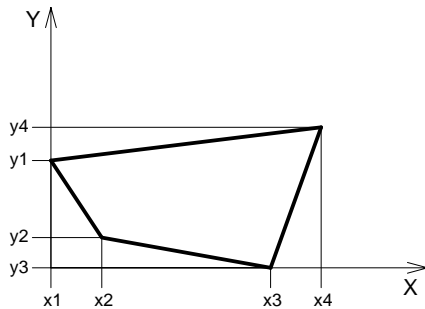
```

Extended wall definition based on the same data structure as the BWALL_ element.

Additional parameters:

vertical_material, horizontal_material: name or index of the vertical/horizontal side materials.

y1, y2, y3, y4: the projected endpoints of the wall lying in the x-y plane as seen below.



log_height, log_offset: additional parameters allowing you to compose a wall from logs. Effective only for straight walls.

status: controls the behavior of log walls.

status = j1 + 2*j2 + 4*j3 + 32*j6 + 64*j7 + 128*j8 + 256*j9

j1: apply right side material on horizontal edges

j2: apply left side material on horizontal edges

j3: start with half log

j6: align texture to wall edges

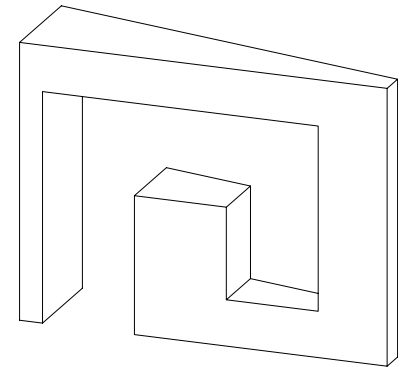
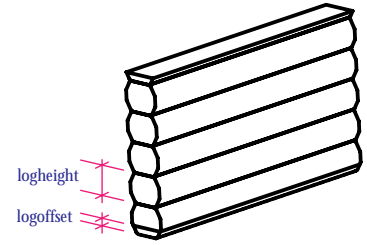
j7: double radius on bended side

j8: square log on the right side

j9: square log on the left side

Example:

```
XWALL_ "Whitewash", "Whitewash", "Whitewash", "Whitewash",
  3.0,
  0.0, 4.0, 4.0, 0.0,
  0.0, 0.0, 0.3, 1.2,
  1.2, 0.0,
  0.0, 0.0,
  15, 15, 15, 15,
  3,
  0.25, 0.0, 1.25, 2.5, -255,
  1.25, 1.5, 2.25, 2.5, -255,
  2.25, 0.5, 3.25, 2.5, -255, 0
```



XWALL_{2}

```
XWALL_{2} 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
```

Extended wall definition based on the same data structure as the XWALL_ element.

Additional parameters:

silldepthi: logical depth of the opening sill. If the j9 bit of the frame_showni parameter is set, the wall side materials wraps the hole polygons, silldepthi defining the separator line between them.

$$\text{frame_showni} = -(1*j1 + 2*j2 + 4*j3 + 8*j4 + 16*j5 + 32*j6 + 64*j7 + 128*j8 + 256*j9 + 512*j10)$$

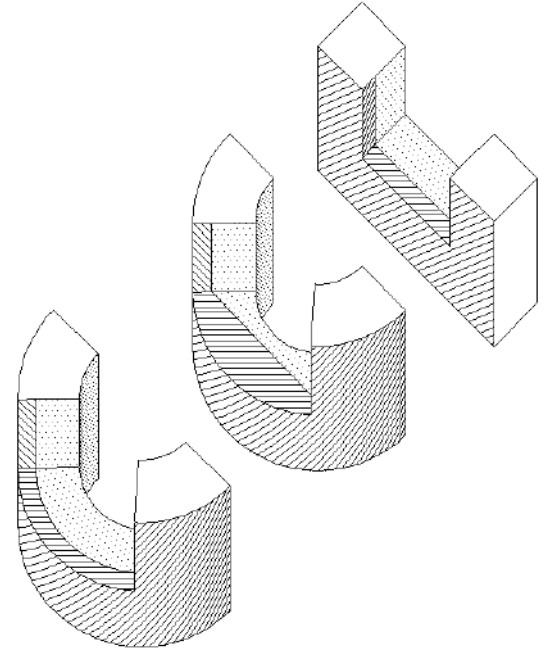
There are two additional values to control the material wrapping. The meaning of the j1, j2 ... j8 values are the same as at the CWALL_ and XWALL_ commands. The j9 value controls the material of the hole polygons. If j9 is 1, the hole inherits the side materials of the wall. The j10 value controls the form of the separator line between the hole materials on the upper and lower polygons of the hole in case of a bent wall. If the j10 value is 1, the separator line will be straight, otherwise curved.

Example:

```

ROTZ 90
xWALL_{2} "C13", "C11", "C12", "C12",
  2, 0, 4, 4, 0,
  0, 0, 1, 1,
  1, 0,
  0, 0,
  15, 15, 15, 15,
  1,
  1, 0.9, 3, 2.1, 0.3, -(255 + 256),
  0,
  0
BODY -1
DEL 1
ADDX 2
xWALL_{2} "C13", "C11", "C12", "C12",
  2, 0, 2 * PI, 2 * PI, 0,
  0, 0, 1, 1,
  0, 0,
  15, 15, 15, 15,
  1,
  1.6, 0.9, 4.6, 2.1, 0.3, -(255 + 256),
  0,
  0
BODY -1
ADDX 4
xWALL_{2} "C13", "C11", "C12", "C12",
  2, 0, 2 * PI, 2 * PI, 0,
  0, 0, 1, 1,
  1, 2,
  0, 0,
  15, 15, 15, 15,
  1,
  1.6, 0.9, 4.6, 2.1, 0.3, -(255 + 256 + 512),
  0,
  0

```



BEAM

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

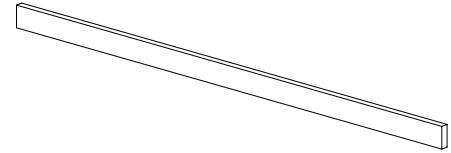
Beam definition. Parameters are similar to those of the XWALL_ element.

Additional parameters:

top_material, bottom_material: top and bottom materials

Example:

```
BEAM 1, 1, 1, 1, 1,
      0.3, 0.0, 7.0, 7.0, 0.0,
      0.0, 0.0, 0.1, 0.1, 0.5,
      15, 15, 15, 15
```



CROOF_

```
CROOF_ top_material, bottom_material, side_material,
         n, xb, yb, xe, ye, height, angle, thickness,
         x1, y1, alpha1, s1,
         ...,
         xn, yn, alphan, sn
```

A sloped roof pitch with custom angle ridges.

top_material, bottom_material, side_material: name/index of the top, bottom and side material.

n: the number of nodes in the roof polygon.

xb, yb, xe, ye: reference line (vector).

height: the height of the roof at the reference line (lower surface).

angle: the rotation angle of the roof plane around the given oriented reference line in degrees (CCW).

thickness: the thickness of the roof measured perpendicularly to the plane of the roof.

x_i, y_i : the coordinates of the nodes of the roof's lower polygon.

α_i : the angle between the face belonging to the edge i of the roof and the plane perpendicular to the roof plane, $-90^\circ < \alpha_i < 90^\circ$. Looking in the direction of the edge of the properly oriented roof polygon, the CCW rotation angle is positive.

The edges of the roof polygon are oriented properly if, in top view, the contour is sequenced CCW and the holes are sequenced CW.

s_i : status code that allows you to control the visibility of polygon edges and side surfaces. You can also define holes and create segments and arcs in the polyline using special constraints.

See *"Status Codes"* on page 137 for details.

Restriction of parameters:

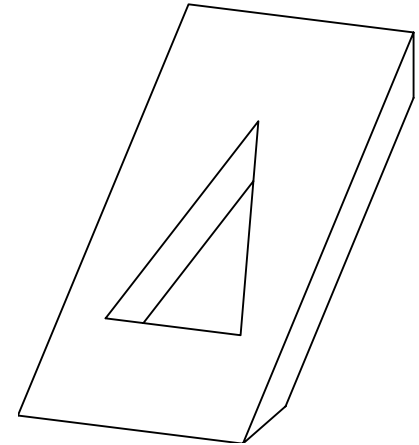
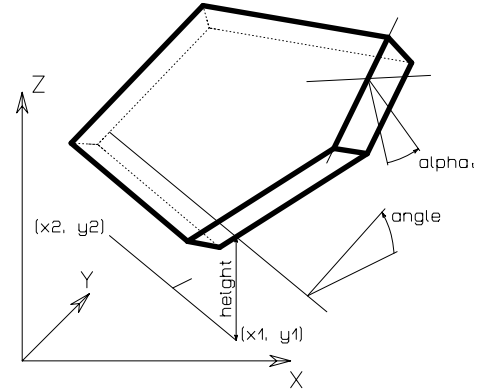
$n \geq 3$

Examples:

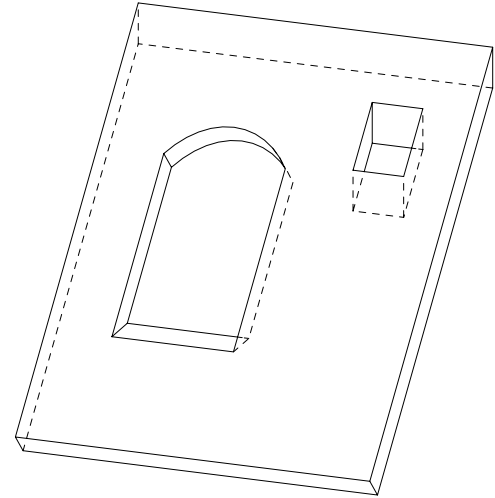
```

CROOF_ 1, 1, 1,           ! materials
        9,
        0, 0,
        1, 0,             ! reference line (xb,yb) (xe,ye)
        0.0,              ! height
        -30,              ! angle
        2.5,              ! thickness
        0, 0, -60, 15,
        10, 0, 0, 15,
        10, 20, -30, 15,
        0, 20, 0, 15,
        0, 0, 0, -1,
        2, 5, 0, 15,
        8, 5, 0, 15,
        5, 15, 0, 15,
        2, 5, 0, -1

```



```
L=0.25
R=(0.6^2+L^2)/(2*L)
A=ASN(0.6/R)
CROOF_ "Roof Tile", "Pine", "Pine",
      16, 2, 0, 0,
      0, 0, 45, -0.2*SQR(2),
      0, 0, 0, 15,
      3.5, 0, 0, 15,
      3.5, 3, -45, 15,
      0, 3, 0, 15,
      0, 0, 0, -1,
      0.65, 1, -45, 15,
      1.85, 1, 0, 15,
      1.85, 2.4-L, 0, 13,
      1.25, 2.4-R, 0, 900,
      0, 2*A, 0, 4015,
      0.65, 1, 0, -1,
      2.5, 2, 45, 15,
      3, 2, 0, 15,
      3, 2.5, -45, 15,
      2.5, 2.5, 0, 15,
      2.5, 2, 0, -1
```



MESH

MESH $a, b, m, n, \text{mask},$
 $z_{11}, z_{12}, \dots, z_{1m},$
 $z_{21}, z_{22}, \dots, z_{2m},$
 \dots
 $z_{n1}, z_{n2}, \dots, z_{nm}$

A simple smooth mesh based on a rectangle with an equidistant net. The sides of the base rectangle are a and b ; the m and n points are along the x and y axes respectively; z_{ij} is the height of the node.

Masking:

$$\text{mask} = j_1 + 4*j_3 + 16*j_5 + 32*j_6 + 64*j_7$$

where j_1, j_3, j_5, j_6, j_7 can be 0 or 1.

j_1 (1): base surface is present.

j_3 (4): side surfaces are present.

j_5 (16): base and side edges are visible.

j_6 (32): top edges are visible.

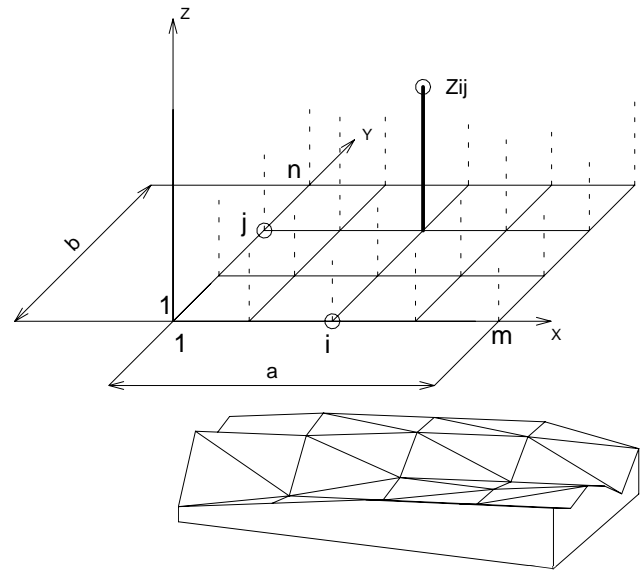
j_7 (64): top edges are visible, top surface is not smooth.

Parameter restrictions:

$$m \geq 2, n \geq 2$$

Examples:

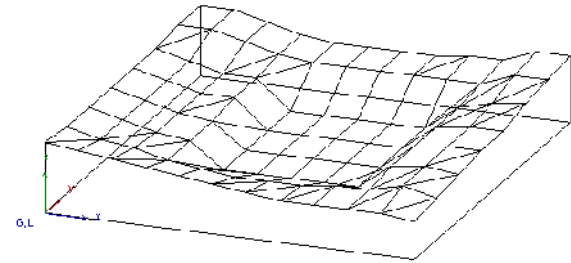
```
MESH 50, 30, 5, 6, 1+4+16+32+64,
      2, 4, 6, 7, 8,
      10, 3, 4, 5, 6,
      7, 9, 5, 5, 7,
      8, 10, 9, 4, 5,
      6, 7, 9, 8, 2,
      4, 5, 6, 8, 6
```



```

MESH 90,100, 12,8, 1+4+16+32+64,
 17,16,15,14,13,12,11,10,10,10,10, 9,
 16,14,13,11,10, 9, 9, 9,10,10,12,10,
 16,14,12,11, 5, 5, 5, 5, 5,11,12,11,
 16,14,12,11, 5, 5, 5, 5, 5,11,12,12,
 16,14,12,12, 5, 5, 5, 5, 5,11,12,12,
 16,14,12,12, 5, 5, 5, 5, 5,11,13,14,
 17,17,15,13,12,12,12,12,12,12,15,15,
 17,17,15,13,12,12,12,12,13,13,16,16

```



ARMC

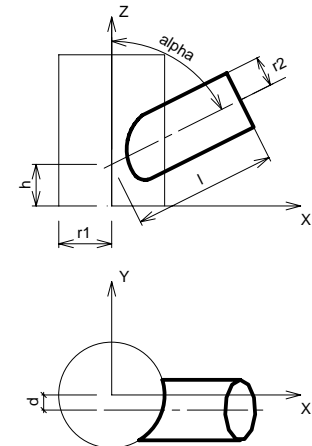
ARMC r1, r2, l, h, d, alpha

A piece of tube starting from another tube; parameters according to the figure (penetration curves are also calculated and drawn). alpha is in degrees.

Restrictions of parameters:

$$r1 \geq r2 + d$$

$$r1 \leq l \cdot \sin(\alpha) - r2 \cdot \cos(\alpha)$$

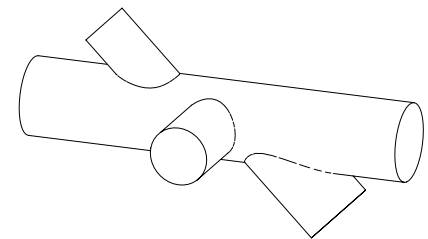


Example:

```

ROTY 90
CYLIND 10,1
ADDZ 6
ARMC 1, 0.9, 3, 0, 0, 45
ADDZ -1
ROTZ -90
ARMC 1, 0.75, 3, 0, 0, 90
ADDZ -1
ROTZ -90
ARMC 1, 0.6, 3, 0, 0, 135

```



ARME

ARME l, r1, r2, h, d

A piece of tube starting from an ellipsoid in the y-z plane; parameters according to the figure (penetration lines are also calculated and drawn).

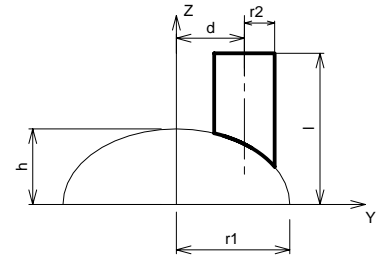
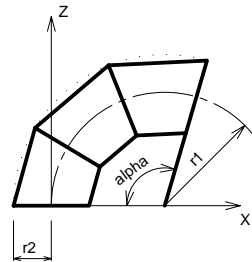
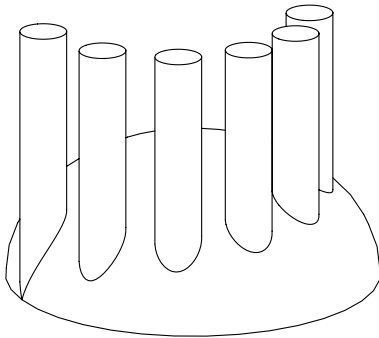
Restrictions of the parameters:

$$r1 \geq r2 + d$$

$$l \geq h * \sqrt{(1 - (r2 - d)^2 / r1^2)}$$

Example:

```
ELLIPS 3,4
FOR i=1 TO 6
  ARME 6,4,0.5,3,3.7-0.2*i
  ROTZ 30
NEXT i
```



ELBOW

ELBOW r1, alpha, r2

A segmented elbow in the x-z plane. The radius of the arc is r1, the angle is alpha and the radius of the tube segment is r2. The alpha value is in degrees.

Restriction of parameters:

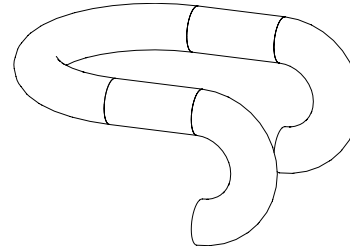
$$r1 > r2$$

Example:

```

ROTY      90
ELBOW    2.5, 180, 1
ADDZ     -4
CYLIND   4, 1
ROTZ     -90
MULZ     -1
ELBOW    5, 180, 1
DEL       1
ADDX     10
CYLIND   4, 1
ADDZ     4
ROTZ     90
ELBOW    2.5, 180, 1

```



PLANAR SHAPES IN 3D

The drawing elements presented in this section can be used in 3D scripts, allowing you to define points, lines, arcs, circles and planar polygons in the three-dimensional space.

HOTSPOT

HOTSPOT *x, y, z* [, unID [, paramReference, flags] [, displayParam]]

A 3D hotspot in the point (*x, y, z*).

unID is the unique identifier of the hotspot in the 3D script. It is useful if you have a variable number of hotspots.

paramReference: parameter that can be edited by this hotspot using the graphical hotspot based parameter editing method.

displayParam: parameter to display in the information palette when editing the paramReference parameter. Members of arrays can be passed as well.

See [“Graphical Editing” on page 131](#) for using HOTSPOT.

LIN_

LIN_ *x1, y1, z1, x2, y2, z2*

A line segment between the points P1 (*x1,y1,z1*) and P2 (*x2,y2,z2*).

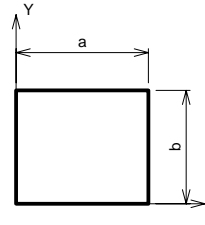
RECT

RECT a, b

A rectangle in the x-y plane with sides a and b.

Restriction of parameters:

a, b \geq 0



POLY

POLY n, x1, y1, ... xn, yn

A polygon with n edges in the x-y plane. The coordinates of nodei are (xi, yi, 0).

Restriction of parameters:

n \geq 3

POLY_

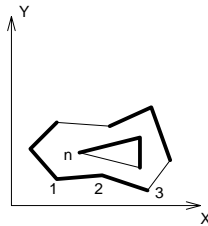
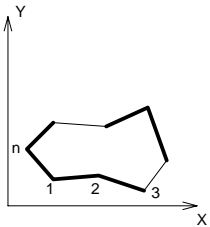
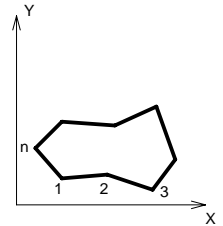
POLY_ n, x1, y1, s1, ... xn, yn, sn

Similar to the normal POLY statement, but any of the edges can be omitted. If si = 0, the edge starting from the (xi,yi) apex will be omitted. If si = 1, the edge will be shown.

si = -1 is used to define holes directly.

Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *"Additional Status Codes"* on page 139 for details.



Restriction of parameters:

$n \geq 3$

PLANE

PLANE $n, x_1, y_1, z_1, \dots, x_n, y_n, z_n$

A polygon with n edges on an arbitrary plane. The coordinates of node i are (x_i, y_i, z_i) . The polygon must be planar in order to get a correct shading/rendering result, but the interpreter does not check this condition.

Restriction of parameters:

$n \geq 3$

PLANE_

PLANE_ $n, x_1, y_1, z_1, s_1, \dots, x_n, y_n, z_n, s_n$

Similar to the normal PLANE statement, but any of the edges can be omitted as in the POLY_ statement.

Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *“Additional Status Codes” on page 139*.

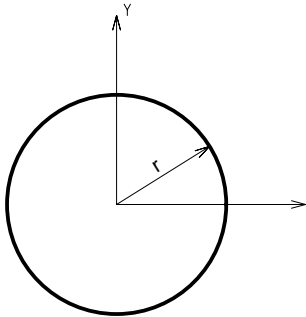
Restriction of parameters:

$n \geq 3$

CIRCLE

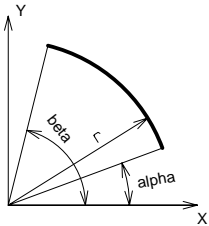
CIRCLE r

A circle in the x-y plane with its center at the origin and with a radius of r .



ARC

ARC r , alpha, beta



An arc (in Wireframe mode) or sector (in other modes) in the x-y plane with its center at the origin from angle alpha to beta with a radius of r . alpha and beta are in degrees.

SHAPES GENERATED FROM POLYLINES

These elements let you create complex 3D shapes using a polyline and a built-in rule. You can rotate, project or translate the given polyline. The resulting bodies are a generalization of some previously described elements like PRISM_ and CYLIND.

Shapes generated from a single polyline:

EXTRUDE
PYRAMID
REVOLVE

Shapes generated from two polylines:

RULED
SWEEP
TUBE
TUBEA

The first polyline is always in the x-y plane. Points are determined by two coordinates; the third value is the status (see below). The second polyline (for RULED and SWEEP) is a space curve. Apices are determined by three coordinate values.

Shape generated from four polylines:

COONS

Shape generated from any number of polylines:

MASS

General restrictions for polylines

- Adjacent vertices must not be coincident (except RULED).

- The polyline must not intersect itself (this is not checked by the program, but hidden line removal and rendering will be incorrect).
- The polylines may be either open or closed. In the latter case, the first node must be repeated after the last one of the contour.

Masking

Mask values are used to show or hide characteristic surfaces and/or edges of the 3D shape. The mask values are specific to each element and you can find a more detailed description in their corresponding sections/chapters.

$$\text{mask} = j1 + 2*j2 + 4*j3 + 8*j4 + 16*j5 + 32*j6 + 64*j7$$

where $j1, j2, j3, j4, j5, j6, j7$ can be 0 or 1.

$j1, j2, j3, j4$ represent whether the surfaces are present (1)
or omitted (0).

$j5, j6, j7$ represent whether the edges are visible (1)
or invisible (0).

$j1$: base surface.

$j2$: top surface.

$j3$: side surface.

$j4$: other side surface.

$j5$: base edges.

$j6$: top edges.

$j7$: cross-section/surface edges are visible, surface is not smooth.

To enable all faces and edges, set mask value to 127.

Status

Status values are used to state whether a given point of the polyline will leave a sharp trace of its rotation path behind.

0: latitudinal arcs/lateral edges starting from the node are all visible.

1: latitudinal arcs/lateral edges starting from the node are used only for showing the contour.

-1: for EXTRUDE only: it marks the end of the enclosing polygon or a hole, and means that the next node will be the first node of another hole.

Additional status codes allow you to create segments and arcs in the polyline using special constraints.

See *"Additional Status Codes"* on page 139 for details.

To create a smooth 3D shape, set all status values to 1. Use status = 0 to create a ridge.

Other values are reserved for future enhancements.

EXTRUDE

EXTRUDE *n*, *dx*, *dy*, *dz*, *mask*, *x1*, *y1*, *s1*,
 ..., *xn*, *yn*, *sn*

General prism using a polyline base in the x-y plane. The displacement vector between bases is (*dx*, *dy*, *dz*).

This is a generalization of the PRISM and SLAB statements. The base polyline is not necessarily closed, as the lateral edges are not always perpendicular to the x-y plane. The base polyline may include holes, just like PRISM_. It is possible to control the visibility of the contour edges.

n: the number of polyline nodes.

mask: controls the existence of the bottom, top and (in the case of an open polyline) side polygon.

si: status of the lateral edges or marks the end of the polygon or of a hole. You can also define arcs and segments in the polyline using additional status code values.

Parameter restriction:

$n > 2$

Masking:

$mask = j1 + 2*j2 + 4*j3 + 16*j5 + 32*j6$

where *j1*, *j2*, *j3*, *j5*, *j6* can be 0 or 1.

j1 (1): base surface is present.

j2 (2): top surface is present.

j3 (4): side (closing) surface is present.

j5 (16): base edges are visible.

j6 (32): top edges are visible.

Status values:

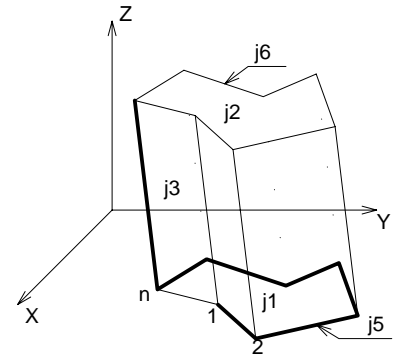
0: lateral edge starting from the node is visible.

1: lateral edges starting from the node are used for showing the contour.

-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.

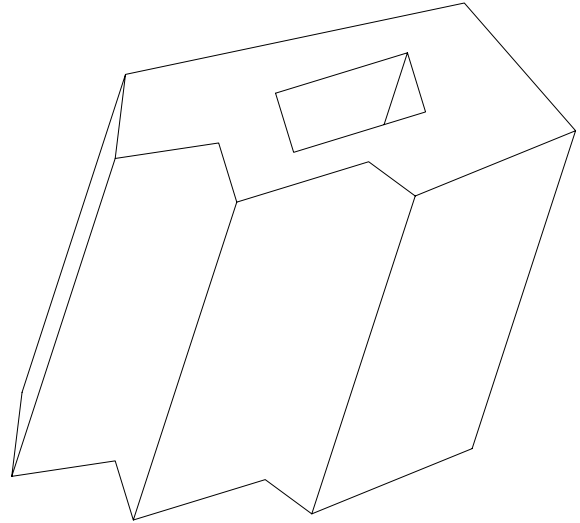
Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See ["Additional Status Codes" on page 139](#) for details.



Examples:

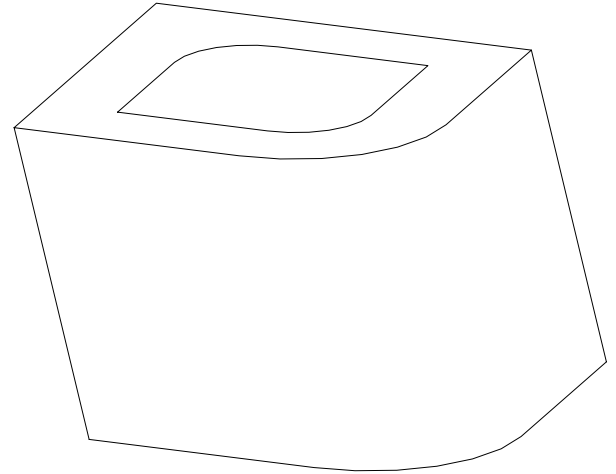
```
EXTRUDE 14, 1, 1, 4, 1+2+4+16+32,  
0, 0, 0,  
1, -3, 0,  
2, -2, 1,  
3, -4, 0,  
4, -2, 1,  
5, -3, 0,  
6, 0, 0,  
3, 4, 0,  
0, 0, -1,  
2, 0, 0,  
3, 2, 0,  
4, 0, 0,  
3, -2, 0,  
2, 0, -1
```




```

A=5: B=5
R=2: S=1
C=R-S
D=A-R
E=B-R
EXTRUDE 28, -1, 0, 4, 1+2+4+16+32,
0, 0, 0,
D+R*SIN(0), R-R*COS(0), 1,
D+R*SIN(15), R-R*COS(15), 1,
D+R*SIN(30), R-R*COS(30), 1,
D+R*SIN(45), R-R*COS(45), 1,
D+R*SIN(60), R-R*COS(60), 1,
D+R*SIN(75), R-R*COS(75), 1,
D+R*SIN(90), R-R*COS(90), 1,
A, B, 0,
0, B, 0,
0, 0, -1,
C, C, 0,
D+S*SIN(0), R-S*COS(0), 1,
D+S*SIN(15), R-S*COS(15), 1,
D+S*SIN(30), R-S*COS(30), 1,
D+S*SIN(45), R-S*COS(45), 1,
D+S*SIN(60), R-S*COS(60), 1,
D+S*SIN(75), R-S*COS(75), 1,
D+S*SIN(90), R-S*COS(90), 1,
A-C, B-C, 0,
R-S*COS(90), E+S*SIN(90), 1,
R-S*COS(75), E+S*SIN(75), 1,
R-S*COS(60), E+S*SIN(60), 1,
R-S*COS(45), E+S*SIN(45), 1,
R-S*COS(30), E+S*SIN(30), 1,
R-S*COS(15), E+S*SIN(15), 1,
R-S*COS(0), E+S*SIN(0), 1,
C, C, -1

```



PYRAMID

PYRAMID $n, h, \text{mask}, x_1, y_1, s_1, \dots, x_n, y_n, s_n$

Pyramid based on a polyline in the x-y plane. The peak of the pyramid is located at $(0, 0, h)$.

n : number of polyline nodes.

mask : controls the existence of the bottom and (in the case of an open polyline) side polygon.

s_i : status of the lateral edges.

Parameter restrictions:

$h > 0$ and $n > 2$

Masking:

$\text{mask} = j_1 + 4*j_3 + 16*j_5$

where j_1, j_3, j_5 can be 0 or 1.

j_1 (1): base surface is present.

j_3 (4): side (closing) surface is present.

j_5 (16): base edges are visible.

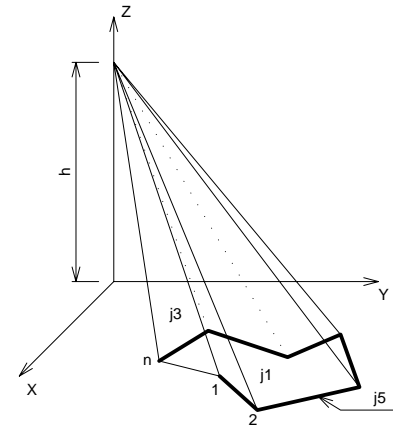
Status values:

0: lateral edges starting from the node are all visible.

1: lateral edges starting from the node are used for showing the contour.

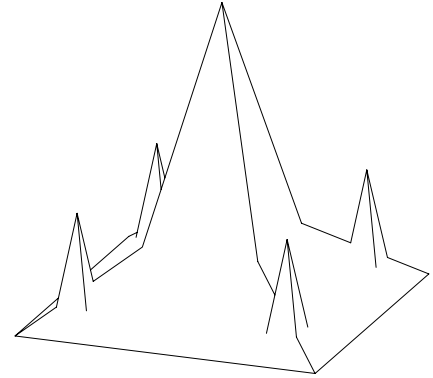
Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *"Additional Status Codes"* on page 139 for details.



Example:

```
PYRAMID 4, 1.5, 1+4+16,  
        -2, -2, 0,  
        -2, 2, 0,  
        2, 2, 0,  
        2, -2, 0  
PYRAMID 4, 4, 21,  
        -1, -1, 0,  
        1, -1, 0,  
        1, 1, 0,  
        -1, 1, 0  
ADDX -1.4  
ADDY -1.4  
GOSUB 100  
ADDX 2.8  
GOSUB 100  
ADDY 2.8  
GOSUB 100  
ADDX -2.8  
GOSUB 100  
END  
100:  
PYRAMID 4, 1.5, 21,  
        -0.25, -0.25, 0,  
        0.25, -0.25, 0,  
        0.25, 0.25, 0,  
        -0.25, 0.25, 0  
RETURN
```



REVOLVE

REVOLVE *n*, *alpha*, *mask*, *x1*, *y1*, *s1*, ... *xn*, *yn*, *sn*

Surface generated by rotating a polyline defined in the x-y plane around the x axis.

n: number of polyline nodes.

alpha: sweep angle in degrees.

mask: controls the existence of the bottom, top and (in the case of $\alpha < 360^\circ$) side polygons.

si: status of the latitudinal arcs.

Parameter restrictions:

$n \geq 2$

$y_i \geq 0.0$

y_i and y_{i+1} (i.e., the y value of two neighboring nodes) cannot be zero at the same time.

Masking:

$mask = j_1 + 2*j_2 + 4*j_3 + 8*j_4 + 16*j_5 + 32*j_6 + 64*j_7$

where $j_1, j_2, j_3, j_4, j_5, j_6, j_7$ can be 0 or 1.

j_1 (1): base surface is present.

j_2 (2): top surface is present.

j_3 (4): side surface is present at initial angle.

j_4 (8): side surface is present at final angle.

j_5 (16): edges on side surface at initial angle are visible.

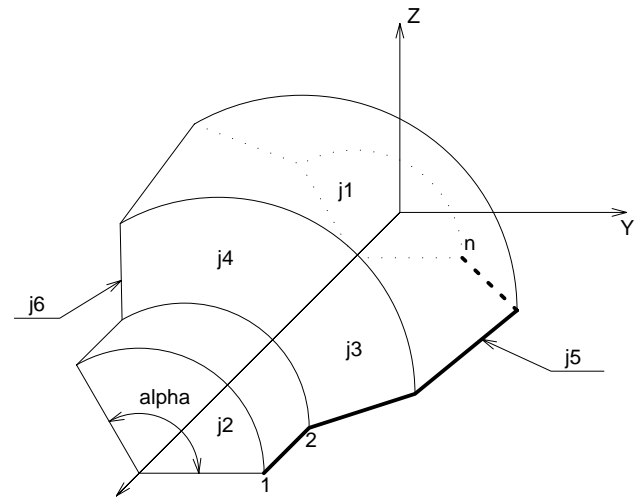
j_6 (32): edges on side surface at final angle are visible.

j_7 (64): cross-section edges are visible, surface is not smooth.

Status values:

0: latitudinal arcs starting from the node are all visible.

1: latitudinal arcs starting from the node are used for showing the contour.

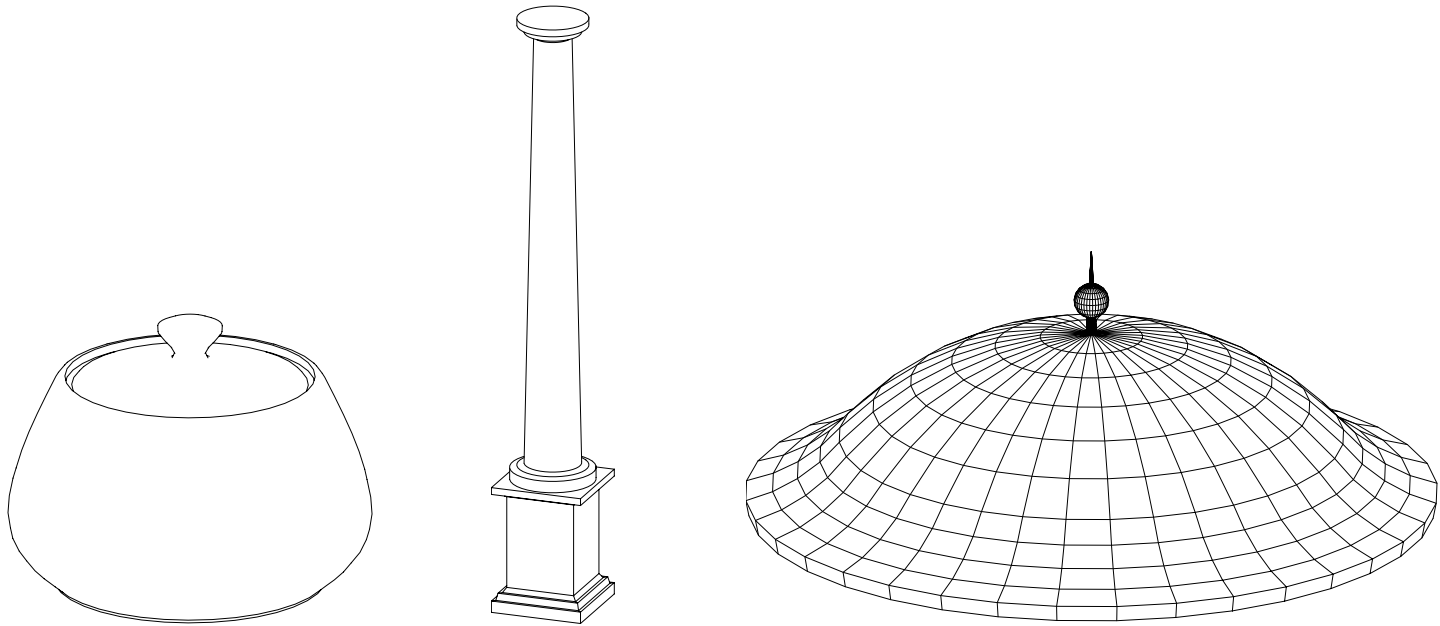


2: when using ArchiCAD/ArchiFM or Z-buffer Rendering Engine and setting Smooth Surfaces, the latitudinal edge belonging to this point defines a break. This solution is equivalent to the definition of additional nodes. The calculation is performed by the compiler. With other rendering methods, it has the same effect as using 0.

Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *“Additional Status Codes” on page 139 for details.*

Examples:



ROTY -90

```
REVOLVE 22, 360, 1+64,  
0, 1.982, 0,  
0.093, 2, 0,  
0.144, 1.845, 0,  
0.220, 1.701, 0,  
0.318, 1.571, 0,  
0.436, 1.459, 0,  
0.617, 1.263, 0,  
0.772, 1.045, 0,  
0.896, 0.808, 0,  
0.987, 0.557, 0,  
1.044, 0.296, 0,  
1.064, 0.030, 0,  
1.167, 0.024, 0,  
1.181, 0.056, 0,  
1.205, 0.081, 0,  
1.236, 0.096, 0,  
1.270, 0.1, 0,  
1.304, 0.092, 0,  
1.333, 0.073, 0,  
1.354, 0.045, 0,  
1.364, 0.012, 0,  
1.564, 0, 0
```

**workaround without
status code 2:**

```

ROTY -90
REVOLVE 26, 180, 16+32,
7, 1, 0,
6.0001, 1, 1,
6, 1, 0,
5.9999, 1.0002, 1,
5.5001, 1.9998, 1,
5.5, 2, 0,
5.4999, 1.9998, 1,
5.0001, 1.0002, 1,
5, 1, 0,
4.9999, 1, 1,
4.0001, 1, 1,
4, 1, 0,
3+COS(15), 1+SIN(15), 1,
3+COS(30), 1+SIN(30), 1,
3+COS(45), 1+SIN(45), 1,
3+COS(60), 1+SIN(60), 1,
3+COS(75), 1+SIN(75), 1,
3, 2, 1,
3+COS(105), 1+SIN(105), 1,
3+COS(120), 1+SIN(120), 1,
3+COS(135), 1+SIN(135), 1,
3+COS(150), 1+SIN(150), 1,
3+COS(165), 1+SIN(165), 1,
2, 1, 0,
1.9999, 1, 0,
1, 1, 0

```

RULED

```

RULED n, mask,
    u1, v1, s1, ... un, vn, sn,
    x1, y1, z1, ... xn, yn, zn

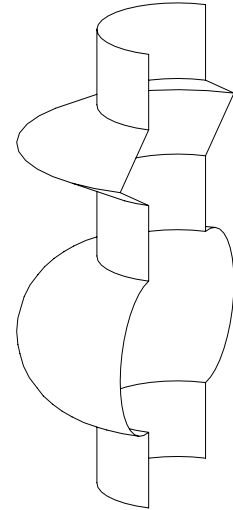
```

**the same result with
status code 2:**

```

ROTY -90
REVOLVE 18, 180, 48,
7, 1, 0,
6, 1, 2,
5.5, 2, 2,
5, 1, 2,
4, 1, 2,
3+COS(15), 1+SIN(15), 1,
3+COS(30), 1+SIN(30), 1,
3+COS(45), 1+SIN(45), 1,
3+COS(60), 1+SIN(60), 1,
3+COS(75), 1+SIN(75), 1,
3, 2, 1,
3+COS(105), 1+SIN(105), 1,
3+COS(120), 1+SIN(120), 1,
3+COS(135), 1+SIN(135), 1,
3+COS(150), 1+SIN(150), 1,
3+COS(165), 1+SIN(165), 1,
2, 1, 2,
1, 1, 0

```



RULED{2}

RULED{2} *n*, *mask*,
 u1, *v1*, *s1*, ... *un*, *vn*, *sn*,
 x1, *y1*, *z1*, ... *xn*, *yn*, *zn*

RULED is a surface based on a planar curve and a space curve having the same number of nodes. Straight segments connect the corresponding nodes of the two polylines.

This is the only GDL element allowing the neighboring nodes to overlap.

The second version, RULED {2}, checks the direction (clockwise or counterclockwise) in which the points of both the top polygon and base polygon were defined, and reverses the direction if necessary. (The original RULED command takes only the base polygon into account, which can lead to errors.)

n: number of polyline nodes in each curve.

mask: controls the existence of the bottom, top and side polygon and the visibility of the edges on the generator polylines. The side polygon connects the first and last nodes of the curves, if any of them are not closed.

ui, *vi*: coordinates of the planar curve nodes.

si: status of the lateral edges.

xi, *yi*, *zi*: coordinates of the space curve nodes.

Parameter restriction:

$n > 1$

Masking:

$mask = j1 + 2*j2 + 4*j3 + 16*j5 + 32*j6 + 64*j7$

where *j1*, *j2*, *j3*, *j5*, *j6*, *j7* can be 0 or 1.

j1 (1): base surface is present.

j2 (2): top surface is present (not effective if the top surface is not planar).

j3 (4): side surface is present (a planar quadrangle or two triangles).

j5 (16): edges on the planar curve are visible.

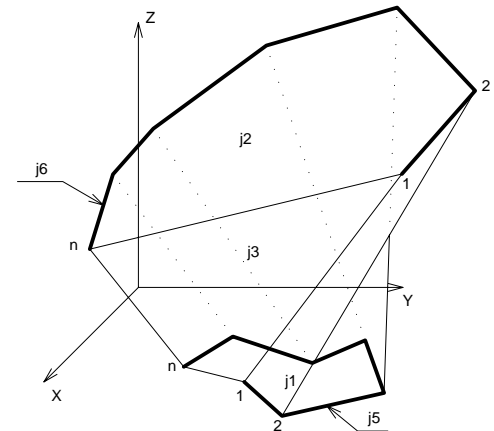
j6 (32): edges on the space curve are visible.

j7 (64): edges on the surface are visible, surface is not smooth.

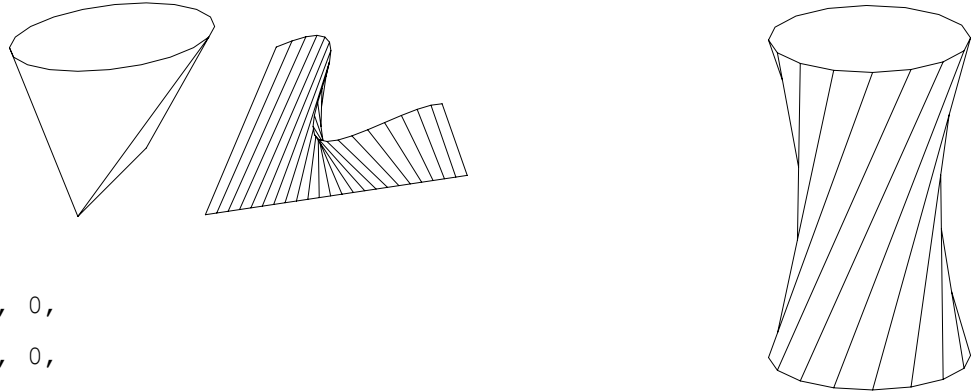
Status values:

0: lateral edges starting from the node are all visible.

1: lateral edges starting from the node are used for showing the contour.



Examples:



R=3

```

RULED 16, 1+2+4+16+32,
COS (22.5)*R, SIN(22.5)*R, 0,
COS (45)*R, SIN(45)*R, 0,
COS (67.5)*R, SIN(67.5)*R, 0,
COS (90)*R, SIN(90)*R, 0,
COS (112.5)*R, SIN(112.5)*R, 0,
COS (135)*R, SIN(135)*R, 0,
COS (157.5)*R, SIN(157.5)*R, 0,
COS (180)*R, SIN(180)*R, 0,
COS (202.5)*R, SIN(202.5)*R, 0,
COS (225)*R, SIN(225)*R, 0,
COS (247.5)*R, SIN(247.5)*R, 0,
COS (270)*R, SIN(270)*R, 0,
COS (292.5)*R, SIN(292.5)*R, 0,
COS (315)*R, SIN(315)*R, 0,
COS (337.5)*R, SIN(337.5)*R, 0,
COS (360)*R, SIN(360)*R, 0,
COS (112.5)*R, SIN(112.5)*R, 1,
COS (135)*R, SIN(135)*R, 1,
COS (157.5)*R, SIN(157.5)*R, 1,
COS (180)*R, SIN(180)*R, 1,
COS (202.5)*R, SIN(202.5)*R, 1,
COS (225)*R, SIN(225)*R, 1,
COS (247.5)*R, SIN(247.5)*R, 1,
COS (270)*R, SIN(270)*R, 1,
COS (292.5)*R, SIN(292.5)*R, 1,
COS (315)*R, SIN(315)*R, 1,
COS (337.5)*R, SIN(337.5)*R, 1,
COS (360)*R, SIN(360)*R, 1,
COS (22.5)*R, SIN(22.5)*R, 1,
COS (45)*R, SIN(45)*R, 1,
COS (67.5)*R, SIN(67.5)*R, 1,
COS (90)*R, SIN(90)*R, 1

```

SWEEP

SWEEP *n, m, alpha, scale, mask,*
u1, v1, s1, ... un, vn, sn,
x1, y1, z1, ... xm, ym, zm

Surface generated by a polyline sweeping along a polyline space curve path.

The plane of the polyline follows the path curve. The space curve has to start from the x-y plane. If this condition is not met, it is moved along the z axis to start on the x-y plane.

The cross-section at point (x_i, y_i, z_i) is perpendicular to the space curve segment between points $(x_{i-1}, y_{i-1}, z_{i-1})$ and (x_i, y_i, z_i) .

SWEEP can be used to model the spout of a teapot and other complex shapes.

n: number of polyline nodes.

m: number of path nodes.

alpha: incremental polyline rotation on its own plane, from one path node to the next one.

scale: incremental polyline scale factor, from one path node to the next one.

mask: controls the existence of the bottom and top polygons' surfaces and edges.

u_i, v_i: coordinates of the base polyline nodes.

s_i: status of the lateral edges.

x_i, y_i, z_i: coordinates of the path curve nodes.

Parameter restrictions:

n > 1
m > 1
z1 < *z2*

Masking:

mask = $j_1 + 2*j_2 + 4*j_3 + 16*j_5 + 32*j_6 + 64*j_7$
 where $j_1, j_2, j_3, j_5, j_6, j_7$ can be 0 or 1.

j_1 (1): base surface is present.

j_2 (2): top surface is present.

j_3 (4): side surface is present.

j_5 (16): base edges are visible.

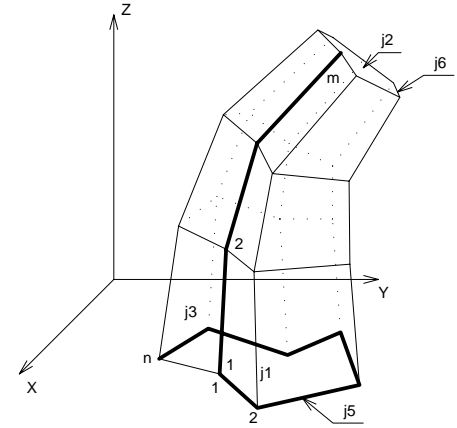
j_6 (32): top edges are visible.

j_7 (64): cross-section edges are visible, surface is articulated.

Status values:

0: lateral edges starting from the node are all visible.

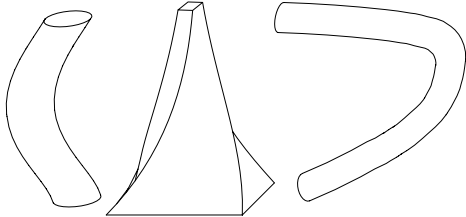
1: lateral edges starting from the node are used for showing the contour.



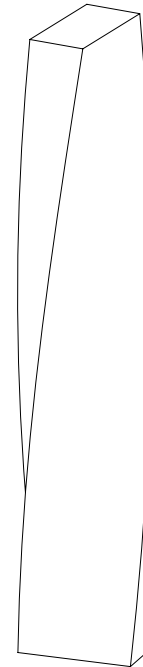
Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *“Additional Status Codes”* on page 139 for details.

Examples:



```
SWEEP 4, 12, 7.5, 1, 1+2+4+16+32,
      -0.5, -0.25, 0,
      0.5, -0.25, 0,
      0.5, 0.25, 0,
      -0.5, 0.25, 0,
      0, 0, 0.5,
      0, 0, 1,
      0, 0, 1.5,
      0, 0, 2,
      0, 0, 2.5,
      0, 0, 3,
      0, 0, 3.5,
      0, 0, 4,
      0, 0, 4.5,
      0, 0, 5,
      0, 0, 5.5,
      0, 0, 6
```



TUBE

```
TUBE n, m, mask,
      u1, w1, s1,
      ...
      un, wn, sn,
      x1, y1, z1, angle1,
      ...
      xm, ym, zm, anglem
```

Surface generated by a polyline sweeping along a space curve path without distortion of the generating cross-section. The internal connection surfaces are rotatable in the U-W plane of the instantaneous U-V-W coordinate system.

V axis: approximates the tangent of the generator curve at the corresponding point,

W axis: perpendicular to the V axis and pointing upward with respect to the local z axis,

U axis: perpendicular to the V and W axes and forms with them a right-hand sided Cartesian coordinate system.

If the V axis is vertical, then the W direction is not correctly defined. The W axis in the previous path node is used for determining a horizontal direction.

The cross-section polygon of the tube measured at the middle of the path segments is always equal to the base polygon (u1, w1, ... un, wn). Section polygons in joints are situated in the bisector plane of the joint segments. The base polygon must be closed.

n: number of the polyline nodes.

m: number of the path nodes.

ui, wi: coordinates of the base polyline nodes.

si: status of the lateral edges.

xi, yi, zi: coordinates of the path curve nodes.

Note: The path comprises two points more than the number of generated sections. The first and the last points determine the position in space of the first and the last surfaces belonging to the TUBE. These points only play a role in determining the normal of the surfaces, they are not actual nodes of the path. 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 were continued in the directions indicated by these.)

anglei: rotation angle of the cross-section.

Masking:

```
mask = j1 + 2*j2 + 16*j5 + 32*j6 + 64*j7
where j1, j2, j5, j6, j7 can be 0 or 1.
```

j1 (1): base surface is present.

j2 (2): end surface is present.

j5 (16): base edges (at x_2, y_2, z_2) are visible.

j6 (32): end edges (at $x_{m-1}, y_{m-1}, z_{m-1}$) are visible.

j7 (64): cross-section edges are visible, surface is articulated.

Parameter restrictions:

$n > 2$ and $m > 3$

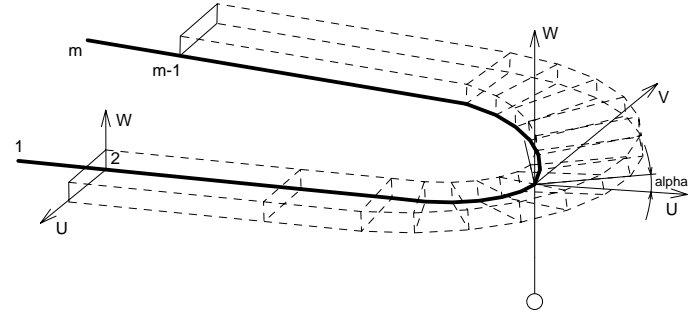
Status values:

0: lateral edges starting from the node are all visible.

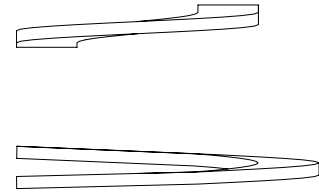
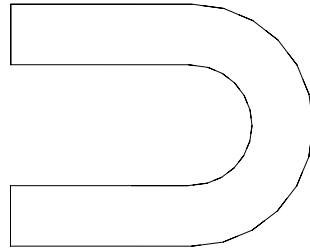
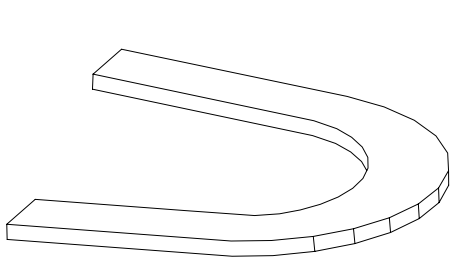
1: lateral edges starting from the node are used for showing the contour.

Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *"Parametric Objects"* on page 199 of *ArchiCAD 9 Reference Guide*.



Examples:



```
TUBE 4, 18, 16+32,
      2.0, 0.0, 0,
      0.0, 0.0, 0,
      0.0, 0.4, 0,
      2.0, 0.4, 0,
      -1, 0, 0, 0,
      0, 0, 0, 0,
      4, 0, 0.1, 0,
      6, 0, 0.15, 0,
      6+4*SIN(15), 4 - 4*COS(15), 0.2, 0,
      6+4*SIN(30), 4 - 4*COS(30), 0.25, 0,
      6+4*SIN(45), 4 - 4*COS(45), 0.3, 0,
      6+4*SIN(60), 4 - 4*COS(60), 0.35, 0,
      6+4*SIN(75), 4 - 4*COS(75), 0.4, 0,
      10, 4, 0.45, 0,
      6+4*SIN(105), 4 - 4*COS(105), 0.5, 0,
      6+4*SIN(120), 4 - 4*COS(120), 0.55, 0,
      6+4*SIN(135), 4 - 4*COS(135), 0.6, 0,
      6+4*SIN(150), 4 - 4*COS(150), 0.65, 0,
      6+4*SIN(165), 4 - 4*cos(165), 0.7, 0,
      6, 8, 0.75, 0,
      0, 8, 1, 0,
      -1, 8, 1, 0
```

```

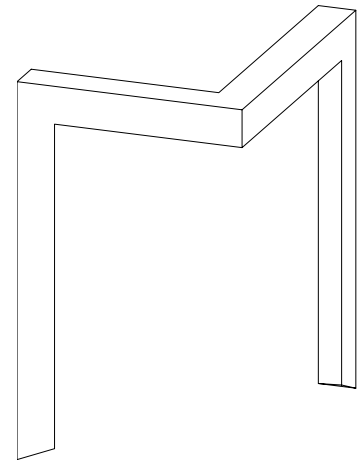
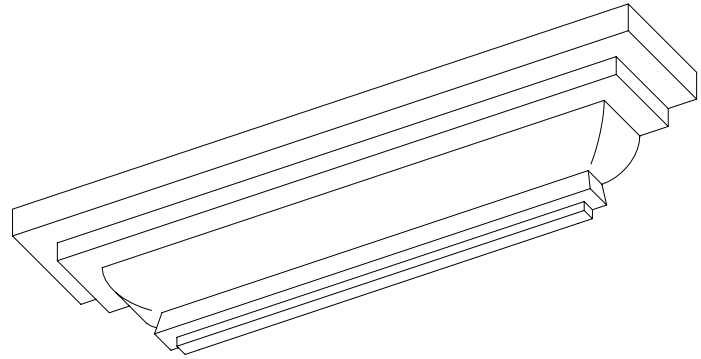
TUBE 14, 6, 1+2+16+32,
      0, 0, 0,
      0.03, 0, 0,
      0.03, 0.02, 0,
      0.06, 0.02, 0,
      0.05, 0.0699, 0,
      0.05, 0.07, 1,
      0.05, 0.15, 901,
      1, 0, 801,
      0.08, 90, 2000,
      0.19, 0.15, 0,
      0.19, 0.19, 0,
      0.25, 0.19, 0,
      0.25, 0.25, 0,
      0, 0.25, 0,
      0, 1, 0, 0,
      0, 0.0001, 0, 0,
      0, 0, 0, 0,
      -0.8, 0, 0, 0,
      -0.8, 0.0001, 0, 0,
      -0.8, 1, 0, 0

```

```

TUBE 3, 7, 16+32,
      0, 0, 0,
      -0.5, 0, 0,
      0, 0.5, 0,
      0.2, 0, -0.2, 0,
      0, 0, 0, 0,
      0, 0, 5, 0,
      3, 0, 5, 0,
      3, 4, 5, 0,
      3, 4, 0, 0,
      3, 3.8, -0.2, 0

```



TUBEA

TUBEA $n, m, \text{mask},$
 $u1, w1, s1,$
 \dots
 $un, wn, sn,$
 $x1, y1, z1,$
 \dots
 xm, ym, zm

TUBEA is a surface generated by a polyline sweeping along a space curve path with a different algorithm than that of the TUBE statement.

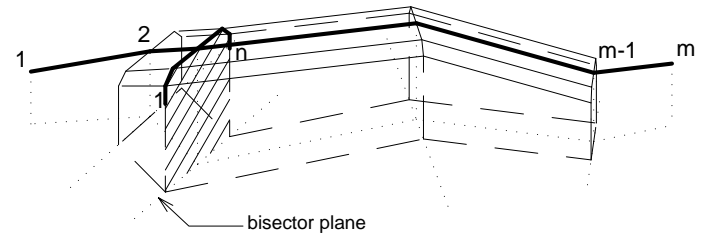
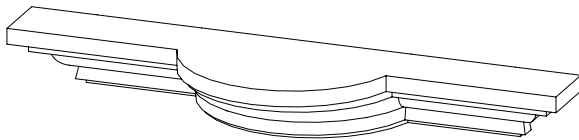
The section polygon generated in each joint of the path curve is equal with the base polygon ($u1, w1, \dots un, wn$) and is situated in the bisector plane of the projections of the joint segments to the local x-y plane. The base polygon can be opened: in this case the section polygons will be generated to reach the local x-y plane as in the case of REVOLVE surfaces.

The cross section of the tube measured at the middle of the path segments can be different from the base polygon.

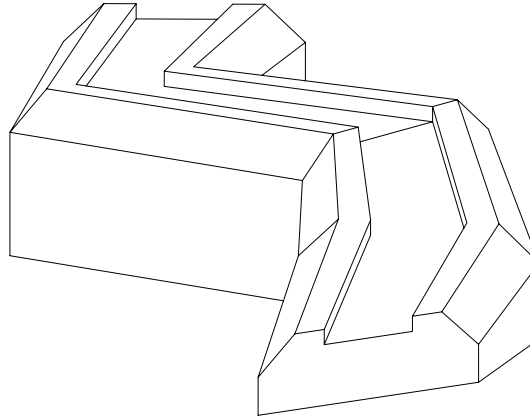
Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *“Additional Status Codes”* on page 139 for details.

Examples:



```
TUBEA 9, 7, 1 + 2 + 16 + 32,  
-1, 1, 0,  
0, 2, 0,  
0.8, 2, 0,  
0.8, 1.6, 0,  
0.8001, 1.6, 1,  
3.2, 1.6, 0,  
3.2, 2, 0,  
4, 2, 0,  
5, 1, 0,  
0, -7, 0,  
0, 0, 0,  
4, 0, 1,  
9, 3, 2.25,  
9, 10, 2.25,  
14, 10, 2.25,  
20, 15, 5
```



COONS

COONS $n, m, \text{mask},$
 $x_{11}, y_{11}, z_{11}, \dots, x_{1n}, y_{1n}, z_{1n},$
 $x_{21}, y_{21}, z_{21}, \dots, x_{2n}, y_{2n}, z_{2n},$
 $x_{31}, y_{31}, z_{31}, \dots, x_{3m}, y_{3m}, z_{3m},$
 $x_{41}, y_{41}, z_{41}, \dots, x_{4m}, y_{4m}, z_{4m}$

A Coons patch generated from four boundary curves.

Masking:

$\text{mask} = 4*j_3 + 8*j_4 + 16*j_5 + 32*j_6 + 64*j_7$

where j_3, j_4, j_5, j_6, j_7 can be 0 or 1.

j_3 (4): edges of the 1st boundary (x_1, y_1, z_1) are visible.

j_4 (8): edges of the 2nd boundary (x_2, y_2, z_2) are visible.

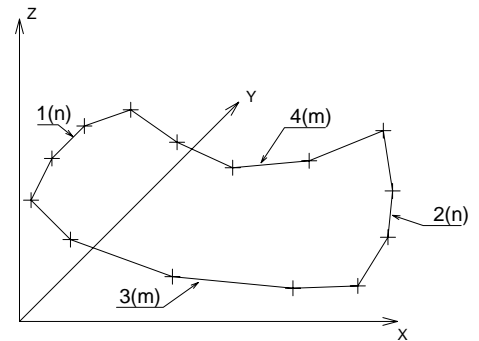
j_5 (16): edges of the 3rd boundary (x_3, y_3, z_3) are visible.

j_6 (32): edges of the 4th boundary (x_4, y_4, z_4) are visible.

j_7 (64): edges on surface are visible, surface is not smooth.

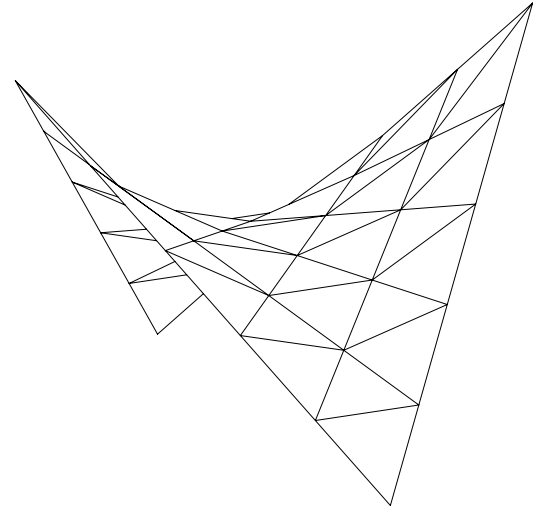
Parameter restrictions:

$n, m > 1$

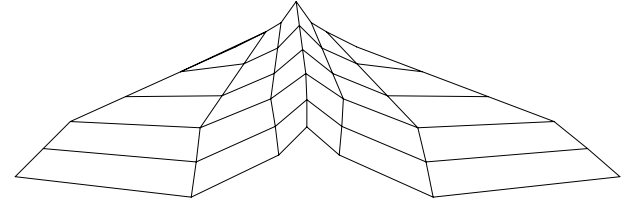


Examples:

```
COONS  6, 6, 4+8+16+32+64,  
! 1st boundary, n=6  
0, 0, 5,  
1, 0, 4,  
2, 0, 3,  
3, 0, 2,  
4, 0, 1,  
5, 0, 0,  
! 2nd boundary, n=6  
0, 5, 0,  
1, 5, 1,  
2, 5, 2,  
3, 5, 3,  
4, 5, 4,  
5, 5, 5,  
! 3rd boundary, m=6  
0, 0, 5,  
0, 1, 4,  
0, 2, 3,  
0, 3, 2,  
0, 4, 1,  
0, 5, 0,  
! 4th boundary, m=6  
5, 0, 0,  
5, 1, 1,  
5, 2, 2,  
5, 3, 3,  
5, 4, 4,  
5, 5, 5
```



```
ROTZ -90
ROTY 90
COONS 7, 6, 4+8+16+32+64,
! 1st boundary, n=7
1, 2, 0,
0.5, 1, 0,
0.2, 0.5, 0,
-0.5, 0, 0,
0.2, -0.5, 0,
0.5, -1, 0,
1, -2, 0,
! 2nd boundary, n=7
6, 10, -2,
6.5, 4, -1.5,
5, 1, -1.2,
4, 0, -1,
5, -1, -1.2,
6.5, -4, -1.5,
6, -10, -2,
! 3rd boundary, m=6
1, 2, 0,
2, 4, -0.5,
3, 6, -1,
4, 8, -1.5,
5, 9, -1.8,
6, 10, -2,
! 4th boundary, m=6
1, -2, 0,
2, -4, -0.5,
3, -6, -1,
4, -8, -1.5,
5, -9, -1.8,
6, -10, -2
```



MASS

```

MASS 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
  
```

The equivalent of the shape generated by the Mesh tool in ArchiCAD.

top_material, bottom_material, side_material: name/index of the top, bottom and side materials

n: the number of nodes in the mass polygon.

m: the number of nodes on the ridges.

h: the height of the skirt (can be negative).

xi, yi, zi: the coordinates of the nodes.

si: similar to the PRISM_ statement. Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *"Additional Status Codes"* on page 139 for details.

Masking:

$$\text{mask} = j1 + 4*j3 + 16*j5 + 32*j6 + 64*j7$$

where $j1, j3, j5, j6, j7$ can be 0 or 1.

$j1$ (1): base surface is present.

$j3$ (4): side surfaces are present.

$j5$ (16): base and side edges are visible.

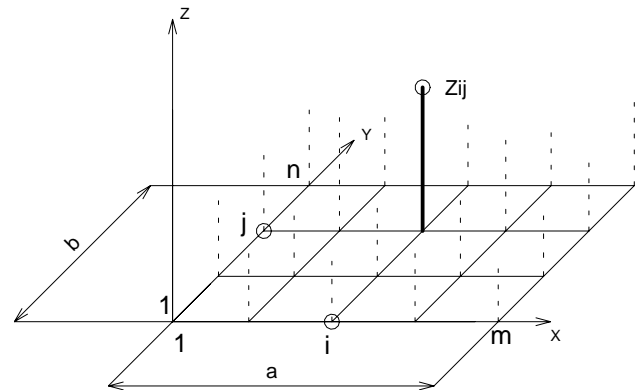
$j6$ (32): top edges are visible.

$j7$ (64): top edges are visible, top surface is not smooth.

$j8$ (128): all ridges will be sharp, but the surface is smooth.

Parameter restrictions:

$n \geq 3, m \geq 0$



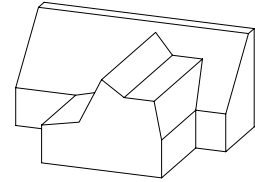
Example:

```

MASS  "Whitewash", "Whitewash", "Whitewash",
      15, 12, 117, -5.0,
      0, 12, 0, 15,
      8, 12, 0, 15,
      8, 0, 0, 15,
      13, 0, 0, 13,
      16, 0, 0, 13,
      19, 0, 0, 13,
      23, 0, 0, 13,
      24, 0, 0, 15,
      24, 12, 0, 15,
      28, 12, 0, 15,
      28, 20, 8, 13,
      28, 22, 8, 15,
      0, 22, 8, 15,
      0, 20, 8, 13,
      0, 12, 0, -1,

      0, 22, 8, 0,
      28, 22, 8, -1,
      23, 17, 5, 0,
      23, 0, 5, -1,
      13, 13, 1, 0,
      13, 0, 1, -1,
      16, 0, 7, 0,
      16, 19, 7, -1,
      0, 20, 8, 0,
      28, 20, 8, -1,
      19, 17, 5, 0,
      19, 0, 5, -1

```



ELEMENTS FOR VISUALIZATION

LIGHT

```
LIGHT red, green, blue, shadow,
      radius, alpha, beta, angle_falloff,
      distance1, distance2,
      distance_falloff [ ADDITIONAL_DATA name1 = value1,
                        name2 = value2, ...]
```

A light source radiates [red, green, blue] colored light from the local origin along the local x axis. The light is projected parallel to the x axis from a point or circle source. It has its maximum intensity within the alpha-angle frustum of a cone and falls to zero at the beta-angle frustum of a cone. This falloff is controlled by the angle_falloff parameter. (Zero gives the light a sharp edge, higher values mean that the transition is smoother.) The effect of the light is limited along the axis by the distance1 and distance2 clipping values. The distance_falloff parameter controls the decrease in intensity depending on the distance. (Zero value means a constant intensity; bigger values are used for stronger falloff.) GDL transformations affect only the starting point and the direction of the light.

The shadow parameter controls the light's shadow casting.

0: light casts no shadows

1: light casts shadows

Parameter restriction:

$\alpha \leq \beta \leq 80^\circ$

The following parameter combinations have special meanings:

radius = 0, alpha = 0, beta = 0

A point light, it radiates light in every direction and does not cast any shadows. The shadow and angle_falloff parameters are ignored, the values shadow = 0, angle_falloff = 0 are supposed.

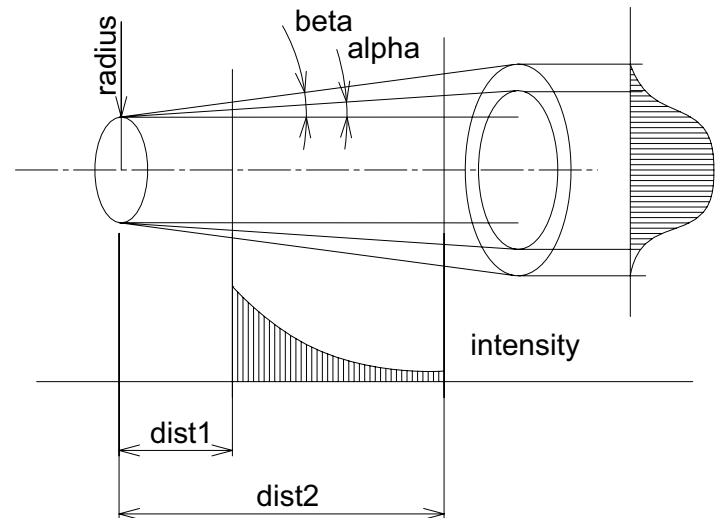
radius > 0, alpha = 0, beta = 0

A directional light.

Light definition can contain optional additional data definitions after the ADDITIONAL_DATA keyword. Additional data has a name (name_i) and a value (value_i), which can be an expression of any type, even an array. If a string parameter name ends with the substring "_file", its value is considered to be a file name and will be included in the archive project.

Different meanings of additional data can be defined and used by ArchiCAD or Add-Ons to ArchiCAD.

See meanings of LightWorks Add-On parameters at <http://www.graphisoft.com/support/developer/documentation/wide/LibraryDevKit/>.



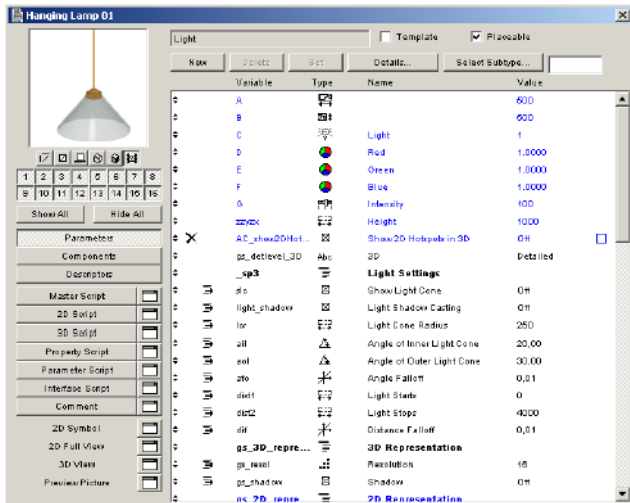
Example:

```

LIGHT 1.0,0.2,0.3,      ! RGB
      1,                ! shadow on
      1.0,              ! radius
      45.0,60.0,        ! angle1, angle2
      0.3,              ! angle_falloff
      1.0,10.0,         ! distance1, distance2
      0.2               ! distance_falloff

```

The library part dialog box for lights in ArchiCAD and ArchiFM:

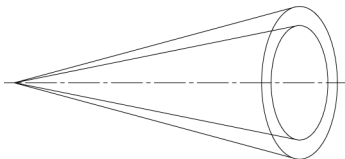


Part of the corresponding GDL script:

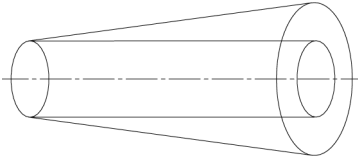
```

IF C = 0 GOTO 10
LIGHT G/100*D, G/100*E, G/100*F, ! RGB
...
10:

```



$r = 0$, $\alpha > 0$, $\beta > 0$



$r > 0$, $\alpha = 0$, $\beta > 0$



$r > 0$, $\alpha = 0$, $\beta = 0$

Light types using different alpha, beta parameters

PICTURE

PICTURE expression, a, b, mask

A picture element for photorendering.

A string type expression means a file name, a numeric expression or the index of a picture stored in the library part. A 0 index is a special value that refers to the preview picture of the library part. Other pictures can only be stored in library parts when saving the project or selected elements containing pictures as GDL Objects.

Indexed picture reference cannot be used in the MASTER_GDL script when attributes are merged into the current attribute set. The image is fitted on a rectangle treated as a RECT in any other 3D projection method.

mask = alpha + distortion

alpha: alpha channel control

0: do not use alpha channel; picture is a rectangle.

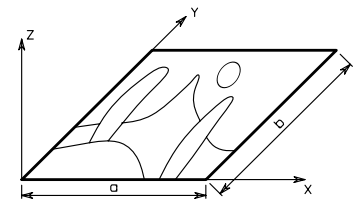
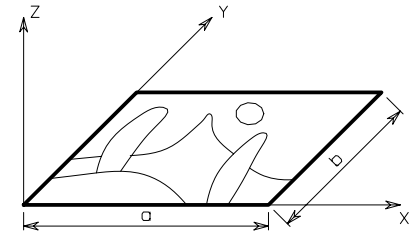
1: use alpha channel; parts of the picture may be transparent.

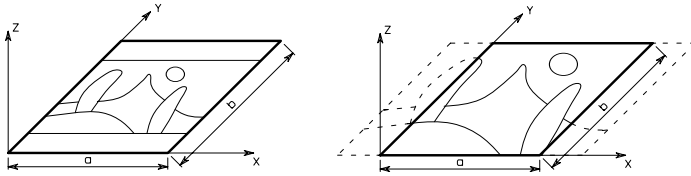
distortion: distortion control

0: fit the picture into the given rectangle.

2: fit the picture in the middle of the rectangle using the natural aspect ratio of the picture.

4: fill the rectangle with the picture in a central position using natural aspect ratio of the picture.





3D TEXT ELEMENTS

TEXT

TEXT *d*, 0, *expression*

A 3D representation of the value of a string or numeric type expression in the current style.

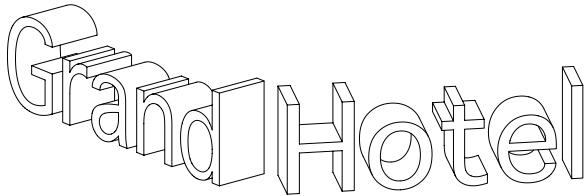
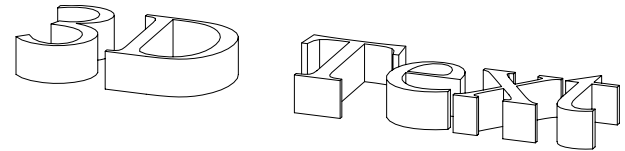
See “[*SET*] *STYLE* ” on page 153 and “[*DEFINE*] *STYLE* ” on page 171.

d: thickness of the characters in meters

In the current version of GDL, the second parameter is always zero.

Examples:

```
DEFINE STYLE "aa" "New York", 3, 7, 0
SET STYLE "aa"
TEXT 0.005, 0, "3D Text"
```



```
name = "Grand"
ROTX 90
ROTY -30
TEXT 0.003, 0, name
ADDX STW (name)/1000
ROTY 60
TEXT 0.003, 0, "Hotel"
```

Note: For compatibility with the 2D GDL script, character heights are always interpreted in millimeters in DEFINE STYLE statements.

RICHTEXT

RICHTEXT *x*, *y*,
height, 0, *textblock_name*

A 3D representation of a previously defined TEXTBLOCK. For more details, see *“Textblock” on page 174*.

x, *y*: X-Y coordinates of the richtext location

height: thickness of the characters in meters

textblock_name: the name of a previously defined TEXTBLOCK

In the current version of GDL, the 4th parameter is always zero.

PRIMITIVE ELEMENTS

The primitives of the 3D data structure are VERT, VECT, EDGE, PGON and BODY. The bodies are represented by their surfaces and the connections between them. The information to execute a 3D cutaway comes from the connection information.

Indexing starts with 1, and a BASE statement or any new body (implicit BASE statement) resets indices to 1. For each edge, the indices of the adjacent polygons (maximum 2) are stored. Edges' orientations are defined by the two vertices determined first and second.

Polygons are lists of edges with an orientation including the indices of the edges. These numbers can have a negative prefix. This means that the given edge is used in the opposite direction. Polygons can include holes. In the list of edges, a zero index indicates a new hole. Holes must not include other holes. One edge may belong to 0 to 2 polygons. In the case of closed bodies, the polygon's orientation is correct if the edge has different prefixes in the edge list of the two polygons.

The normal vectors of the polygons are stored separately. In the case of closed bodies, they point from the inside to the outside of the body. The orientation of the edge list is counterclockwise (mathematical positive), if you are looking at it from the outside. The orientation of the holes is opposite to that of the parent polygon. Normal vectors of an open body must point to the same side of the body.

To determine the inside and outside of bodies they must be closed. A simple definition for a closed body is the following: each edge has exactly two adjacent polygons.

The efficiency of the cutting, hidden line removal or rendering algorithms is lower for open bodies. Each compound three-dimensional element with regular parameters is a closed body in the internal 3D data structure.

Contour line searching is based on the status bits of edges and on their adjacent polygons. This is automatically set for compound curved elements but it is up to you to specify these bits correctly in the case of primitive elements.

In the case of a simplified definition (*vect* = 0 or *status* < 0 in a PGON) the primitives that are referred to by others must precede their reference. In this case, the recommended order is:

```
VERT (TEVE)
EDGE
(VECT)
PGON (PIPG)
COOR
BODY
```

Searching for adjacent polygons by the edges is done during the execution of the body statement.

The numbering of VERTs, EDGEs, VECTs and PGONs is relative to the last (explicit or implicit) BASE statement.

Status values are used to store special information about primitives. Each single bit usually has an independent meaning in the status, but there are some exceptions.

Given values can be added together. Other bit combinations than the ones given below are strictly reserved for internal use. The default for each status is zero.

VERT

VERT x, y, z

A node in the x-y-z space, defined by three coordinates.

TEVE

TEVE x, y, z, u, v

Extension of the VERT statement including a texture coordinate definition. Can be used instead of the VERT statement if user-defined texture coordinates are required instead of the automatic texture wrapping (see *“COOR” on page 97* statement).

x, y, z: coordinates of a node

u, v: texture coordinates of the node

(u, v) coordinates for each vertex of the current body must be specified and each vertex should have only one texture coordinate. If VERT and TEVE statements are mixed inside a body definition, (u, v) coordinates are ineffective.

Note: The (u, v) texture coordinates are only effective in photorenderings, and not for vectorial fill mapping.

VECT

VECT x, y, z

Definition of the normal vector of a polygon by three coordinates. In case of a simplified definition (vect=0 in a PGON), these statements can be omitted.

EDGE

EDGE vert1, vert2, pgon1, pgon2, status

Definition of an edge.

vert1, vert2: index of the endpoints. The vert1 and vert2 indices must be different and referenced to previously defined VERTs.

pgon1, pgon2: indices of the neighboring polygons. Zero and negative values have special meanings:

0: sidemost or standalone edge.

< 0: possible neighbors will be searched for.

Status bits:

1: invisible edge.

2: edge of a curved surface.

Reserved status bits for future use:

4: first edge of a curved surface (only together with 2).

8: last edge of a curved surface (only together with 2).

16: the edge is an arc segment.

32: first segment of an arc (only together with 16).

64: last segment of an arc (only together with 16).

PGON

PGON n, vect, status, edge1, edge2, ... edgen

Polygon definition.

n: number of edges in the edge list.

vect: index of the normal vector. It must refer to a previously defined VECT.

Note: If vect = 0, the program will calculate the normal vector during the analysis.

The edge1, edge2, ... edgen indices must refer to previously defined EDGES. A zero value means the beginning or the end of a hole definition.

A negative index changes the direction of the stored normal vector or edge to the opposite in the polygon. (The stored vector or edge does not change; other polygons can refer to it using the original orientation with a positive index.)

Status bits:

1: invisible polygon.

2: polygon of a curved surface.

16: concave polygon.

32: polygon with hole(s).

64: hole(s) are convex (only together with 32).

Reserved status bits for future use:

4: first polygon of a curved surface (only together with 2).

8: last polygon of a curved surface (only together with 2).

If the status value is negative, the engine will calculate the status of the polygon (like concave polygon or polygon with hole).

$n = 0$ is allowed for special purposes.

PIPG

PIPG *expression*, *a*, *b*, *mask*, *n*, *vect*,
status,
edge1, *edge2*, ... *edgen*

Picture polygon definition. The first four parameters are the same as in the PICTURE element; the remaining ones are the same as in the PGON element.

COOR

COOR *wrap*, *vert1*, *vert2*, *vert3*, *vert4*

Local coordinate system of a BODY for the fill and texture mapping.

wrap: wrapping mode + projection type

Wrapping modes:

1: planar

2: box

3: cylindrical

4: spherical

5: same as the cylindrical fill mapping, but in rendering the top and the bottom surface will get a circular mapping

Projection types:

256: the fill always starts at the origin of the local coordinate system.

1024: quadratic texture projection (recommended).

2048: linear texture projection based on the average distance.

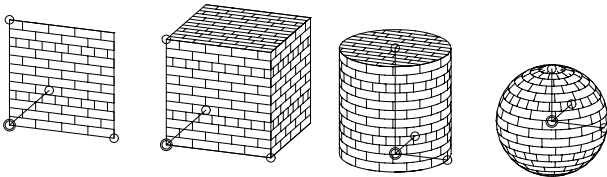
4096: linear texture projection based on normal triangulation.

Note: The last three values are only effective with custom texture coordinate definitions (see *"TEVE" on page 95*).

vert1: index of a VERT, representing the origin of the local coordinate system.

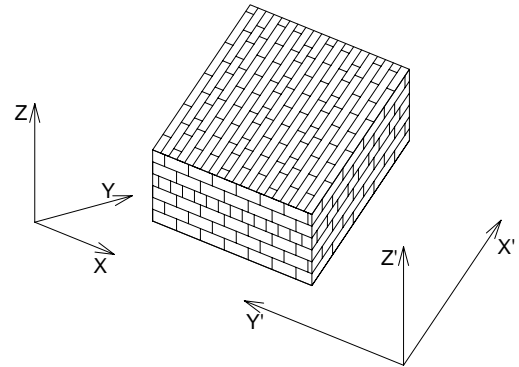
vert2, vert3, vert4: indices of VERTs defining the three coordinate axes

Use a minus sign (-) before VERT indices if they are used only for defining the local coordinate system.

*Example for custom texture axes:*

```
CSLAB_ "Face brick", "Face brick", "Face brick",
      4, 0.5,
      0, 0, 0, 15,
      1, 0, 0, 15,
      1, 1, 1, 15,
      0, 1, 1, 15
```

```
BASE
VERT 1, 0, 0      !#1
VERT 1, 1, 1     !#2
VERT 0, 0, 0     !#3
VERT 1, 0, 1     !#4
COOR 2, -1, -2, -3, -4
BODY 1
```



BODY

BODY status

Composes a body defined with the above primitives.

Status bits:

- 1: closed body.
- 2: body including curved surface(s).
- 4: surface model: when the body is cut, no surface originates on the cutting plane.
- 32: body always casts shadow independently from automatic preselection algorithm.
- 64: body never casts shadow.

If neither 32 nor 64 are set, the automatic shadow preselection is performed.

See *"SHADOW"* on page 156.

If the status value is negative, the engine will calculate the status of the body.

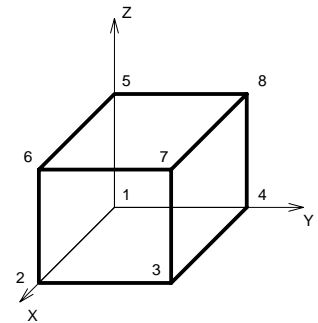
Example:

1: Complete description

```

VERT 0.0, 0.0, 0.0      !#1
VERT 1.0, 0.0, 0.0      !#2
VERT 1.0, 1.0, 0.0      !#3
VERT 0.0, 1.0, 0.0      !#4
VERT 0.0, 0.0, 1.0      !#5
VERT 1.0, 0.0, 1.0      !#6
VERT 1.0, 1.0, 1.0      !#7
VERT 0.0, 1.0, 1.0      !#8
EDGE 1, 2, 1, 3, 0      !#1
EDGE 2, 3, 1, 4, 0      !#2
EDGE 3, 4, 1, 5, 0      !#3
EDGE 4, 1, 1, 6, 0      !#4
EDGE 5, 6, 2, 3, 0      !#5
EDGE 6, 7, 2, 4, 0      !#6
EDGE 7, 8, 2, 5, 0      !#7
EDGE 8, 5, 2, 6, 0      !#8
EDGE 1, 5, 6, 3, 0      !#9
EDGE 2, 6, 3, 4, 0      !#10
EDGE 3, 7, 4, 5, 0      !#11
EDGE 4, 8, 5, 6, 0      !#12
VECT 1.0, 0.0, 0.0      !#1
VECT 0.0, 1.0, 0.0      !#2
VECT 0.0, 0.0, 1.0      !#3
PCGN 4, -3, 0, -1, -4, -3, -2 !#1 !VERT1, 2, 3, 4

```



GDL Reference Guide

```

PGON 4, 3, 0, 5, 6, 7, 8      !#2      !VERT5,6,7,8
PGON 4, -2, 0, 1, 10, -5, -9  !#3      !VERT1,2,5,6
PGON 4, 1, 0, 2, 11, -6, -10 !#4      !VERT2,3,6,7
PGON 4, 2, 0, 3, 12, -7, -11 !#5      !VERT3,4,7,8
PGON 4, -1, 0, 4, 9, -8, -12 !#6      !VERT1,4,5,8
BODY 1                          !CUBE

```

2: (no direct reference to the polygons or the vectors, they will be calculated)

```

VERT 0.0, 0.0, 0.0          !#1
VERT 1.0, 0.0, 0.0          !#2
VERT 1.0, 1.0, 0.0          !#3
VERT 0.0, 1.0, 0.0          !#4
VERT 0.0, 0.0, 1.0          !#5
VERT 1.0, 0.0, 1.0          !#6
VERT 1.0, 1.0, 1.0          !#7
VERT 0.0, 1.0, 1.0          !#8
EDGE 1, 2, -1, -1, 0         !#1
EDGE 2, 3, -1, -1, 0         !#2
EDGE 3, 4, -1, -1, 0         !#3
EDGE 4, 1, -1, -1, 0         !#4
EDGE 5, 6, -1, -1, 0         !#5
EDGE 6, 7, -1, -1, 0         !#6
EDGE 7, 8, -1, -1, 0         !#7
EDGE 8, 5, -1, -1, 0         !#8
EDGE 1, 5, -1, -1, 0         !#9
EDGE 2, 6, -1, -1, 0         !#10
EDGE 3, 7, -1, -1, 0         !#11
EDGE 4, 8, -1, -1, 0         !#12
PGON 4, 0, -1, -1, -4, -3, -2 !#1
                                !VERT1,2,3,4
PGON 4, 0, -1, 5, 6, 7, 8    !#2
                                !VERT5,6,7,8
PGON 4, 0, -1, 1, 10, -5, -9 !#3
                                !VERT1,2,5,6
PGON 4, 0, -1, 2, 11, -6, -10 !#4
                                !VERT2,3,6,7
PGON 4, 0, -1, 3, 12, -7, -11 !#5
                                !VERT3,4,7,8
PGON 4, 0, -1, 4, 9, -8, -12 !#6
                                !VERT1,4,5,8
BODY -1                          !CUBE

```

BASE

BASE

Resets counters for low-level geometric elements (VERT, TEVE, VECT, EDGE, PGON and PIPG) statements. Implicitly issued after every compound element definition.

CUTTING IN 3D

CUTPLANE

```
CUTPLANE [x, y, z [, side]]
    [statement1
    ...
    statementn]
CUTEND
```

or

```
CUTPLANE angle
    [statement1
    ...
    statementn]
CUTEND
```

Creates a cutting plane and removes the cut parts of enclosed shapes. CUTPLANE may have a different number of parameters.

If CUTPLANE has the following number of parameters:

0: x-y plane;

1: cutting plane goes across x axis, angle is between cutting plane and x-y plane;

2: cutting plane is parallel to z axis, crosses x axis and y axis at the given values;

3: cutting plane crosses the x, y and z axes at the given values;

4: the first three parameters are as above, with the addition of the side value as follows:

side = 0: removes parts above cutting plane (default);

side = 1: removes parts below cutting plane; in case of x-y, x-z, y-z, removes the parts in the negative direction of the axis.

The cut (without the side parameter) removes parts above the cutting plane. If the first three parameters define the x-y, x-z or y-z plane (for example, 1.0, 1.0, 0.0 defines the x-y plane), the parts in the positive direction of the third axis are removed.

Any number of statements can be added between CUTPLANE and CUTEND. It is also possible to include CUTPLANES in macros.

CUTPLANE parameters refer to the current coordinate system.

Transformations between CUTPLANE and CUTEND have no effect on this very cutting plane, but any successive CUTPLANEs will be transformed. Therefore, it is recommended to use as many transformations to set up the CUTPLANE as necessary, then delete these transformations before you define the shapes to cut.

Pairs of CUTPLANE-CUTEND commands can be nested, even within loops. If the final CUTEND is missing, its corresponding CUTPLANE will be effective on all shapes until the end of the script.

CUTPLANEs in macros affect shapes in the macro only, even if CUTEND is missing.

If a macro is called between CUTPLANE and CUTEND, the shapes in the macro will be cut.

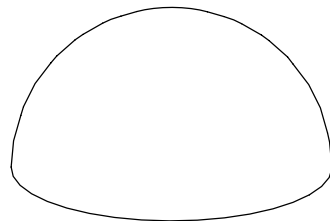
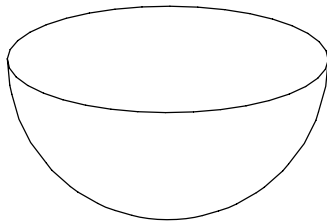
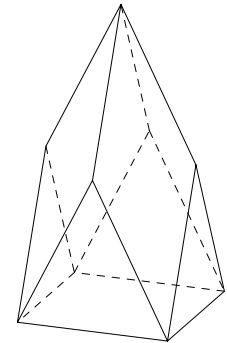
Current material, pen and fill settings are effective on the cut surfaces.

Note: If CUTPLANE is not closed with CUTEND, all shapes may be entirely removed. That's why you always get a warning message about missing CUTENDs.

If transformations used only to position the CUTPLANE are not removed, you may think that the CUTPLANE is at a wrong position when, in reality, it is the shapes that have moved away.

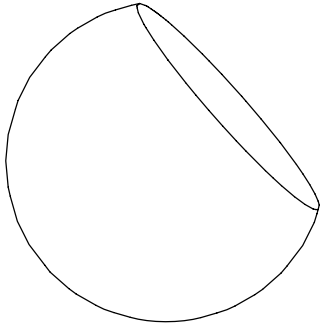
Examples:

```
CUTPLANE 2, 2, 4
CUTPLANE -2, 2, 4
CUTPLANE -2, -2, 4
CUTPLANE 2, -2, 4
  ADD -1, -1, 0
  BRICK 2, 2, 4
  DEL 1
CUTEND
CUTEND
CUTEND
CUTEND
```

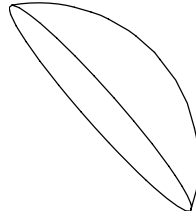


```
CUTPLANE
  SPHERE 2
CUTEND
```

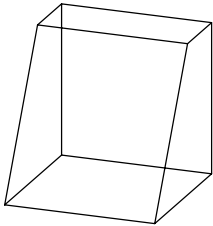
```
CUTPLANE 1, 1, 0, 1
  SPHERE 2
CUTEND
```



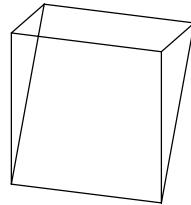
```
CUTPLANE 1.8, 1.8,
          1.8
  SPHERE 2
CUTEND
```



```
CUTPLANE 1.8, 1.8,
          1.8, 1
  SPHERE 2
CUTEND
```



```
CUTPLANE 60
  BRICK 2, 2, 2
CUTEND
```



```
CUTPLANE -120
  BRICK 2, 2, 2
CUTEND
```

CUTPOLY

```
CUTPOLY n,
  x1, y1, ... xn, yn
  [, x, y, z]
  [statement1
  statement2
  ...
  statementn]
CUTEND
```

Similarly to the CUTPLANE command, parameters of CUTPOLY refer to the current coordinate system. The polygon cannot be self-intersecting. The direction of cutting is the Z axis or an optional (x, y, z) vector can be specified.

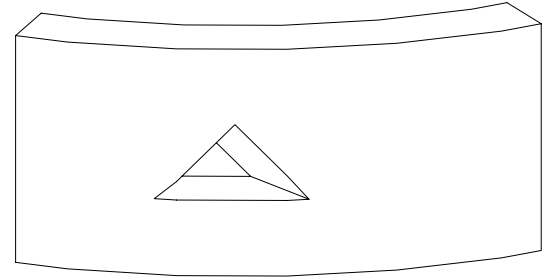
The parameters define an infinite “tube”: the polygon is the cross-section of the tube, the direction of cutting is the direction of the tube. Everything inside the tube is removed.

Examples:

```

ROTX 90
MULZ -1
CUTPOLY 3,
        0.5, 1,
        2, 2,
        3.5, 1,
        -1.8, 0, 1
      DEL 1
      BPRISM - "Red brick", "Red brick", "Face brick",
              4, 0.9, 7,
              0.0, 0.0, 15,
              6.0, 0.0, 15,
              6.0, 3.0, 15,
              0.0, 3.0, 15
CUTEND

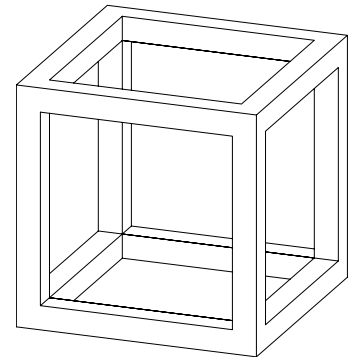
```



```

A=1.0
D=0.1
CUTPOLY 4,
        D, D,
        A-D, D,
        A-D, A-D,
        D, A-D
ROTX 90
CUTPOLY 4,
        D, D,
        A-D, D,
        A-D, A-D,
        D, A-D
      DEL 1
      ROTY 90
      CUTPOLY 4,
              D, D,
              A-D, D,
              A-D, A-D,
              D, A-D
      DEL 1
      BLOCK A, A, A
CUTEND

```

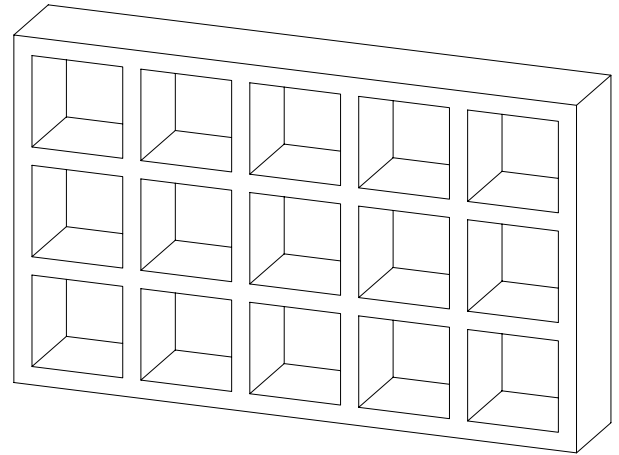


```

CUTEND
CUTEND

ROTX 90
FOR I=1 TO 3
  FOR J=1 TO 5
    CUTPOLY 4,
      0, 0, 1, 0,
      1, 1, 0, 1
    ADDX 1.2
  NEXT J
  DEL 5
  ADDY 1.2
NEXT I
DEL NTR()-1
ADD -0.2, -0.2, 0
BRICK 6.2, 3.8, 1
FOR K=1 TO 15
  CUTEND
NEXT K
DEL TOP

```



CUTPOLYA

```

CUTPOLYA n, status, d,
  x1, y1, mask1, ... xn, yn, maskn [,
  x, y, z]
  [statement1
  statement2
  ...
  statementn]

```

CUTEND

Similar to the CUTPOLY definition, but with the possibility to control the visibility of the edges of the generated polygons. The cutting form is a half-infinite tube with the defined polygonal cross-section. If the end of the cutting form hangs down into the body, it will cut out the corresponding area.

status: controls the treatment of the generated cut polygons

1: use the attributes of the body for the generated polygons and edges.

2: generated cut polygons will be treated as normal polygons.

d: the distance between the local origin and the end of the half-infinite tube.

$d = 0$ means a cut with an infinite tube.

maski: similar to the PRISM_ statement.

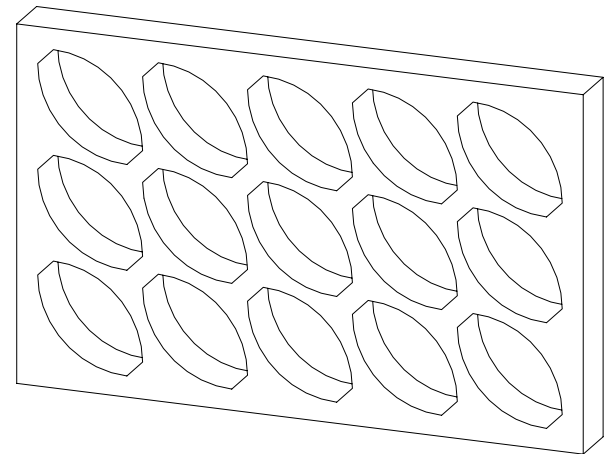
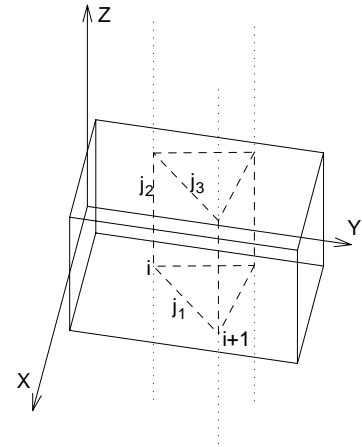
$\text{maski} = j1 + 2 * j2 + 4 * j3 + 64 * j7$

Example:

```

ROTX 90
FOR I=1 TO 3
  FOR J=1 TO 5
    CUTPOLYA 6, 1, 0,
              1, 0.15, 5,
              0.15, 0.15, 900,
              0, 90, 4007,
              0, 0.85, 5,
              0.85, 0.85, 900,
              0, 90, 4007
    ADDX 1
  NEXT J
  DEL 5
  ADDY 1
NEXT I
DEL NTR()-1
ADD -0.2, -0.2, 0
BRICK 5.4, 3.4, 0.5
FOR K=1 TO 15
  CUTEND
NEXT K
DEL TOP

```



CUTSHAPE

```
CUTSHAPE d
    [statement1
    statement2
    ...
    statementn]
```

CUTEND

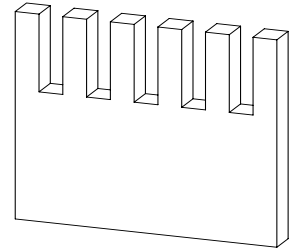
Cutting shape definition, similar to the cutting plane definition.

If $d = 0.0$ the cutting shape is the x-y plane. The cut removes the part above the x-y plane.

$d < 0.0$ means an L shape cut. The part above the x-y plane with $x \geq 0$ is removed.

$d > 0.0$ means a U shape cut. Similar to the L shape cut, the part above the x-y plane with $0 \leq x \leq d$ is removed.

```
FOR I = 1 TO 5
  ADDX 0.4 * I
  ADDZ 2.5
  CUTSHAPE 0.4
  DEL 2
  ADDX 0.4
NEXT I
DEL TOP
BRICK 4.4, 0.5, 4
FOR I = 1 TO 5
  CUTEND
NEXT I
```



CUTFORM

```
CUTFORM n, method, status,
    rx, ry, rz, d,
    x1, y1, mask1,
    ...
    xn, yn, maskn
```

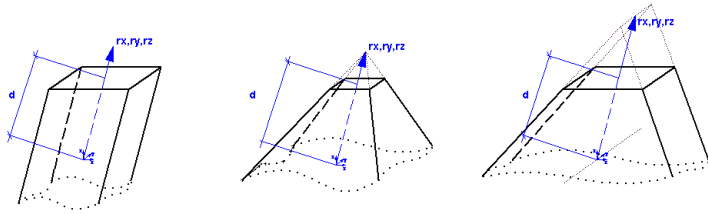
Similar to the CUTPOLYA definition, but with the possibility to control the form and extent of the cutting body.

method: Controls the form of the cutting body

1: prism shaped

2: pyramidal

3: wedge-shaped cutting body. The direction of the wedge's top edge is parallel to the Y axis and its position is in rx, ry, rz (ry is ignored.)



status: Controls the extent of the cutting body and the treatment of the generated cut polygons and new edges.

status = $j1 + 2*j2 + 8*j4 + 16*j5 + 32*j6 + 64*j7 + 128*j8$

j1: use the attributes of the body for the generated polygons and edges

j2: generated cut polygons will be treated as normal polygons

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. (can only be used with the CUTFORM command)

j7: edges generated by the bottom of the cutting body will be invisible

j8: edges generated by the top of the cutting body will be invisible

rx, ry, rz: defines the direction of cutting if the cutting form is prism-shaped, or the top of the pyramid if the method of cutting is pyramidal

d: defines the distance along rx, ry, rz to the end of the cut. If the cut is infinite, this parameter has no effect. If the cut is finite, then the start of the cutting body will be at the local coordinate system and the body will end at a distance of d along the direction defined by rx, ry, rz

If the cut is semi-infinite, then the start of the cutting body will be at a distance of d along the direction defined by rx, ry, rz, and the direction of the semi-infinite cut will be in the opposite direction defined by rx, ry, rz.

mask: Defines the visibility of the edges of the cutting body

maski = $j1 + 2*j2 + 4*j3 + 8*j4 + 16*j5 + 64*j7$

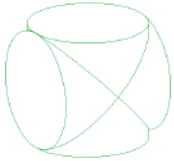
j1: the polygon will create a visible edge upon entry into the body being cut

- j2: the lengthwise edge of the cutting form will be visible
- j3: polygon will create a visible edge upon exiting the body being cut
- j4: the bottom edge of the cutting form will be visible
- j5: the top edge of the cutting form will be visible
- j7: controls the viewpoint dependent visibility of the lengthwise edge

SOLID GEOMETRY COMMANDS

GDL is capable of performing specialized 3D operations between solids represented by groups. These operations can be one of the following:

- **ADDGROUP**: forming the Boolean union of two solids;



- **SUBGROUP**: forming the Boolean difference of two solids;



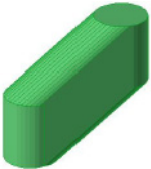
- **ISECTGROUP**: forming the Boolean intersection of two solids;



- **ISECTLINES:** calculating the intersection lines of two solids;



- **SWEEPGROUP:** sweeping a solid along a vector.



A GDL solid is composed of one or more lumps that appear as separated bodies in the model. A lump has exactly one outer shell and may contain voids. (Voids can be described as “negative” inner shells inside a lump.) The solid in the drawing below is composed of two lumps in such a way that one of them contains a void.

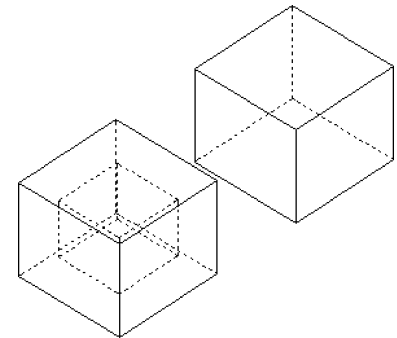
GDL bodies such as BLOCK, SPHERE, etc., appear as outer shells in groups. By means of the following construction the user is capable of putting more than one shell in a solid (note the BODY -1 statement):

```
GROUP "myGroup"
  BLOCK 1,1,1
  BODY -1
  ADDX 1
  BLOCK 1,1,1
ENDGROUP
```

The above solid contains two lumps; each of them is composed of one shell. Voids can be defined by means of primitives, or can occur as a result of a Boolean difference (e.g. subtracting a small cube from the middle of a big one).

See also *“Primitive Elements” on page 94.*

Although group operations are intended to work with solid objects, they can be applied to surfaces, wireframes or hybrid models, too. (Hybrid models are basically surfaces that may contain edges without neighboring faces.) The result of the operations on such models are summarized in the following tables:



Union (base » tool)

	solid base	surface base	wireframe base	hybrid base
solid tool	solid result	surface result (merging)	wireframe result (merging)	hybrid result (merging)
surface tool	surface result (merging)	surface result (merging)	hybrid result (merging)	hybrid result (merging)
wireframe tool	wireframe result (merging)	hybrid result (merging)	wireframe result (merging)	hybrid result (merging)
hybrid tool	hybrid result (merging)	hybrid result (merging)	hybrid result (merging)	hybrid result (merging)

Difference (base \ tool)

	solid base	surface base	wireframe base	hybrid base
solid tool	solid result	surface result	wireframe result	hybrid result
surface tool	solid base (no effect)	surface base (no effect)	wireframe base (no effect)	hybrid base (no effect)
wireframe tool	solid base (no effect)	surface base (no effect)	wireframe base (no effect)	hybrid base (no effect)
hybrid tool	solid base (no effect)	surface base (no effect)	wireframe base (no effect)	hybrid base (no effect)

Intersection (base « tool)

	solid base	surface base	wireframe base	hybrid base
solid tool	solid result	surface result	wireframe result	hybrid result
surface tool	surface result	empty result	empty result	empty result
wireframe tool	wireframe result	empty result	empty result	empty result
hybrid tool	hybrid result	empty result	empty result	empty result

Intersection lines (base « tool)

	solid base	surface base	wireframe base	hybrid base
solid tool	wireframe result	wireframe result	empty result	wireframe result
surface tool	wireframe result	empty result	empty result	empty result
wireframe tool	empty result	empty result	empty result	empty result
hybrid tool	wireframe result	empty result	empty result	empty result

Sweeping

solid	surface	wireframe	hybrid
valid result	surface base (no effect)	wireframe base (no effect)	hybrid base (no effect)

Surfaces can be explicitly generated by using the MODEL SURFACE command, or implicitly by leaving out non-neighboring face polygons from the model. Wireframes are produced either by using the MODEL WIRE statement or by defining objects without face polygons. Hybrid models can only be generated indirectly by leaving out neighboring face polygons from the model.

In the majority of the cases the required model is solid. GDL bodies appear as shells in group definitions, so in order to achieve fast and reliable operation, the geometric correctness of the generated shells is a critical issue. Handling degenerated objects loads the GDL engine and causes the desired operation to take more time to complete. The main rule to be considered regarding the efficient use of GDL group operations can be summarized as follows: *model by conforming to existing physical presence of spatial objects*. In practice this can be expressed by the following guidelines:

- Avoid self-intersecting objects.
- Avoid self-touching objects (apply small gaps).
- Avoid zero-sized portions of objects (apply small thickness).

According to the above, these rules are to be followed for shells (defined by bodies), not for solids (defined by groups). (The solid produced by the script in the Group construction above is modeled properly, since the constituent shells touch each other but the shells, themselves, are geometrically correct.)

GROUP

GROUP "name"

Beginning of a group definition. All bodies until the next ENDGROUP statement will be part of the “name” group. Groups are not actually generated (placed), they can be used in group operations or placed explicitly using the PLACEGROUP command. Group definitions cannot be nested, but macro calls containing group definitions and PLACEGROUP commands using other groups can be included.

Group names must be unique inside the current script. Transformations, cutplanes outside the group definition have no effect on the group parts; transformations, cutplanes used inside have no effect on the bodies outside the definition. Group definitions are transparent to attribute DEFINEs and SETs (pens, materials, fills); attributes defined/set before the definition and those defined/set inside the definition are all effective.

ENDGROUP

ENDGROUP

End of a group definition.

ADDGROUP

ADDGROUP (g_expr1, g_expr2)

SUBGROUP

SUBGROUP (g_expr1, g_expr2)

ISECTGROUP

ISECTGROUP (g_expr1, g_expr2)

ISECTLINES

ISECTLINES (g_expr1, g_expr2)

Group operations: addition, subtraction, intersection, intersection lines. The return value is a new group, which can be placed using the **PLACEGROUP** command, stored in a variable or used as a parameter in another group operation. Group operations can be performed between previously defined groups or groups result from any other group operation. g_expr1, g_expr2 are group type expressions. Group type expressions are either group names (string expressions) or group type variables or any combination of these in operations which result in groups.

PLACEGROUP

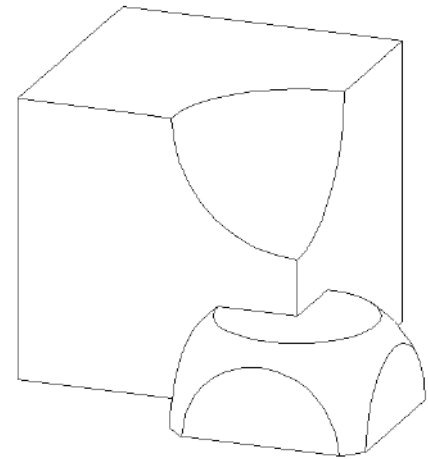
PLACEGROUP g_expr

Placing a group is the operation in which bodies are actually generated. Cutplanes and transformations are effective, the group expression is evaluated and the resulting bodies are stored in the 3D data structure.

KILLGROUP

KILLGROUP g_expr

Clears the bodies of the specified group from the memory. After a **KILLGROUP** operation the group becomes empty. Clearing is executed automatically at the end of the interpretation or when returning from macro calls. For performance reasons this command should be used when a group is no longer needed.



Example:

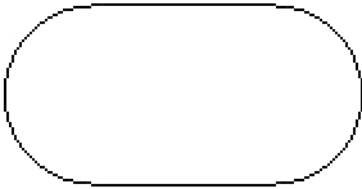
```
GROUP "box"
  BRICK 1, 1, 1
ENDGROUP
GROUP "sphere"
  ADDZ 1
  SPHERE 0.45
  DEL 1
ENDGROUP
GROUP "semisphere"
  ELLIPS 0.45, 0.45
ENDGROUP
GROUP "brick"
  ADD -0.35, -0.35, 0
  BRICK 0.70, 0.70, 0.35
  DEL 1
ENDGROUP
! Subtracting the "sphere" from the "box"
result_1=SUBGROUP("box","sphere")
! Intersecting the "semisphere" and the "brick"
result_2=ISECTGROUP("semisphere","brick")
! Adding the generated result_3=ADDGROUP(result_1,result_2)
PLACEGROUP result_3
KILLGROUP "box"
KILLGROUP "sphere"
KILLGROUP "semisphere"
KILLGROUP "brick"
```


SWEEPGROUP

SWEEPGROUP (*g_expr*, *x*, *y*, *z*)

Returns a group that is created by sweeping the group parameter along the given direction. The command works for solid models only.

Example:



```
GROUP "a"
  SPHERE 1
ENDGROUP
PLACEGROUP SWEEPGROUP ("a", 2, 0, 0)
```

BINARY 3D

BINARY

BINARY mode [, section]

Special command to include inline binary objects into a GDL macro. A set of vertices, vectors, edges, polygons, bodies and materials is read from a special section of the library part file. These are transformed according to the current transformations and merged into the 3D model. The data contained in the binary section is not editable by the user.

mode: defines pencolor and material attribute definition usage

0: the current PEN and MATERIAL settings are in effect.

1: the current PEN and MATERIAL settings have no effect. The library part will be shown with the stored colors and material definitions. Surface appearance is constant.

2: the stored PEN and MATERIAL settings are used, non-defined materials are replaced by current settings.

3: the stored PEN and MATERIAL settings are used, non-defined materials are replaced by the stored default attributes.

section: index of the binary part, from 1 to 16

By using 0 for the section index, you can refer simultaneously to all the existing binary parts.

Only sections with an index value of 1 can be saved from within GDL, BINARY commands without the section argument will also refer to this. Other section indexes can be used by third party tools.

If you open files with a different data structure (e.g, DXF or ZOOM) their 3D description will be converted into binary format.

You can save a library part in binary format from the main Library Part editing window through the *Save as...* command. If the *Save in binary format* checkbox is marked in the *Save as...* dialog box, the GDL text of the current library part will be replaced with a binary description.

Hint: Saving the 3D model after a 3D cutaway operation in binary format will save the truncated model. This way, you can create cut shapes.

You can only save your library part in binary format if you have already generated its 3D model.

By replacing the GDL description of your library part with a binary description you can considerably reduce the 3D conversion time of the item. On the other hand, the binary 3D description is not parametric and takes more disk space than an algorithmic GDL script.

2D SHAPES

This chapter presents the commands used for generating shapes in 2D from simple forms such as lines and arcs to complex polygons and splines, and the definition of text elements in 2D. It also covers the way binary data is handled in 2D and the projection of the shape created by a 3D script into the 2D view, thereby ensuring coherence between the 3D and 2D appearance of objects. Further commands allow users to place graphic elements into element lists created for calculations.

DRAWING ELEMENTS

HOTSPOT2

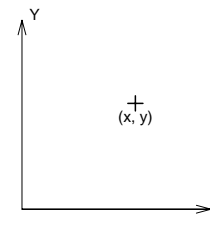
HOTSPOT2 x , y [, unID [, paramReference, flags][, displayParam]]

unID is the unique identifier of the hotspot in the 2D Script. Useful if you have a variable number of hotspots.

paramReference: parameter that can be edited by this hotspot using the graphical hotspot based parameter editing method.

displayParam: parameter to display in the information palette when editing the paramReference parameter. Members of arrays can be passed as well.

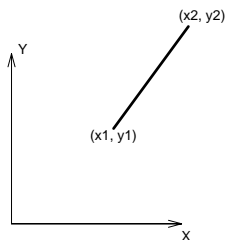
See [“Graphical Editing” on page 131](#) for information on using HOTSPOT2.



LINE2

LINE2 $x1$, $y1$, $x2$, $y2$

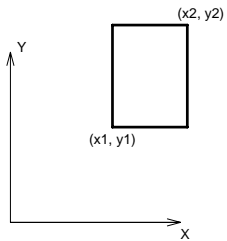
Line definition between two points.



RECT2

RECT2 $x1, y1, x2, y2$

Rectangle definition by two nodes.



POLY2

POLY2 $n, frame_fill, x1, y1, \dots, xn, yn$

An open or closed polygon with n nodes.

Restriction of parameters:

$n \geq 2$

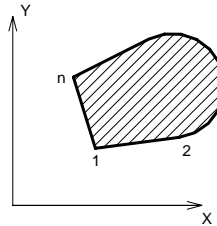
$frame_fill = j1 + 2*j2 + 4*j3$

where $j1, j2, j3$ can be 0 or 1.

$j1$ (1): contour only

$j2$ (2): fill only

$j3$ (4): close an open polygon



POLY2_

POLY2_ n, frame_fill, x1, y1, s1, ... xn, yn, sn

Similar to the normal POLY2 statement, but any of the edges can be omitted. If $s_i = 0$, the edge starting from the (x_i, y_i) apex will be omitted. If $s_i = 1$, the vertex should be shown. $s_i = -1$ is used to define holes directly. You can also define arcs and segments in the polyline using additional status code values.

Restriction of parameters:

$n \geq 2$

frame_fill = $j_1 + 2*j_2 + 4*j_3 + 8*j_4 + 32*j_6 + 64*j_7$

where j_1, j_2, j_3 can be 0 or 1.

j_1 (1): contour only

j_2 (2): fill only

j_3 (4): close an open polygon

j_4 (8): local fill orientation

j_6 : fill is cut fill (default is drafting fill)

j_7 : fill is cover fill (only if $j_6 = 0$, default is drafting fill)

Status values:

$s = j_1 + 16*j_5 + 32*j_6$

where j_1, j_5, j_6 can be 0 or 1.

j_1 (1): next segment is visible

j_5 (16): next segment is inner line (if 0, generic line)

j_6 (32): next segment is contour line (effective only if j_5 is not set)

-1: end of a contour

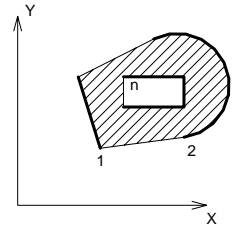
Default line property for POLY2_ lines is 0 (generic line), LINE_PROPERTY statement has no effect on POLY2_ edges.

Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *“Additional Status Codes” on page 139 for details.*

POLY2_A

POLY2_A n, frame_fill, fill_pen,
x1, y1, s1, ..., xn, yn, sn



POLY2_B

```
POLY2_B n, frame_fill, fill_pen,  
        fill_background_pen,  
        x1, y1, s1, ..., xn, yn, sn
```

Advanced versions of the POLY2_ command, with additional parameters: the fill pen and the fill background pen. All other parameters are similar to those described at the POLY2_ statement. Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *“Additional Status Codes” on page 139* for details.

POLY2_B{2}

```
POLY2_B{2} n, frame_fill, fill_pen,  
           fill_background_pen,  
           fillOrigoX, fillOrigoY,  
           fillAngle,  
           x1, y1, s1, ..., xn, yn, sn
```

Advanced version of the POLY2_B command where the fill pen, background pen, origin and direction can be defined.

```
frame_fill = j1 + 2*j2 + 4*j3 + 8*j4
```

where j_1, j_2, j_3, j_4 can be 0 or 1.

j_1 (1): contour only

j_2 (2): fill only

j_3 (4): close an open polygon

j_4 (8): local fill orientation

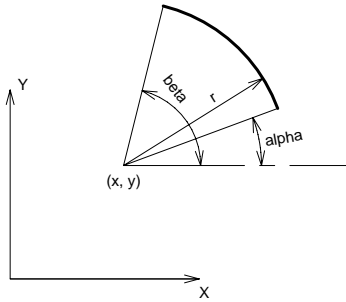
Additional status codes allow you to create segments and arcs in the planar polyline using special constraints.

See *“Additional Status Codes” on page 139* for details.

ARC2

ARC2 x , y , r , α , β

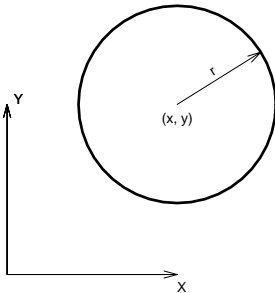
An arc with its centerpoint at (x, y) from the angle α to β , with a radius of r .
 α and β are in degrees.



CIRCLE2

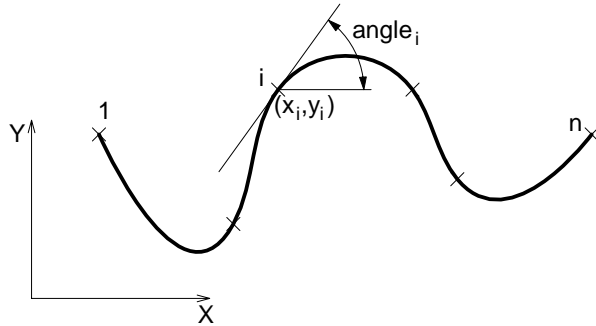
CIRCLE2 x , y , r

A circle with its center at (x, y) , with a radius of r .



SPLINE2

```
SPLINE2 n, status, x1, y1,
        angle1, ..., xn, yn, anglen
```



Restriction:

$n \geq 2$

Spline, with n control points. The tangent of the spline in the control point (x_i, y_i) is defined by $angle_i$, the angle with the x axis in degrees.

Status values:

0: default

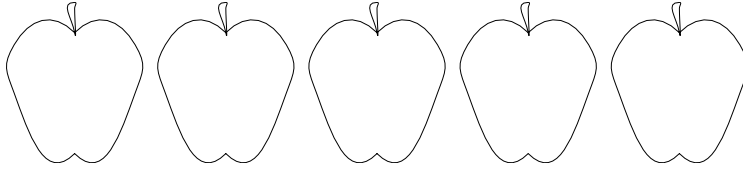
1: closed spline; the last and first nodes of the spline will become connected, thus closing the spline.

2: automatically smoothed spline; the angle parameter value of the nodes between the first and the last node is not used when generating the spline. An internal autosmoothing algorithm is used.

Examples:

```
SPLINE2 5, 2,
        0, 0, 60,
        1, 2, 30,
        1.5, 1.5, -30,
        3, 4, 45,
        4, 3, -45
```





```

n = 5
FOR I = 1 TO n
  SPLINE2 4, 0,
    0.0, 2.0, 135.0,
    -1.0, 1.8, 240.0,
    -1.0, 1.0, 290.0,
    0.0, 0.0, 45.0
  MUL2 -1.0, 1.0
  SPLINE2 4, 0,
    0.0, 2.0, 135.0,
    -1.0, 1.8, 240.0,
    -1.0, 1.0, 290.0,
    0.0, 0.0, 45.0
  DEL 1
  SPLINE2 4, 0,
    0.0, 2.0, 100.0,
    0.0, 2.5, 0.0,
    0.0, 2.4, 270.0,
    0.0, 2.0, 270.0
  ADD2 2.5, 0
NEXT I

```

SPLINE2A

```

SPLINE2A n, status, x1, y1, angle1, length_previous1, length_next1,
  ...
  xn, yn, anglen, length_previousn,
  length_nextn

```

An extension of the SPLINE2 statement (Bézier spline), used mainly in automatic 2D script generation because of its complexity.

For more details, see *"Drawing Splines" in the ArchiCAD 9 Reference Guide*.

Status codes:

0: default

1: closed spline; the last and first nodes of the spline will become connected, thus closing the spline

2: Automatically smoothed spline; the angle, length_previous_i and length_next_i parameter values of the nodes between the first and the last node are not usedⁱ when generating the spline. An internal autosmoothing algorithm is used.

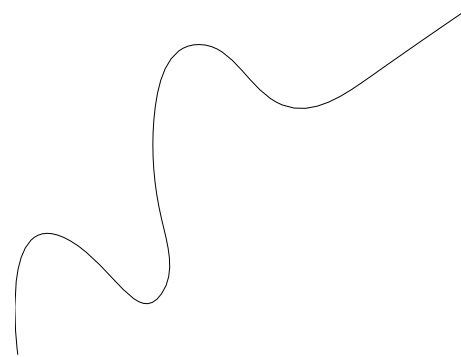
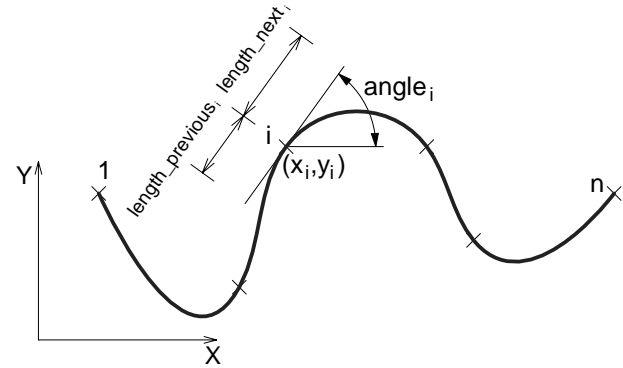
x_i, y_i: control point coordinates

length_previous_i, length_next_i: tangent lengths for the previous and the next control points

angle_i: tangent direction angle

Example:

```
SPLINE2A 9, 2,
0.0, 0.0, 0.0, 0.0, 0.0,
0.7, 1.5, 15, 0.9, 1.0,
1.9, 0.8, 72, 0.8, 0.3,
1.9, 1.8, 100, 0.3, 0.4,
1.8, 3.1, 85, 0.4, 0.5,
2.4, 4.1, 352, 0.4, 0.4,
3.5, 3.3, 338, 0.4, 0.4,
4.7, 3.7, 36, 0.4, 0.8,
6.0, 4.6, 0, 0.0, 0.0
```



PICTURE2

PICTURE2 expression, a, b, mask

PICTURE2{2}

PICTURE2{2} expression, a, b, mask

Can be used in 2D similarly to the PICTURE command in 3D. Unlike in 3D, the mask values have no effect on 2D pictures.

A string type expression means a file name, a numerical expression means an index of a picture stored in the library part. A 0 index is a special value, it refers to the preview picture of the library part. For PICTURE2{2} mask = 1 means that exact white colored pixels are transparent. Other pictures can only be stored in library parts when saving the project or selected elements containing pictures as GDL objects.

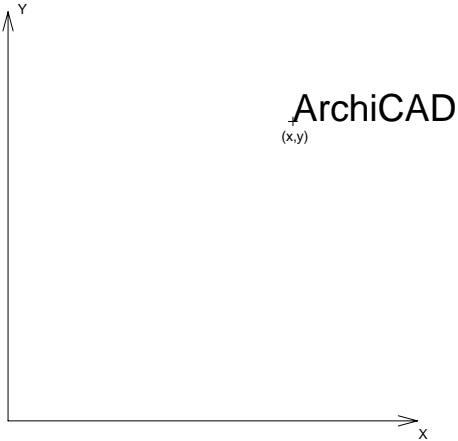
TEXT ELEMENT

TEXT2

TEXT2 x, y, expression

The value of the calculated numerical or string type expression is written in the set style at the x, y coordinates.

See also commands *“[SET] STYLE ” on page 153* and *“DEFINE STYLE ” on page 171*.



RICHTEXT2

RICHTEXT2 x, y, textblock_name

Place a previously defined TEXTBLOCK.

For more details, see *“Textblock” on page 174*.

x, y: X-Y coordinates of the richtext location

textblock_name: the name of a previously defined TEXTBLOCK

BINARY 2D

FRAGMENT2

FRAGMENT2 fragment_index,
use_current_attributes_flag

The fragment with the given index is inserted into the 2D Full View with the current transformations.

use_current_attributes_flag: 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.

FRAGMENT2

FRAGMENT2 ALL,use_current_attributes_flag

All the fragments are inserted into the 2D Full View with the current transformations.

use_current_attributes_flag: defines whether or not the current attributes will be used.

0: the fragments appears with the color, line type and fill type defined for them.

1: the current settings of the script are used instead of the color, line type and fill type of the fragments.

3D PROJECTIONS IN 2D

PROJECT2

PROJECT2 projection_code, angle, method

PROJECT2{2}

PROJECT2{2} projection_code, angle,method [,backgroundColor, fillOrigoX, fillOrigoY, filldirection]

Creates a projection of the 3D script in the same library part and adds the generated lines to the 2D parametric symbol. The 2nd version PROJECT2{2}, together with a previous SET FILL command, allows the user to control the fill background, origin and direction of the resulting drawing from the 2D script.

projection_code: the type of projection

3: Top view

4: Side view

6: Frontal axonometry

7: Isometric axonometry

8: Monometric axonometry

9: Dimetric axonometry

-3: Bottom view

-6: Frontal bottom view

-7: Isometric bottom view

-8: Monometric bottom view

-9: Dimetric bottom view

angle: the azimuth angle set in the 3D Projection Settings dialog box.

method: the chosen imaging method

1: wireframe

2: hidden lines (analytic)

3: shading

16: addition modifier, draws vectorial hatches (effective only in hidden line and shaded mode)

32: addition modifier, use current attributes instead of attributes from 3D (effective only in shading mode)

64: addition modifier, local fill orientation (effective only in shading mode)

128: addition modifier: lines are all inner lines (effective only together with 32). Default is generic.

256: addition modifier: lines are all contour lines (effective only together with 32, if 128 is not set). Default is generic.

512: addition modifier: fills are all cut (effective only together with 32). Default is drafting fills.

1024: addition modifier: fills are all cover (effective only together with 32, if 256 is not set). Default is drafting fills.

BackgroundColor: background color of the fill

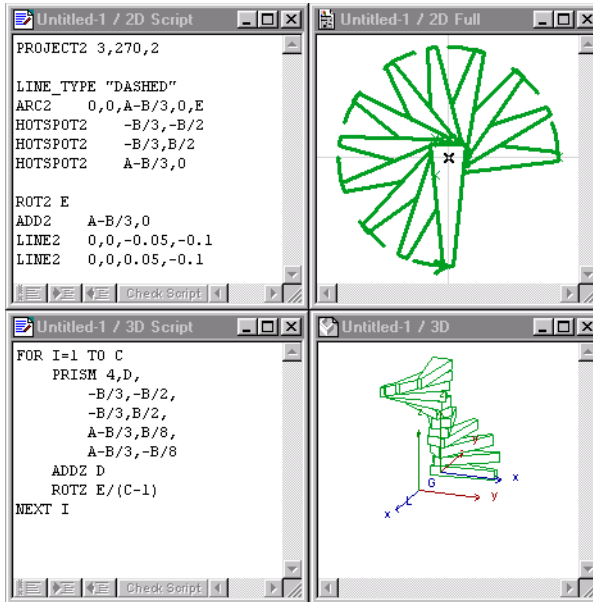
fillOrigoX: X coordinate of the fill origin

fillOrigoY: Y coordinate of the fill origin

filldirection: direction angle of fill

Note: SET FILL is effective for PROJECT2{2}

Example:



DRAWINGS IN THE LIST

These commands only take effect when a list of elements is created in ArchiCAD.

When the library part is a special property type library part and is in some way associated to a library part (Object, Door, Window or Light) placed on the floor plan, including the following commands in its 2D script will refer to the 2D and 3D part of that library part. This is a virtual reference that is resolved during the listing process, using the 2D or 3D script of the currently listed element.

DRAWING2

DRAWING2 [expression]

Depending on the value of the expression, creates a drawing of the library part (expression = 0, default) or the label of the element (expression = 1) associated with the Property Object containing this command.

DRAWING3

DRAWING3 projection_code, angle, method

DRAWING3{2}

DRAWING3{2} projection_code, angle, method [,backgroundColor, origoX, origoY, fillldirection]

Similarly to PROJECT2, creates a projection of the 3D script of the library part associated with the property library part containing this command. All parameters are similar to those of PROJECT2 and PROJECT2{2}.

New method flags in DRAWING3{2}:

3: shading

32: use current attributes instead of attributes from 3D

64: local fill orientation

GRAPHICAL EDITING

Hotspot-based interactive graphical editing of length and angle type GDL parameters.

HOTSPOT-BASED EDITING COMMANDS

HOTSPOT

HOTSPOT *x*, *y*, *z* [, *unID* [, *paramReference*, *flags*] [, *displayParam*]]

HOTSPOT2 *x*, *y* [, *unID* [, *paramReference*, *flags*][, *displayParam*]]

unID: unique identifier, which must be unique among the hotspots defined in the library part.

paramReference: parameter that can be edited by this hotspot using the graphical hotspot based parameter editing method.

displayParam: parameter to display in the information palette when editing the *paramReference* parameter. Members of arrays can be passed as well.

Examples of valid arguments:

D, *Arr*[5], *Arr*[2*I+3][D+1], etc.

flags: hotspot's type + hotspot's attribute,

type:

- 1: length type editing, base hotspot
- 2: length type editing, moving hotspot
- 3: length type editing, reference hotspot (always hidden)
- 4: angle type editing, base hotspot
- 5: angle type editing, moving hotspot
- 6: angle type editing, center of angle (always hidden)
- 7: angle type editing, reference hotspot (always hidden)

attribute can be a combination of the following values or zero:

128: hide hotspot (meaningful for types: 1,2,4,5)

256: editable base hotspot (for types: 1,4)

512: reverse the angle in 2D (for type 6)

To edit a length type parameter, three hotspots must be defined with types 1, 2 and 3. The positive direction of the editing line is given by the vector from the reference hotspot to the base hotspot. The moving hotspot must be placed along this line at a distance determined by the associated parameter's value, measured from the base hotspot.

To edit an angle type parameter, four hotspots must be defined with types 4, 5, 6 and 7. The plane of the angle is perpendicular to the vector that goes from the center hotspot to the reference hotspot. The positive direction in measuring the angle is counter-clockwise if we look at the plane from the reference hotspot. In 2D the plane is already given, so the reference hotspot is ignored, and the positive direction of measuring the

angle is by default counter-clockwise. This can be changed to clockwise by setting the 512 attribute flag for the center hotspot (type 6). To be consistent, the vectors from the center hotspot to the moving and the base hotspots must be perpendicular to the vector from the center to the reference hotspot. The moving hotspot must be placed at an angle determined by the associated parameter measured from the base hotspot around the center hotspot.

If several sets of hotspots are defined to edit the same parameter, hotspots are grouped together in the order of the execution of the hotspot commands. If the editable attribute is set for a base hotspot, the user can also edit the parameter by dragging the base hotspot. Since the base hotspot is supposed to be fixed in the object's coordinate frame (i.e. its location must be independent of the parameter that is attached to it), the whole object is dragged or rotated along with the base point. (As the parameter's value is changing, the moving hotspot will not change its location.)

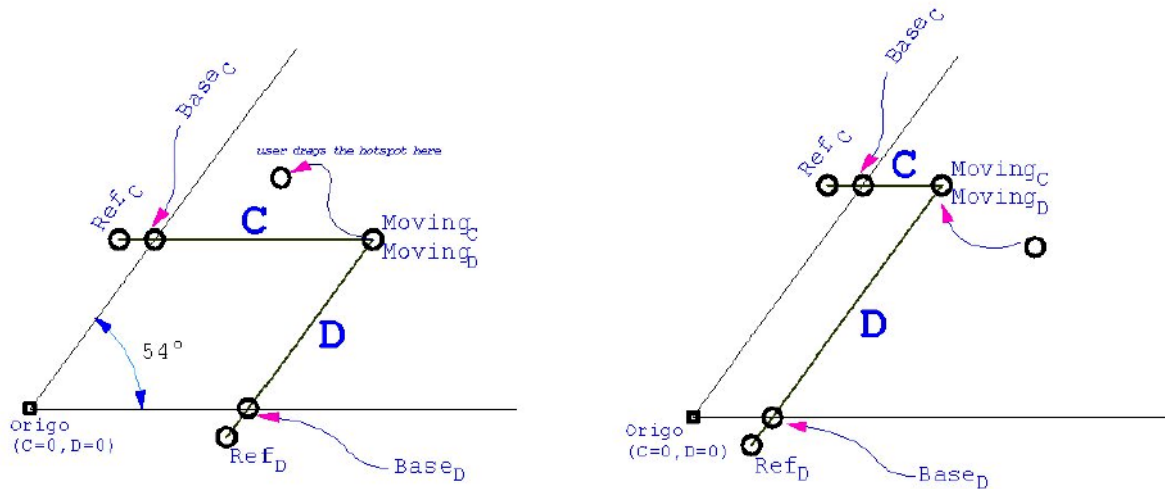
Two or three length type sets of hotspots can be combined to allow editing of two or three parameters with only one dragging. If two are combined, the motion of the hotspot is no longer constrained to a line but to the plane determined by the two lines of each set of length editing hotspots. In 3D, the combination of three sets of length editing hotspots allows the hotspot to be placed anywhere in space. The two lines must not be parallel to each other, and the three lines must not be on the same plane. A combined parameter editing operation is started if, at the location of the picked point, there are two or three editable hotspots (moving or editable base) with different associated parameters. If parameters are designed for combined editing, the base and reference hotspots are not fixed in the object's coordinate frame, but must move as the other parameter's value changes.

See illustration and example 2.

Example 1, angle editing in 2D:

```
LINE2 0, 0, A, 0
LINE2 0, 0, A*COS(angle), A*SIN(angle)
ARC2 0, 0, 0.75*A, 0, angle
HOTSPOT2 0, 0, 1, angle, 6
HOTSPOT2 0.9*A, 0, 2, angle, 4
HOTSPOT2 0.9*A*COS(angle), 0.9*A*SIN(angle), 3,
          angle, 5
```

Example 2, combined length type editing with 2 parameters:



```

RECT2 0, 0, A, B
RECT2 0, 0, sideX, sideY
HOTSPOT2 sideX, 0, 1, sideY, 1
HOTSPOT2 sideX, -0.1, 2, sideY, 3
HOTSPOT2 sideX, sideY, 3, sideY, 2
HOTSPOT2 0, sideY, 4, sideX, 1
HOTSPOT2 -0.1, sideY, 5, sideX, 3
HOTSPOT2 sideX, sideY, 6, sideX, 2

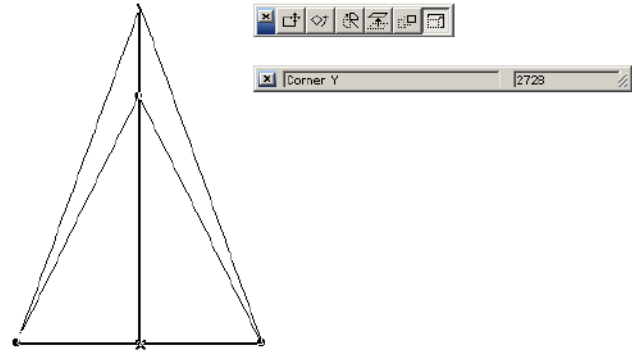
```

Example 3, simple length type editing with 1 parameter:

```

2D SCRIPT:
HOTSPOT2 -1, 0, 1
HOTSPOT2 1, 0, 2
HOTSPOT2 0, 0, 3, corner_y, 1+128
HOTSPOT2 0, -1, 4, corner_y, 3
HOTSPOT2 0, corner_y, 5, corner_y, 2
LINE2 -1, 0, 1, 0
LINE2 -1, 0, 0, corner_y
LINE2 1, 0, 0, corner_y
3D SCRIPT:
HOTSPOT -1, 0, 0, 1
HOTSPOT -1, 0, 0.5, 2
HOTSPOT 1, 0, 0, 3
HOTSPOT 1, 0, 0.5, 4
HOTSPOT 0, 0, 0, 5, corner_y, 1+128
HOTSPOT 0, -1, 0, 6, corner_y, 3
HOTSPOT 0, corner_y, 0, 7, corner_y, 2
HOTSPOT 0, 0, 0.5, 8, corner_y, 1+128
HOTSPOT 0, -1, 0.5, 9, corner_y, 3
HOTSPOT 0, corner_y, 0.5, 10, corner_y, 2
PRISM_ 4, 0.5,
      -1, 0, 15,
       1, 0, 15,
       0, corner_y, 15,
      -1, 0, -1

```



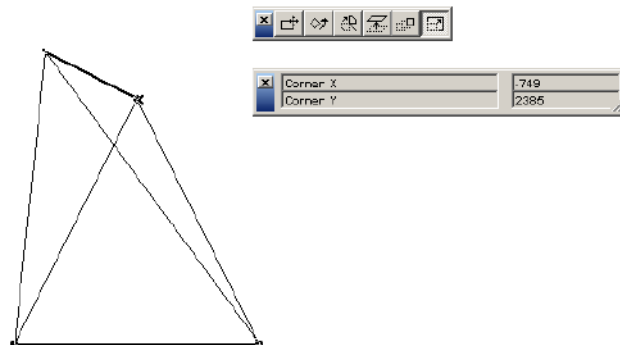
Example 4: combined length type editing with 2 parameters:

```

2D SCRIPT:
HOTSPOT2 -1, 0, 1
HOTSPOT2 1, 0, 2
HOTSPOT2 corner_x, 0, 3, corner_y, 1+128
HOTSPOT2 corner_x, -1, 4, corner_y, 3
HOTSPOT2 corner_x, corner_y, 5, corner_y, 2
HOTSPOT2 0, corner_y, 3, corner_x, 1+128
HOTSPOT2 -1, corner_y, 4, corner_x, 3
HOTSPOT2 corner_x, corner_y, 5, corner_x, 2
LINE2 -1, 0, 1, 0
LINE2 -1, 0, corner_x, corner_y
LINE2 1, 0, corner_x, corner_y

3D SCRIPT:
HOTSPOT -1, 0, 0, 1
HOTSPOT -1, 0, 0.5, 2
HOTSPOT 1, 0, 0, 3
HOTSPOT 1, 0, 0.5, 4
HOTSPOT corner_x, 0, 0, 5, corner_y, 1+128
HOTSPOT corner_x, -1, 0, 6, corner_y, 3
HOTSPOT corner_x, corner_y, 0, 7, corner_y, 2
HOTSPOT 0, corner_y, 0, 8, corner_x, 1+128
HOTSPOT -1, corner_y, 0, 9, corner_x, 3
HOTSPOT corner_x, corner_y, 0, 10, corner_x, 2
HOTSPOT corner_x, 0, 0.5, 11, corner_y, 1+128
HOTSPOT corner_x, -1, 0.5, 12, corner_y, 3
HOTSPOT corner_x, corner_y, 0.5, 13, corner_y, 2
HOTSPOT 0, corner_y, 0.5, 14, corner_x, 1+128
HOTSPOT -1, corner_y, 0.5, 15, corner_x, 3
HOTSPOT corner_x, corner_y, 0.5, 16, corner_x, 2
PRISM_ 4, 0.5,
      -1, 0, 15,
      1, 0, 15,
      corner_x, corner_y, 15,
      -1, 0, -1

```



HOTLINE2

HOTLINE2 x1, y1, x2, y2

Status line definition between two points.

HOTARC2

HOTARC2 *x, y, r, startangle, endangle*

Status arc definition with its centerpoint at (x, y) from the angle startangle to endangle, with a radius of r.

STATUS CODES

Status codes introduced in the following pages allow users to create segments and arcs in planar polylines using special constraints.

Planar polylines with status codes at nodes are the basis of many GDL elements:

POLY_, PLANE_, PRISM_, CPRISM_, BPRISM_, FPRISM_, HPRISM_, SPRISM_, SLAB_, CSLAB_, CROOF_, EXTRUDE_, PYRAMID, REVOLVE, SWEEP, TUBE, TUBEA

Status codes allow you:

- to control the visibility of planar polyline edges
- to define holes in the polyline
- to control the visibility of side edges and surfaces
- to create segments and arcs in the polyline

STATUS CODE SYNTAX

The *si* number is a binary integer (between 0 and 127) or -1.

$s = j1 + 2*j2 + 4*j3 + 8*j4 + 16*j5 + 32*j6 + 64*j7$ [+ *a_code*]

where *j1*, *j2*, *j3*, *j4*, *j5*, *j6*, *j7* can be 0 or 1.

The *j1*, *j2*, *j3*, *j4* numbers represent whether the vertices and the sides are present (1) or omitted (0):

j1: lower horizontal edge

j2: vertical edge

j3: upper horizontal edge

j4: side

j5: horizontal edge in line elimination (for PRISM_ shapes only)

j6: vertical edge in line elimination (for PRISM_ shapes only)

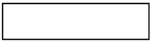
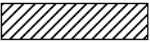


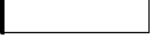

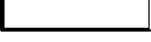


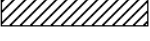
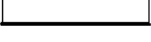


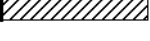


j7: special additional status value effective only when *j2*=1 and controls the viewpoint dependent visibility of the current vertical edge

j2=0: the vertical edge is always invisible

j2=1 and *j7*=1: the vertical edge is only visible when it is a contour observed from the current direction of view

j2=1 and *j7*=0: the vertical edge is always visible

Possible status values (the heavy lines denote visible edges):

<i>invisible surface</i>	<i>visible surface</i>
0 	8 
1 	9 
2 	10 
3 	11 
4 	12 
5 	13 
6 	14 
7 	15 

`a_code`: additional status code (optional) which allow you to create segments and arcs in the polyline.

`si=-1` is used to define holes directly into the prism. It marks the end of the contour and the beginning of a hole inside of the contour. It is also used to indicate the end of one hole's contour and the beginning of another. Coordinates before that value must be identical to the coordinates of the first point of the contour/hole. If you have used the -1 mask value, the last mask value in the parameter list must be -1, marking the end of the last hole.

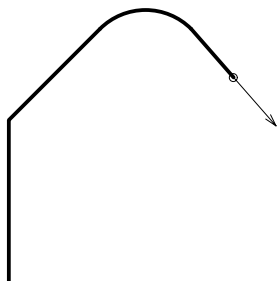
The holes must be disjunct and internal intersections are forbidden in the polygon for a correct shading/rendering result.

ADDITIONAL STATUS CODES

The following additional status codes allow you to create segments and arcs in the polyline using special constraints. They refer to the next segment or arc. Original status code(s) are only effective where they are specified (a “+s” is included after the additional code).

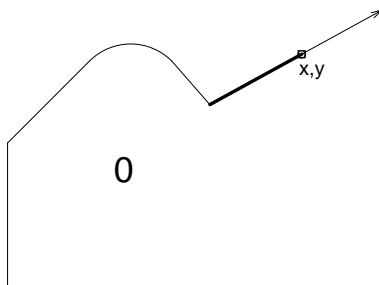
Note: Resolution of arcs is controlled by directives described in the “Attributes” chapter. In case of the POLY2_ statement, if the resolution is greater than 8, it generates real arcs; otherwise all generated arcs will be segmented.

Previous part of the polyline: current position and tangent is defined



Segment by absolute endpoint

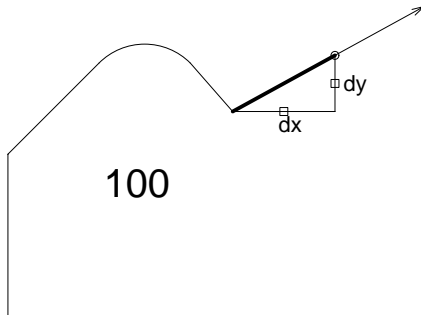
x, y, s
where $0 < s < 100$



Segment by relative endpoint

$dx, dy, 100+s,$

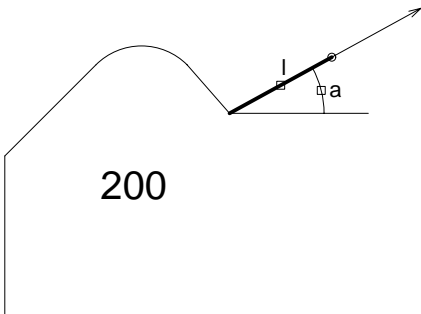
where $0 < s < 100$



Segment by length and direction

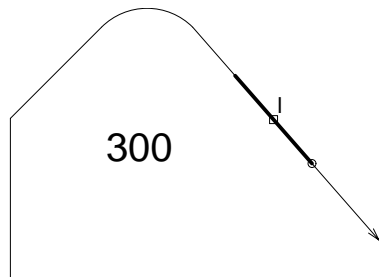
$l, a, 200+s,$

where $0 < s < 100$



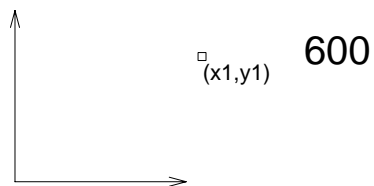
Tangential segment by length

$l, 0, 300+s,$
where $0 < s < 100$



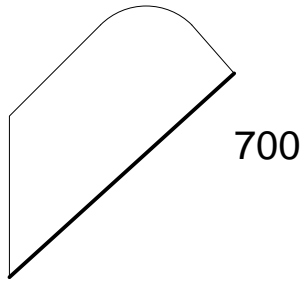
Set start point

$x1, y1, 600,$



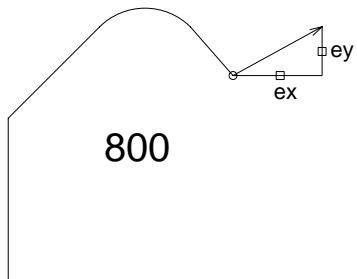
Close polyline

0, 0, 700,



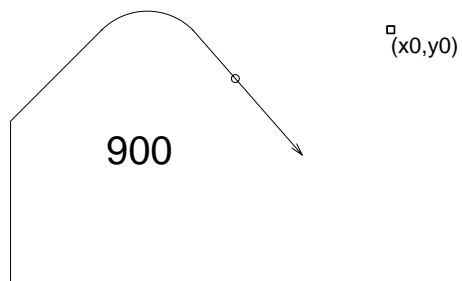
Set tangent

ex, ey, 800,



Set centerpoint

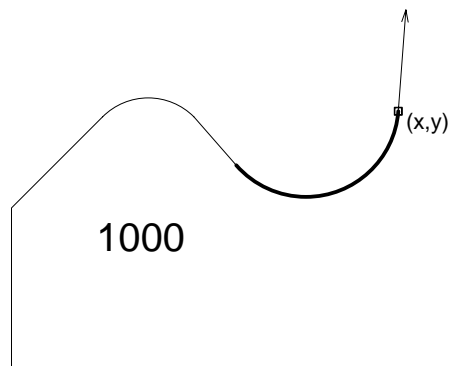
$x0, y0, 900,$



Tangential arc to endpoint

$x, y, 1000+s,$

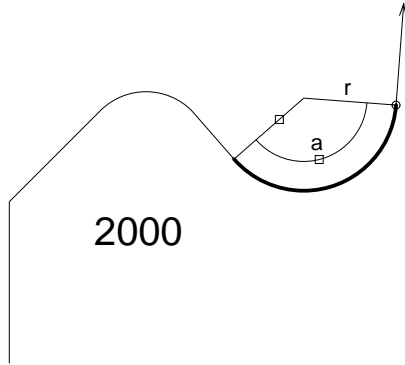
where $0 < s < 100$



Tangential arc by radius and angle

$r, a, 2000+s,$

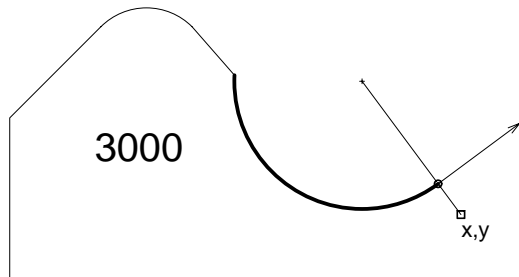
where $0 < s < 100$



Arc using centerpoint and point on the final radius

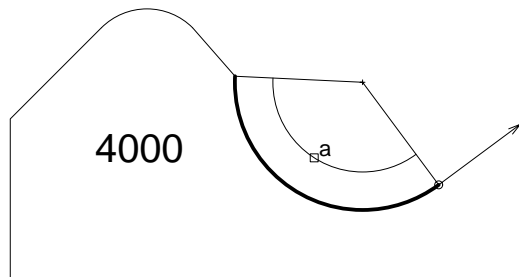
$x, y, 3000+s,$

where $0 < s < 100$



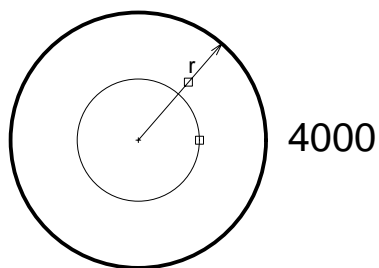
Arc using centerpoint and angle

$0, a, 4000+s,$
 where $0 < s < 100$



Full circle using centerpoint and radius

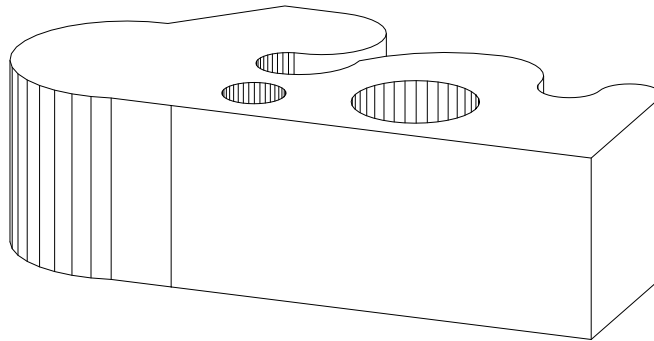
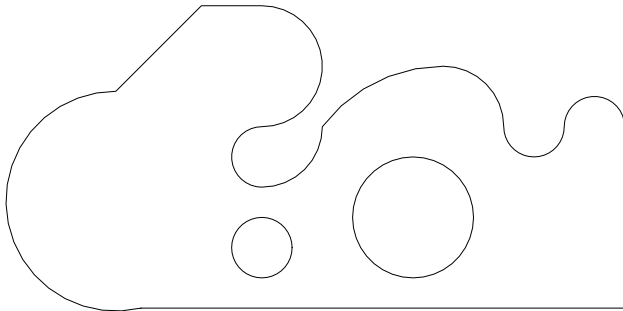
$r, 360, 4000+s,$
 where $0 < s < 100$



In this case the s status refers to the whole circle.

All angle values are in degrees. Omitted coordinates marked by 0 (for codes 300, 700, 4000) can have any value.

Examples:



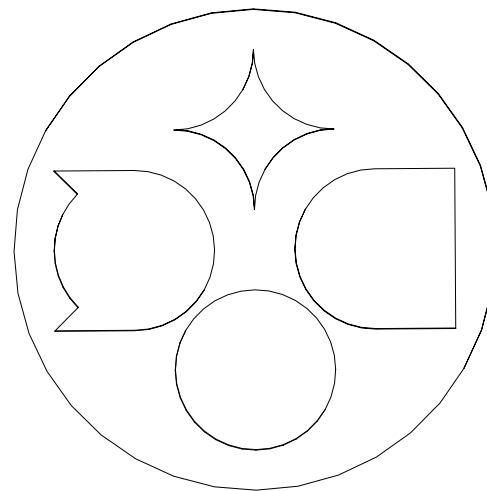
```

EXTRUDE 21, 0, 0, 3, 1+2+4+16+32,
0, 0, 0,
7, 0, 0,
7, 3, 1,
6, 3, 1000,      ! tangential arc to endpoint
5, 3, 1001,      ! tangential arc to endpoint
1, 90, 2000,     ! tangential arc by radius and angle
2, 3, 1001,      ! tangential arc to endpoint
1, 3, 900,       ! set centerpoint
1, 2, 3000,      ! arc using startpoint, centerpoint and point on final radius
1, 2.5, 900,     ! set centerpoint
0, -180, 4001,   ! arc using start point, centerpoint and angle
1, 5, 1000,      ! tangential arc to endpoint
-1, 0, 100,      ! segment by (dx, dy)
2, 225, 200,     ! segment by (len, angle)
-1, 0, 800,      ! set tangent
-1, 0, 1000,     ! tangential arc to endpoint
0, 0, -1,        ! end of contour
1, 1, 900,       ! set centerpoint
0.5, 360, 4000, ! full circle by centerpoint and radius
3.5, 1.5, 900,  ! set centerpoint
1, 360, 4001    ! full circle by centerpoint and radius

```



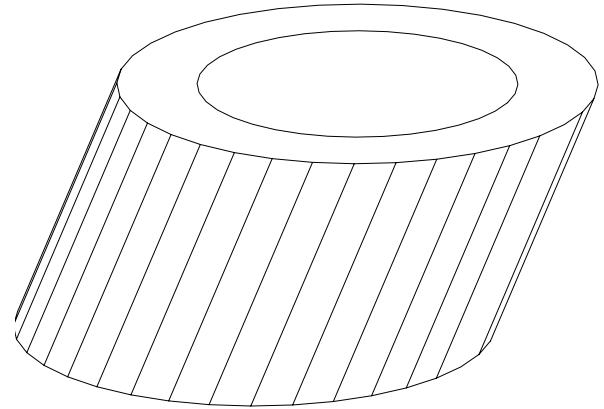
```
EXTRUDE 2+5+10+10+2, 0, 0, 3, 1+2+4+16+32,  
0, 0, 900,  
3, 360, 4001,  
2.5, -1, 0,  
2.5, 1, 0,  
1.5, 1, 1,  
1.5, -1, 1001,  
2.5, -1, -1,  
0, 2.5, 600,  
0, -1, 800,  
1, 1.5, 1001,  
-1, 0, 800,  
0, 0.5, 1001,  
0, 1, 800,  
-1, 1.5, 1001,  
1, 0, 800,  
0, 2.5, 1001,  
0, 2.5, 700,  
-1.5, 0, 900,  
-2.5, 0, 600,  
-2.5, 1, 3000,  
-2.5, 1, 0,  
-1.5, 1, 0,  
-1.5, -1, 1001,  
-2.5, -1, 0,  
SQRT(2)-1, 45, 200,  
-2.5, 0, 3000,  
-2.5, 0, 700,  
0, -1.5, 900,  
1, 360, 4000
```



```

EXTRUDE 3, 1, 1, 3, 1+2+4+16+32,
0, 0, 900,
3, 360, 4001,
2, 360, 4000

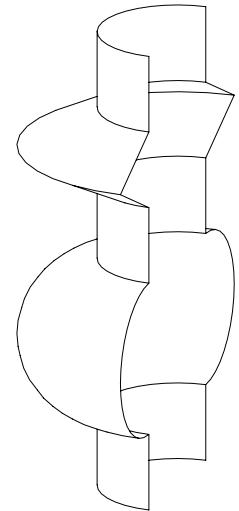
```



```

ROTY -90
REVOLVE 9, 180, 16+32,
7, 1, 0,
6, 1, 0,
5.5, 2, 0,
5, 1, 0,
4, 1, 0,
3, 1, 900, ! set centerpoint
0, 180, 4001, ! arc using startpoint, centerpoint and angle
2, 1, 0,
1, 1, 0

```



ATTRIBUTES

In the first part of this chapter, directives influencing the interpretation of GDL statements are presented. Directives may define the smoothness used for cylindrical elements, representation mode in the 3D view or the assignment of an attribute (color, material, text style, etc.) for the subsequent shapes. Inline attribute definition is covered in the second part. This feature allows you to assign to your objects customized materials, textures, fill patterns, line types and text styles that are not present in the current attribute set of your project.

DIRECTIVES

The influence of directives on the interpretation of the subsequent GDL statements remains in effect until the next directive or the end of the script. Called scripts inherit the current settings: the changes have local influence. Returning from the script resets the settings as they were before the macro call.

Directives for 3D and 2D Scripts

[LET]

[**LET**] varnam = n

Value assignment. The LET directive is optional. The variable will store the evaluated value of n.

RADIUS

RADIUS radius_min, radius_max

Sets smoothness for cylindrical elements and arcs in polylines.

A circle with a radius of r is represented:

- if $r < \text{radius_min}$, by a hexagon,
- if $r \geq \text{radius_max}$, by a 36-edged polygon,
- if $\text{radius_min} < r < \text{radius_max}$, by a polygon of $(6+30*(r-\text{radius_min})/(\text{radius_max}-\text{radius_min}))$ edges.

Arc conversion is proportional to this.

After a RADIUS statement, $\alpha\psi\psi$ previous RESOL and TOLER statements lose their effect.

Parameter restriction:

$r_{\text{min}} \leq r_{\text{max}}$

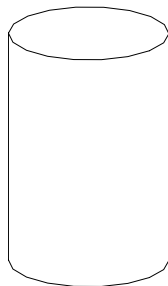
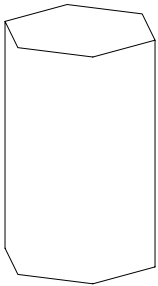
Examples:

RADIUS 1.1, 1.15

RADIUS 0.9, 1.15

CYLIND 3.0, 1.0

CYLIND 3.0, 1.0



RESOL

RESOL n

Sets smoothness for cylindrical elements and arcs in polylines. Circles are converted to regular polygons having n sides.

Arc conversion is proportional to this.

After a RESOL statement, any previous RADIUS and TOLER statements lose their effect.

Parameter restriction:

$n \geq 3$

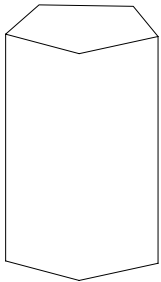
Default:

RESOL 36

Examples:

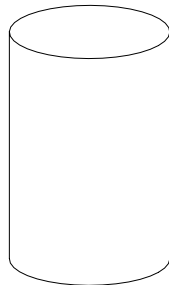
RESOL 5

CYLIND 3.0, 1.0



RESOL 36

CYLIND 3.0, 1.0



TOLER

TOLER d

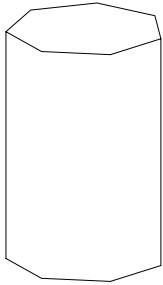
Sets smoothness for cylindrical elements and arcs in polylines. The error of the arc approximation (i.e., the greatest distance between the theoretical arc and the generated chord) will be smaller than d.

After a TOLER statement, any previous RADIUS and RESOL statements lose their effect.

Examples:

```
TOLER 0.1
CYLIND 3.0, 1.0
```

```
TOLER 0.01
CYLIND 3.0, 1.0
```



Note: The RADIUS, RESOL and TOLER directives set smoothness for cylindrical 3D elements (CIRCLE, ARC, CYLIND, SPHERE, ELLIPS, CONE, ARMC, ARME, ELBOW, REVOLVE) and arcs in 2D polylines using curved edges.

See *“Additional Status Codes”* on page 139.

PEN

PEN n

Sets the color.

Parameter restriction:

$0 < n \leq 255$

Default:

PEN 1

if there is no PEN statement in the script.

(For library parts, default values come from the library part's settings. If the script refers to a non existing index, PEN 1 becomes the default setting.)

LINE_PROPERTY

LINE_PROPERTY *expr*

Defines the property for all subsequently generated lines in the 2D script (RECT2, LINE2, ARC2, CIRCLE2, SPLINE2, SPLINE2A, POLY2, FRAGMENT2 commands) until the next LINE_PROPERTY statement. Default value is generic.

expr possible values:

0: all lines are generic lines

1: all lines are inner

2: all lines are contour

[SET] STYLE

[SET] STYLE *name_string*

[SET] STYLE *index*

All the texts generated afterwards will use that style until the next SET STYLE statement.

The index is a constant referring to a style stack in the internal data structure. This stack is modified during GDL analysis and can also be modified from within the program. The use of the index instead of the style name is only recommended with the prior use of the IND function.

Default:

SET STYLE 0

(application font, size 5 mm, anchor = 1, normal face) if there is no SET STYLE statement in the script.

Directives Used in 3D Scripts Only

MODEL

MODEL WIRE

MODEL SURFACE

MODEL SOLID

Sets the representation mode in the current script.

MODEL WIRE: only wireframe, no surfaces or volumes. Objects are transparent.

MODEL SURFACE, MODEL SOLID: The generation of the section surfaces is based on the relation of the boundary surfaces, so that both methods generate the same 3D internal data structure. Objects are opaque.

The only distinction can be seen after cutting away a part of the body:

MODEL SURFACE: the inside of bodies will be visible,

MODEL SOLID: new surfaces may appear.

Default:

```
MODEL SOLID
```

To illustrate the three modeling methods, consider the following three blocks:

```
MODEL WIRE
```

```
BLOCK 3,2,1
```

```
ADDY 4
```

```
MODEL SURFACE
```

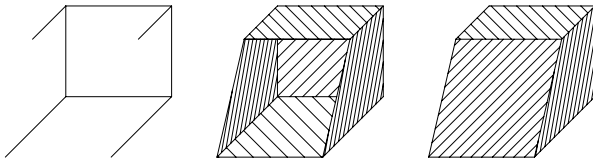
```
BLOCK 3,2,1
```

```
ADDY 4
```

```
MODEL SOLID
```

```
BLOCK 3,2,1
```

After cutting them with a plane:



[SET] MATERIAL

```
[SET] MATERIAL name_string
```

```
[SET] MATERIAL index
```

All the surfaces generated afterwards will represent that material until the next MATERIAL statement. Surfaces in the

```
BPRISM_, CPRISM_, FPRISM_, HPRISM_
SPRISM_, CSLAB_, CWALL_, BWALL_, XWALL_,
CROOF_, MASS
```

bodies are exceptions to this rule.

The index is a constant referring to a material stack in the internal data structure. This stack is modified during GDL analysis and can also be modified from within the program. The use of the index instead of the material name is only recommended with the prior use of the IND function.

index 0 has a special meaning: surfaces use the color of the current pen and they have a matte appearance.

Default:

MATERIAL 0

if there is no MATERIAL statement in the script.

(For Library parts, default values are read from the Library part's settings. If the script refers to a non-existing index, MATERIAL 0 becomes the default setting.)

See also IND function description in the "Miscellaneous" > "Requests" on page 233.

SECT_FILL

SECT_FILL fill, fill_background_pen,
fill_pen, contour_pen

Defines the fill used for the 3D elements subsequently generated in the Section/Elevation window.

fill: fill name or index number

fill_background_pen: fill background pencolor number

fill_pen: fill pencolor number

contour_pen: fill contour pencolor number

SHADOW

SHADOW keyword_1[, keyword_2]

Controls the shadow casting of the elements in PhotoRendering and in vectorial shadow casting.

keyword_1: ON, AUTO or OFF

keyword_2: ON or OFF

ON: all the subsequent elements will cast shadows in all circumstances.

OFF: none of the subsequent elements will cast shadows in any circumstance.

AUTO: shadow casting will be determined automatically.

- Setting SHADOW OFF for hidden parts will spare memory space and processing time.
- Setting SHADOW ON ensures that even tiny details will cast shadows.

The optional second keyword controls the appearance of shadows on surfaces.

- SHADOW keyword_1, OFF disables vectorial shadows on the following surfaces.
- SHADOW keyword_1, ON switches back vectorial shadows.

Default:

SHADOW AUTO

SHADOW OFF

BRICK 1, 1, 1

ADDX 2

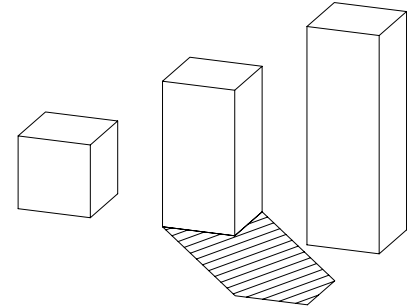
SHADOW ON

BRICK 1, 1, 2

ADDX 2

SHADOW OFF

BRICK 1, 1, 3



Directives Used in 2D Scripts Only

DRAWINDEX

DRAWINDEX number

Defines the drawing order of 2D Script elements. Elements with a smaller drawindex will be drawn first.

Restriction of parameters:

`0 < number <= 50`

(In the current version of GDL only the 10, 20, 30, 40 and 50 DRAWINDEX values are valid. Other values will be rounded to these.)

If no DRAWINDEX directive is present, the default drawing order is the following:

- 1 Figures
- 2 Fills
- 3 Lines
- 4 Text elements

[SET] FILL

[SET] FILL name_string

[SET] FILL index

All the 2D polygons generated afterwards will represent that fill until the next SET FILL statement.

The index is a constant referring to a fill stack in the internal data structure. This stack is modified during GDL analysis and can also be modified from within the program. The use of the index instead of the fill name is only recommended with the prior use of the IND function.

Default:

`SET FILL 0`

i.e., empty fill, if there is no SET FILL statement in the script.

See also IND function description in the [“Miscellaneous”](#) > [“Requests”](#) on page 233.

[SET] LINE_TYPE

[SET] LINE_TYPE name_string

[SET] LINE_TYPE index

All the 2D lines generated afterwards will represent that line type (in lines, arcs, polylines) until the next SET LINE_TYPE statement.

The index is a constant that refers to a line type stack in the internal data structure. This stack is modified during GDL analysis and can also be modified from the program. The use of the index instead of the line type name is only recommended with the prior use of the IND function.

Default:

```
SET LINE_TYPE 1
```

i. e., solid line, if there is no SET LINE_TYPE statement in the script.

See also IND function description in the *“Miscellaneous”* > *“Requests”* on page 233.

INLINE ATTRIBUTE DEFINITION

Attributes in can be created using the material, fill and line type dialog boxes. These floor plan attributes can be referenced from any GDL script. Attributes can also be defined in GDL scripts. There are two different cases:

- Attribute definition in the MASTER_GDL script. The MASTER_GDL script is interpreted when the library that contains it is loaded in the memory. The MASTER_GDL attributes are merged into the floor plan attributes; attributes with the same names are not replaced. Once the MASTER_GDL is loaded, the attributes defined in it can be referenced from any script.
- Attribute definition in library parts. The materials and textures defined this way can be used in the script and its second generation scripts. Fills and line types defined and used in the 2D script have the same behavior as if they were defined in the MASTER_GDL script.

The **Check GDL Script** command in the script window helps to verify whether the material, fill, line type or style parameters are correct.

When a material, fill, line type or style is different in the 3D interpretation of the library part from the intended one, but there is no error message, this probably means that one or more of the parameter values are incorrect. The **Check GDL Scripts** command will help you with detailed messages to find these parameters.

Materials

DEFINE MATERIAL

```
DEFINE MATERIAL name type, parameter1,  
parameter2, ... parametern
```

Note: This command can contain additional data definition.

See *“Additional Data”* on page 175 for details.

Any GDL script can include material definitions prior to the first reference to that material name. This material can only be used for 3D elements in its own script and its second generation scripts.

name: name of the material.

type: type of the material. The actual number (n) of parameters that define the material is different, depending on the type. The meaning of the parameters and their limits are explained in the examples' comments.

0: general definition, n=16

1: simple definition, n=9 (extra parameters are constants or calculated from given values)

2-7: predefined material types, n=3

The three values are the RGB components of the surface color. Other parameters are constants or calculated 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.

Zero value for a texture index allows you to define materials without a vectorial hatch or texture.

Examples:

```
DEFINE MATERIAL "water" 0,
  0.5284, 0.5989, 0.6167,
!  surface RGB [0.0..1.0]
  1.0, 0.5, 0.5, 0.9,
!  ambient, diffuse, specular, transparent
!  coefficients [0.0..1.0]
2.0,
!  shining [0.0..100.0]
  1,
```

```
! transparency attenuation [0.0..4.0]
  0.5284, 0.5989, 0.6167,
! specular RGB [0.0..1.0]
  0, 0, 0,
! emission RGB [0.0..1.0]
  0.0
! emission attenuation [0.0..65.5]
DEFINE MATERIAL "asphalt" 1,
  0.1995, 0.2023, 0.2418,
! surface RGB [0.0..1.0]
  1.0, 1.0, 0.0, 0.0,
! ambient, diffuse, specular,transparent
! coefficients [0.0..1.0]
  0,
! shining [0..100]
  0
! transparency attenuation [0..4]
DEFINE MATERIAL "matte red" 2,
  1.0, 0.0, 0.0
! surface RGB [0.0..1.0]
DEFINE MATERIAL "Red Brick" 10,
  0.878294, 0.398199, 0.109468,
  0.58, 0.85, 0.0, 0.0,
  0,
  0.0,
  0.878401, 0.513481, 0.412253,
  0.0, 0.0, 0.0,
  0,
IND(FILL, "common brick")
! fill index
DEFINE MATERIAL "Yellow Brick+*" 20,
  1, 1, 0,
! surface RGB [0.0 .. 1.0]
  0.58, 0.85, 0, 0,
! ambient, diffuse, specular,transparent
! coefficients [0.0 .. 1.0]
  0,
! shining [0.0 .. 100.0]
  0,
! transparency attenuation [0.0 .. 4.0]
  0.878401, 0.513481, 0.412253,
! specular RGB [0.0 .. 1.0]
  0, 0, 0,
! emission RGB [0.0 .. 1.0]
```

```

0,
! emission attenuation [0.0 .. 65.5]
  IND(FILL, "common brick"), 61,
  IND(TEXTURE, "Brick")
! Fill index, color index, texture index

```

DEFINE MATERIAL BASED_ON

```

DEFINE MATERIAL name BASED_ON orig_name PARAMETERS name1 = expr1 [,
  ...][ADDITIONAL_DATA name1 = expr1 [, ...]]

```

Material definition based on an existing material. Specified parameters of the original material will be overwritten by the new values, other parameters remain untouched. Using the command without actual parameters results in a material exactly the same as the original, but with a different name. Parameter values of a material can be obtained using the *“REQUEST{2} (“Material_info”, name_or_index, param_name, value_or_values)”* function.

orig_name: name of the original material (name of an existing, previously defined in GDL or floor plan material)

namei: material parameter name to be overwritten by a new value. Names corresponding to parameters of material definition:

```

gs_mat_surface_r, gs_mat_surface_g, gs_mat_surface_b (surface RGB [0.0..1.0])
  gs_mat_ambient (ambient coefficient [0.0..1.0])
  gs_mat_diffuse (diffuse coefficient [0.0..1.0])
  gs_mat_specular (specular coefficient [0.0..1.0])
  gs_mat_transparent (transparent coefficient [0.0..1.0])
  gs_mat_shining (shininess [0.0..100.0])
  gs_mat_transp_att (transparency attenuation [0.0..4.0])
  gs_mat_specular_r, gs_mat_specular_g, gs_mat_specular_b (specular color RGB [0.0..1.0])
  gs_mat_emission_r, gs_mat_emission_g, gs_mat_emission_b (emission color RGB [0.0..1.0])
  gs_mat_emission_att (emission attenuation [0.0..65.5])
  gs_mat_fill_ind (fill index)
  gs_mat_fillcolor_ind (fill color index)
  gs_mat_texture_ind (texture index)

```

expr1: new value to overwrite the specified parameter of the material. Value ranges are the same as at the material definition.

Example:

```
n = REQUEST{2} ("Material_info", "Brick-Face", "gs_mat_emission_rgb ", em_r, em_g, em_b)
em_r = em_r + (1 - em_r) / 3
em_g = em_g + (1 - em_g) / 3
em_b = em_b + (1 - em_b) / 3
DEFINE MATERIAL "Brick-Face light" BASED_ON "Brick-Face" PARAMETERS gs_mat_emission_r = em_r,
gs_mat_emission_g = em_g, gs_mat_emission_b = em_b

SET MATERIAL "Brick-Face"
BRICK a, b, zzyzx
ADDX a
SET MATERIAL "Brick-Face light"
BRICK a, b, zzyzx
```

DEFINE TEXTURE

DEFINE TEXTURE name expression, x, y, mask, angle

Any GDL script can include texture definition prior to the first reference to that texture name. The texture can be used only in the script in which it was defined and its subsequent second generation scripts.

name: name of the texture

expression: picture associated with the texture. A string expression means a file name, a numerical expression an index of a picture stored in the library part. A 0 index is a special value which refers to the preview picture of the library part.

x: logical width of the texture

y: logical height of the texture

mask: $j_1 + 2*j_2 + 4*j_3 + 8*j_4 + 16*j_5 + 32*j_6 + 64*j_7 + 128*j_8 + 256*j_9$
where $j_1, j_2, j_3, j_4, j_5, j_6, j_7, j_8, j_9$ can be 0 or 1.

Alpha channel controls ($j_1... j_6$):

j1: alpha channel changes the transparency of texture

j2: Bump mapping or surface normal perturbation.

Bump mapping uses the alpha channel to determine the amplitude of the surface normal.

j3: alpha channel changes the diffuse color of texture

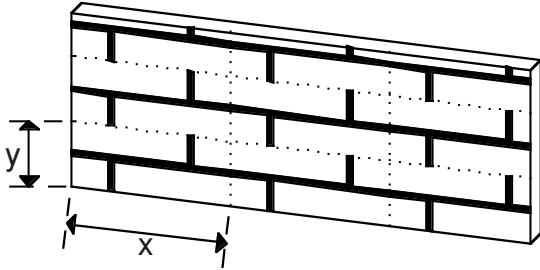
j4: alpha channel changes the specular color of texture

j5: alpha channel changes the ambient color of texture

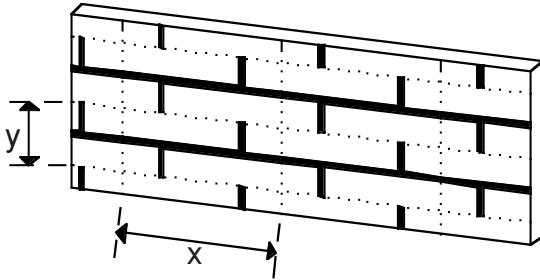
j6: alpha channel changes the surface color of texture

Connection controls ($j_7... j_9$):

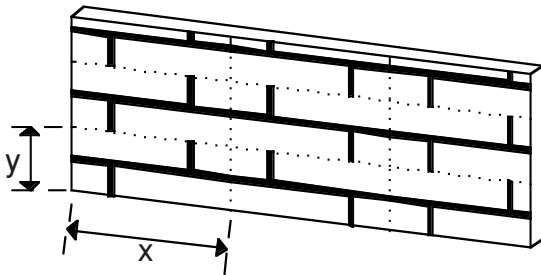
If the value is zero, normal mode is selected:



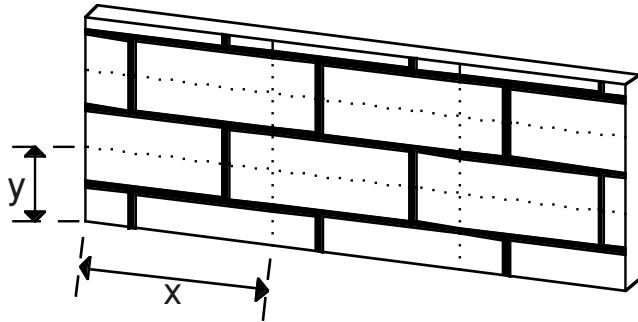
j7: the texture will be shifted randomly.



j8: mirroring in 'x' direction



j9: mirroring in 'y' direction



angle: angle of the rotation.

Example:

```
DEFINE TEXTURE "Brick" "Brick.PICT", 1.35, 0.3, 256+128, 35.0
```

Fills

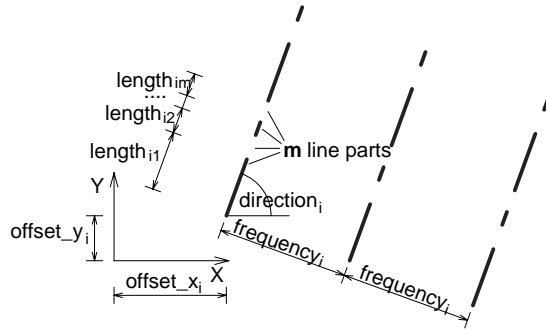
DEFINE FILL

```
DEFINE FILL name [FILLTYPES_MASK fill_types,] pattern1, pattern2, pattern3, pattern4,
  pattern5, pattern6, pattern7, pattern8,
  spacing, angle, n,
  frequency1, direction1, offset_x1, offset_y1, m1,
  length11, ... length1m,
  ...
  frequencyn, directionn, offset_xn,
  lengthn1, ... lengthnm
```

Note: This command can contain additional data definition.

See *“Additional Data” on page 175* for details.

Any GDL script may include fill definitions prior to the first reference to that fill name. The fill defined this way can be used only in the script in which it was defined and its subsequent second generation-scripts.



name: name of the fill

fill_types = $j1 + 2 * j2 + 4 * j3$

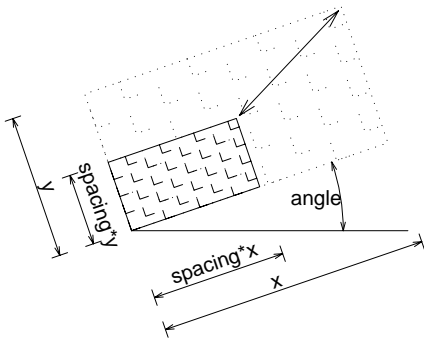
j1: cut fills

j2: cover fills

j3: drafting fills

If the j bit is set, the defined fill can be used in ArchiCAD correspondig to its specified type. Default is all fills (0).

pattern definition: pattern1, pattern2, pattern3, pattern4, pattern5, pattern6, pattern7, pattern8: 8 numbers between 0 and 255 representing binary values. Defines the bitmap pattern of the fill.



spacing: hatch spacing - defines a global scaling factor for the whole fill. All values will be multiplied by this number in both the x and y direction.

angle: global rotation angle in degrees

n: number of hatch lines

frequencyi: frequency of the line (the distance between two lines is spacing * frequencyi)

diri: direction angle of the line in degrees

offset_xi, offset_yi: offset of the line from the origin

mi: number of line parts

lengthij: length of the line parts (the real length is spacing * lengthij). Line parts are segments and spaces following each other. First line part is a segment, zero length means a dot.

The **bitmap pattern** is only defined by the pattern1... pattern8 parameters and is used when the display options for Polygon Fills are set to "Bitmap Pattern" (Options menu). To define it, choose the smallest unit of the fill, and represent it as dots and empty spaces using a rectangular grid with 8x8 locations. The 8 pattern parameters are decimal representations of the binary values in the lines of the grid (a dot is 1, an empty space is 0).

The **vectorial hatch** is defined by the second part of the fill definition as a collection of dashed lines repeated with a given frequency (frequencyi). Each line of the collection is described by its direction (directioni), its offset from the origin (offset_xi, offset_yi) and the dashed line definition which contains segments and spaces with the given length (lengthij) following each other.

Note: Only simple fills can be defined with the DEFINE FILL command. There is no possibility to define symbol fills.

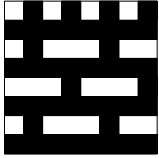
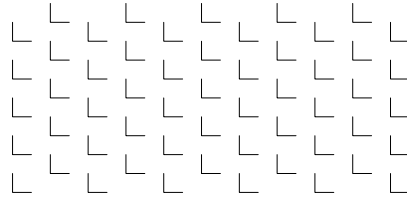
Example:

```
DEFINE FILL "brick" 85, 255, 136, 255,
  34, 255, 136, 255,
  0.08333, 0.0, 4,
  1.0, 0.0, 0.0, 0.0, 0,
  3.0, 90.0, 0.0, 0.0, 2,
  1.0, 1.0,
  3.0, 90.0, 1.5, 1.0, 4,
  1.0, 3.0, 1.0, 1.0,
  1.5, 90.0, 0.75, 3.0, 2,
  1.0, 5.0
```

Bitmap pattern:

```
Pattern: Binary value:
pattern1 = 85      01010101      . . . .
pattern2 = 255    11111111      .....
pattern3 = 136    10001000      . .
pattern4 = 255    11111111      .....
pattern5 = 34     00100010      . .
pattern6 = 255    11111111      .....
pattern7 = 136    10001000      . .
pattern8 = 255    11111111      .....

```

View:*Vectorial batch:*

DEFINE FILLA

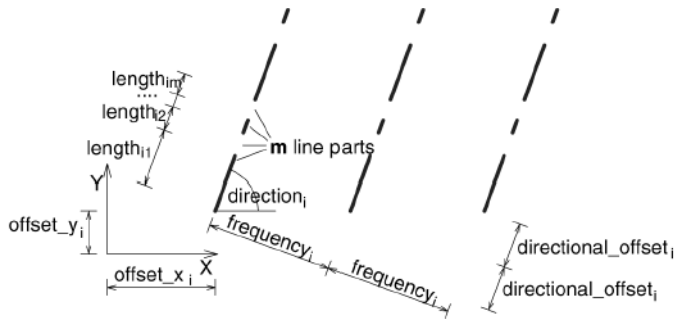
```

DEFINE FILLA name [FILLTYPES_MASK fill_types,] pattern1, pattern2, pattern3, pattern4, pattern5,
  pattern6, pattern7, pattern8, spacing_x, spacing_y, angle, n, frequency1,
  directional_offset1, direction1,
  offset_x1, offset_y1, m1, length11,
  ...
  length1m, ... frequencyn,
  directional_offsetn, directionn,
  offset_xn, offset_yn, mn,
  lengthn1, ... lengthnm

```

Note: This command can contain additional data definition.

See [“Additional Data” on page 175](#) for details.



An extended DEFINE FILL statement.

Additional parameters:

spacing_x, spacing_y: spacing factor in the x and y direction, respectively. These two parameters define a global scaling factor for the whole fill. All values in the x direction will be multiplied by spacing_x and all values in the y direction will be multiplied by spacing_y.

directional_offseti: the offset of the beginning of the next similar hatch line, measured along the line's direction. Each line of the series will be drawn at a distance defined by frequency_i with an offset defined by directional_offset_i. The real length of the offset will be spacing * directional_offset_i.

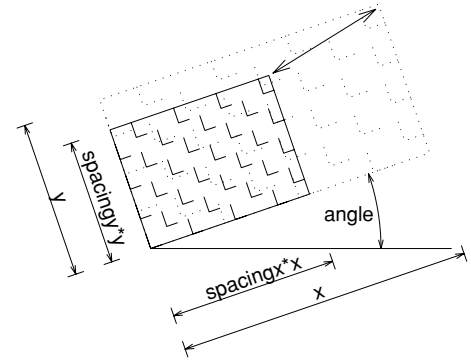
Example:

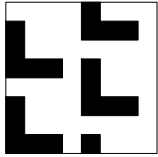
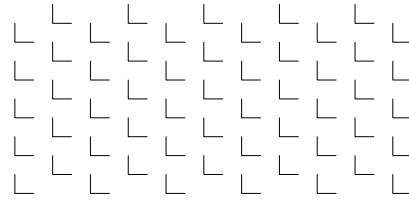
```
DEFINE FILLA "TEST" 8, 142, 128, 232,
                8, 142, 128, 232,
                0.5, 0.5, 0, 2,
                2, 1, 90, 0,
                0, 2, 1, 1,
                1, 2, 0, 0, 0,
                2, 1, 3

FILL "TEST"
POLY2 4, 6,
      -0.5, -0.5, 12, -0.5,
      12, 6, -0.5, 6
```

Bitmap pattern:

Pattern:	Binary value:
pat1 = 8	00001000 .
pat2 = 142	10001110
pat3 = 128	10000000 .
pat4 = 232	11101000
pat5 = 8	00001000 .
pat6 = 142	10001110
pat7 = 128	10000000 .
pat8 = 232	11101000



View:*Vectorial batch:*

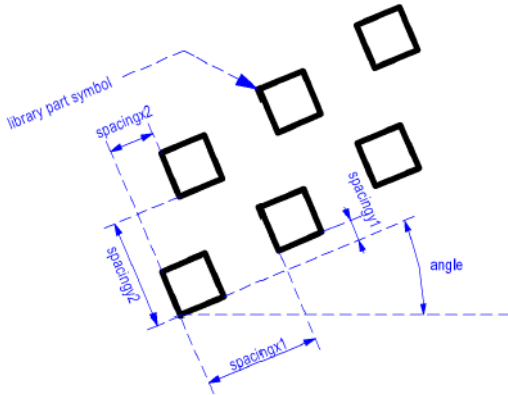
DEFINE SYMBOL_FILL

```

DEFINE SYMBOL_FILL name [FILLTYPES_MASK fill_types,]
    pat1, pat2, pat3, pat4, pat5, pat6, pat7, pat8,
    spacingx1, spacingy1,
    spacingx2, spacingy2, angle,
    scaling1, scaling2, macro_name PARAMETERS [name1 = value1, ... namen = valuen]
  
```

Note: This command can contain additional data definition.

See *“Additional Data” on page 175* for details.



An extended DEFINE FILL statement, which allows you to include a library part drawing in a fill definition. The usage of macro_name and the parameters are the same as for the CALL command.

Parameters:

spacingx1, spacingx2: horizontal spacings

spacingy1, spacingy2: vertical spacings

scaling1: horizontal scale

scaling2: vertical scale

macro_name: the name of the library part.

DEFINE SOLID_FILL**DEFINE SOLID_FILL** name [**FILLTYPES_MASK** fill_types]

Defines a solid fill.

Note: This command can contain additional data definition.*See “Additional Data” on page 175 for details..***DEFINE EMPTY_FILL****DEFINE EMPTY_FILL** name [**FILLTYPES_MASK** fill_types]

Defines an empty fill.

Note: This command can contain additional data definition.*See “Additional Data” on page 175 for details.***Line Types****DEFINE LINE_TYPE****DEFINE LINE_TYPE** name spacing, n,
length1, ... lengthn**Note:** This command can contain additional data definition.*See “Additional Data” on page 175 for details..*

Any GDL script may include line type definitions prior to the first reference to that line-type name. The line type defined this way can be used only for 2D elements in the script in which it was defined and its subsequent second generation scripts.

name: name of the line type

spacing: spacing factor

n: number of the line parts

lengthi: length of the line parts (the real length is spacing * lengthi). Line parts consist of segments and spaces. First line part is a segment, zero length means a dot.

Note: Only simple line types can be defined in the DEFINE LINE_TYPE command, that is, ones consisting only of segments and spaces, there is no possibility to define symbol lines.

Example:

```
DEFINE LINE_TYPE "line - - ." 1,
  6, 0.005, 0.002, 0.001, 0.002, 0.0, 0.002
```

DEFINE SYMBOL_LINE

DEFINE SYMBOL_LINE name dash, gap, macro_name PARAMETERS [name1 = value1, ... namen = valuen]

Note: This command can contain additional data definition.

See *“Additional Data” on page 175 for details.*

An extended DEFINE LINE statement, which allows you to include a library part drawing in a line definition. The usage of macro_name and the parameters are the same as for the CALL command.

Parameters:

dash: scale of both line components

gap: gap between each component

Styles

DEFINE STYLE

DEFINE STYLE name font_family, size, anchor, face_code

Recommended to be used with the TEXT2 and TEXT commands.

GDL scripts may include style definitions prior to the first reference to that style name. The style defined this way can be used only in the script in which it was defined and its subsequent second generation scripts.

name: name of the style

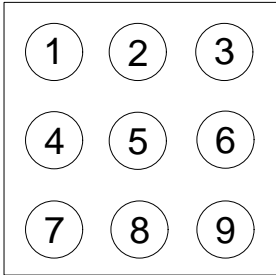
font_family: name of the used font family (e.g., Garamond)

size: height of the “I” character in millimeters or m in model space

If the defined style is used with the TEXT2 and TEXT commands, size means character heights in millimeters.

If used with PARAGRAPH strings in the RICHTEXT2 and RICHTEXT commands, size meaning millimeters or meters depends on the fixed_height parameter of the TEXTBLOCK definition, while the outline and shadow face_code values and the anchor values are not effective.

anchor: code of the position point in the text



face_code: a combination of the following values:

- 0 normal
- 1 **bold**
- 2 *italic*
- 4 underline
- 8 outline
- 16 shadow

Note: The outline and shadow values are effective only on Macintosh platform and up to the 8.1 version of ArchiCAD.

DEFINE STYLE {2}

DEFINE STYLE{2} name font_family, size, face_code

New version of style definition, recommended to be used with PARAGRAPH definitions.

name: name of the style

font_family: name of the used font family (e.g., Garamond)

size: height of the characters in mm or m in model space

face_code: a combination of the following values:

- 0 normal
- 1 **bold**
- 2 *italic*
- 4 underline
- 32 ^{superscript}
- 64 _{subscript}
- 128 ~~strikethrough~~

If the defined style is used with the TEXT2 command, size means character heights in millimeters, while the superscript, subscript and strikethrough face_code values are not effective. If used with PARAGRAPH strings in the RICHTEXT2 and RICHTEXT commands, size meaning millimeters or meters depends on the fixed_height parameter of the TEXTBLOCK definition.

Paragraph

PARAGRAPH

```
PARAGRAPH name alignment, firstline_indent,
    left_indent, right_indent, line_spacing [,
    tab_size1, ...]
    [PEN index]
    [[SET] STYLE style1]
    [[SET] MATERIAL index]
    'string1'
    'string2'
    ...
    'string n'
    [PEN index]
    [[SET] STYLE style2]
    [[SET] MATERIAL index]
    'string1'
    'string2'
    ...
    'string n'
    ...
```

ENDPARAGRAPH

GDL scripts may include paragraph definitions prior to the first reference to that paragraph name. The paragraph defined this way can be used only in the script in which it was defined and its subsequent second generation scripts. A paragraph is defined to be a sequence of an arbitrary number of strings (max 256 characters long each) with different attributes: style, pen and material (3D). If no attributes are specified inside the paragraph definition, actual (or default) attributes are used. The new lines included in a paragraph string (using the special character '\n') will automatically split the string into identical paragraphs, each containing one line. Paragraph definitions can be referenced by name in the TEXTBLOCK command. All length type parameters (firstline_indent, left_indent, right_indent, tab_position) meaning millimeters or meters depends on the fixed_height parameter of the TEXTBLOCK definition.

name: name of the paragraph

alignment: alignment of the paragraph strings. Possible values:

1: left aligned, 2: center aligned, 3: right aligned, 4: full justified

firstline_indent: first line indentation, in mm or m in model space

left_indent: left indentation, in mm or m in model space

right_indent: right indentation, in mm or m in model space

line_spacing: line spacing factor. The default distance between the lines (character size + distance to the next line) defined by the actual style will be multiplied by this number.

tab_positioni: consecutive tabulator positions (each realative to the beginning of the paragraph), in mm or m in model space. Tabulators in the paragraph strings will snap to these positions. If no tabulator positions are specified, default values are used (12.7 mm).

Textblock

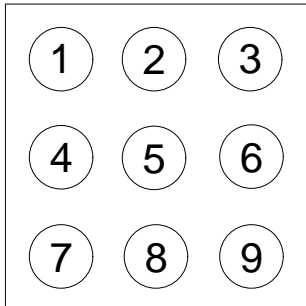
TEXTBLOCK

TEXTBLOCK name width, anchor, angle, width_factor, charspace_factor, fixed_height, 'string_expr1' [, 'string_expr2', ...]

Textblock definition. GDL scripts may include textblock definitions prior to the first reference to that textblock name. The textblock defined this way can be used only in the script in which it was defined and its subsequent second generation scripts. A textblock is defined to be a sequence of an arbitrary number of strings or paragraphs which can be placed using the *“RICHTEXT2” on page 125* and *“RICHTEXT” on page 94*. Use the *“REQUEST (TEXTBLOCK_INFO; textblock_name, width, height)”* to obtain information on the calculated width and height of a TEXTBLOCK.

width: textblock width in mm or m in model space , if 0 it will be calculated automatically

anchor: code of the position point in the text



angle: rotation angle of the textblock in degrees

width_factor: character width factor. Character widths defined by the actual style will be multiplied by this number.

charspace_factor: character space factor. The horizontal distance between two characters will be multiplied by this number.

fixed_height: possible values

- 1: the placed TEXTBLOCK will be scale-independent and all specified length type parameters will mean millimeters
- 0: the placed TEXTBLOCK will be scale-dependent and all specified length type parameters will mean meters in model space.

string_expri: means paragraph name if it was previously defined, simple string otherwise (with default paragraph parameters).

Additional Data

Attribute definitions can contain optional additional data definitions after the `ADDITIONAL_DATA` keyword. The additional data must be entered after the previously defined parameters of the attribute command. An additional data has a name (`namei`) and a value (`valuei`), which can be an expression of any type, even an array. If a string parameter name ends with the substring “_file”, its value is considered to be a file name and will be included in the archive project. Different meanings of additional data can be defined and used by ArchiCAD or Add-Ons to ArchiCAD.

See meanings of LightWorks Add-On parameters at <http://www.graphisoft.com/support/developer/documentation/wide/LibraryDevKit/>.

Additional data definition is available in the following commands:

DEFINE MATERIAL

DEFINE MATERIAL parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]

DEFINE FILL

DEFINE FILL parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]

DEFINE FILL_A

DEFINE FILL_A parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]

DEFINE SYMBOL_FILL

DEFINE SYMBOL_FILL parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]

DEFINE LINE_TYPE

DEFINE LINE_TYPE parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]

DEFINE SYMBOL_LINE

DEFINE SYMBOL_LINE parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]

External file dependence

FILE_DEPENDENCE "name1" [, "name2", ...]

You can give a list of external files on which your GDL script depends on. File names should be constant strings.

All files specified here will be included in the archive project (like constant macro names used in *CALL* statements and constant picture names used in various GDL commands). The command works on this level only: if the specified files are library parts, their called maro files will not be included.

The command can be useful in cases when external files are referenced at custom places in the GDL script, for example:

ADDITIONAL_DATA file parameters, data files in file operations.

NON-GEOMETRIC SCRIPTS

In addition to the 3D and 2D script windows that define the appearance of the GDL Object, further scripts are available for adding complementary information to it. These are the Properties Script used for quantity calculations, the Parameter Script that includes the list of possible values for different parameters, and the User Interface Script for creating a custom interface for parameter entry. The commands available for all these script types are detailed on the following pages.

THE PROPERTIES SCRIPT

Library parts have a GDL window reserved for the Properties script. This script allows you to make library part properties dependent on parameters, and, through a directive, define their place in the final component list. By using a few commands, it is possible to define in the script local descriptors and components (created in the Properties windows of earlier ArchiCAD versions). Descriptors and components from external databases can also be referenced. Code lengths cannot exceed 32 characters.

In the Properties script, you can use any GDL command that does not generate a shape.

DATABASE_SET

DATABASE_SET set_name [descriptor_name, component_name, unit_name, key_name, criteria_name, list_set_name]

Database set definition or Database set selection. If this command is placed in a MASTER_GDL script, it will define a Database set containing Descriptor, Component, Unit, Key, Criteria and List Scheme files.

This Database set name can then be referenced from Properties Scripts using the same command with only the set_name parameter as a directive, by selecting the actual Database set that REF COMPONENTs and REF DESCRIPTORs refer to. The default Database set name is “Default Set”, and will be used if no other set has been selected. The default Database set file names are: DESCDATA, COMPDATA, COMPUNIT, LISTKEY, LISTCRIT, LISTSET.

Scripts can include any number of DATABASE_SET selections.

set_name: Database set name

descriptor_name: Descriptor data file name

component_name: Component data file name

unit_name: Unit data file name

key_name: Key data file name

criteria_name: Criteria file name

list_set_name: List Scheme file name

DESCRIPTOR

DESCRIPTOR name [, code, keycode]

Local descriptor definition. Scripts can include any number of DESCRIPTORS.

name: can extend to more than one line. New lines can be defined by the character ‘\n’ and tabulators by ‘\t’. Adding ‘\’ to the end of a line allows you to continue the string in the next line without adding a new line. Inside the string, if the ‘\’ character is doubled (\\), it will lose its control function and simply mean ‘\’.

The length of the string (including the new line characters) cannot exceed 255 characters: additional characters will be simply cut by the compiler. If you need a longer text, use several DESCRIPTORS.

code: string, defines a code for the descriptor

keycode: string, reference to a key in an external database.

The key will be assigned to the descriptor.

REF DESCRIPTOR

REF DESCRIPTOR code [, keycode]

Reference by code and keycode string to a descriptor in an external database.

COMPONENT

COMPONENT name, quantity, unit [, proportional_with, code, keycode, unitcode]

Local component definition. Scripts can include any number of COMPONENTS.

name: the name of the component (max. 128 characters)

quantity: a numeric expression

unit: the string used for unit description

proportional_with: a code between 1 and 6. When listing, the component quantity defined above will be automatically multiplied by a value calculated for the current listed element:

- 1: item
- 2: length
- 3: surface A
- 4: surface B
- 5: surface
- 6: volume

code: string, defines a code for the component

keycode: string, reference to a key in an external database. The key will be assigned to the component.

unitcode: string, reference to a unit in an external database that controls the output format of the component quantity. This will replace the locally defined unit string.

REF COMPONENT

REF COMPONENT code [, keycode [, numeric_expression]]

Reference by code and keycode string to a component in an external database. The value to multiply by in the component database can be overwritten by the optional numeric expression specified here.

BINARYPROP

BINARYPROP

Binaryprop is a reference to the binary properties data (components and descriptors) defined in the library part in the Components and Descriptors sections.

DATABASE_SET directives have no effect on the binary data.

SURFACE3D ()

SURFACE3D ()

The Surface 3D () function gives you the surface of the 3D shape of the library part.

Warning: If you place two or more shapes in the same location with the same parameters, this function will give you the total sum of all shapes' surfaces.

VOLUME3D ()

VOLUME3D ()

The Volume 3D () function gives you the volume of the 3D shape of the library part.

Warning: If you place two or more shapes in the same location with the same parameters, this function will give you the total sum of all shapes' volumes.

POSITION

POSITION position_keyword

Effective only in the Component List.

Changes only the type of the element the following descriptors and components are associated to. If there are no such directives in the Properties script, descriptors and components will be listed with their default element types.

Keywords are the following:

WALLS
COLUMNS
BEAMS
DOORS
WINDOWS

OBJECTS
CEILS
PITCHED_ROOFS
LIGHTS
HATCHES
ROOMS
MESHES

A directive remains valid for all succeeding DESCRIPTORS and COMPONENTs until the next directive is ascribed. A script can include any number of directives.

Example:

```
DESCRIPTOR "\tPainted box.\n\t Properties:\n\t\t - swinging doors\n\t\t - adjustable height\n\t\t - scratchproof"  
REF DESCRIPTOR "0001"  
s = SURFACE3D () !wardrobe surface  
COMPONENT "glue", 1.5, "kg"  
COMPONENT "handle", 2*c, "nb"!c number of doors  
COMPONENT "paint", 0.5 * s, "kg"  
POSITION WALLS  
REF COMPONENT "0002"
```

DRAWING

DRAWING

DRAWING: Refers to the drawing described in the 2D script of the same library part. Use it to place drawings in your bill of materials.

THE PARAMETER SCRIPT

Parameter lists are sets of possible numerical or string values. They can be applied to the parameters as defined in the Parameter Script of the Library Part or in the MASTER_GDL script. The parameter has to be of simple type. Type compatibility is verified by the GDL compiler.

The Parameter Script will be interpreted each time a value list type parameter value is to be changed, and the possible values defined in the script will appear in a pop-up menu.

VALUES

VALUES "fillparam_name" [**FILLTYPES_MASK** fill_types,] value_definition1 [, value_definition2, ...]

name: the name of the parameter

fill_types = j1 + 2 * j2 + 4 * j3

j1: cut fills

j2: cover fills

j3: drafting fills

Can only be used for fill type parameters. If the j bit is set, the fill popup corresponding to the "fillparam_name" parameter will automatically contain only the fills of the specified type. Default is all fills (0).

value_definitioni: value definition, can be:

expressioni: numerical or string expression, or

CUSTOM: keyword, meaning that any custom value can be entered, or

RANGE left_delimiter [expression₁], [expression₂]
 right_delimiter [**STEP** step_start_value,
 step_value]

range definition, with optional step

left_delimiter: [, meaning '>=', or (, meaning '>'

expression1: lower limit expression

expression2: upper limit expression

right_delimiter:], meaning '<=', or), meaning '<'

step_start_value: starting value

step_value: step value

Examples:

```
VALUES "par1" 1, 2, 3
VALUES "par2" "a", "b"
VALUES "par3" 1, CUSTOM, SIN (30)
VALUES "par4" 4,RANGE(5, 10],12,RANGE(,20]
    STEP 14.5, 0.5, CUSTOM

! Example to read all string values from a file
! and use it in a value list
DIM sarray[]
! file in the library, containing parameter data
filename = "ProjectNotes.txt"
ch1 = OPEN ("text", filename, "MODE=RO, LIBRARY")
i = 1
j = 1
sarray[1] = ""
! collect all strings
DO
n = INPUT (ch1, i, 1, var)
IF n > 0 AND VARTYPE (var) = 2 THEN
sarray[j] = var
j = j + 1
ENDIF
i = i + 1
WHILE n > 0
CLOSE ch1
! parameter popup with strings read from the file
VALUES "RefNote" sarray
```

PARAMETERS

```
PARAMETERS name1 = expression1 [,
    name2 = expression2, ...,
    namen = expressionn]
```

namei: the name of the parameter

expressioni: the new value of the parameter

Using this command, the parameter values of a Library Part can be modified by the Parameter Script.

The modification will only be effective for the next interpretation. Commands in macros refer to the caller's parameters. If the parameter is a value list, the value chosen will be either an existing value, the custom value, or the first value from the value list.

In addition, the global string variable GLOB_MODPAR_NAME contains the name of the last user-modified parameter.

LOCK

LOCK name1 [, name2, ..., namen]

Locks the named parameter in the settings dialog box. A locked parameter will appear grayed in the dialog box and its value cannot be modified by the user.

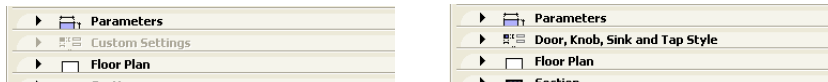
HIDEPARAMETER

HIDEPARAMETER name1 [, name2, ..., namen]

Hides the named parameter(s) and its child parameters in the settings dialog box. A parameter hidden using this command in the parameter script will automatically disappear from the parameter list.

THE USER INTERFACE SCRIPT

Using the following GDL commands, you can define a custom interface for a Library Part's Custom Settings panel in the settings dialog box. If you click the "Set as default" button in the Library Part editor, the custom interface will be used by default in the Object's (Door's, Window's, etc.) settings dialog box. Parameters with custom control are not hidden automatically on the original parameter list, but they can be hidden manually in the library part editor.



The origin of the coordinate system is in the top-left corner. Sizes and coordinate values are measured in pixels.

UI_DIALOG

UI_DIALOG title [, size_x, size_y]

Defines the title of the dialog box. Currently, the size of the available area is fixed at 444 x 266 pixels, and the size_x and size_y parameters are not used.

Restriction: The Interface Script can contain only one UI_DIALOG command.

UI_PAGE

UI_PAGE page_number

Page directive, defines the page that the interface elements are placed on. Page numbering starts at 1. Moving between pages can be defined by creating two buttons with the UI_NEXT and UI_PREV commands.

If there is no UI_PAGE command in the Interface Script, each element will be placed on the first page by default.

Warning: Any break of continuity in the page list forces the insertion of a new page without buttons, and therefore no possibility to go to any other page from there.

UI_BUTTON

UI_BUTTON *type, text, x, y, width, height*

Button definition on the current page. Buttons can be used for moving from page to page. They are not generated automatically and have to be defined for each page.

type: type of the button as followstext: the text that should appear on the button

x, y: the position of the button

width, height: width and height of the button in pixels

UI_SEPARATOR

UI_SEPARATOR *x1, y1, x2, y2*

Generates a separator rectangle. The rectangle becomes a single (vertical or horizontal) separator line if $x1 = x2$ or $y1 = y2$

x1, y1: upper left node coordinates (starting point coordinates of the line)

x2, y2: lower right node coordinates (endpoint coordinates of the line)

UI_GROUPBOX

UI_GROUPBOX *text, x, y, width, height*

A groupbox is a rectangular separator. It can be used to visually group logically related parameters.

text: the title of the groupbox

x, y: the position of upper left corner

width, height: width and height in pixels

UI_PICT

UI_PICT *expression, x, y [,width, height]*

Picture element in the dialog box. The picture file must be located in one of the loaded libraries.

expression: file name or index number of the picture stored in the library part. The index 0 refers to the preview picture of the library part.

x, y: position of the top left corner of the picture.

width, height: optional width and height in pixels; by default, the picture's original width and height values will be used.

UI_STYLE

UI_STYLE *fontsize, face_code*

All the UI_OUTFIELDS and UI_INFIELDS generated after this keyword will represent this style until the next UI_STYLE statement.

fontsize: one of the following font size values

- 0: small
- 1: extra small
- 2: large

face_code: similar to the STYLE definition, but the values cannot be used in combination.

- 0: normal
- 1: **bold**
- 2: *italic*
- 4: underline
- 8: outline
- 16: shadow

UI_OUTFIELD

UI_OUTFIELD *expression, x, y, width, height*

Generates a static text.

expression: numerical or string expression

x, y: position of the text block's top left corner

width, height: width and height of the pixels

UI_INFIELD

```
UI_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]
```

UI_INFIELD {2}

```
UI_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]
```

Generates an edit text or a pop-up menu for the parameter input. A pop-up is generated if the parameter type is value list, material, fill, line type or pencolor.

If the optional parameters of the command are present, value lists can be alternatively displayed as thumbnail view fields. Thumbnail view fields display the specified images and associated texts, have scrollbars at their borders and allow the selection of one single item at a time, just like in a pop-up menu.

The Interface Script is rebuilt with the new value after any parameter is modified.

name: parameter name as string expression for `UI_INFIELD` or parameter name with optional actual index values if array for `UI_INFIELD{2}`

x, y: the position of the edit text, pop-up or control

width, height: width and height in pixels

version_flag: reserved, always 1

picture_name: name of the common image file containing a matrix of concatenated images, or empty string

images_number: number of images in the matrix

rows_number: number of rows of the matrix

cell_x, cell_y: width and height of a cell within the thumbnail view field, including image and text

image_x, image_y: width and height of the image in the cell

expression_image*i*: index of image number *i* in the matrix, or individual file name. If a common image file name was specified, indices must be used here. Combination of indices and individual file names does not work.

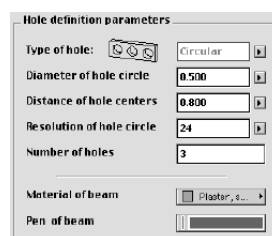
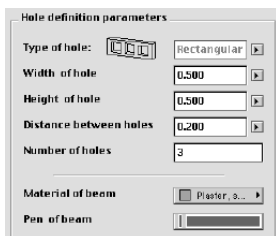
text*i*: text in cell number *i*

Example:

```

IF c THEN
  UI_DIALOG "Hole definition parameters"
  UI_OUTFIELD "Type of hole:",15,40,180,20
  UI_INFIELD "D",190,40,105,20
  IF D="Rectangular" THEN
    UI_PICT "rect.pict",110,33,60,30
    UI_OUTFIELD "Width of hole",15,70,180,20
    UI_INFIELD "E", 190,70,105,20
    UI_OUTFIELD "Height of hole",15,100,180,20
    UI_INFIELD "F", 190,100,105,20
    UI_OUTFIELD "Distance between
      holes",15,130,180,20
    UI_INFIELD "G", 190,130,105,20
  ELSE
    UI_PICT "circle.pict",110,33,60,30
    UI_OUTFIELD "Diameter of hole
      circle",15,70,180,20
    UI_INFIELD "J", 190,70,105,20
    UI_OUTFIELD "Distance of hole
      centers",15,100,180,20
    UI_INFIELD "K", 190,100,105,20
    UI_OUTFIELD "Resolution of hole
      circle",15,130,180,20
    UI_INFIELD "M", 190,130,105,20
  ENDIF
  UI_OUTFIELD "Number of holes",15,160,180,20
  UI_INFIELD "I", 190,160,105,20
ENDIF
UI_SEPARATOR 50,195,250,195
UI_OUTFIELD "Material of beam", 15,210,180,20
UI_INFIELD "MAT", 190,210,105,20
UI_OUTFIELD "Pen of beam", 15,240,180,20
UI_INFIELD "P", 190,240,105,20

```



EXPRESSIONS AND FUNCTIONS

All parameters of GDL shapes can be the result of calculations. For example, you can define that the height of the cylinder is five times the radius of the cylinder, or prior to defining a cube, you can move the coordinate system in each direction by half the size of the cube, in order to have the initial origin in the center of the cube rather than in its lower left corner. To define these calculations, GDL offers a large number of mathematical tools: expressions, operators and functions.

EXPRESSIONS

You can write compound expressions in GDL statements. Expressions can be of numerical and string type. They are constants, variables, parameters or function calls and any combination of these in operators. Round bracket pairs (()) (precedence 1) are used to override the default precedence of the operators.

Simple type variables can be given numerical and string values, even in the same script, and can be used in numerical and string type expressions respectively. Operations resulting in strings CANNOT be used directly as macro names in macro calls, or as attribute names in material, fill, line type or style definitions. Variables given a string value will be treated as such and can be used wherever string values are required. If later in the script the same variable is given a numerical value, it will be usable in numerical expressions only until it is given a string value again. Where possible, in the precompilation process the type of the expressions is checked.

GDL supports one and two dimensional arrays. Variables become *arrays* after a declaration statement, in which their dimensions are specified.

DIM

```
DIM var1[dim_1], var2[dim_1][dim_2], var3[ ],  
      var4[ ][ ], var5[dim_1][ ],  
      var5[ ][dim_2]
```

After the DIM keyword there can be any number of variable names separated by commas. var1, var2, ... are the array names, while the numbers between the brackets represent the dimensions of the array (numerical constants). Variable expressions cannot be used as dimensions. If they are missing, the array is declared to be dynamic (one or both dimensions).

Library part parameters can also be arrays. Their actual dimensions are specified in the library part dialog. Parameter arrays do not have to be declared in the script and they are dynamic by default. When referencing the library part using a CALL statement, the actual values of an array parameter can be an array with arbitrary dimensions.

The elements of the arrays can be referenced 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
```

Writing the array name without actual indices means referencing 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! This may cause an unexpected out of memory error in some cases. Each index - even of a possibly wrong, huge value - is considered valid, since the interpreter is unable to detect the error condition. A non-existing dynamic array element is 0 (numerical).

Arrays having 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, group). 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.

VARDIM1(expr)

VARDIM1 (expr)

VARDIM2(expr)

VARDIM2 (expr)

These functions return as integers the actual dimension values for the (array) expression specified as a parameter. They must be used if you want to handle correctly all actual elements of a dynamic array or an array parameter. If no element of a dynamic array was previously set, the return value is 0. For one-dimensional arrays VARDIM2 returns 0.

Examples for numeric expressions:

```
Z
5.5
(+15)
-X
A*(B+C)
SIN(X+Y)*Z
A+R*COS(I*D)
5' 4"
SQR (x^2 + y^2) / (1 - d)
a + b * sin (alpha)
height * width
```

Examples for string expressions:

```
"Constant string"
name + STR ("%m", i) + "." + ext
string_param <> "Mode 1"
```

Examples for expressions using array values:

```
DIM tab [5], tab2 [3][4] ! declaration
tab [1] + tab [2]
tab2 [2][3] + A
PRINT tab

DIM f1 [5], v1[], v2[][]
v1[3] = 3 !v1[1] = 0, v1[2] = 0, array of 3
           ! elements
v2[2][3] = 23 ! all other elements(2 X 3) = 0
PRINT v1, v2

DIM f1 [5], v1[], v2[][]
FOR i = 1 TO VARDIM1(f1)
    f1 [i] = i
NEXT I
v1 = f1
v2 [1] = f1
PRINT v1, v2
```

OPERATORS

The operators below are listed in order of decreasing precedence. The evaluation of an expression begins with the highest precedence operator and from left to right.

Arithmetical Operators

<code>^</code> (or <code>**</code>)	Power of precedence 2
<code>*</code>	Multiplication precedence 3
<code>/</code>	Division precedence 3
MOD (or <code>%</code>)	Modulo (remaining part) precedence 3 $X \text{ MOD } Y = X - Y * \text{INT} (X/Y)$
<code>+</code>	Addition precedence 4
<code>-</code>	Subtraction precedence 4

Note: `+` (addition) can also be applied to string expressions: the result is the concatenation of the strings.

The result of the `'/'` (Division) is always a real number, while the result of the other operations depends on the type of the operands: if all operands are integer, the result will be integer, otherwise real.

Relational Operators

<code>=</code>	Equal precedence 5
<code><</code>	Less than precedence 5
<code>></code>	Greater than precedence 5
<code><=</code>	Less than or equal precedence 5
<code>>=</code>	Greater than or equal precedence 5
<code><></code> (or <code>#</code>)	Not equal precedence 5

Note: These operators can be used between any two string expressions also (string comparison is case sensitive). The result is an integer, 1 or 0. There is not recommended to use the `'='` (Equal), `'<='` (Less than or equal), `'>='` (Greater than or equal), `'<>'` (or `#`) (Not equal) operators with real operands, as these operations can result in precision problems.

Boolean Operators

AND (or <code>&</code>)	Logical and precedence 6
OR (or <code> </code>)	Logical inclusive or precedence 7
EXOR (or <code>@</code>)	Logical exclusive or precedence 8

Note: Boolean operators work with integer numbers. In consequence, 0 means “false”, while any other number means “true”. The value of a logical expression is also integer, i.e., 1 for “true” and 0 for “false”. There is not recommended to use boolean operators with real operands, as these operations can result in precision problems.

FUNCTIONS

Arithmetical Functions

ABS

ABS (x) Returns the absolute value of x (integer if x integer, real otherwise).

CEIL

CEIL (x) Returns the smallest integral value that is not smaller than x (always integer).
(e.g., `CEIL(1.23) = 2`; `CEIL(-1.9) = -1`).

INT

INT (x) Returns the integral part of x (always integer). (e.g., `INT(1.23) = 1`, `INT(-1.23) = -2`).

FRA

FRA (x) Returns the fractional part of x (integer 0 if x integer, real otherwise). (e.g.,
`FRA(1.23) = 0.23`, `FRA(-1.23) = 0.77`).

ROUND_INT

ROUND_INT (x) Returns the rounded integer part of x. 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)`

SGN

SGN (x) Returns +1 integer if x positive, -1 integer if x negative, otherwise 0 integer.

SQR

SQR (x) Returns the square root of x (always real).

Circular Functions

These functions use degrees for arguments (COS, SIN, TAN) and for return values (ACS, ASN, ATN).

ACS

ACS (x) Returns the arc cosine of x. ($-1.0 \leq x \leq 1.0$; $0^\circ \leq \text{ACS}(x) \leq 180^\circ$).

ASN

ASN (x) Returns the arc sine of x. ($-1.0 \leq x \leq 1.0$; $-90^\circ \leq \text{ASN}(x) \leq 90^\circ$).

ATN

ATN (x) Returns the arc tangent of x. ($-90^\circ \leq \text{ATN}(x) \leq 90^\circ$).

COS

COS (x) Returns the cosine of x.

SIN

SIN (x) Returns the sine of x.

TAN

TAN (x) Returns the tangent of x.

PI

PI Returns Ludolph's constant. ($p = 3.1415926\dots$).

Note: All return values are *real*.

Transcendental Functions

EXP

EXP (x) Returns the x^{th} power of e ($e = 2.7182818$).

LGT

LGT (x) Returns the base 10 logarithm of x.

LOG

LOG (x) Returns the natural logarithm of x.

Note: All returned values are *real*.

Boolean Functions

NOT

NOT (x) Returns false (=0 integer) if x is true (<>0), and true (=1 integer) if x is false (=0) (logical negation).

Note: Parameter value should be *integer*.

Statistical Functions

MIN

MIN (x1,x2, ... xn) Returns the smallest of an unlimited number of arguments.

MAX

MAX (x1,x2, ... xn) Returns the largest of an unlimited number of arguments.

RND

RND (x) Returns a random value between 0.0 and x (x > 0.0) always *real*.

Bit functions

BITTEST

BITTEST (x, b)

Returns 1 if the b bit of x is set, 0 otherwise.

BITSET

BITSET (x, b [, expr])

expr can be 0 or different, the default value is 1. Sets the b bit of x to 1 or 0 depending on the value of the specified expression, and returns the result. Parameter value should be *integer*, returned value is *integer*.

Special Functions

Special functions (besides global variables) can be used in the script to communicate with the program. They either ask the current state and different preferences settings of the program, or refer to the current environment of the library part. Request calls can also be used to communicate with GDL extensions.

There are two types of special functions: requests and the IND function:

REQ (parameter_string)

REQUEST (question_name, name | index, variable1 [, variable2,...])

IND (MATERIAL, name_string)

IND (FILL, name_string)

IND (LINE_TYPE, name_string)

IND (STYLE, name_string)

IND (TEXTURE, name_string)

The return value of the requests is always the number of successfully retrieved values (integer), while the type of the retrieved values is defined by each request in part.

IND functions return an attribute index (integer) value.

For more details, see IND function description in the “Miscellaneous” > “Requests” on page 233.

String Functions

STR

STR (numeric_expression, length, fractions)

STR

STR (format_string, numeric_expression)

STR{2}

STR{2} (format_string, numeric_expression [, extra_accuracy_string])

The first form of the function creates a string from the current value of the numeric expression. The minimum number for numerical characters in the string is length, while fractions represents the numbers following the floating point. If the converted value has more than length characters, it is expanded as required. If it has fewer characters, it is padded on the left (length > 0) or on the right (length < 0).

Example:

```
A=4.5
B=2.345
TEXT2 0, 2, STR(A, 8, 2) ! 4.50
TEXT2 0, 1, STR(B, 8, 2) ! 2.34
TEXT2 0, 0, STR(A*B, 8, 2)! 10.55
```

In the second and third case, the `format_string` can either be a variable or a constant. If the format is empty, it is interpreted as meters, with an accuracy of three decimals (display 0 wholes). If the extra accuracy flags are set in the `format_string`, then the `STR{2}` function will return the corresponding extra accuracy string in the 3rd parameter.

The format_string can be as seen below:

```
%[0 or more flags][field_width][.precision] conv_spec
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 justify (default)
-        left justify
+        explicit plus sign
(space)  in place of a + sign
'*' 0    extra accuracy Off (default)
'*' 1    extra accuracy .5
'*' 2    extra accuracy .25
'*' 3    extra accuracy .1
'*' 4    extra accuracy .01
'*' 5    extra 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):

```
'0'      display 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)
```

`field_width`: unsigned decimal integer, the minimum number of characters to generate

`precision`: unsigned decimal integer, the number of fraction digits to generate

conv_spec (conversion specifier):

e: exponential format (meter)

m: meters

mm: millimeters

cm: centimeters

ffi: feet & fractional inches

fdi: feet & 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

Examples:

```

h = 23
nr = 0.345678
TEXT2 0, h, STR ("%m", nr) !0.346
TEXT2 0, h-1, STR ("%#10.2m", nr) !35
TEXT2 0, h-2, STR ("% .4cm", nr) !34.5678
TEXT2 0, h-3, STR ("%12.4cm", nr)!34.5678
TEXT2 0, h-4, STR ("% .6mm", nr)!345.678000
TEXT2 0, h-5, STR ("% +15e", nr)!+3.456780e-01
TEXT2 0, h-6, STR ("% ffi", nr) !1'-2"
TEXT2 0, h-7, STR ("%0.16 ffi", nr) !1'-1 5/8"
TEXT2 0, h-8, STR ("% .3fdi", nr) ! 1'-1.609"
TEXT2 0, h-9, STR ("% -10.4df", nr) ! 1.1341'
TEXT2 0, h-10, STR ("%0.64fi", nr) !13 39/64"
TEXT2 0, h-11, STR ("% +12.4di", nr)!+13.6094"
TEXT2 0, h-12, STR ("%# .3sqm", nr) ! 346
TEXT2 0, h-13, STR ("% +sqcm", nr) !+3,456.78
TEXT2 0, h-14, STR ("% .2sqmm", nr)! 345,678.00
TEXT2 0, h-15, STR ("% -12sqf", nr) !3.72
TEXT2 0, h-16, STR ("%10sqi", nr)! 535.80

```

```

alpha = 88.657
TEXT2 0, h-17, STR ("% +10.3dd", alpha)!+88.657°
TEXT2 0, h-18, STR ("% .1dms", alpha)!88°39'
TEXT2 0, h-19, STR ("% .2dms", alpha) !88°39'25"
TEXT2 0, h-20, STR ("%10.4gr", alpha) ! 98.5078G
TEXT2 0, h-21, STR ("%rad", alpha) !1.55R
TEXT2 0, h-22, STR ("% .2surv", alpha) !N 1°20'35" E

```

SPLIT

SPLIT (string, format, variable1 [, variable2, ..., variablen])

Splits the string parameter according to the format in one or more numeric or string parts. The split process stops when the first non-matching part is encountered. Returns the number of successfully read values (*integer*).

string: the string to be split

format: any combination of constant strings, %s and %n -s. Parts in the string must fit the constant strings, %s denotes any string value delimited by spaces or tabs, while %n denotes any numeric value.

variable_i: the variable names to store the split string parts.

Example:

```
ss = "3 pieces 2x5 beam"
n = SPLIT (ss, "%n pieces %nx%n %s", num, ss1, size1, ss2, size2, name)
IF n = 6 THEN
    PRINT num, ss1, size1, ss2, size2, name      ! 3 pieces 2 x 5 beam
ELSE
    PRINT "ERROR"
ENDIF
```

STW

STW (string_expression)

Returns the (*real*) width of the string in meters displayed in the current style. The width in meters, at current scale, is $STW(\text{string_expression}) / 1000 * A_$.

Example:



```
DEFINE STYLE "own" "Monaco", 180000 / A_, 0, 0
SET STYLE "own"
string = "abcd"
width = STW (string) / 1000 * A_
n = REQUEST ("Height_of_style", "own", height)
height = height / 1000 * A_
text2 0,0, string
rect2 0,0, width, -height
```

STRLEN

STRLEN (string_expression)

Returns the (*integer*) length of the string (the number of characters)

STRSTR

STRSTR (string_expression1, string_expression2)

Returns the (*integer*) position of the first appearance of the second string in the first string. If the first string doesn't contain the second one, the function returns 0.

STRSUB

STRSUB (string_expression, start_position, characters_number)

Returns a substring of the string parameter that begins at the position given by the start_position parameter and its length is characters_number characters.

Example:

```
ss = ""
  n = REQUEST ("Linear_dimension", "", ss)
  unit = ""
IF STRSTR (ss, "m") > 0 THEN unit = "m"
IF STRSTR (ss, "mm") > 0 THEN unit = "mm"
IF STRSTR (ss, "cm") > 0 THEN unit = "cm"
TEXT2 0, 0, STR (ss, a) + " " + unit !1.00 m
string = "Flowers.PICT"
len = STRLEN (string)
n = STRSTR (string, ".")
TEXT2 0, -1, STRSUB (string, 1, n - 1) !Flowers
TEXT2 0, -2, STRSUB (string, len - 4, 5) !.PICT
```

CONTROL STATEMENTS

This chapter reviews the GDL commands available for controlling loops and subroutines in scripts and introduces the concept of buffer manipulation designed to store parameter values for further use. It also explains how to use objects as macro calls and how to display calculated expressions on screen.

FLOW CONTROL STATEMENTS

FOR

FOR variable_name = initial_value **TO** end_value [**STEP** step_value]

First statement of a FOR loop. If the STEP keyword and the step_value are missing, the step is assumed to be 1.

A global variable is not allowed as a loop control variable.

Example:

```
FOR I=1 TO 10 STEP 2
    PRINT I
NEXT I
```

NEXT

NEXT variable_name

Last statement of a FOR loop.

The loop variable varies from the initial_value to the end_value by the step_value increment (or decrement) in each execution of the body of the loop (statements between the FOR and NEXT statements). If the loop variable exceeds the value of the end_value, the program executes the statement following the NEXT statement.

Note: Changing the step_value during the execution of the loop has no effect.

The two program fragments below are equivalent:

```
! 1st
  A = B
1: IF C > 0 AND A > D OR C < 0 AND A < D THEN 2
  PRINT A
  A = A + C
  GOTO 1
2:
! 2nd
  FOR A = B TO D STEP C
    PRINT A
  NEXT A
```

The above example shows that `step_value = 0` causes an infinite loop.

Only one `NEXT` statement is allowed after a `FOR` statement. You can exit the loop with a `GOTO` (or `IF ... GOTO`) statement and to return after leaving, but you cannot enter a loop skipping the `FOR` statement.

DO

DO

```
[statement1
statement2
...
statementn]
```

WHILE condition

The statements between the keywords are executed as long as the condition is true.

The condition is checked after each execution of the statements.

WHILE condition DO

```
[statement1
statement2
...
statementn]
```

ENDWHILE

The statements between the keywords are executed as long as the condition is true.

The condition is checked before each execution of the statements.

REPEAT

```

    [statement1
    statement2
    ...
    statementn]

```

UNTIL condition

The statements between the keywords are executed until the condition becomes true.

The condition is checked after each execution of the statements.

Example:

The following four sequences of GDL commands are equivalent:

```

! 1st
FOR i = 1 TO 5 STEP 1
    BRICK 0.5, 0.5, 0.1
    ADDZ 0.3
NEXT i
! 2nd
i = 1
DO
    BRICK 0.5, 0.5, 0.1
    ADDZ 0.3
    i = i + 1
WHILE i <= 5
! 3rd
i = 1
WHILE i <= 5 DO
    BRICK 0.5, 0.5, 0.1
    ADDZ 0.3
    i = i + 1
ENDWHILE
! 4th
i = 1
REPEAT
    BRICK 0.5, 0.5, 0.1
    ADDZ 0.3
    i = i + 1
UNTIL i > 5

```

IF

IF condition **THEN** label

IF condition **GOTO** label

IF condition **GOSUB** label

Conditional jump statement. If the value of the condition expression is 0, the command has no effect, otherwise execution continues at the label.

Examples:

```
IF A THEN 28
IF I > J GOTO 200+I*J
IF I > 0 GOSUB 9000
IF condition THEN statement [ELSE statement]
or
IF condition THEN [statement1
statement2
...
statementn]
[ELSE
statementn+1
statementn2
...
statementn+m]
ENDIF
```

If you write only one command after keywords **THEN** and/or **ELSE** in the same row, there is no need for **ENDIF**. A command after **THEN** or **ELSE** in the same row means a definite **ENDIF**.

If there is a new row after **THEN**, the successive commands (all of them until the keyword **ELSE** or **ENDIF**) will only be executed if the expression in the condition is true (other than zero). Otherwise, the commands following **ELSE** will be carried out. If the **ELSE** keyword is absent, the commands after **ENDIF** will be carried out.

Example:

```
IF a = b THEN height = 5 ELSE height = 7
IF needdoors THEN
    CALL "door_macro" PARAMETERS
    ADDX a
ENDIF
IF simple THEN
    HOTSPOT2 0, 0
    RECT2 a, 0, 0, b
ELSE PROJECT2 3, 270, 1
IF name = "Sphere" THEN
    ADDY b
    SPHERE 1
ELSE
    ROTX 90
    TEXT 0.002, 0, name
ENDIF
```

GOTO

GOTO label

Unconditional jump statement. The program executes a branch to the statement denoted by the value of the label.

Example:

```
GOTO K+2
```

GOSUB

GOSUB label

Internal subroutine call where the label is the entry point of the subroutine.

RETURN

RETURN

Return from an internal subroutine.

END / EXIT

END

EXIT

End of the current GDL script. The program terminates or returns to the level above. It is possible to use several ENDS or EXITS in a GDL file.

BREAKPOINT

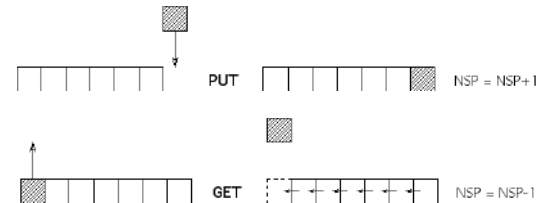
BREAKPOINT expression

With this command, you can specify a breakpoint in the GDL script. The GDL debugger will stop at this command if the value of the parameter (a numeric expression) is true (1) and the Enable Breakpoints option of the debugger is checked. In “normal” execution mode, the GDL interpreter simply steps over this command.

PARAMETER BUFFER MANIPULATION

The parameter buffer is a built-in data structure that may be used if some values (coordinates, for example) change after a definite rule that can be described using a mathematical expression. This is useful if, for instance, you want to store the current values of your variables.

The parameter buffer is an infinitely long array in which you can store numeric values using the PUT command. The PUT command stores the given values at the end of the buffer. These values can later be used (by the GET and USE commands) in the order in which they were entered (i. e., the first stored value will be the first one used). A GET(n) or USE(n) command is equivalent with n values separated by commas. This way, they can be used in any GDL parameter list where n values are needed.



PUT

PUT expression [, expression, ...]

Store the given values in the given order in the internal parameter buffer.

GET

GET (n)

Use the next n values from the internal parameter buffer and then disregard them.

USE

USE (n)

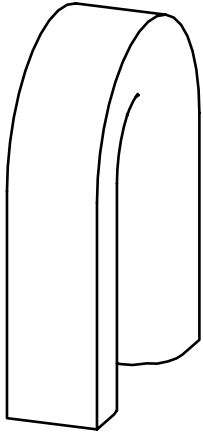
Use the next n values from the internal parameter buffer without deleting them. Following USE and GET functions can use the same parameter sequence.

NSP

NSP

Returns the number of stored parameters in the internal buffer.

Example for using the parameter buffer:



```
R=2: B=6: C=4: D=10
N=12
```

```
S=180/N
FOR T=0 TO 180 STEP S
  PUT R+R*COS(T), C-R*SIN(T), 1
NEXT T
```

```
FOR I=1 TO 2
  EXTRUDE 3+NSP/3, 0,0,D, 1+16,
    0, B, 0,
    2*R, B, 0,
    USE(NSP),
    0, B, 0
  MULY -1
NEXT I
DEL 1
ADDZ D
REVOLVE 3+NSP/3, 180, 0,
  0, B, 0,
  2*R, B, 0,
  GET(NSP),
  0, B, 0
```


The full description:

R=2: B=6: C=4: D=10

```

FOR I=1 TO 2
  EXTRUDE 16, 0,0,D, 1+16,
    0, B, 0,
    2*R, B, 0,
    2*R, C, 1,
    R+R*COS(15), C-R*SIN(15), 1,
    R+R*COS(30), C-R*SIN(30), 1,
    R+R*COS(45), C-R*SIN(45), 1,
    R+R*COS(60), C-R*SIN(50), 1,
    R+R*COS(75), C-R*SIN(75), 1,
    R+R*COS(90), C-R*SIN(90), 1,
    R+R*COS(105), C-R*SIN(105), 1,
    R+R*COS(120), C-R*SIN(120), 1,
    R+R*COS(135), C-R*SIN(135), 1,
    R+R*COS(150), C-R*SIN(150), 1,
    R+R*COS(165), C-R*SIN(165), 1,
    0, B, 1,
    0, B, 0
  MULLY -1
NEXT I
DEL 1
ADDZ D
REVOLVE 16, 180, 0,
  0, B, 0,
  2*R, B, 0,
  2*R, C, 1,
  R+R*COS(15), C-R*SIN(15), 1,
  R+R*COS(30), C-R*SIN(30), 1,
  R+R*COS(45), C-R*SIN(45), 1,
  R+R*COS(60), C-R*SIN(50), 1,
  R+R*COS(75), C-R*SIN(75), 1,
  R+R*COS(90), C-R*SIN(90), 1,
  R+R*COS(105), C-R*SIN(105), 1,
  R+R*COS(120), C-R*SIN(120), 1,
  R+R*COS(135), C-R*SIN(135), 1,
  R+R*COS(150), C-R*SIN(150), 1,
  R+R*COS(165), C-R*SIN(165), 1,
  0, B, 1,
  0, B, 0

```

MACRO OBJECTS

Although the 3D objects you may need can always be broken down into complex or primitive elements, sometimes it is desirable to define these complex elements specifically for certain applications. These individually defined elements are called macros.

CALL

CALL macro_name_string [,parameter_list]

CALL macro_name_string **PARAMETERS** [name1=value1 , ... namen=valuen]

CALL macro_name **PARAMETERS ALL**

Macro names cannot be longer than 31 characters.

Macro names can be string constants, string variables or parameters. String operations cannot be used with a macro call as a macro name.

Warning: If string variables or parameters are used as macro names, the called macro may not be included in the archive project, unless the “Include All Parts of Loaded Libraries” option is checked.

The macro name must be put between quotation marks (“;”;“), unless it matches the definition of identifiers, i.e., it begins with a letter or a ‘_’ or ‘~’ character and contains only letters, numbers and the ‘_’ and ‘~’ characters. 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 macro name.

A macro name itself also can be used as a command, without the CALL keyword:

macro_name [parameter_list]

macro_name **PARAMETERS** [name1=value1, ... namen=valuen]

macro_name **PARAMETERS ALL**

The first type of macro call can be used with simple GDL text files as well as any library part, on the condition that its parameter list contains only single-letter parameters (A ... Z). This form of macro call can be used for compatibility with previous versions, but we recommend the second type. The meaning of the parameter list is the following: the value of parameter A will be the first value in the list, the value of parameter B will be the second value, and so on. If the (library part) macro does not have a single-letter parameter corresponding to the value, interpretation will continue by skipping this value, but you will get a warning from the program. No string type expressions are allowed with this method.

The second type can be used with full featured library parts or plain GDL text files. After the PARAMETERS keyword you need to list the parameter names of the called macro 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 macro. Array parameters have to be given full array values. If a parameter name in the parameter list cannot be found in the called macro, you will get an error message. Parameters of the called macro that are not listed in the macro call will be given their original default values as defined in the library part called as a macro.

The third type can only be used with full featured library parts. In this case there is no need to specify the parameters one by one; all parameters of the caller are passed to the called macro. For a parameter which cannot be found in the called macro, the default value will be used.

A GDL macro has its own environment which depends on its calling order. The current values of the

MODEL, RADIUS, RESOL, TOLER, PEN, LINE_TYPE, MATERIAL, FILL, STYLE, SHADOW

options and the current transformation are all valid in the macro. You can use or modify them, but the modifications will only have an effect locally. They do not take effect on the level the macro was called from.

Giving parameters to a macro call means an implicit value assignment on the macro's level.

The parameters A and B are generally used for resizing objects.

Examples:

```
CALL "leg" 2, , 5 ! A = 2, B = 0, C = 5
leg 2, , 5
CALL "door-1" PARAMETERS height = 2, a = 25.5,
    name = "Director"
CALL "door-1" PARAMETERS
    ! use parameter default values
"door-1" PARAMETERS
```

In summary: whenever you do not need a parameter with a long name or of string type, using the text type GDL can be sufficient. This type of GDL can only be called with the first type of macro call, since it does not have a parameter list. On the other hand, if you do not wish to limit your macro parameter names to letters from A to Z, or if you want to include strings in the parameter list, your macro must be a library part and called according to the second type of GDL syntax.

THE OUTPUT STATEMENT

PRINT

PRINT expression [, expression, ...]

Writes all of its arguments in a dialog box. Arguments can be strings or numeric expressions of any number in any sequence, separated by commas.

Examples:

```
PRINT "loop-variable:", I
PRINT J, K-3*L
PRINT "Beginning of interpretation"
PRINT a * SIN (alpha) + b * COS (alpha)
PRINT "Parameter values: ", "a = ", a, ", b = ", b
PRINT name + STR ("%m", i) + "." + ext
```

FILE OPERATIONS

The following keywords allow you to open external files for reading/writing and to manipulate them by putting/getting values from/to GDL scripts. This process necessarily involves using special Add-On extensions. Text files can be handled by the "TEXT GDL I/O" Add-On. Add-Ons for other file types can be developed by third parties.

See also "GDL Text I/O Add-On" in the "Miscellaneous".

OPEN

OPEN (filter, filename, parameter_string)

filter: string, the name of an existing extension

filename: string, the name of the file

parameter_string: string, it contains the specific separation characters of the operational extension and the mode of opening. Its contents are interpreted by the extension.

Opens a file as directed. Its return value is a positive integer that will identify the specific file. This value, the channel number, will be the file's reference number in succeeding instances. To include the referenced file in the archive project, use the `"FILE_DEPENDENCE "name1" [, "name2", ...]"` command with the file name.

INPUT

INPUT (channel, recordID, fieldID, variable1 [, variable2,...])

recordID, fieldID: the string or numeric type starting position of the reading, its contents are interpreted by the extension

The number of given parameters defines the number of values from the starting position read from the file identified by the channel value. The parameter list must contain at least one value. This function puts the read values into the parameters as ordered. These values can be of numeric or string type, independent of the parameter type defined for storage.

The return value is the number of the successfully read values. When encountering an end of file character, -1 is returned.

VARTYPE

VARTYPE (expression)

Returns 1 if the type of the expression is numerical, 2 if it is a string.

Useful when reading values in variables with the INPUT command, which can change the type of the variables according to the current values.

The type of these variables is not checked during the compilation process.

OUTPUT

OUTPUT channel, recordID, fieldID, expression1 [, expression2, ...]

recordID, fieldID: the string or numeric type starting position of the writing; its contents are interpreted by the extension.

Writes as many values into the file identified by the channel value from the given position as there are defined expressions. There has to be at least one expression. The type of values is the same as those of the expressions.

CLOSE

CLOSE channel

Closes the file identified by the channel value.

MISCELLANEOUS

GDL can also handle a number of operations on external files through special Add-On applications. The commands used to achieve this are described in this chapter and illustrated with an example.

GLOBAL VARIABLES

The global variables make it possible to store special values of the model. This allows you to access geometric information about the environment of the GDL macro. For example, you can access the wall parameters when defining a window which has to fit into the wall. Global variables are not stacked during macro calls.

General environment information

GLOBAL_SCRIPT_TYPE <i>1-property script, 2-2D script, 3-3D script, 4-not implemented, 5-parameters script, 6-master script</i>	T~	type of current script
GLOBAL_CONTEXT <i>1-library part editor, 2-floor plan, 3-3D view, 4-section/ elevation, 5-settings dialog, 6-list, 7 - detail drawing, 22 - feedback mode from the floor plan, 23 - feedback mode from a 3Dview, 24 - feedback mode from a section/ elevation, 43 - generating as an operator from a 3D view, 44 - generating as an operator from a section/ elevation, 46 - generating as an operator from a list</i>		context of appearance
GLOBAL_SCALE <i>according to the current window</i>	A_	drawing scale
GLOBAL_NORTH_DIR <i>relative to the default project coordinate system according to the settings made in the Sun... dialog</i>	U~	project North direction
GLOBAL_DRAWING_BGD_PEN <i>the best matching pen from the current palette to the background color of the current window</i>		pen of the drawing background color
GLOBAL_MODPAR_NAME <i>in the settings dialog or library part editor, including parameters modified through editable hotspots. Valid only in parameter scripts.</i>		name of the last modified parameter
GLOBAL_WORLD_ORIGO_OFFSET_X, GLOBAL_WORLD_ORIGO_OFFSET_Y <i>Position of the project origin relative to the world origin. See “Example illustrating the usage of the GLOBAL_WORLD_ORIGO_... globals:” on page 231.</i>		

Story information

GLOBAL_HSTORY_ELEV <i>home story is the one the object is placed on</i>	B_	elevation of the home story
GLOBAL_HSTORY_HEIGHT <i>home story is the one the object is placed on</i>	Q_	height of the home story
GLOBAL_CSTORY_ELEV <i>current story is the one currently shown in the Floor Plan window</i>	Q~	elevation of the current story
GLOBAL_CSTORY_HEIGHT <i>current story is the one currently shown in the Floor Plan window</i>	R~	height of the current story
GLOBAL_CH_STORY_DIST <i>current story is the one currently shown in the Floor Plan window</i>	S~	relative position of the current story to the home story

Fly-through information

GLOB_FRAME_NR	N_	current frame number in animation
<i>valid only for animation, 0 for still images</i>		
GLOB_FIRST_FRAME	O_	first frame index in fly-through
<i>valid only for animation, 0 for still images</i>		
GLOB_LAST_FRAME	P_	last frame index in fly-through
<i>valid only for animation, 0 for still images</i>		
GLOB_EYEPOS_X	K~	current camera position (x)
<i>valid only in perspective projection for both animation and still images</i>		
GLOB_EYEPOS_Y	L~	current camera position (y)
<i>valid only in perspective projection for both animation and still images</i>		
GLOB_EYEPOS_Z	M~	current camera position (z)
<i>valid only in perspective projection for both animation and still images</i>		
GLOB_TARGPOS_X	N~	current target position (x)
<i>valid only in perspective projection for both animation and still images</i>		
GLOB_TARGPOS_Y	O~	current target position (y)
<i>valid only in perspective projection for both animation and still images</i>		
GLOB_TARGPOS_Z	P~	current target position (z)
<i>valid only in perspective projection for both animation and still images</i>		
GLOB_SUN_AZIMUTH		sun azimuth
<i>according to the settings in the Sun... dialog box</i>		
GLOB_SUN_ALTITUDE		sun altitude
<i>according to the settings in the Sun... dialog box</i>		

General element parameters

GLOB_LAYER		layer of the element
	<i>name of the layer the element is assigned to</i>	
GLOB_ID		user ID of the element
	<i>ID as set in the settings dialog box</i>	
GLOB_INTID		internal ID of the element
	<i>the internal unique ID generated by the program (cannot be controlled by the user)</i>	
GLOB_ELEVATION	J_	base elevation of the element
	<i>relative to the home story (excluding door, window: sill height, according to current settings)</i>	
GLOB_ELEM_TYPE		element type, for labels and property objects contains the type of the parent element
	<i>0 - none (individual label), 1-object, 2-lamp, 3-window, 4-door, 5-wall, 6-column, 7-slab, 8-roof, 9-fill, 10-mesh, 11-none, 12 - beam</i>	

Object, Lamp, Door, Window parameters

SYMB_LINETYPE		line type of the library part
	<i>applied as the default line type of the 2D symbol</i>	
SYMB_FILL		fill type of the library part
	<i>applied on cut surfaces of library parts in section/ elevation windows</i>	
SYMB_FILL_PEN		pen of the fill of the library part
	<i>applied on cut surfaces of library parts in section/ elevation windows</i>	
SYMB_FBGD_PEN		pen of the background of the fill of the library part
	<i>applied on cut surfaces of library parts in section/ elevation windows</i>	
SYMB_SECT_PEN		pen of the library part in section
	<i>applied on contours of cut surfaces of library parts in section/ elevation windows</i>	
SYMB_VIEW_PEN	L_	default pen of the library part
	<i>applied on all edges in 3D window and on edges on view in section/ elevation windows</i>	
SYMB_MAT	M_	default material of the library part
SYMB_POS_X	X~	position of the library part (x)
	<i>relative to the project origin (excluding door, window and wall end: relative to the startpoint of the including wall)</i>	
SYMB_POS_Y	Y~	position of the library part (y)
	<i>relative to the project origin (excluding door, window and wall end: relative to the startpoint of the including wall) Note: see "Doors and Windows" on page 242 specials for orientation of Y and Z axes</i>	
SYMB_POS_Z	Z~	position of the library part (z)
	<i>relative to the project origin (excluding door, window and wall end: relative to the startpoint of the including wall) Note: see "Doors and Windows" on page 242 specials for orientation of Y and Z axes</i>	

Object, Lamp parameters

SYMB_ROTANGLE	W~	rotation angle of the library part
<i>numeric rotation from within the settings dialog is performed around the current anchor point</i>		
SYMB_MIRRORED	V~	library part mirrored
<i>0-no , 1-yes (mirroring is performed around the current anchor point.) Always 0 for wall ends, except when the origin of the local coordinate system is in a non-rectangular vertex of a trapezoidal wall's polygon.</i>		

Object, Lamp, Door, Window parameters - available for listing and labels only

SYMB_A_SIZE	nominal length/width of library part
<i>length of object/ lamp, width of window/ door (fixed parameter)</i>	
SYMB_B_SIZE	nominal width/height of library parts
<i>width of object/ lamp, height of window/ door (fixed parameter)</i>	

Object, Lamp parameters - available for listing and labels only

SYMB_Z_SIZE	nominal height of the library part
<i>if a user parameter is named in $\%x$ format then it will be used for nominal height, otherwise 0</i>	

Window, Door and Wall End parameters

WIDO_REVEAL_ON		window/door reveal is on
		<i>0-reveal is off, 1-reveal is on</i>
WIDO_SILL	K_	sill depth of the window/door
		<i>Reveal Depth as set in the Reveal tab page of Window/Door Settings dialog box for curved walls: in radial direction at nominal sized opening corner</i>
WIDO_SILL_HEIGHT		Window/door nominal sill height
WIDO_RSIDE_SILL_HEIGHT		Window/door sill height on the reveal side
WIDO_OPRSIDE_SILL_HEIGHT		Window/door sill height on the side opposite to the reveal side
WIDO_RIGHT_JAMB	B~	window/door jamb on the right side
		<i>as set in the Reveal tab page of Window/Door Settings dialog box</i>
WIDO_LEFT_JAMB		window/door jamb on the left side
		<i>as set in the Reveal tab page of Window/Door Settings dialog box</i>
WIDO_THRES_DEPTH	C~	window/door sill/threshold depth
		<i>as set in the Reveal tab page of Window/Door Settings dialog box</i>
WIDO_HEAD_DEPTH	D~	window/door head depth
		<i>as set in the Reveal tab page of Window/Door Settings dialog box</i>
WIDO_HEAD_HEIGHT		Window/door nominal head height
WIDO_RSIDE_HEAD_HEIGHT		Window/door head height on the reveal side
WIDO_OPRSIDE_HEAD_HEIGHT		Window/door head height on the side opposite to the reveal side
WIDO_REVEAL_SIDE	E~	reveal side is opposite to the opening side
		<i>1=yes, 0=no - when placing an element, the default value is 0 for windows, 1 for doors</i>
WIDO_FRAME_THICKNESS	F~	frame thickness of window/door
		<i>when flipping doors/windows, they will be mirrored then relocated automatically by this value</i>
WIDO_POSITION	H~	offset of the door/window
		<i>angle or distance between the axis of the opening or wall end and the normal vector at the wall's starting point</i>
WIDO_ORIENTATION		window/door opening orientation
		<i>left/right - it will work fine only if the door/window was created according to local standards</i>
WIDO_MARKER_TXT		window/door marker text
		<i>as set in the Window/Door Dimensioning subdialog from within the Door/Window Settings dialog</i>
WIDO_SUBFL_THICKNESS		subfloor thickness (sill correction)
		<i>as set in the Parameters tab page of the Window/Door Settings dialog box</i>
WIDO_PREFIX		window/door sill height prefix
		<i>as set in the Window/Door Dimensioning subdialog from within the Door/Window Settings dialog</i>
WIDO_CUSTOM_MARKER		window/door custom marker switch
		<i>1-parameters can be used in the 2D script while the automatic dimension is not present</i>
WIDO_ORIG_DIST	R_	distance of the local origin from the center of curvature of the wall
		<i>distance of the local origin from the centerpoint of the curved wall, 0 for straight walls. Negative for wall ends 0 at the ending point of the curved wall.</i>
WIDO_PWALL_INSET		parapet wall inset

Window, Door parameters - available for listing and labels only

WIDO_RSIDE_WIDTH	Window/door opening width on the reveal side
WIDO_OPRSIDE_WIDTH	Window/door opening width on the side opposite to the reveal side
WIDO_RSIDE_HEIGHT	Window/door opening height on the reveal side
WIDO_OPRSIDE_HEIGHT	Window/door opening height on the side opposite to the reveal side
WIDO_RSIDE_SURF	Window/door opening surface on the reveal side
WIDO_OPRSIDE_SURF	Window/door opening surface on the side opposite to the reveal side
WIDO_N_RSIDE_WIDTH	Nominal window/door opening width on the reveal side
WIDO_N_OPRSIDE_WIDTH	Nominal window/door opening width on the side opposite to the reveal side
WIDO_N_RSIDE_HEIGHT	Nominal window/door opening height on the reveal side
WIDO_N_OPRSIDE_HEIGHT	Nominal window/door opening height on the side opposite to the reveal side
WIDO_N_RSIDE_SURF	Nominal window/door opening surface on the side opposite to the reveal side
WIDO_N_OPRSIDE_SURF	Nominal window/door opening surface on the side opposite to the reveal side
WIDO_VOLUME	Window/door opening volume
WIDO_GROSS_SURFACE	Window/door opening nominal surface
WIDO_GROSS_VOLUME	Window/door opening nominal volume

Lamp parameters - available for listing and labels only

LIGHT_ON <i>0-light is off, 1-light is on: as set in the Lamp Settings dialog box (fixed parameter)</i>	light is on
LIGHT_RED <i>as set in the Lamp Settings dialog box (fixed parameter)</i>	red component of the light color
LIGHT_GREEN <i>as set in the Lamp Settings dialog box (fixed parameter)</i>	green component of the light color
LIGHT_BLUE <i>as set in the Lamp Settings dialog box (fixed parameter)</i>	blue component of the light color
LIGHT_INTENSITY <i>as set in the Lamp Settings dialog box (fixed parameter)</i>	light intensity

Label parameters

LABEL_POSITION	position of the label <i>array[3][2] containing the coordinates of the 3 points defining the label position</i>
LABEL_CUSTOM_ARROW	use symbol arrow option on/off <i>1 if the Use symbol arrow checkbox is checked, 0 otherwise</i>
LABEL_ARROW_PEN	pen of the arrow in the settings dialog box
LABEL_ARROWHEAD_PEN	pen of the arrowhead in the settings dialog box
LABEL_FONT_NAME	font name in the settings dialog box
LABEL_TEXT_SIZE	text size in the settings dialog box
LABEL_TEXT_PEN	pen of the text in the settings dialog box
LABEL_FONT_STYLE	font style in the settings dialog box <i>0-normal, 1-bold, 2-italic</i>
LABEL_FRAME_ON	label frame on/off <i>1 if the label frame is checked, 0 otherwise</i>
LABEL_ANCHOR_POS	label anchor position <i>0 - middle, 1 - top, 2 - bottom according to settings dialog box</i>
LABEL_ROTANGLE	Rotation angle in the settings dialog box
LABEL_TEXT_ALIGN	Text alignment in the label settings dialog box <i>1: left aligned, 2: center aligned, 3: right aligned, 4: full justified</i>
LABEL_TEXT_LEADING	Line spacing factor in the label settings dialog box
LABEL_TEXT_WIDTH_FACT	Width factor as set in the label settings dialog box
LABEL_CHARSPACE_FACT	Spacing factor, as set in the label settings dialog box

Wall parameters - available for Doors/Windows

WALL_RESOL <i>effective in 3D only</i>	J~	3D resolution of a curved wall
WALL_THICKNESS <i>in case of inclined walls: the wall thickness at the opening axis (local z axis)</i>	C_	thickness of the wall
WALL_START_THICKNESS		Start thickness of the wall
WALL_END_THICKNESS		End thickness of the wall
WALL_INCL <i>the angle between the two inclined wall surfaces - 0 for common straight walls</i>		inclination of the wall surfaces
WALL_HEIGHT	D_	height of the wall
WALL_MIN_HEIGHT		Minimum height of the wall
WALL_MAX_HEIGHT		Maximum height of the wall
WALL_MAT_A <i>this can vary from opening to opening placed in the same wall</i>	G_	material of the wall on the side opposite to the opening side
WALL_MAT_B <i>this can vary from opening to opening placed in the same wall</i>	H_	material of the wall on the opening side
WALL_MAT_EDGE	I_	material of the edges of the wall
WALL_LINETYPE <i>applied on the contours only in the floor plan window</i>		line type of the wall
WALL_FILL <i>fill index, first skin of a composite structure</i>	A~	fill type of the wall
WALL_FILL_PEN	F_	pen of the wall fill
WALL_COMPS_NAME <i>name of the composite structure, range of 1 to 8, 0 if single fill applied</i>		composite structure of the wall
WALL_SKINS_PARAMS <i>array with 12 columns: fill, thickness, (old contour pen), pen of fill, pen of fill background, core status, upper line pen, lower line pen, upper line type, lower line type, end face pen, fill orientation and up to 8 rows, core status: 0 - not part, 1 - part, 3 - last skin of core, fill orientation: 0 - global, 1 - local</i>		parameters of the composite wall skins
WALL_SECT_PEN <i>applied on contours of cut surfaces of walls both in floor plan and section/ elevation windows</i>	E_	pen of the contours of the wall in section
WALL_VIEW_PEN <i>applied on all edges in 3D window and on visible edges in section/ elevation windows</i>		pen of the contours of the wall on view
WALL_FBGD_PEN		pen of the background of the fill of the wall
WALL_DIRECTION <i>straight walls: the direction of the reference line, curved walls: the direction of the chord of the arc</i>		direction of the wall
WALL_POSITION <i>the position of the wall's starting point relative to the project origin</i>		absolute coordinates of the wall

Wall parameters - available for listing and labels only

WALL_LENGTH_A	length of the wall on the reference line side
WALL_LENGTH_B	length of the wall on the side opposite to the reference line
WALL_CENTER_LENGTH	Length of the wall at the center
WALL_AREA	Area of the wall
WALL_PERIMETER	Perimeter of the wall
WALL_SURFACE_A	surface of the wall on the reference line side
WALL_SURFACE_B	surface of the wall on the side opposite to the reference line
WALL_GROSS_SURFACE_A	Gross surface of the wall on the reference line side
WALL_GROSS_SURFACE_B	Gross surface of the wall on side opposite to the reference line
WALL_EDGE_SURF	surface of the edge of the wall
WALL_VOLUME	volume of the wall
WALL_GROSS_VOLUME	Gross volume of the wall
WALL_DOORS_NR	number of doors in the wall
WALL_WINDS_NR	number of windows in the wall
WALL_HOLES_NR	number of empty openings
WALL_DOORS_SURF	surface of doors in the wall
WALL_WINDS_SURF	surface of windows in the wall
WALL_HOLES_SURF	surface of empty openings in the wall
WALL_HOLES_SURF_A	Analytic surface of openings on the reference line side
WALL_HOLES_VOLUME	Analytic volume of openings in the wall
WALL_WINDS_WID	combined width of the windows in the wall
WALL_DOORS_WID	combined width of the doors in the wall
WALL_COLUMNS_NR	number of columns in the wall

Column parameters - available for listing and labels only

COLU_CORE	core/veneer properties <i>serves compatibility: it is only effective in the property script of .CPS (Column.Properties) files</i>
COLU_HEIGHT	height of the column
COLU_MIN_HEIGHT	Minimum height of the column
COLU_MAX_HEIGHT	Maximum height of the column
COLU_VENEER_WIDTH	thickness of the column veneer
COLU_CORE_X	Width of the core
COLU_CORE_Y	Depth of the core
COLU_DIM1	1st dimension of the column
COLU_DIM2	2nd dimension of the column
COLU_MAT	material of the column
Note: Wall wrapping will replace column material with the materials of the connecting walls	
COLU_LINETYPE	line type of the column <i>applied on the contours only in the floor plan window</i>
COLU_CORE_FILL	fill of the column core
COLU_VENEER_FILL	fill of the column veneer
COLU_SECT_PEN	pen of the contours of the column in section <i>applied on contours of cut surfaces in both floor plan and section/elevation windows</i>
COLU_VIEW_PEN	pen of the column on view <i>applied on all edges in 3D window and on visible edges in section/elevation windows</i>
COLU_CORE_FILL_PEN	pen of the fill of the column core
COLU_CORE_FBGD_PEN	pen of the background of the fill of the column core
COLU_VENEER_FILL_PEN	pen of the fill of the column veneer
COLU_VENEER_FBGD_PEN	pen of the background of the fill of the column veneer
COLU_PERIMETER	Perimeter of the column
COLU_AREA	Area of the column
COLU_VOLUME	Volume of the column
COLU_GROSS_VOLUME	Gross volume of the column
COLU_CORE_SURF	surface of the column core
COLU_CORE_GROSS_SURF	Gross surface of the column
COLU_CORE_VOL	volume of the column core

COLU_CORE_GROSS_VOL	Gross volume of the core
COLU_VENEER_SURF	surface of the column veneer
COLU_VENEER_GROSS_SURF	Gross surface of the veneer
COLU_VENEER_VOL	volume of the column veneer
COLU_VENEER_GROSS_VOL	Gross volume of the veneer
COLU_CORE_TOP_SURF	Surface of the core top
COLU_CORE_BOT_SURF	Surface of the core bottom
COLU_VENEER_TOP_SURF	Surface of the veneer top
COLU_VENEER_BOT_SURF	Surface of the veneer top
COLU_CORE_GROSS_TOPBOT_SURF	Gross surface of the core top and bottom
COLU_VENEER_GROSS_TOPBOT_SURF	Gross surface of the veneer top and bottom

Beam parameters - available for listing and labels only

BEAM_THICKNESS	thickness of the beam
BEAM_HEIGHT	height of the beam
BEAM_REFLINE_OFFSET	offset of the reference line relative to the axes of the beam
BEAM_PRIORITY	3D intersection priority index number
BEAM_MAT_RIGHT	material of the beam on the right side of the reference line
BEAM_MAT_LEFT	material of the beam on the left side of the reference line
BEAM_MAT_TOP	material of the beam on the top
BEAM_MAT_BOTTOM	material of the beam at the bottom
BEAM_MAT_END	material of the beam at both ends
BEAM_OUTLINE_LINETYPE	line type of the beam outline
BEAM_AXES_LINETYPE	line type of the beam axes
BEAM_FILL	fill type of the beam
BEAM_FILL_PEN	pen of the beam fill
BEAM_SECT_PEN	pen of the contours of the beam in section
BEAM_FBGD_PEN	pen of the background of the fill of the beam
BEAM_DIRECTION	the direction of the beam reference line
BEAM_POSITION	absolute coordinates of the beam axis starting point
BEAM_LENGTH_RIGHT	length of the beam on the right side of the reference line
BEAM_LENGTH_LEFT	length of the beam on the left side of the reference line
BEAM_RIGHT_SURF	surface of the beam on the right side of the reference line
BEAM_LEFT_SURF	surface of the beam on the left side of the reference line
BEAM_TOP_SURF	surface of the top of the beam
BEAM_BOTTOM_SURF	surface of the bottom of the beam
BEAM_END_SURF	surface of both ends of the beam
BEAM_VOLUME	volume of the beam
BEAM_HOLES_NR	number of holes in the beam
BEAM_HOLES_SURF	total surface of holes in the beam
BEAM_HOLE_EDGE_SURF	total surface of hole edges in the beam
BEAM_HOLES_VOLUME	total volume of holes in the beam

Slab parameters - available for listing and labels only

SLAB_THICKNESS	thickness of the slab
SLAB_MAT_TOP	material of the top surface of the slab
SLAB_MAT_EDGE	material of the edges of the slab
SLAB_MAT_BOTT	material of the bottom surface of the slab
SLAB_LINETYPE	line type of the slab
SLAB_FILL	fill of the slab
	<i>fill index - its value is negative in case of a composite structure</i>
SLAB_FILL_PEN	pen of the fill of the slab
SLAB_FBGD_PEN	pen of the background of the fill of the slab
SLAB_COMPS_NAME	composite structure of the slab
	<i>name of the composite structure</i>
SLAB_SKINS_NUMBER	number of composite slab skins
	<i>range of 1 to 8, 0 if single fill applied</i>
SLAB_SKINS_PARAMS	parameters of the composite slab skins
	<i>array with 12 columns: fill, thickness, (old contour pen), pen of fill, pen of fill background, core status, upper line pen, lower line pen, upper line type, lower line type, end face pen, fill orientation and up to 8 rows, core status: 0 - not part, 1 - part, 3 - last skin of core, fill orientation: 0 - global, 1 - local</i>
SLAB_SECT_PEN	pen of the contours of the slab in section
	<i>applied on contours of cut surfaces in both floor plan and section/elevation windows</i>
SLAB_VIEW_PEN	pen of the slab
	<i>applied on all edges in 3D window and on visible edges in section/elevation windows</i>
SLAB_TOP_SURF	top surface of the slab
SLAB_GROSS_TOP_SURF	Gross surface of the slab top
SLAB_BOT_SURF	bottom surface of the slab
SLAB_GROSS_BOT_SURF	Gross surface of the slab bottom
SLAB_EDGE_SURF	surface of the edges of the slab
SLAB_GROSS_EDGE_SURF	Gross surface of the slab edges
SLAB_PERIMETER	perimeter of the slab
SLAB_VOLUME	volume of the slab
SLAB_GROSS_VOLUME	Gross volume of the slab
SLAB_SEGMENTS_NR	number of segments of the slab
SLAB_HOLES_NR	number of holes in the slab
SLAB_HOLES_AREA	area of holes in the slab
SLAB_HOLES_PRM	perimeter of holes in the slab

Roof parameters - available for listing and labels only

ROOF_THICKNESS	thickness of the roof
ROOF_ANGLE	slope of the roof
ROOF_MAT_TOP	material of the top surface of the roof
ROOF_MAT_EDGE	material of the edges of the roof
ROOF_MAT_BOTT	material of the bottom surface of the roof
ROOF_LINETYPE	line type of the roof
<i>applied on the contours only in the floor plan window</i>	
ROOF_FILL	fill of the roof
<i>fill index - its value is negative in case of a composite structure</i>	
ROOF_FILL_PEN	pen of the fill of the roof
ROOF_FBGD_PEN	pen of the background of the fill of the roof
ROOF_COMPS_NAME	composite structure of the roof
<i>name of the composite structure</i>	
ROOF_SKINS_NUMBER	number of composite roof skins
<i>range of 1 to 8, 0 if single fill applied</i>	
ROOF_SKINS_PARAMS	parameters of the composite roof skin
<i>array with 12 columns: fill, thickness, (old contour pen), pen of fill, pen of fill background, core status, upper line pen, lower line pen, upper line type, lower line type, end face pen, fill orientation and up to 8 rows, core status: 0 - not part, 1 - part, 3 - last skin of core, fill orientation: 0 - global, 1 - local</i>	
ROOF_SECT_PEN	pen of the contours of the roof in section
<i>applied on contours of cut surfaces of walls both in floor plan and section/elevation windows</i>	
ROOF_VIEW_PEN	pen of the roof on view
<i>applied on all edges in 3D window and on edges on view in section/elevation windows</i>	
ROOF_BOTTOM_SURF	bottom surface of the roof
ROOF_GROSS_BOTTOM_SURF	Gross surface of the roof bottom
ROOF_TOP_SURF	top surface of the roof
ROOF_GROSS_TOP_SURF	Gross surface of the roof top
ROOF_EDGE_SURF	surface of the edge of the roof
ROOF_GROSS_EDGE_SURF	Gross surface of the roof edges
ROOF_PERIMETER	perimeter of the roof
ROOF_VOLUME	volume of the roof
ROOF_GROSS_VOLUME	Gross volume of the roof
ROOF_SEGMENTS_NR	number of segments of the roof
ROOF_HOLES_NR	number of holes in the roof
ROOF_HOLES_AREA	area of holes in the roof
ROOF_HOLES_PRM	perimeter of holes in the roof

Fill parameters - available for listing and labels only

FILL_LINETYPE	line type of the fill
FILL_FILL	fill type of the fill
FILL_FILL_PEN	pen of the fill pattern of the fill
FILL_PEN	pen of the fill
FILL_FBGD_PEN	pen of the background of the fill
FILL_SURF	area of the fill
FILL_PERIMETER	perimeter of the fill
FILL_SEGMENT_NR	number of segments of the fill
FILL_HOLES_NR	number of holes in the fill
FILL_HOLES_PRM	perimeter of holes in the fill
FILL_HOLES_AREA	area of holes in the fill

Mesh parameters - available for listing and labels only

MESH_TYPE	type of the mesh
<i>1 - closed body, 2 - top & edge, 3 - top surface only</i>	
MESH_BASE_OFFSET	offset of the bottom surface to the base level
MESH_USEREDGE_PEN	pen of the user defined ridges of the mesh
MESH_TRIEDGE_PEN	pen of the triangulated edges of the mesh
MESH_SECT_PEN	pen of the contours of the mesh in section
<i>applied on contours of cut surfaces of walls both in floor plan and section/elevation windows</i>	
MESH_VIEW_PEN	pen of the contours on view
<i>applied on all edges in 3D window and on edges on view in section/elevation windows</i>	
MESH_MAT_TOP	material of the top surface of the mesh
MESH_MAT_EDGE	material of the edges of the mesh
MESH_MAT_BOTT	material of the bottom surface of the mesh
MESH_LINETYPE	line type of the mesh
<i>applied on the contours only in the floor plan window</i>	
MESH_FILL	fill type of the mesh
MESH_FILL_PEN	pen of the fill of the mesh
MESH_FBGD_PEN	pen of the background of the fill of the mesh
MESH_BOTTOM_SURF	bottom surface of the mesh
MESH_TOP_SURF	top surface of the mesh
MESH_EDGE_SURF	surface of the edge of the mesh
MESH_PERIMETER	perimeter of the mesh
MESH_VOLUME	volume of the mesh
MESH_SEGMENTS_NR	number of segments of the mesh
MESH_HOLES_NR	number of holes in the mesh
MESH_HOLES_AREA	area of holes in the mesh
MESH_HOLES_PRM	perimeter of holes in the mesh

Free users' globals

GLOB_USER_1	S_	
GLOB_USER_2	T_	
GLOB_USER_3	U_	
GLOB_USER_4	V_	
GLOB_USER_5	W_	
GLOB_USER_6	X_	
GLOB_USER_7	Y_	
GLOB_USER_8	Z_	
GLOB_USER_9	G~	
GLOB_USER_10	I~	free variables 1 to 10 are initalized to number by default
GLOB_USER_11		
GLOB_USER_12		
GLOB_USER_13		
GLOB_USER_14		
GLOB_USER_15		
GLOB_USER_16		
GLOB_USER_17		
GLOB_USER_18		
GLOB_USER_19		
GLOB_USER_20		free variables 11 to 20 are initalized to string by default

Example illustrating the usage of the GLOB_WORLD_ORIGO... globals:

```
GLOB_WORLD_ORIGO... globals:
ADD2 -GLOB_WORLD_ORIGO_OFFSET_X - SYMB_POS_X, -GLOB_WORLD_ORIGO_OFFSET_X -SYMB_POS_Y
LINE2 -0.1, 0.0, 0.1, 0.0
LINE2 0.0, -0.1, 0.0, 0.1
HOTSPOT2 0.0, 0.0, 1
TEXT2 0, 0, "( 0.00 ; 0.00 )"
TEXT2 0, 0.5, "World Origo"
DEL TOP
if ABS(GLOB_WORLD_ORIGO_OFFSET_X) > 0.01 OR ABS(GLOB_WORLD_ORIGO_OFFSET_Y) > 0.01 THEN
  ADD2 = SYMB_POS_X, - SYMB_POS_Y
  LINE2 -0.1, 0.0, 0.1, 0.0
  LINE2 0.0, -0.1, 0.0, 0.1
  HOTSPOT2 0.0, 0.0, 2
  TEXT2 0, 0, "(" + STR (GLOB_WORLD_ORIGO_OFFSET_X, 9, 4) + "; " + STR
(GLOB_WORLD_ORIGO_OFFSET_Y, 9, 4) + " )"
  TEXT2 0, 0.5, "Virtual Origo"
  DEL TOP
ENDIF
if ABS(GLOB_WORLD_ORIGO_OFFSET_X + SYMB_POS_X) > 0.01 OR ABS(GLOB_WORLD_ORIGO_OFFSET_Y +
SYMB_POS_Y) > 0.01 THEN
  LINE2 -0.1, 0.0, 0.1, 0.0
  LINE2 0.0, -0.1, 0.0, 0.1
  HOTSPOT2 0.0, 0.0, 3
  TEXT2 0, 0, "(" + STR (GLOB_WORLD_ORIGO_OFFSET_X + SYMB_POS_X, 9, 4) + "; " + STR
(GLOB_WORLD_ORIGO_OFFSET_Y + SYMB_POS_Y, 9, 4) + " )" TEXT2 0, 0.5, "Object Placement"
ENDIF
```

Old Global Variables

Old global variable names can be used; however, the use of the new names is recommended. Each old global corresponds to a new variable with a long name.

A_	GLOB_SCALE	F~	WIDO_FRAME_THICKNESS
B_	GLOB_HISTORY_ELEV	G~	GLOB_USER_9
C_	WALL_THICKNESS	H~	WIDO_POSITION
D_	WALL_HEIGHT	I~	GLOB_USER_10
E_	WALL_SECT_PEN	J~	WALL_RESOL
F_	WALL_FILL_PEN	K~	GLOB_EYEPOS_X
G_	WALL_MAT_A	L~	GLOB_EYEPOS_Y
H_	WALL_MAT_B	M~	GLOB_EYEPOS_Z
I_	WALL_MAT_EDGE	N~	GLOB_TARGPOS_X
J_	GLOB_ELEVATION	O~	GLOB_TARGPOS_Y
K_	WIDO_SILL	P~	GLOB_TARGPOS_Z
L_	SYMB_VIEW_PEN	Q~	GLOB_CSTORY_ELEV
M_	SYMB_MAT	R~	GLOB_CSTORY_HEIGHT
N_	GLOB_FRAME_NR	S~	GLOB_CH_STORY_DIST
O_	GLOB_FIRST_FRAME	T~	GLOB_SCRIPT_TYPE
P_	GLOB_LAST_FRAME	U~	GLOB_NORTH_DIR
Q_	GLOB_HISTORY_HEIGHT	V~	SYMB_MIRRORED
R_	WIDO_ORIG_DIST	W~	SYMB_ROTANGLE
S_	GLOB_USER_1	X~	SYMB_POS_X
T_	GLOB_USER_2	Y~	SYMB_POS_Y
U_	GLOB_USER_3	Z~	SYMB_POS_Z
V_	GLOB_USER_4		
W_	GLOB_USER_5		
X_	GLOB_USER_6		
Y_	GLOB_USER_7		
Z_	GLOB_USER_8		
A~	WALL_FILL		
B~	WIDO_RIGHT_JAMB		
C~	WIDO_THRES_DEPTH		
D~	WIDO_HEAD_DEPTH		
E~	WIDO_REVEAL_SIDE		

REQUESTS

REQ

REQ (parameter_string)

Asks the current state of the program. Its parameter - the question - is a string. The GDL interpreter answers with a numeric value. If it does not understand the question, the answer is negative.

List of current questions:

"GDL_version"

version number of the GDL compiler/interpreter.

Warning: it is not the same as the ArchiCAD version.

"Program"

code of the program (e.g, 1: ArchiCAD).

"Serial_number"

the serial number of the keyplug.

"Model_size"

size of the current 3D data structure in bytes.

"Red_of_material name"

"Green_of_material name"

"Blue_of_material name"

Defines the given material's color components in RGB values between 0 and 1.

"Red_of_pen index"

"Green_of_pen index"

"Blue_of_pen index"

Defines the given pen's color components in RGB values between 0 and 1.

"Pen_of_RGB r g b"

Defines the index of the pen closest to the given color. The r, g and b constants' values are between 0 and 1.

REQUEST

REQUEST (question_name, name | index, variable1 [, variable2,...])

The first parameter represents the question string while the second represents the object of the question (if it exists) and can be of either string or numeric type (for example, the question can be “Rgb_of_material” and its object the material’s name, or “Rgb_of_pen” and its object the index of the pen). The other parameters are variable names in which the return values (the answers) are stored. The function’s return value is the number of the answer (in the case of a badly formulated question or a nonexisting name, the value will be 0).

REQUEST ("Name_of_program", "", program_name)

Returns in the given variable the name of the program, e. g., “ArchiCAD”, etc.

Example: Printing the name of the program

```
n=REQUEST("Name_of_program", "", program_name)
PRINT program_name
```

REQUEST ("Name_of_macro", "", my_name)

REQUEST ("Name_of_main", "", main_name)

After executing these function calls, the my_name variable will contain the name of the macro, while main_name will contain the name of the main macro (if it doesn’t exist, empty string).

REQUEST ("ID_of_main", "", id_string)

For library parts placed on the floor plan, returns the identifier set in the tool’s settings dialog box in the id_string variable (otherwise empty string).

REQUEST ("Name_of_plan", "", name)

Returns in the given variable the name of the current project.

REQUEST ("Story", "", index, story_name)

Returns in the index and story_name variables the index and the name of the current story.

REQUEST ("Home_story", "", index,
story_name)

Returns in the index and story_name variables the index and the name of the home story.

REQUEST ("Story_info", expr, nStories,
index1, name1, elev1, height1 [,
index2, name2, ...])

Returns the story information in the given variables: number of stories and story index, name, elevation, height to next successively. If expr is a numerical expression, it means a story index: only the number of stories and the information on the specified story is returned. If expr is a string expression, it means that information on all stories is requested. The return value of the function is the number of successfully retrieved values.

Example:

```
DIM t[]
  n = REQUEST ("STORY_INFO", "", nr, t)
  FOR i = 1 TO nr
    nr = STR ("%0m", t [4 * (i - 1) + 1])
    name = t [4 * (i - 1) + 2]
    elevation = STR ("%m", t [4 * (i - 1) + 3])
    height = STR ("%m", t [4 * (i - 1) + 4])
    TEXT2 0, -i, nr + "," + name + "," + elevation + "," + height
  NEXT i
```

REQUEST ("Internal_id", "", id)

Returns in the id variable the internal id of the library part.

REQUEST ("Linear_dimension", "",
format_string)

REQUEST ("Angular_dimension", "",
format_string)

REQUEST ("Angular_length_dimension", ""
format_string)

REQUEST ("Radial_dimension", "",
format_string)

REQUEST ("Level_dimension", "",
format_string)

REQUEST ("Elevation_dimension", "",
format_string)

REQUEST ("Window_door_dimension", "",
format_string)

REQUEST ("Sill_height_dimension", "",
format_string)

REQUEST ("Area_dimension", ""
format_string)

REQUEST ("Calculation_length_unit", "",
format_string)

REQUEST ("Calculation_area_unit", "",
format_string)

```
REQUEST ("Calculation_volume_unit", "",  
         format_string)
```

```
REQUEST ("Calculation_angle_unit", "",  
         format_string)
```

With these requests, you can learn the dimension formats set in the Options/Preferences/Dimensions and Calculation Units dialog boxes. These requests return a format string that can be used as the first parameter in the STR () function.

Example:

```
format = ""  
num = 60.55  
REQUEST ("Angular_dimension", "", format)  
! "%.2dd"  
TEXT2 0, 0, STR (format, num)!60.55
```

```
REQUEST ("Clean_intersections", "", state)
```

Returns the state of the Clean Intersections feature from the Options menu (1 when turned on, 0 when off)

```
REQUEST ("Zone_category", "", name, code)
```

For zones, returns the name and the code string of the current zone category.

```
REQUEST ("Zone_relations", "", category_name, code, name, number [ ,category_name2,  
         code2, name2, number2])
```

Returns in the given variables the zone category name and code and the name and number of the zone where the library part containing this request is located. For doors and windows, there can be a maximum of two zones. The return value of the request is the number of successfully retrieved values (0 if the library part is not inside any zone).

```
REQUEST ("Zone_colus_area", "", area)
```

Returns in the area variable the total area of the columns placed in the current zone. Effective only for Zone Stamps.

```
REQUEST ("Custom_auto_label", "", name)
```

Returns in the name variable the name of the custom auto label of the library part or an empty string if it does not exist.

```
REQUEST ("Rgb_of_material", name, r, g, b)
```

```
REQUEST ("Rgb_of_pen", penindex, r, g, b)
```

```
REQUEST ("Pen_of_RGB", "r g b", penindex)
```

Like the REQ () function (but in just one call), returns in the specified variables the value of the r, g, b components of the material and pen, or the index of the pen corresponding to the given RGB values.

```
REQUEST ("Height_of_style", name,  
         height [, descent, leading])
```

Returns in the given variables 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).

REQUEST ("Style_info", name, fontname [, size, anchor, face_or_slant])

Returns information in the given variables on the previously defined style (see style parameters at the *"DEFINE STYLE"* on page 171. Can be useful in macros to collect information on the style defined in a main script.

REQUEST ("Name_of_material", index, name)

Returns in the variable the material name identified by index.

REQUEST ("Name_of_fill", index, name)

Returns in the name variable the fill name identified by index.

REQUEST ("Name_of_line_type", index, name)

Returns in the given variable the line name identified by index.

REQUEST ("Name_of_style", index, name)

Returns in the given variable the name of the style identified by index.

If index < 0, it refers to a material, fill, line type or style defined in the GDL script or the MASTER_GDL file. A call of a request with index = 0 returns in the variable the name of the default material or line type. (Empty string for fill and style.)

The return value of the request is the number of successfully retrieved values (1 if no error occurred, 0 for error when the index is not valid).

REQUEST ("window_door_show_dim", " ", show)

Returns 1 in the show variable if Options/Display Options/Doors & Windows is set to "Show with Dimensions", 0 otherwise. Can be used to hide/show custom dimensions according to the current Display Options.

REQUEST ("name_of_listed", " ", name)

Returns in the name variable the name of the library part associated with the property type library part containing this request. For elements (Walls, Slabs, etc.), the name is an empty string.

REQUEST ("window_door_zone_relev", " ", out_direction)

Effective only for Doors and Windows. Use it as complement to the "ZONE_RELATIONS" request. Returns 1 in the out_direction variable if the Door/Window opening direction is in that of the first room identified by the "ZONE_RELATIONS" request, 2 if the opening direction is towards the second room. It also returns 2 if there is only one room and the opening direction is to the outside.

REQUEST ("matching_properties", type, name1, name2, ...)

If type = 1, returns in the given variables individually associated property library part names, otherwise property library part names associated by criteria. If used in an associative label, the function returns the properties of the element the label is associated with.

REQUEST (extension_name, parameter_string, variable1, variable2, ...)

If the question isn't one of those listed above, the REQUEST() function will attempt to use it as an extension-specific name. If this extension is in the Add-Ons folder, it will be used to get as many values for as many variable names as are specified. The parameter string is interpreted by the extension.

REQUEST ("Constr_Fills_display", "", optionVal)

Returns in the given variable the value of the Cut Fills Display option as set in the Options > Display Options > Cut Fills (previous Construction Fills). Possible values are:

1 - Empty

2 - No Fills

4 - Solid

5 - Bitmap Pattern

6 - Vectorial Hatching

The function return value is the number of successfully retrieved values, 0 if an error occurred.

REQUEST ("Working_length_unit", "", format_string)

REQUEST ("Working_angle_unit", "", format_string)

With these requests, you can learn the working unit formats as set in the Options > Preferences > Working Units dialog box. They return a format string that can be used as the first parameter in the STR () function. The requests work only when interpreting the parameter or the user interface scripts.

IND (MATERIAL, name_string)

IND (FILL, name_string)

IND (LINE_TYPE, name_string)

IND (STYLE, name_string)

IND (TEXTURE, name_string)

This function returns the current index of the material, fill, line type or style and texture attribute. The main use of the resulting number is to transfer it to a macro that requires the same attribute as the calling macro. The result is negative for temporary definitions and positive for global definitions.

See *"Attributes" on page 118 of ArchiCAD 9 Reference Guide.*

See also *"Inline Attribute Definition" on page 158.*

REQUEST ("ASSOCLP_PARVALUE", expr, name_or_index, type, flags, dim1, dim2, p_values)

Returns information in the given variables on the library part parameter with which the library part containing this request is associated. Can be used in property objects, labels and marker objects.

The function return value is the number of successfully retrieved values, 0 if the specified parameter does not exist or an error occurred.

expr: the request's object, associated library part parameter name or index expression

name_or_index: returns the index or the name of the parameter, depending on the previous expression type (returns index if a parameter name, name if the index is specified)

type: parameter type, possible values

- 1: boolean
- 2: integer
- 3: real number
- 4: string
- 5: length
- 6: angle
- 7: line
- 8: material
- 9: fill
- 10: pen color
- 11: light switch
- 12: rgb color
- 13: light intensity
- 14: separator
- 15: title

flags:

`flags = j1 + 2 * j2 + 64 * j7 + 128 * j8,`
 where j_i can be 0 or 1

- j_1 (1): child
- j_2 (2): bold
- j_7 (64): disabled
- j_8 (128): hidden

`dim1, dim2`: returns the parameter dimensions, both 0 if simple, $\dim_1 > 0$, $\dim_2 = 0$ if one dimensional array, both > 0 if two dimensional array. \dim_1 is the number of rows, \dim_2 the number of columns

`p_values`: returns the parameter value or array of values. The array elements are returned successively, row by row as a one dimensional array, independently of the dimensions of the variable specified to store it. If the variable is not a dynamic array, there are as many elements stored as there is room for (for a simple variable only one, the first element). If values is a two dimensional dynamic array, all elements are stored in the first row.

REQUEST ("ASSOCLP_NAME", "", name)

Returns in the given variable the name of the library part associated with the label or marker object. For elements (Walls, Slabs, etc.) the name is an empty string.

REQUEST ("ASSOCEL_PROPERTIES ", parameter_string, nr_data, data)

Returns, in the given variables, own property data or the element properties which the library part containing this request is associated to (in labels and associative marker objects). The function return value is the number of successfully retrieved values, 0 if no property data was found or an error occurred. The function does not work in property objects during the listing process.

parameter_string: a combination of keywords separated by commas representing the requested fields of the property data records. Records will be ordered accordingly. Possible values:

```
ISCOMP
DBSETNAME
KEYCODE
KEYNAME
CODE
NAME
FULLNAME
QUANTITY
TOTQUANTITY
UNITCODE
UNITNAME
UNITFORMATSTR
PROPOBJNAME
```

nr_data: returns the number of the data items

data: returns the property data, records containing and being ordered by the fields specified in the parameter string. Values are returned as a one dimensional array which contains the requested record fields successively, independently of the dimensions of the variable specified to store it. If the variable is not a dynamic array, there are as many elements stored as there is room for (for a simple variable only one, the first element). If values is a two dimensional dynamic array, all elements are stored in the first row.

Example:

```
DIM DATA []
n = REQUEST ("ASSOCEL_PROPERTIES", "iscomp, code, name", nr, data)
      IF nr = 0 THEN
        TEXT2 0, 0, "No properties"
      ELSE
        j = 0
        FOR i = 1 TO nr
          IF (i) MOD 3 = 0 THEN
            TEXT2 0, -j, DATA [i] ! name
            j = j + 1
          ENDIF
        NEXT i
      ENDIF
```

```
REQUEST ("REFERENCE_LEVEL_DATA", "",
  name1, elev1, name2, elev2, name3, elev3)
```

Returns in the given variables the names and elevations of the reference levels as set in the Preferences/Working Units/Reference Levels dialog. The function return value is the number of successfully retrieved values, 0 if an error occurred.

```
REQUEST ("ANCESTRY_INFO", expr, name [,
  guid, parent_name1, parent_guid1,
  ...,
  parent_namen, parent_guidn)
```

Ancestry information on a library part

If *expr* = 0, returns in the given variables the name and the global unic identifier of the library part containing this request function. Optionally the function returns the names and global unic identifiers of the parents of the library part (parent_name*i*, parent_guid*i*). If the parent templates are not loaded their names will be empty strings.

If *expr* = 1, returns information on the library part replaced by the template containing this function. In this case if the template is not actually replacing, no values are returned.

The return value of the request is the number of successfully retrieved values.

Example:

```
DIM strings[]
n = REQUEST ("ANCESTRY_INFO", 1, name, guid, strings)
IF n > 2 THEN
  ! data of replaced library part
  TEXT2 0, -1, "replacing: " + name + ' ' + guid
  ! parents
  l = -2
  FOR i = 1 to n - 2 STEP 2
    TEXT2 0, 1, strings [i]
    l = l - 1
  NEXT i
ENDIF
```

```
REQUEST ('TEXTBLOCK_INFO',
  textblock_name, width, height)
```

Returns in the given variables the sizes in x and y direction of a previously defined TEXTBLOCK. The sizes are in mm or in m in model space depending on the fixed_height parameter value of the TEXTBLOCK (millimeters if 1, meters in model space if 0). If width was 0, the request returns the calculated width and height, if width was specified in the TEXTBLOCK definition, returns the calculated height corresponding to that width.

```
REQUEST{2} ("Material_info", name_or_index,
  param_name, value_or_values)
```

```
REQUEST{2} ("Material_info", name_or_index,  
           extra_param_name,  
           value_or_values)
```

Returns information in the given variable(s) on a parameter (or extra parameter, see *“Additional Data” on page 175*) of the specified material. RGB information is returned in three separate variables, texture information is returned in the following variables: file_name, width, height, mask, rotation_angle corresponding to the texture definition. All other parameter information is returned in single variables. Possible material parameter names corresponding to parameters of the material definition:

```
gs_mat_surface_rgb (surface R, G, B [0.0..1.0])  
gs_mat_ambient (ambient coefficient [0.0..1.0])  
gs_mat_diffuse (diffuse coefficient [0.0..1.0])  
gs_mat_specular (specular coefficient [0.0..1.0])  
gs_mat_transparent (transparent coefficient [0.0..1.0])  
gs_mat_shining (shininess [0.0..100.0])  
gs_mat_transp_att (transparency attenuation [0.0..4.0])  
gs_mat_specular_rgb (specular color R, G, B [0.0..1.0])  
gs_mat_emission_rgb (emission color R, G, B [0.0..1.0])  
gs_mat_emission_att (emission attenuation [0.0..65.5])  
gs_mat_fill_ind (fill index)  
gs_mat_fillcolor_ind (fill color index)  
gs_mat_texture (texture index)
```

Example:

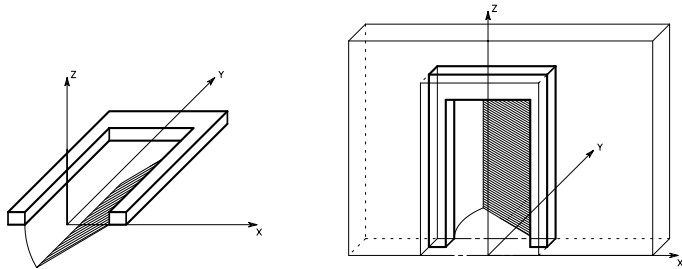
```
REQUEST{2} ("Material_info",  
           "Brick-Face", "gs_mat_ambient", a)  
REQUEST{2} ("Material_info", 1,  
           "gs_mat_surface_rgb", r, g, b)  
REQUEST{2} ("Material_info",  
           "Brick-Face", "gs_mat_texture",  
           file_name, w, h, mask, alpha)  
REQUEST{2} ("Material_info",  
           "My-Material", "my_extra_parameter", e)
```

DOORS AND WINDOWS

This section discusses the various special options related to the creation of Door/Window library elements.

General Guidelines

Once a door/window is inserted into a wall, the default position of these library parts' coordinate system is rotated so that the x-y plane is vertical and the z axis points horizontally into the wall. The origin is placed on the bottom center of the wall opening, on the exterior side of the wall. This way, doors/windows can be easily modeled by elements in the x-y plane. See the illustrations below.



Because of the special behavior of these library parts, the 2D symbol is generated from a special built-in projection otherwise not accessible by users (an upside-down side view from a 90 degree direction). The symbol and the 3D shape are fitted to the Door/Window origin by the lower (y) center (x) of the bounding box, but no adjustment is made along the z axis to enable users to design doors/windows extending beyond the wall in either z direction.

Considering these rules, here are some hints that will help you construct doors/windows that will work properly:

- When constructing the door/window in the floor plan window, visualize it as if you are looking at it from the inside of the wall it will be inserted into.
- Think of the project zero level as the external surface of the wall.
- Elements that should be inside the wall, like the window frame, should be above the zero level.
- Door panels opening to the outside should be below the zero level.

Creation of Door/Window Library Parts

When creating Door/Window type library parts, several possibilities exist, presenting different problems:

- Creation of rectangular doors/windows in straight walls
- Creation of non-rectangular doors/windows in straight walls
- Creation of rectangular doors/windows in curved walls
- Creation of non-rectangular doors/windows in curved walls

Rectangular Doors/Windows in Straight Walls

This is the easiest and most straightforward way of creating doors and windows. The use of simple GDL commands such as PRISM_ or RECT is recommended.

If you want to match the surface materials of door/window elements to those of the wall, the bottom surface of the elements should match the outside, and the top surface the inside of the wall. You can achieve this from your scripts using the G_, H_ and I_ global variables representing the materials of the wall into which the door/window is placed. In the 2D script, the E_, F_ and A~ global variables can be useful, as these give you the pen numbers of the wall contour and fill plus the index number of the fill of the wall on the floor plan into which the door/window is placed. With composite walls, you have to use the corresponding global variables.

See *“Miscellaneous” on page 215* for details.

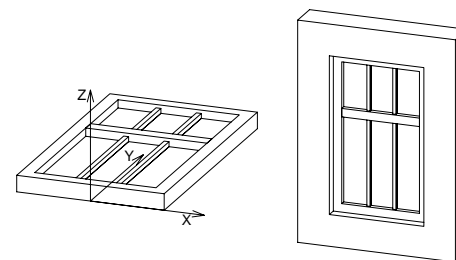
The object libraries come with a large set of door/window macros. These GDL scripts contain common building elements which are used by many doors/windows in the library. There are macros for generating commonly-used frames, panels and many other types of door/window parts. Open some door/window library parts to see what kind of macros they call and what type of parts those macros generate.

Example:

```

A=0.9: B=1.5: C=0.1: D=0.08
E=0.08: F=0.9: G=0.03: H=3
PRISM 10,C,
  -A/2, 0, 15, A/2, 0, 15,
  A/2, B, 15, -A/2, B, 15,
  -A/2, 0, -1,
  -A/2+D, D, 15, A/2-D, D, 15,
  A/2-D, B-D, 15, -A/2+D, B-D, 15,
  -A/2+D, D, -1
ADD -A/2+D, F, 0
BRICK A-2*D, E, C
ADD -G/2, -F+D, C/2
GOSUB 1
ADDZ -G
GOSUB 1
DEL 2
MATERIAL "Glass"
ADD0, -F+D, C/2
RECT A-2*D, F-D
ADDY F-D+E
RECT A-2*D, B-F-E-D
END
1: FOR I=1 TO H-1
  ADDX (A-2*D)/3
  BLOCK G, F-D, G
  ADDY F+E-D
  BLOCK G, B-F-D-E, G
  DEL 1
NEXT I
DEL H-1
RETURN

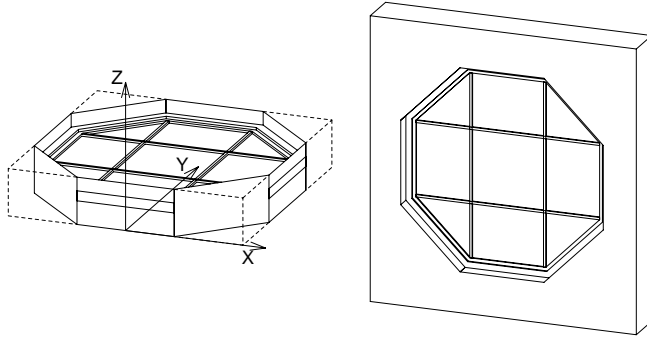
```



Non-Rectangular Doors/Windows in Straight Walls

When working with doors/windows, it is important to know that placing a door/window always cuts a rectangular hole into the wall. The size of this hole is determined by the A and B parameters of the door/window library part. However, when the door/window is not rectangular in elevation, it does not entirely fill the cut rectangular hole. The solution to this is to use the WALLHOLE or WALLNICHE command to define a polygon shape to be cut into the wall where the door/window is placed. There are two solutions for this:

- The 3D script has to contain parts that generate those parts of the wall that fill the hole between the door/window body and the edges of the rectangular wall cut. In this case, special attention must be paid to the visibility of the edges of these fillings.



- With the WALLHOLE or WALLNICHE command, you can define a polygon shape to be cut into the wall where the door/window is placed.

WALLHOLE

```
WALLHOLE n, status,
          x1, y1, mask1,
          ...
          xn, yn, maskn
          [, x, y, z]
```

n: the number of polygon nodes

status:

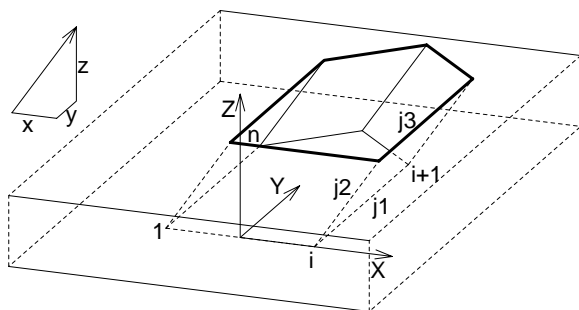
- 1: use the attributes of the body for the generated polygons and edges
- 2: generated cut polygons will be treated as normal polygons

xi, yi: cross-section polygon coordinates

maski: similar to the CUTPOLYA statement:

$\text{mask}_i = j_1 + 2 * j_2 + 4 * j_3 + 64 * j_7$

x, y, z: optional direction vector (default is door/window Z axis)



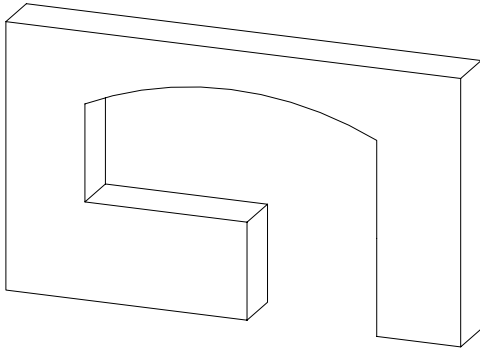
This command can be used in doors'/windows' 3D script to cut custom hole(s) in the wall they are placed into. During the 3D generation of the current wall, the 3D script of all its doors/windows is interpreted without model generation to collect the WALLHOLE commands. If they exist, the current wall will be cut using an infinite tube with the polygonal cross-section and direction defined in the script. There can be any number of WALLHOLEs for any door/window, so it is possible to cut more holes for the same door/window, even intersecting ones. If at least one WALLHOLE command is interpreted in a door/window 3D script, no rectangular opening will be generated for that door/window.

Note: The 3D reveal will not be generated automatically for custom holes, you have to generate it from the script.

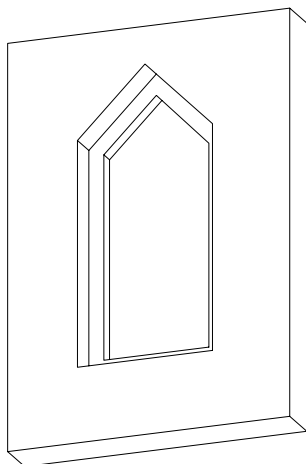
The hole customized this way will only be visible in 3D, because WALLHOLE commands do not have any effect in 2D. A 2D representation can be scripted if needed (used with "framing in plan" off).

The use of convex polygonal cross-sections is recommended; using concave polygons may result in strange shadings/renderings or cut errors. Convex polygons can be combined to obtain concave ones.

Examples:



```
RESOL 72
L1=2.7: L2=1.2: H1=2.1: H2=0.3: H3=0.9
R= ((L1/2)^2+H2^2)/(2*H2)
A=ATN((L1/2)/(R-H2))
WALLHOLE 5,1,
  -L1/2,H3,15,
  L1/2,H3,15,
  L1/2,H1-H2,13,
  0,H1-R,915,
  0,2*A,4015
WALLHOLE 4,1,
  L1/2-L2,0,15,
  L1/2,0,15,
  L1/2,H3,15,
  L1/2-L2,H3,15
```



```

WALLHOLE  5,1,
          -0.45, 0, 15,
           0.45, 0, 15,
           0.45, 1.5, 15,
           0, 1.95, 15,
          -0.45, 1.5, 15

```

```

PRISM     12, 0.1,
          -0.45, 0, 15,
           0.45, 0, 15,
           0.45, 1.5, 15,
           0, 1.95, 15,
          -0.45, 1.5, 15,
          -0.45, 0, -1,
          -0.35, 0.1, 15,
           0.35, 0.1, 15,
           0.35, 1.45, 15,
           0, 1.80, 15,
          -0.35, 1.44, 15,
          -0.35, 0.1, -1

```

WALLNICHE

WALLNICHE *n*, *method*, *status*,
 rx, *ry*, *rz*, *d*,
 x1, *y1*, *mask1*,
 ...
 xn, *yn*, *maskn*

Similar to the CUTFORM definition.

method: Controls the form of the cutting body

1: prism shaped

2: pyramidal

3: wedge-shaped cutting body. The direction of the wedge's top edge is parallel to the Y axis and its position is in *rx*, *ry*, *rz* (*ry* is ignored).

status: Controls the extent of the cutting body and the treatment of the generated cut polygons and new edges.

$status = j1 + 2*j2 + 8*j4 + 16*j5 + 32*j6 + 64*j7 + 128*j8$

j1: use the attributes of the body for the generated polygons and edges

j2: generated cut polygons will be treated as normal polygons

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. (can only be used with the CUTFORM command)

j7: edges generated by the bottom of the cutting body will be invisible

j8: edges generated by the top of the cutting body will be invisible

rx,ry,rz: defines the direction of cutting if the cutting form is prism-shaped, or the top of the pyramid if the method of cutting is pyramidal

d: defines the distance along *rx,ry,rz* to the end of the cut. If the cut is infinite, this parameter has no effect. If the cut is finite, then the start of the cutting body will be at the local coordinate system and the body will end at a distance of *d* along the direction defined by *rx,ry,rz*

If the cut is semi-finite, then the start of the cutting body will be at a distance of *d* along the direction defined by *rx,ry,rz* and the direction of the semi-infite cut will be in the opposite direction defined by *rx,ry,rz*.

mask: Defines the visibility of the edges of the cutting body:

$$\text{mask}_i = j_1 + 2*j_2 + 4*j_3 + 8*j_4 + 16*j_5 + 64*j_7$$

j1: the polygon will create a visible edge upon entry into the body being cut

j2: the lengthwise edge of the cutting form will be visible

j3: the polygon will create a visible edge upon exiting the body being cut

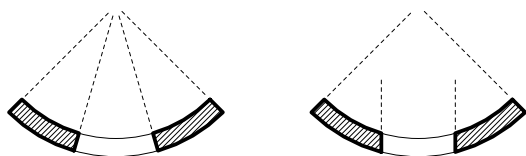
j4: the bottom edge of the cutting form will be visible

j5: the top edge of the cutting form will be visible

j7: controls the viewpoint dependent visibility of the lengthwise edge

Rectangular Doors/Windows in Curved Walls

When placing doors/windows into curved walls, the sides of the hole cut into the wall can vary according to the picture below.



The hole in the wall on the left is created when the program automatically cuts the hole for the door/window. In this case the sides will be of radial direction. On the right, the hole is cut using the WALLHOLE command in the 3D Script of the door/window object. The object itself needs to be written by taking these factors into consideration.

Another thing to consider is whether the door/window placed into the curved wall is a straight or a curved one.



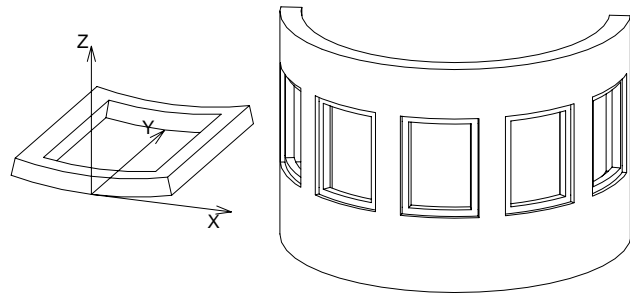
In the case of a straight door/window, as on the left above, the thickness and width of the object and the thickness of the wall are closely related, since above a certain dimension the object would fall outside of the wall. When using true curved doors/windows, this problem doesn't occur.

Example:

```

RESOL 72
ROTX -90
MULY -1
C= 0.12: Z=(360*A)/(2*R_*PI)
Y= (360*C)/(2*R_*PI)
A1= 270+Z/2: A2=270-Z/2
GOSUB 1
ADDZ B
MULZ -1
GOSUB 1
DEL 2
ADDZ C
GOSUB 2
MULX -1
GOSUB 2
END
1:
PRISM 9, C,
  COS(A2)*R_, SIN(A2)*R_+R_, 11,
  COS(A2+Y)*R_, SIN(A2+Y)*R_+R_, 13,
  0, R_, 900,
  0, Z-2*Y, 4009,
  COS(A1)*R_, SIN(A1)*R_+R_, 11,
  COS(A1)*(R_-0.1), SIN(A1)*(R_-0.1)+R_, 11,
  COS(A1-Y)*(R_-0.1), SIN(A1-Y)*(R_-0.1)+R_, 13,
  0, -(Z-2*Y), 4009,
  COS(A2)*(R_-0.1), SIN(A2)*(R_-0.1)+R_, 11
RETURN
2:
PRISM 4, B-2*C,
  COS(A2)*R_, SIN(A2)*R_+R_, 10,
  COS(A2+Y)*R_, SIN(A2+Y)*R_+R_, 15,
  COS(A2+Y)*(R_-0.1), SIN(A2+Y)*(R_-0.1)+R_, 10,
  COS(A2)*(R_-0.1), SIN(A2)*(R_-0.1)+R_, 10
RETURN

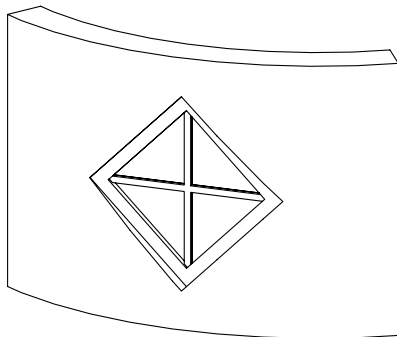
```



Non-Rectangular Doors/Windows in Curved Walls

The general guidelines given for rectangular doors/windows in curved walls applies here, too.

Example:



```

C=0.1: D=0.025
Z=A/2-SQR(2)*C: Y=A/2-SQR(2)*C-D
ADDY A/2
WALLHOLE 4, 1,
    0, -A/2, 15,
    A/2, 0, 15,
    0, A/2, 15,
    -A/2, 0, 15
PRISM_ 10, 0.1,
    0, -A/2, 15,
    A/2, 0, 15,
    0, A/2, 15,
    -A/2, 0, 15,
    0, -A/2, -1,
    0, -Z, 15,
    Z, 0, 15,
    0, Z, 15,
    -Z, 0, 15,
    0, -Z, -1
ADDZ 0.02
GOSUB 1
ADDZ 0.03
GOSUB 1
ADDY -Z

```

```
SET MATERIAL "Glass"  
ROTZ 45  
RECT SQR(2)*Z, SQR(2)*Z  
END  
1:  
PRISM_ 16, 0.03,  
      0, -Z, 15,  
      D, -Y, 15,  
      D, -D, 15,  
      Y, -D, 15,  
      Z, 0, 15,  
      Z, D, 15,  
      D, D, 15,  
      D, Y, 15,  
      0, Z, 15,  
      -D, Y, 15,  
      -D, D, 15,  
      -Y, D, 15,  
      -Z, 0, 15,  
      -Y, -D, 15,  
      -D, -D, 15,  
      -D, -Y, 15  
RETURN
```

GDL CREATED FROM THE FLOOR PLAN

Saving the floor plan as a GDL script or library part will result GDL elements. You can use these GDL scripts as templates for your custom library parts.

KEYWORDS

Common Keywords

Operators, Functions

FOR, NEXT
DO, WHILE, ENDWHILE
REPEAT, UNTIL
IF, THEN, ELSE, ENDIF
GOTO
GOSUB
RETURN
END
EXIT
PUT
GET
USE
NSP
CALL, PARAMETERS
PRINT
OPEN
INPUT
VARTYPE
OUTPUT
CLOSE
DIM
BREAKPOINT

Reserved Keywords

The keywords listed below are reserved; they exist for compatibility reasons or are not publicized.

BAS
BOX
FILTER
GDLBIN
LIN
LINE
NOD
NODE
ORIGO
PARS
PLOTMAKER
PLOTTER
RECT_
SFLINE
TET
TETRA
TRI
WALL_
VOCA_
UI_OK
UI_CANCEL

3D Use Only

ADDX, ADDY, ADDZ
 ADD
 MULX, MUY, MULZ
 MUL
 ROTX, ROTY, ROTZ
 ROT
 XFORM

HOTSPOT
 LIN_
 RECT
 POLY, POLY_
 PLANE, PLANE_
 CIRCLE
 ARC

BLOCK, BRICK
 CYLIND
 SPHERE
 ELLIPS
 CONE
 PRISM, PRISM_, CPRISM_, BPRISM_, FPRISM_,
 SPRISM_
 SLAB, SLAB_, CSLAB_
 CWALL_, BWALL_, XWALL_
 WALLHOLE
 BEAM
 CROOF_
 ARMC
 ARME
 ELBOW
 EXTRUDE
 PYRAMID
 REVOLVE

RULED
 SWEEP
 TUBE, TUBEA
 COONS
 MESH
 MASS
 LIGHT
 PICTURE
 TEXT
 VERT, TEVE
 VECT
 EDGE
 PGON, PIPG
 COOR
 BODY
 BASE
 BINARY

CUTPLANE
 CUTSHAPE
 CUTPOLY
 CUTPOLYA
 CUTEND

DEFINE MATERIAL
 DEFINE TEXTURE
 [SET] MATERIAL
 SHADOW
 MODEL

SECT_FILL

2D Use Only

ADD2
MUL2
ROT2

HOTSPOT2
LINE2
RECT2
POLY2, POLY2_, POLY2_A, POLY2_B
ARC2
CIRCLE2
SPLINE2, SPLINE2A

PICTURE2
TEXT2
FRAGMENT2

PROJECT2

DEFINE FILL
DEFINE FILLA
DEFINE LINE_TYPE

[SET] FILL
[SET] LINE_TYPE
DRAWINDEX
DRAWING2
DRAWING3

2D and 3D Use

DEL
[LET]
RADIUS
RESOL
TOLER
PEN
DEFINE STYLE
[SET] STYLE

Non-Geometric Scripts

Property Script

DATABASE_SET
DESCRIPTOR
COMPONENT
REF
SURFACE3D
VOLUME3D

POSITION

WALLS
COLUMNS
BEAMS
DOORS
WINDOWS
OBJECTS
PITCHED_ROOFS
HIP_ROOFS
LIGHTS
HATCHES
ROOMS
MESHES

DRAWING
BINARYPROP

Parameter Script

VALUES
PARAMETERS
LOCK

Interface Script

UI_DIALOG
UI_PAGE
UI_BUTTON
UI_PREV
UI_NEXT
UI_GROUPBOX
UI_SEPARATOR
UI_PICT
UI_STYLE
UI_OUTFIELD
UI_INFIELD

Alphabetical List of Current GDL Keywords

A

ABS (x) Returns the absolute value of x (integer if x integer, real otherwise).

ACS (x) Returns the arc cosine of x. ($-1.0 \leq x \leq 1.0$; $0^\circ \leq \text{ACS}(x) \leq 180^\circ$).

ADD dx, dy, dz

ADD2 x, y

ADDGROUP (g_expr1, g_expr2)

ADDX dx

ADDY dy

ADDZ dz

AND (or &) Logical and precedence 6

ARC r, alpha, beta

ARC2 x, y, r, alpha, beta

ARMC r1, r2, l, h, d, alpha

ARME l, r1, r2, h, d

ASN (x) Returns the arc sine of x. ($-1.0 \leq x \leq 1.0$; $-90^\circ \leq \text{ASN}(x) \leq 90^\circ$).

ATN (x) Returns the arc tangent of x. ($-90^\circ \leq \text{ATN}(x) \leq 90^\circ$).

B

BASE

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

BINARY mode [, section]

BINARYPROP

BITSET (x, b [, expr])

BITTEST (x, b)

BLOCK a, b, c

BODY status

BPRISM top_material, bottom_material, side_material,
n, h, radius, x1, y1, s1, ... xn, yn, sn

BREAKPOINT expression

BRICK a, b, c

BWALL 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

C

CALL macro_name **PARAMETERS ALL**

CALL macro_name_string [,parameter_list]

CALL macro_name_string **PARAMETERS** [name1=value1 , ... namen=valuen]

CEIL (x)Returns the smallest integral value that is not smaller than x (always integer).
(e.g., CEIL(1.23) = 2; CEIL (-1.9) = -1).

CIRCLE r

CIRCLE2 x, y, r 117

CLOSE channel

COMPONENT name, quantity, unit [, proportional_with, code, keycode, unitcode]

CONE h, r1, r2, alpha1, alpha2

COONS *n*, *m*, *mask*,
 x11, *y11*, *z11*, ... *x1n*, *y1n*, *z1n*,
 x21, *y21*, *z21*, ... *x2n*, *y2n*, *z2n*,
 x31, *y31*, *z31*, ... *x3m*, *y3m*, *z3m*,
 x41, *y41*, *z41*, ... *x4m*, *y4m*, *z4m* 94

COOR *wrap*, *vert1*, *vert2*, *vert3*, *vert4*

COS (*x*) Returns the cosine of *x*.

CPRISM *top_material*, *bottom_material*, *side_material*,
 n, *h*, *x1*, *y1*, *s1*, ... *xn*, *yn*, *sn*

CROOF *top_material*, *bottom_material*, *side_material*,
 n, *xb*, *yb*, *xe*, *ye*, *height*, *angle*, *thickness*,
 x1, *y1*, *alpha1*, *s1*,
 ...,
 xn, *yn*, *alphan*, *sn*

CSLAB *top_material*, *bottom_material*, *side_material*,
 n, *h*, *x1*, *y1*, *z1*, *s1*, ... *xn*, *yn*, *zn*, *sn*

CUTFORM *n*, *method*, *status*,
 rx, *ry*, *rz*, *d*,
 x1, *y1*, *mask1*,
 ...,
 xn, *yn*, *maskn*

CUTPLANE [*x*, *y*, *z* [, *side*]]
 [*statement1*
 ...
 statementn]
CUTEND

CUTPLANE *angle*
 [*statement1*
 ...
 statementn]
CUTEND

```
CUTPOLY n,  
    x1, y1, ... xn, yn  
    [, x, y, z]  
    [statement1  
    statement2  
    ...  
    statementn]  
CUTEND  
  
CUTPOLYA n, status, d,  
    x1, y1, mask1, ... xn, yn, maskn [,  
    x, y, z]  
    [statement1  
    statement2  
    ...  
    statementn]  
  
CUTSHAPE d  
    [statement1  
    statement2  
    ...  
    statementn]  
  
CWALL _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  
  
CYLIND h, r
```

D

DATABASE_SET set_name [descriptor_name, component_name, unit_name, key_name, criteria_name, list_set_name]

DEFINE EMPTY_FILL name [**FILLTYPES_MASK** fill_types]

DEFINE FILL name [**FILLTYPES_MASK** fill_types,] pattern1, pattern2, pattern3, pattern4, pattern5, pattern6, pattern7, pattern8, spacing, angle, n, frequency1, direction1, offset_x1, offset_y1, m1, length11, ... length1m, ... frequencyn, directionn, offset_xn, lengthn1, ... lengthnm

DEFINE FILL parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]

DEFINE FILL_A parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]

DEFINE FILLA name [**FILLTYPES_MASK** fill_types,] pattern1, pattern2, pattern3, pattern4, pattern5, pattern6, pattern7, pattern8, spacing_x, spacing_y, angle, n, frequency1, directional_offset1, direction1, offset_x1, offset_y1, m1, length11, ... length1m, ... frequencyn, directional_offsetn, directionn, offset_xn, offset_yn, mn, lengthn1, ... lengthnm

DEFINE LINE_TYPE name spacing, n, length1, ... lengthn

DEFINE LINE_TYPE parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]

DEFINE MATERIAL name BASED_ON orig_name PARAMETERS name1 = expr1 [, ...][ADDITIONAL_DATA name1 = expr1 [, ...]]

DEFINE MATERIAL name type, parameter1, parameter2, ... parametern

DEFINE MATERIAL parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]

DEFINE SOLID_FILL name [**FILLTYPES_MASK** fill_types]

DEFINE STYLE name font_family, size, anchor, face_code

```
DEFINE STYLE{2} name font_family, size, face_code
DEFINE SYMBOL_FILL name [FILLTYPES_MASK fill_types,]
    pat1, pat2, pat3, pat4, pat5, pat6, pat7, pat8,
    spacingx1, spacingy1,
    spacingx2, spacingy2, angle,
    scaling1, scaling2, macro_name PARAMETERS [name1 = value1, ... namen = valuen]
DEFINE SYMBOL_FILL parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]
DEFINE SYMBOL_LINE name dash, gap, macro_name PARAMETERS [name1 = value1, ... namen = valuen]
DEFINE SYMBOL_LINE parameters [ADDITIONAL_DATA name1 = value1, name2 = value2, ...]
DEFINE TEXTURE name expression, x, y, mask, angle
DEL n [, begin_with]
DEL TOP
DESCRIPTOR name [,code, keycode]
DIM var1[dim_1], var2[dim_1][dim_2], var3[ ],
    var4[ ][ ], var5[dim_1][ ],
    var5[ ][dim_2]
DO
    [statement1
    statement2
    ...
    statementn]
DRAWINDEX number
DRAWING2 [expression]
DRAWING3 projection_code, angle, method
DRAWING3{2} projection_code, angle, method [,backgroundColor, origoX, origoY,
    fillldirection]
DRAWING
```

E

EDGE vert1, vert2, pgon1, pgon2, status

ELBOW r1, alpha, r2

ELLIPS h, r

END

ENDGROUP

EXIT

EXOR (or @) Logical exclusive or precedence 8

EXP (x) Returns the x th power of e (e = 2.7182818).

EXTRUDE n, dx, dy, dz, mask, x1, y1, s1,
..., xn, yn, sn

F

FILE_DEPENDENCE "name1" [, "name2", ...]

FOR variable_name = initial_value TO end_value [STEP step_value]

FPRISM top_material, bottom_material, side_material, hill_material,
n, thickness, angle, hill_height,
x1, y1, s1,
...
xn, yn, sn

FRA (x) Returns the fractional part of x (integer 0 if x integer, real otherwise). (e.g.,
FRA(1.23) = 0.23, FRA(-1.23) = 0.77).

FRAGMENT2 fragment_index,
use_current_attributes_flag

FRAGMENT2 ALL, use_current_attributes_flag

G

GET (n)

GOSUB label

GOTO label

GROUP "name"

H

HIDEPARAMETER name1 [, name2, ..., namen]

HOTARC2 x, y, r, startangle, endangle

HOTLINE2 x1, y1, x2, y2

HOTSPOT x, y, z [, unID [, paramReference, flags] [, displayParam]]

HOTSPOT2 x, y [, unID [, paramReference, flags][, displayParam]]

HPRISM top_mat, bottom_mat, side_mat,
hill_mat,
n, thickness, angle, hill_height, status,
x1, y1, s1,
...,
xn, yn, sn

I

IF condition **GOSUB** label

IF condition **GOTO** label

IF condition **THEN** label

IND (FILL, name_string)

IND (LINE_TYPE, name_string)

IND (MATERIAL, name_string)

IND (STYLE, name_string)

IND (TEXTURE, name_string)

INPUT (channel, recordID, fieldID, variable1 [, variable2,...])

INT (x) Returns the integral part of x (always integer). (e.g., INT(1.23) = 1, INT(-1.23) = -2).

ISECTGROUP (g_expr1, g_expr2)

ISECTLINES (g_expr1, g_expr2)

K

KILLGROUP g_expr

L

[LET] varnam = n

LGT (x) Returns the base 10 logarithm of x.

LIGHT red, green, blue, shadow,
radius, alpha, beta, angle_falloff,
distance1, distance2,
distance_falloff [ADDITIONAL_DATA name1 = value1,
name2 = value2, ...]

LIN_ x1, y1, z1, x2, y2, z2

LINE_PROPERTY expr

LINE2 x1, y1, x2, y2

LOCK name1 [, name2, ..., namen]

LOG (x) Returns the natural logarithm of x.

M

MASS 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

MAX (x1,x2, ... xn) Returns the largest of an unlimited number of arguments.

MESH a, b, m, n, mask,
z11, z12, ... z1m,
z21, z22, ... z2m,
...
zn1, zn2, ... znm

MIN (x1,x2, ... xn) Returns the smallest of an unlimited number of arguments.

MODEL SOLID

MODEL SURFACE

MODEL WIRE

MUL mx, my, mz

MUL2 x, y

MULX mx

MULY my

MULZ mz

N

NEXT variable_name

NOT (x) Returns false (=0 integer) if x is true (<>0), and true (=1 integer) if x is false (=0) (logical negation).

NSP

NTR ()

O

OPEN (filter, filename, parameter_string)

OR (or |) Logical inclusive or precedence 7

OUTPUT (ch, recordID, fieldID, var1, var2...)

OUTPUT channel, recordID, fieldID, expression1 [, expression2, ...]

P

PARAGRAPH name alignment, firstline_indent,
left_indent, right_indent, line_spacing [,
tab_size1, ...]
[PEN index]
[[SET] STYLE style1]
[[SET] MATERIAL index]
'string1'
'string2'
...
'string n'
[PEN index]
[[SET] STYLE style2]
[[SET] MATERIAL index]
'string1'
'string2'
...
'string n'
...

PARAMETERS name1 = expression1 [,
name2 = expression2, ...,
namen = expressionn]

PEN n

PGON n, vect, status, edg1, edge2, ... edgen

PI Returns Ludolph's constant. (p = 3.1415926...).

PICTURE expression, a, b, mask

PICTURE2 expression, a, b, mask

PICTURE2{2} expression, a, b, mask

PIPG expression, a, b, mask, n, vect,
status,
edg1, edge2, ... edgen

PLACEGROUP g_expr

PLANE n, x1, y1, z1, ... xn, yn, zn

PLANE_ n, x1, y1, z1, s1, ... xn, yn, zn, sn

POLY n, x1, y1, ... xn, yn

POLY_ n, x1, y1, s1, ... xn, yn, sn 61

POLY2 n, frame_fill, x1, y1, ... xn, yn

POLY2_ n, frame_fill, x1, y1, s1, ... xn, yn, sn

POLY2_A n, frame_fill, fill_pen,
x1, y1, s1, ..., xn, yn, sn

POLY2_B n, frame_fill, fill_pen,
fill_background_pen,
x1, y1, s1, ..., xn, yn, sn

POLY2_B{2} n, frame_fill, fill_pen,
fill_background_pen,
fillOrigoX, fillOrigoY,
fillAngle,
x1, y1, s1, ..., xn, yn, sn

POSITION position_keyword

PRINT expression [, expression, ...]

PRISM n, h, x1, y1, ... xn, yn

PRISM_ n, h, x1, y1, s1, ... xn, yn, sn

PROJECT2 projection_code, angle, method

PROJECT2{2} projection_code, angle,method [,backgroundColor, fillOrigoX,
fillOrigoY, filldirection]

PUT expression [, expression, ...]

PYRAMID n, h, mask, x1, y1, s1, ... xn, yn, sn

R

RADIUS radius_min, radius_max

RECT a, b

RECT2 x1, y1, x2, y2 114

REF COMPONENT code [, keycode [, numeric_expression]]

REF DESCRIPTOR code [, keycode]

REPEAT

[statement1

statement2

...

statementn]

REQ (parameter_string)

REQ (parameter_string)

REQUEST ("ANCESTRY_INFO", expr, name [,
guid, parent_name1, parent_guid1,
...,
parent_name_n, parent_guid_n)

REQUEST ("Angular_dimension", "",
format_string)

REQUEST ("Angular_length_dimension", ""
format_string)

REQUEST ("Area_dimension", ""
format_string)

REQUEST ("ASSOCEL_PROPERTIES ", parameter_string, nr_data, data)

REQUEST ("ASSOCLP_PARVALUE", expr, name_or_index, type, flags, dim1, dim2, values)

REQUEST ("Calculation_angle_unit", "",
format_string)

REQUEST ("Calculation_area_unit", "",
format_string)

REQUEST ("Calculation_length_unit", "",
format_string)

```
REQUEST ("Calculation_volume_unit", "",
        format_string)
REQUEST ("Constr_Fills_display", "", optionVal)
REQUEST ("Elevation_dimension", "",
        format_string)
REQUEST ("Height_of_style", name,
        height [, descent, leading])
REQUEST ("Level_dimension", "",
        format_string)
REQUEST ("matching_properties", type, name1, name2, ...)
REQUEST ("Name_of_program", "", program_name)
REQUEST ("Radial_dimension", "",
        format_string)
REQUEST ("REFERENCE_LEVEL_DATA", "",
        name1, elev1, name2, elev2, name3, elev3)
REQUEST ("Sill_height_dimension", "",
        format_string)
REQUEST ("Style_info", name, fontname [, size, anchor, face_or_slant])
REQUEST ("Window_door_dimension", "",
        format_string)
REQUEST ("window_door_zone_relev", " ", out_direction)
REQUEST ("Working_angle_unit", "", format_string)
REQUEST ("Working_length_unit", "", format_string)
REQUEST ("Zone_relations", "", category_name, code, name, number [, category_name2,
        code2, name2, number2])
REQUEST ('TEXTBLOCK_INFO',
        textblock_name, width, height)
REQUEST (extension_name, parameter_string, variable1, variable2, ...)
REQUEST (question_name, name | index, variable1 [, variable2,...])
REQUEST (question_name, name | index, variable1 [, variable2,...])
```

REQUEST{2} ("Material_info", name_or_index,
extra_param_name,
value_or_values)

REQUEST{2} ("Material_info", name_or_index,
param_name, value_or_values)

RESOL n

RETURN

REVOLVE n, alpha, mask, x1, y1, s1, ... xn, yn, sn

RICTEXT x, y,
height, 0, textblock_name

RICTEXT2 x, y, textblock_name

RND (x) Returns a random value between 0.0 and x (x > 0.0) always real.

ROT x, y, z, alpha

ROT2 alpha

ROTX alphax

ROTY alphay

ROTZ alphaz

ROUND_INT (x)

RULED n, mask,
u1, v1, s1, ... un, vn, sn,
x1, y1, z1, ... xn, yn, zn

RULED{2} n, mask,
u1, v1, s1, ... un, vn, sn,
x1, y1, z1, ... xn, yn, zn

S

[SET] FILL index

[SET] FILL name_string

[SET] LINE_TYPE index

[SET] LINE_TYPE name_string

[SET] MATERIAL index

[SET] MATERIAL name_string

[SET] STYLE index

[SET] STYLE name_string

SECT_FILL fill, fill_background_pen,
fill_pen, contour_pen

SGN (x) Returns +1 integer if x positive, -1 integer if x negative, otherwise 0 integer.

SHADOW keyword_1[, keyword_2]

SIN (x) Returns the sine of x.

SLAB n, h, x1, y1, z1, ... xn, yn, zn

SLAB_ n, h, x1, y1, z1, s1, ... xn, yn, zn, sn

SPHERE r

SPLINE2 n, status, x1, y1,
angle1, ..., xn, yn, anglen

SPLINE2A n, status, x1, y1, angle1, length_previous1, length_next1,
...
xn, yn, anglen, length_previousn,
length_nextn 119

SPLIT (string, format, variable1 [, variable2, ..., variablen])

SPRISM_ top_material, bottom_material, side_material,
n, xb, yb, xe, ye, h, angle,
x1, y1, s1, ... xn, yn, sn

SQR (x) Returns the square root of x (always real).

STR (numeric_expression, length, fractions)

STR (format_string, numeric_expression)

STR{2} (format_string, numeric_expression [, extra_accuracy_string])

STRLEN (string_expression)

STRSTR (string_expression1, string_expression2)

STRSUB (string_expression, start_position, characters_number)

STW (string_expression)

SUBGROUP (g_expr1, g_expr2)

SURFACE3D ()

SWEEP n, m, alpha, scale, mask,
u1, v1, s1, ... un, vn, sn,
x1, y1, z1, ... xm, ym, zm

SWEEPGROUP (g_expr, x, y, z)

T

TAN (x) Returns the tangent of x.

TEVE x, y, z, u, v

TEXT d, 0, expression

TEXT2 x, y, expression

TEXTBLOCK name width, anchor, angle, width_factor, charspace_factor, fixed_height,
'string_expr1' [, 'string_expr2', ...]

TOLER d

TUBE n, m, mask,
u1, w1, s1,
...
un, wn, sn,
x1, y1, z1, angle1,
...
xm, ym, zm, anglem

TUBEA n, m, mask,
u1, w1, s1,
...
un, wn, sn,
x1, y1, z1,
...
xm, ym, zm

U

UI_BUTTON type, text, x, y, width, height

UI_DIALOG title [, size_x, size_y]

UI_GROUPBOX text, x, y, width, height

UI_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]

UI_OUTFIELD expression,x,y,width,height

UI_PAGE page_number

UI_PICT expression, x, y [,width, height]

UI_SEPARATOR x1, y1, x2, y2

UI_STYLE fontsize, face_code

USE (n)

V

VALUES "fillparam_name" [**FILLTYPES_MASK** fill_types,] value_definition1 [, value_definition2,
 ...]

VARDIM1 (expr)

VARDIM2 (expr)

VARTYPE (expression)

VECT x, y, z

VERT x, y, z

VOLUME3D ()

W

WALLHOLE n, status,
x1, y1, mask1,
...
xn, yn, maskn
[, x, y, z]

WALLNICHE n, method, status,
rx, ry, rz, d,
x1, y1, mask1,
...
xn, yn, maskn

WHILE condition DO
[statement1
statement2
...
statementn]

X

XFORM a11, a12, a13, a14,
 a21, a22, a23, a24,
 a31, a32, a33, a34

XWALL_ 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

XWALL_{2} 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

Parameter Naming Convention

Because of the subtype hierarchy, the child library parts automatically inherit all parameters of the parent. (Read more about subtypes and parameter in the ArchiCAD User Guide). Parameters are identified by their name, so inherited and original parameters can have the same name. It is the responsibility of the library author to avoid conflicts by using descriptive parameter names prefixed with abbreviated library part names. For handler parameters and user-defined parameters, Graphisoft has introduced a parameter naming convention in its libraries.

Note: Handlers add extra functionality to library parts (e.g. doors and windows cut holes in walls).

Parameters names with the prefix **ac_** are reserved for special parameters associated with ArchiCAD handlers (e.g. **ac_corner_window**). Check the standard ArchiCAD Library subtype templates for the complete list.

Standard Graphisoft parameter names are marked with the **gs_** prefix (e.g. **gs_frame_pen**). Please check the AC library parts for reference. Use these parameters in your GDL scripts to ensure full compatibility with Graphisoft libraries.

FM_ is reserved for ArchiFM (e.g. **FM_Type**) and **HVAC_** is assigned to HVAC for ArchiCAD parameters (e.g. **HVAC_Manufacturer**).

GDL DATA I/O ADD-ON

The “GDL Data In/Out” Add-On allows you to access a simple kind of database by using GDL commands. Otherwise this Add-On is similar to the “Text GDL In/Out” Add-On.

Description of Database

The database is a text file in which the records are stored in separate lines. The database can be queried and modified based on a single key. The key and the other items are separated by a character (specified in the OPEN command).

The length of the lines does not need to be the same and even the number of columns in the records may be different.

If a database is open for writing then there should be enough space beside the database file for duplicating the whole file.

Opening and closing a database may be time consuming, so consecutive closing and opening of a database should be avoided.

Large databases (with more than some hundred thousand records) should be ordered by the key values.

A database can be opened, queried, modified and closed by this Add-On using the OPEN, INPUT, OUTPUT and CLOSE GDL commands.

Opening a Database

```
channel = OPEN (filter, filename, paramstring)
```

filter: the internal name of the Add-On, in this case "DATA"

filename: the name of the database file to be opened

paramstring: add-on specific parameter, contains separator characters and file opening mode parameters

Opens the database. If the database file is to be opened for modification and the file does not exist, it creates a new file. If the database file is to be opened for reading and the file does not exist, an error message is displayed.

Its return value is a positive integer that will identify the specific database. This value will be the database's future reference number.

If the database is opened before open command, it will generate a channel number only.

The paramstring may contain the following:

SEPARATOR: after the keyword between single quotation marks (") you can define a character that you want to use in your text file (both in case of writing and reading) for the separation of columns. A special case is the tabulator character ('\t').

MODE: after the keyword the mode of opening has to follow.

There are three modes of opening:

- RO (read only)
- WA (read, modify)
- WO (read, modify) Empties the database if exists.

DIALOG: the 'filename' parameter is handling as a file-identifier, otherwise it is a full-path-name. The file-identifier is a simple string, which will be corresponded to an existing file by the Add-On during a standard 'Open/Save as' dialog. These correspondence is stored by the Add-On and do not ask again except the file is not available. If the open-mode is read only, the Add-On put an Open dialog to select an existing document. Otherwise the Add-On put an alert-dialog to select between the 'Create' and 'Browse' option:

- Browse: search an existing data-file (Open dialog)
- Create: create a new data-file (Save as Dialog).

Always put a comma (,) between SEPARATOR, MODE and DIALOG.

LIBRARY: If the LIBRARY keyword is present in the parameter string, the data file has to be in the loaded library.

If you use keywords that don't exist, if the separator characters given are wrong or if there is nothing in the parameter string, the extension will use the default settings:

```
SEPARATOR = '\t', MODE = RO.
```

Example:

```
ch1 = OPEN ("DATA", "file1",
           "SEPARATOR=';', MODE = RO", DIALOG)
ch2 = OPEN ("DATA", "file2", "")
ch3 = OPEN ("DATA", "newfile",
           "SEPARATOR = '\t', MODE = WA")
```

Reading Values from Database

```
INPUT (channel, recordID, fieldID, var1  
      [, var2, ...])
```

recordID: key value (numeric or string)

fieldID: the column number in the given
record (the smallest number : 1
refers to the item after the key
value)

var1,...: variables to receive the read record items

Queries the database based on the key value.

If it finds the record then it reads items from the record starting from the given column and puts the read values into the parameters in sequence.

In the parameter list there has to be at least one value. The values can be of numeric or string type independently of the parameter type defined for them. The return value is the number of successfully read values.

If there are more parameters than values, the parameters without corresponding values will be set to zero. In case of empty columns (i.e. if there is nothing between the separator characters) the parameters will be set to zero.

If it finds no record it returns (-1).

Example:

```
nr = INPUT (ch1, "key1", 1, v1, v2, v3)  
! input of three values from the first column  
! (after the key) of the record containing the  
! key "key1"  
PRINT nr, v1, v2, v3
```


Writing Values into Database

OUTPUT channel, recordID, fieldID, expr1 [, expr2, ...]

recordID: key value (numeric or string)

fieldID: flag: specify 0 (or ≤ 0) to delete a record, specify 1 (or > 0) to create or modify a record

expr1, ...: new item values of the found or new record in case of deletion these values are ignored

In case of record creation or modification it sets the record belonging to the given key value. The record will contain the given values in the same sequence as they appear in the command. The values can be of numeric or string type. There has to be at least one expression.

In case of deletion the record belonging to the given key value is removed from the database. The expression values are ignored, however at least one should be specified.

Example:

```
string = "Date: 19.01.1996"
```

```
a = 1.5
```

```
OUTPUT ch2, "keyA", 1, "New record"
```

```
OUTPUT ch2, "keyA", 1, "Modified record"
```

```
OUTPUT ch2, "keyA", 0, 0 ! deletes the record
```

```
OUTPUT ch2, "keyB", 1, a, string
```

Closing Database

CLOSE channel

channel: channel value

Closes the database identified by the channel value.

GDL DATE'TIME ADD-ON

The Date'Time extension allows you to set various formats for the current date and time set on your computer.

The Add-On works the same way the GDL file operations. You have to open a channel, read the information and close the channel.

This Add-On is also available by using the REQUEST GDL command, in which case the sequence of commands OPEN, INPUT and CLOSE is called internally. This is the simplest way to obtain the date/time information, with just a single GDL command line:

```
REQUEST ("Date'Time", format, datetimestring)
```

The second parameter of the Request function is the same as that described in the OPEN function paramstring parameter.

Opening Channel

```
channel = OPEN (filter, filename, paramstring)
```

filter: the internal name of the Add-On, in this case "Date'Time"

filename: unused (there is no need to open any file to get the system date and time)

paramstring: add-on specific parameter, contains the desired output format of the date and time

Its return value is a positive integer that will identify the opened channel. This value will become the channel's future reference number. The paramstring can contain specifiers and other characters.

The specifiers are replaced with date and time values as follows:

%a	abbreviated weekday name
%A	full weekday name
%b	abbreviated month name
%B	full month name
%c	date and time in the form: 01:35:56 PM Wednesday, March 27, 1996
%d	day of the month as a decimal number (01-31)
%H	hour (24-hour clock), as a decimal number (00-23)
%I	hour (12-hour clock), as a decimal number (01-12)
%j	day of the year, as a decimal number (001-366)
%m	month, as a decimal number (01-12)
%M	minute, as a decimal number (00-59)
%P	AM/PM designation for a 12-hour clock
%S	second, as a decimal number (00-61)
%U	week number of the year (with Sunday as the first day of the first week), as a decimal number
%w	weekday, as a decimal number (0 (Sunday)-6 (Saturday))
%W	week number of the year (with Monday as the first day of the first week), as a decimal number (00-53)
%x	date in the form Wednesday, March 27, 1996
%X	time in the form 01:35:56 PM
%y	year without century, as a decimal number (00-99)
%Y	year with century, as a decimal number
%Z	GDL ignores this specifier. According to the standard, it prints the time zone if it can be determined
%%	the % character

Example:

```
dstr = ""
ch = OPEN ("DateTime", "", "%w/%m/%d/%Y, %H:%M%P")
n = INPUT (ch, "", "", dstr)
CLOSE (ch)
PRINT dstr !it prints 3/03/27/1996, 14:36 PM
```

Reading Information

```
n = INPUT (channel, "", "", datetimestr)
```

channel: channel value

datetimestr: string type value

It reads a string type value which represents the date and/or time in the format given at the OPEN sequence. The second and third parameters are unused (they can be empty strings or 0-s as well).

The return value is the number of successfully read values, in this case 1.

Closing Channel

```
CLOSE channel
```

Closes the channel identified by the channel value.

GDL FILE MANAGER I/O ADD-ON

The “GDL File Manager In-Out” Add-On allows you to scan a folder for the contained files/subfolders from a GDL script.

Specify the folder you would like to scan by using the OPEN command.

Get the first/next file/folder name in the specified folder by using the INPUT command.

Finish folder scanning by using the CLOSE command.

Specifying Folder

```
channel = OPEN (filter, filename, paramstring)
```

channel : folder id

filter : the internal name of the Add-On, in this case "FileMan"

filename : - the name of folder to be scanned (OS dependent path) - folder id string (in DIALOG mode - see later)

paramstring : Add-on specific parameter.

The parameters in paramString must be separated by commas (,).

1. parameter: FILES/FOLDERS

What would you like to search for?

2. parameter (optional): DIALOG

Indicates that the folder is given by a file id string instead of a file path.

When this is the case, at the first time (and each time when the corresponding file path seems to be invalid) the user will be faced a dialog box to set the id string - file path correspondence, which will be stored.

For example, the line

```
folder = OPEN( "FileMan", "c:\\\\
```

opens the root directory of the C drive (on a PC) for file-scanning.

Getting File/Folder Name

```
n = INPUT (channel, recordID, fieldID, var1 [,
          var2, ...])
channel : folder id (returned by the OPEN command)
recordID : 0 (reserved for further development)
fieldID : 0 (reserved for future development)
var1, ... : variable(s) to receive the file/folder name(s)
n : the number of succesfully filled variables
```

For example, the line

```
n = INPUT (folder, 0, 0, fileName)
```

fetches the next file name from the specified folder, and returns 1.

If there are no more files/subfolders the variable n will be set to zero.

Finishing Folder Scanning

```
CLOSE (channel)
```

Closes the folder identified by the channel value.

Example: Listing a single folder

The following code segment (as the 2D script section of an object, for example) lists the files in the folder specified by the MyFavouriteFolder identifier. At first usage, the user will have to assign an existing folder to this identifier. Later, MyFavouriteFolder id will represent that folder.

```
topFolder = open( "FileMan", "MyFavouriteFolder", "files, dialog" )
y = 0
n = input( topFolder, 0, 0, fileName )
while n = 1 do
  text2 0, y, fileName
  y = y - 0.6
  n = input( topFolder, 0, 0, fileName )
endwhile
close( topFolder )
```

GDL TEXT I/O ADD-ON

The GDL Text In/Out Add-On allows you to open external text files for reading/writing and to manipulate them by putting/getting values from/to GDL scripts.

This Add-On interprets the strings on the parameter list of the OPEN, INPUT, OUTPUT commands from the GDL script.

It assumes that a folder named “ArchiCAD Data Folder” exists beside ArchiCAD for user defined files. (The name of this folder is defined in the Add-On resource fork, therefore it can be localized.) If a folder with that name doesn't exist, the Add-On will create one. The folder can contain subfolders where the extension will look for existing files. It can read and write TEXT type files.

Opening File

```
channel = OPEN (filter, filename, paramstring)
```

filter: the internal name of the Add-On, in this case "TEXT"

filename: the name of the file to be opened

paramstring: add-on specific parameter, contains separator characters and file opening mode parameters

Opens the file. If the file into which you want to write doesn't exist, it creates the file. If a file to be read doesn't exist, an error message is displayed.

Its return value is a positive integer that will identify the specific file. This value will be the file's future reference number.

The paramstring can contain the following:

SEPARATOR: after the keyword between single quotation marks (‘’) you can assign a character to use in the text file (for both writing and reading) to separate columns.

Special cases are the tabulator (‘\t’) and the new row (‘\n’) characters.

MODE: the mode of opening has to follow this keyword. There are only three modes of opening:

RO (read only)

WA (write only, append at the end of the file)

WO (write only, overwrite) the data previously stored in the file will be lost!

A file cannot be open for reading and writing at the same time.

DIALOG: If this keyword is present, a dialog box will appear in which you can enter a file name.

FULLPATH: If this keyword is present, the file name will be interpreted as a full path name.

LIBRARY: If this keyword is present, the data file must be in the loaded library.

Always put a comma (,) between the keywords.

If you use keywords that don't exist, if the separator characters given are wrong or there is nothing in the parameter string, the extension uses the default settings: SEPARATOR = '\t', MODE=RO.

Example:

```
ch1 = OPEN ("TEXT", "file1", "SEPARATOR = ';', MODE = RO")
ch2 = OPEN ("TEXT", "file2", "")
ch3 = OPEN ("TEXT", "file3", "SEPARATOR = '\n', MODE = WO")
```

Reading Values

```
INPUT (channel, recordID, fieldID,
      var1 [, var2, ...])
```

channel: channel value

recordID: the row number (numeric or string)

fieldID: the column number in the given row

var1,...: variables to receive the read record items

It reads as many values from the given starting position of the file identified by the channel value as many parameters are given. In the parameter list there has to be at least one value. The function puts the read values into the parameters in sequence. The values can be of numeric or string type independently of the parameter type defined for them.

The return value is the number of successfully read values, in case of end of file (-1).

Both the row and the column numbers have to be positive integers, otherwise you will get an error message.

If the row or column numbers are incorrect, the input will not be carried out. (n = 0)

If the row and the column can be identified, as many values shall be input from the given starting position as many parameters are given, or if there are more parameters than values, the parameters without corresponding values will be set to zero.

In case of empty columns (i.e. if there is nothing between the separator characters) the parameters will be set to zero.

Example:

```
nr = INPUT (ch1, 1, 1, v1, v2, v3) ! input of three values from the first ! column of the
first row
PRINT nr, v1, v2, v3
```

Writing Values

OUTPUT channel, recordID, fieldID, expr1 [, expr2, ...]

channel: channel value

recordID: if positive, the output values will be followed by a new row

fieldID: no role, its value is not used

expr1: values to output

Outputs as many values into the file identified by the channel value from the given position as many expressions are defined. There has to be at least one expression. The types of the output values are the same as those of the expressions.

In case of a text extension, the OUTPUT will either (depending on the mode of opening) overwrite the file or add to the end of the file the given expressions to consecutive positions using between them the separator characters defined when opening the file. In this case, the given position is not interpreted.

The recordID is used to direct the new rows in the output.

If the recordID is positive, the output values will be followed by a new row, otherwise the last value will be followed by a separator character.

Example:

```
string = "Date: 19.01.1996"
```

```
a = 1.5
```

```
OUTPUT ch2, 1, 0, string ! string followed by a new row
```

```
OUTPUT ch2, 0, 0, a, a + 1, a + 2! separator character after a + 2 ! without new row
```


Closing File

CLOSE channel

channel: channel value

Closes the data base identified by the channel value.

Example

A GDL object that will simply copy the contents of the "f1" file both into the "f2" and the "f3" files, but will write all the values tabulated in "f1" into a separate row in both "f2" and "f3".

```
ch1 = open ("TEXT", "f1", "mode = ro")
ch2 = open ("TEXT", "f2", "separator = '\n', mode = wo")
ch3 = open ("TEXT", "f3", "separator = '\n', mode = wo")
i = 1
1:
n = input (ch1, i, 1, var1, var2, var3, var4)
if n <> -1 then
output ch2, 1, 0, var1, var2, var3, var4
output ch3, 1, 0, var1, var2, var3, var4
i = i + 1
goto 1
else
goto 2
endif
2:
close ch1
close ch2
close ch3
end
```

PROPERTY GDL ADD-ON

The purpose of this add-on is to make an ArchiCAD property database accessible from GDL scripts. You can open database tables and query their contents, just like you would do it with SQL. You can query single records and multiple records (lists). Note that you cannot modify the database, and you cannot append records to it.

For the detailed description of the property database please refer to the "Calculation Guide" in ArchiCAD Help.

OPEN

Syntax: OPEN ("PROP", "database set name", ["database files"])

Parameters: <database set name> is an arbitrary name that will identify a set of database files in subsequent OPEN calls.

<database files> is a list of text files that are part of the property database. This parameter is optional, if you have previously assigned a <database set name> to the files you would like to read. The order of the files is fixed: <key file>, <component file>, <descriptor file>, <unit file>. You don't need to give full paths, because ArchiCAD will look up these files for you in the active libraries. If you use long filenames, put them between quotes (' or ").

Return value: channel number

Opens a communication channel to the given database files. The content of the database files are read into memory for faster access. As long as it is open modifications to the property database will not be accessible from this add-on. This is usually not a problem though.

Examples:

1.

```
channel = OPEN ("PROP", "sample", "'AC 8_KEY.txt', 'AC 8_COMP.txt', 'AC 8_DESC.txt', 'AC 8_UNIT.txt'")
```

This opens a database that consists of the files above (those are the files of the ArchiCAD 7.0 Property database), and names it "sample". Note that inside the third parameter you must use a different quotation character (you can use " and ').

2.

```
channel = OPEN ("PROP", "sample", "")
```

This command can be issued after explicitly opening the database files (like in example 1), but before closing it. This lets you use the explicit command at one place in the Master_GDL script, and use the shorter version later.

CLOSE

Syntax: CLOSE (channel_number)

Return value: none

Closes the previously opened communication channel.

INPUT

Syntax: INPUT (channel_number, "query type", "field list", variable1[, ...])

Parameters: <channel number> is a valid communication channel number given by a previous OPEN command.

<query type> specifies the query you would like to execute. The add-on understands the following keywords:

Single-record queries: KEY, <keycode> – query the record from the key database where <keycode> is the value of the keycode attribute.

Valid fields: KEYCODE, KEYNAME

UNIT, <unitcode> – query the record from the unit database where <unitcode> is the value of the unit code attribute.

Valid fields: UNITCODE, UNITNAME, UNITFORMATSTR

COMP, <keycode>, <code> – query the record from the unit database where <keycode> is the key code attribute value, and <code> is the component code attribute value.

Valid fields: KEYCODE, KEYNAME, CODE, NAME, QUANTITY, QUANTITYSTR, UNITCODE, UNITNAME, UNITFORMATSTR

DESC, <keycode>, <code> – query the record from the unit database where <keycode> is the key code attribute value, and <code> is the descriptor code attribute value.

Valid fields: KEYCODE, KEYNAME, CODE, NAME, NUMOFLINES, FULLNAME

Listing queries: KEYLIST – list all records in the key database

Valid fields: KEYCODE, KEYNAME

UNITLIST – list all records in the unit database

Valid fields: UNITCODE, UNITNAME, UNITFORMATSTR

COMPLIST[, <keycode>] – list all records in the component database, or if <keycode> is given, then only those records are listed whose keycode equals <keycode>

Valid fields: KEYCODE, KEYNAME, CODE, NAME, QUANTITY, QUANTITYSTR, UNITCODE, UNITNAME, UNITFORMATSTR

DESCLIST[, keycode] – list all records in the descriptor database, or if <keycode> is given, then only those records are listed whose keycode equals <keycode>

Valid fields: KEYCODE, KEYNAME, CODE, NAME, NUMOFLINES, FULLNAME

COMPDESCLIST[, <keycode>] – list all records in the component and the descriptor database, or if <keycode> is given, then only those records are listed whose keycode equals <keycode>.

Valid fields: ISCOMP, KEYCODE, KEYNAME, CODE, NAME, QUANTITY, QUANTITYSTR, UNITCODE, UNITNAME, UNITFORMATSTR, NUMOFLINES, FULLNAME

Use this query with care! If either field is not valid in a database (eg. FULLNAME in the component database) it will be simply left out from the resulting list (you should be aware of that).

<field list> lists the database attributes whose values you would like to see in the output. If the output is a list, it will be sorted in the order of the fields listed here.

The following fields can be used:

KEYCODE – key code attribute.

Type: string

Usable in queries: KEY, COMP, DESC, KEYLIST, COMPLIST, DESCLIST, COMPDESCLIST

KEYNAME – key name attribute.

Type: string

Usable in queries: KEY, COMP, DESC, KEYLIST, COMPLIST, DESCLIST, COMPDESCLIST

UNITCODE – unit code attribute

Type: string

Usable in queries: UNIT, COMP, UNITLIST, COMPLIST, COMPDESCLIST

UNITNAME – unit name attribute

Type: string

Usable in queries: UNIT, COMP, UNITLIST, COMPLIST, COMPDESCLIST

UNITFORMATSTR – GDL format string of the unit

Type: string

Usable in queries: UNIT, COMP, UNITLIST, COMPLIST, COMPDESCLIST

CODE – component or descriptor code attribute (depends on the query)

Type: string

Usable in queries: COMP, DESC, COMPLIST, DESCLIST, COMPDESCLIST

NAME – name of component or the first line of a descriptor record

Type: string

Usable in queries: COMP, DESC, COMPLIST, DESCLIST, COMPDESCLIST

QUANTITY – quantity of a component as a number (for calculations)

Type: number

Usable in queries: COMP, COMPLIST, COMPDESCLIST

QUANTITYSTR – quantity of a component in string format

Type: string

Usable in queries: COMP, COMPLIST, COMPDESCLIST

NUMOFLINES – number of lines in a descriptor record

Type: number

Usable in queries: DESC, DESCLIST

FULLNAME – the whole descriptor record

Type: string(s)

Usable in queries: DESC, DESCLIST

ISCOMP – tells you whether the next record is a component or a descriptor

Type: number (1 if component, 0 if descriptor)

Usable in queries: COMPDESCLIST

<variables> will hold the result of the query upon completion. You can list several variables if you know exactly how many you need (eg. with single queries) or you can specify a dynamic array. The records are listed sequentially.

Examples:

1.

```
INPUT (channel, "KEY, 001", "KEYNAME", keyname)
```

This is a simple query: the name of the key with '001' code is put into the keyname variable.

2.

```
INPUT (channel, "DESC, 004, 10", "NUMOFLINES, FULLNAME", desc_txt)
```

The descriptor record with keycode '004' and code '10' is processed, the number of lines of the description text and the text itself is put into the desc_txt array. The result is:

```
desc_txt[1] = <numoflines> (number)
```

```
desc_txt[2] = <first row of description> (string)
```

```
...
```

```
desc_txt[<numoflines+1>] = <last row of description
```

3.

```
INPUT (channel, "COMPLIST", "NAME, KEYNAME, QUANTITY", comp_list)
```

Create a component list, sort it by the name field, then by the keyname and finally by the quantity field and put it into the comp_list array. The result is:

```
complist[1] = <name1> (string)
```

```
complist[2] = <keyname1> (string)
```

```
complist[3] = <quantity1> (number)
```

```
complist[4] = <name2> (string)
```

```
... etc.
```

4.

```
INPUT (channel, "COMPDESCLIST, 005", "ISCOMP, KEYNAME, NAME, QUANTITY", x_list)
```

Creates a common component and descriptor list, which means that records from both tables are listed where <keycode> is '005'. The output is:

```
x_list[1] = 0 (number, 0 -> it is descriptor)
```

```
x_list[2] = <name1> (string -> descriptors do not have <keyname> field, so it is left out)
```

```
x_list[3] = 0 (number, descriptors do not have quantity field)
```

```
...
```

```
x_list[(n*2)-1] = 1 (number -> there were n-1 descriptors listed, now come the components)
```

```
x_list[n*2] = <keyname_n> (string)
```

```
... etc.
```

OUTPUT

This command is not implemented in this add-on, since property databases are read-only.

GDL XML EXTENSION

This extension allows reading, writing and editing XML files. It implements a subset of the Document Object Model (DOM) interface. XML is a text file that uses tags to structure data into a hierarchical system, similar to HTML. An XML document can be modeled by a hierarchical tree structure whose nodes contain the data of the document. The following node types are known by the extension:

- *Element*: what is between a start-tag and an end-tag in the document, or for an empty-element it can be an empty-element tag. Elements have a name, may have attributes, and usually but not necessarily have content. It means that element type nodes can have child nodes. Attributes are held in an attribute list where each attribute has a different name and a text value.
- *Text*: a character sequence. It cannot have child nodes.
- *Comment*: text between the comment delimiters: `<!-- the comment itself -->`. In the text of the comment each `'` character must be followed by a character different from `'`. It also means that the following is illegal: `<!-- comment --->`. Comment type nodes cannot have child nodes.
- *CDATASection*: text between the CDATA section delimiters: `<![CDATA[the text itself]]>`. In a CDATA section characters that have special meaning in an XML document need not (and must not) be escaped. The only markup recognized is the closing `]]>`. CDATA section nodes cannot have child nodes.
- *Entity-reference*: reference to a predefined entity. Such a node can have a read-only subtree and this subtree gives the value of the referenced entity. During the parsing of the document it can be chosen that entity references are translated into text nodes.

On the top level it is obligatory to have exactly one element type node (the root), and there can be several comment type nodes, as well. The document type node of the DOM interface is not available through the extension's interface.

For each node in the tree there is a name and a value string associated whose meanings depend on the type of the node:

	name:	value:
Element:	name of the tag	"" (empty string)
Text:	"#text"	the text content of the node
Comment:	"#comment"	the text content of the node
CDATASection:	"#cdata-section"	the text content of the node
Entity-reference:	name of the referenced entity	"" (empty string)

For each node type the extension defines string keywords that can be passed to the extension in certain instructions:

Element:	ELEM
Text:	TEXT
Comment:	CMT
CDATA section:	CDATA
Entity reference:	EREF

The success or error code of an OPEN, INPUT or OUTPUT command can be retrieved by the GetLastError instruction of the INPUT command.

Opening XML Document

The OPEN command:

`channel = OPEN (filter, filename, parameter_string)`

filter: file extension. This should be 'XML'.

filename: name and path of the file to open (or create), or an identifier name if the file is opened through a dialog box and the file's location is given by the user.

parameter_string: a sequence of character flags that determine the open-mode:

- **'r'**: open in read-only mode. In general only the INPUT command can be used.
- **'e'**: entity references are not translated into text nodes in the tree. Without this flag there are no entity-references in the document structure.
- **'v'**: validity check is performed during reading in and writing out. If a DTD exists in the document, the document's structure must agree with it. Without this flag a well-structured but invalid document can be read in and written out without error message.
- **'n'**: create a new file. If the file exists, the open will fail. (After the OPEN the CreateDocument instruction must be the first to execute.)
- **'w'**: overwrite file with empty document if it exists. If it doesn't exist, a new file will be created. (After the OPEN the CreateDocument instruction must be the first to execute.)
- **'d'**: the file is obtained from the user in a dialog box. In later runs it will be associated with the identifier given in the filename parameter of the OPEN command. (If the identifier is already associated to a file, the dialog box will not be opened to the user.)
- **'f'**: the filename parameter contains a full path.
- **'l'**: the file is in the loaded library parts.

channel: used to identify the connection in subsequent I/O commands.

If you want to open an existing XML file for modification, then none of the 'r', 'n' and 'w' flags must be set in the parameter string. Only one of the 'd', 'f' and 'l' flags should be set. If none of these flags is set then filename is considered to be a path relative to the user's documents folder.

Reading XML Document

DOM is an object-oriented model that cannot be adapted to a BASIC-like language like GDL directly. To represent the nodes in the hierarchy tree we define position descriptors. When we want to walk through the nodes of the tree, first we have to request a new position descriptor from the extension. Originally a new descriptor points to the root element. The descriptor is in fact a 32 bit identification number whose value has no interest for the GDL script. The position it refers to can be changed as we move from one node in the tree to another.

The INPUT command:

INPUT (ch, recordID, fieldID, var1, var2...)

ch: channel returned by the OPEN command

recordID: instruction name plus parameters

fieldID: usually a position descriptor var1, var2,...: optional list of variables receiving returned data.

INPUT instructions:

1 GetLastError: retrieve the result of the last operation

recordID: "GetLastError"

fieldID: ignored

return values:

var1: error code / ok

var2: the explanation text of error / ok

2 NewPositionDesc: request for a new position descriptor

recordID: "NewPositionDesc"

fieldID: ignored

return value: var1: the new position descriptor (initially refers to the root)

3 CopyPositionDesc: request for a new position descriptor whose starting node is taken from another descriptor.

recordID: "CopyPositionDesc"

fieldID: an existing position descriptor

return value: var1: the new position descriptor (initially refers to where the descriptor given in fieldID refers to)

4 ReturnPositionDesc: when a position descriptor is no longer needed.

recordID: "ReturnPositionDesc"

fieldID: the position descriptor.

Call this instruction when a position descriptor received from the NewPositionDesc or CopyPositionDesc instructions is no longer used.

5 **MoveToNode:** change the position of a descriptor. (and retrieve the data of the new node)

This instruction can be used for navigating in the tree hierarchy.

recordID: "MoveToNode searchmode nodename nodetype nodenumber"

fieldID: position descriptor

searchmode (or movemode): the nodename parameter must contain a path that determines an element or entity reference node in the xml document.

The path is relative to the node given in fieldID. The delimiter is the '.' character (which is otherwise an accepted character in an element's name so this doesn't work for all cases). The '..' string in the path means a step to the parent node. The starting node can be different from an element or entity reference node, in which case the path must begin with '..' to step back. If there are several element nodes on the same level with the same name then the first one is chosen.

For the following move-modes the rest of the parameters must not be present:

ToParent: moves to the parent of the node given in fieldID.

ToNextSibling: moves to the next node on the same level.

ToPrevSibling: moves to the previous node on the same level.

ToFirstChild: moves to the first descendant of the fieldID node.

ToLastChild: moves to the last descendant of the fieldID node.

The following are the search-modes for which the rest of the parameters may occur, but they have default values if not present:

FromNextSibling: searching starts from the next node on the same level and it moves forward.

FromPrevSibling: searching starts from the node before fieldID and it moves backward on the same level.

FromFirstChild: searching starts from the first descendant of the fieldID node and moves forward.

FromLastChild: searching starts from the last descendant of the fieldID node and moves backward.

nodename: the searching considers those nodes only whose name or value matches nodename. The * and ? characters in nodename are considered as wildcard characters. For element and entity reference type nodes the name is compared, while for text, comment and CDATA section nodes the value is compared. Default value: *

nodetype: the searching considers those nodes only whose type is allowed by nodetype. The * means all types are allowed. Otherwise the type keywords can be combined with the + character to form the nodetype (it must be one word without spaces, like TXT+CDATA.) The default value is *

nodenumber: if there are several matching nodes, this gives the number of the searched node in the sequence of matching nodes. (Starts from 1) Default value: 1

return values:

var1: name of the node

var2: value of the node

var3: type keyword of the node

Example:

We want to move backwards on the same level to the 2nd node that is an element or an entity reference and whose name starts with K:

```
INPUT (ch, "MoveToNode FromPrevSibling K* ELEM+EREF 2", posDesc, name, val, type)
```

6 GetNodeData: retrieve the data of a given node.

recordID: "GetNodeData"

fieldID: the position descriptor

return values:

var1: name of the node

var2: value of the node

var3: type keyword of the node

7 NumberofChildNodes: gives the number of child nodes of a given node

recordID: "NumberofChildNodes nodetype nodename"

fieldID: position descriptor

The following optional parameters can narrow the set of child nodes considered:

nodetype: allowed node types as defined in the MoveToNode instruction

nodename: allowed node names or values as defined in the MoveToNode instruction

return values:

var1: number of child nodes

8 NumberofAttributes: returns the number of attributes of an element node.

recordID: "NumberofAttributes attrname"

fieldID: position descriptor (must refer to an element node)

attrname: if present, it can narrow the set of attributes considered as only those attributes will be counted whose names (and not the values) match attrname. In attrname the * and ? characters are considered wildcard characters.

return values:

var1: number of attributes

9 GetAttribute: return the data of an attribute of an element node

recordID: "GetAttribute attrname attrnumber"

fieldID: position descriptor (must refer to an element node)

optional parameters:

attrname: give the name of the attribute. The * and ? are considered wildcard characters. Default value: *

attrnumber: If several attribute matches attrname, attrnumber chooses the attribute in the sequence of matching attributes. (Counting starts from 1.) Default value: 1

return values:

var1: value of the attribute

var2: name of the attribute

10 Validate: check the validity of the document.

The validity is not checked during a document modification instruction. It is checked during writing back the file to disk if the 'v' flag was set in the open-mode string. A validity check can be forced any time by the Validate instruction, however it can consume considerable amount of time and memory so it is not advisable to do so after every modification.

recordID: "Validate"

fieldID: ignored

Modifying XML Document

OUTPUT (ch, recordID, fieldID, var1, var2...)

ch: channel returned by the OPEN command

recordID: instruction name plus parameters

fieldID: usually a position descriptor

var1, var2,...: additional input data

OUTPUT instructions:

Most of the OUTPUT instructions are invalid for files opened in read-only mode.

1 CreateDocument:

recordID: "CreateDocument"

fieldID: ignored

var1: name of the document. This will be the tagname of the root element, as well.

CreateDocument is allowed only if the file was opened in new-file or overwrite mode. In these modes this instruction must be the first to be executed in order to create the XML document.

2 NewElement: insert a new element type node in the document

recordID: "NewElement insertpos"

fieldID: a position descriptor relative to which the new node is inserted

var1: name of the new element (element tag-name)

insertpos can be:

AsNextSibling: new element is inserted after the position given in fieldID

AsPrevSibling: new element is inserted before the position given in fieldID

AsFirstChild: new element is inserted as the first child of the node given in fieldID (which must be an element node)

AsLastChild: new element is inserted as the last child of the node given in fieldID (which must be an element node)

3 NewText: insert a new text node in the document

recordID: "NewText insertpos"

fieldID: position descriptor

var1: text to be inserted

*See also the NewElement instruction.***4 NewComment:** insert a new comment node in the document

recordID: "NewComment insertpos"

fieldID: position descriptor

var1: text of the comment to be inserted

*See also the NewElement instruction.***5 NewCDATASection:** insert a new CDATA section node in the document

recordID: "NewCDATASection insertpos"

fieldID: position descriptor

var1: text of the CDATA section to be inserted

*See also the NewElement instruction.***6 Copy:** make a copy of a subtree of the document under some node

recordID: "Copy insertpos"

fieldID: position descriptor relative to which the subtree is inserted

var1: position descriptor giving the node of the subtree to be copied

insertpos: same as in the NewElement instruction

The copied subtree remains unchanged. Position descriptors pointing to some node in the copied subtree will point to the same node after the copy.

7 Move: replace some subtree in the document to some other location

recordID: "Move insertpos"

fieldID: position descriptor relative to which the subtree is inserted

var1: position descriptor giving the node of the subtree to be moved

insertpos: same as in the NewElement instruction

The original subtree is deleted. Position descriptors pointing to some node in the moved subtree will point to the same node in the new position of the subtree.

8 Delete: delete a node and its subtree from the document

recordID: "Delete"

fieldID: position descriptor giving the node to delete

All position descriptors pointing to some node in the deleted subtree become invalid.

9 SetNodeValue: change the value of a node

recordID: "SetNodeValue"

fieldID: position descriptor, it must refer to either a text, a comment or a CDATA section type node

var1: new text value of the node

10 SetAttribute: change an attribute of an element node or create a new one

recordID: "SetAttribute"

fieldID: position descriptor, it must refer to an element type node

var1: name of the attribute

var2: text value of the attribute

If the element already has an attribute with this name then its value is changed, otherwise a new attribute is added to the element's list of attributes.

11 RemoveAttribute: removes an attribute of an element node

recordID: "RemoveAttribute"

fieldID: position descriptor, it must refer to an element type node

var1: name of the attribute to remove

12 Flush: write the current document back to file

recordID: "Flush"

fieldID: ignored

If the file was opened in validate mode, then only a valid document is saved.

13 ChangeFileName: associate another file with the current document

recordID: "ChangeFileName"

fieldID: new file path

var1: gives how fieldID should be interpreted. If var1 is an empty string, fieldID contains a path relative to the user's documents folder. 'd' means the file's location is obtained from the user from a file dialog box (see the command open-mode flags "*OPEN*" on page 292). 'l' means the file is taken from the loaded libraries. 'f' means fieldID contains a full path.

This instruction can be called even if the file was opened in read-only mode. In this case after the execution the document loses the read-only attribute, so it can be modified and saved to the new file location.

Error codes and messages:

0: "Ok"
-1: "Add-on Initialization Failed"
-2: "Not Enough Memory"
-3: "Wrong Parameter String"
-4: "File Dialog Error"
-5: "File Does Not Exist"
-6: "XML Parse Error"
-7: "File Operation Error"
-8: "File Already Exists"
-9: "This channel is not open"
-10: "Syntax Error"
-11: "Open Error"
-12: "Invalid Position Descriptor"
-13: "Invalid Node Type for this Operation"
-14: "No Such Node Found"
-15: "Internal Error"
-16: "Parameter Error"
-17: "No Such Attribute Found"
-18: "Invalid XML Document"
-19: "Unhandled Exception"
-20: "Read-Only Document"
-21: "CreateDocument Not Allowed"
-22: "Document Creation Failed"
-23: "Setting NodeValue Failed"
-24: "Move Not Allowed"
-25: "Delete Not Allowed"
-26: "SetAttribute Not Allowed"
-27: "Format File Error"
-28: "Insertion (or Copy) Not Allowed"
-29: "Node Creation Failed"
-30: "Bad String"
-31: "Invalid Name"

INDEX

Numerics

2D script 13
2D symbol
 generating Door/Window ~ 243
3D script 13

A

ABS 193
ACS 194
ADD 29
ADD2 27
ADDGROUP 109, 112
ADDITIONAL_DATA 175
ADDX 28
ADDY 28
ADDZ 28
advanced commands and features 18
AND 192
arc definition 121
ARC2 121
ArchiCAD 14
 Component Lists in ~ 179
 Element Lists in ~ 128
ArchiFM 14
ARMC 58
ARME 59
arrays for parameters 25
ASN 194
ATN 194
attributes
 defining ~ 21

B

BackgroundColor 127
base 10 logarithm 194
basic syntactic elements 23
BEAM 53
beam definition 53
BINARY 13, 115
binary 2D data 13
binary 3D data 13
binary data reference 179
binary properties data 14

BINARYPROP 14, 179
bitmap pattern 166
BITSET 195
BITTEST 195
BLOCK 33
block definition 33
bodies 94
BODY 99
body definition with primitives 99
Boolean difference 110
BPRISM_ 40
BREAKPOINT 208
breakpoint definition in script 208
BRICK 33
BWALL_ 48

C

CALL 212
CEIL 193
changing element type in listing 179
Character strings 24
characters available in GDL scripts 23
circle definition 121
CIRCLE2 121
CLOSE 214
closing a file 214
colons in GDL scripts 23
commas in GDL scripts 23
comments 14
complex transformation matrix 31
COMPONENT 178, 179
components 14
 ~ definition 178
 ~ reference 179
conditions 204
CONE 35
cone frustum definition 35
COONS 85
Coons patch generation 85
COORD 97
coordinate system definition
 local ~ for primitives 97
coordinate transformations 27
 basic ~ 16

 intermediate level ~ 17
COS 194
CPRISM_ 39
CSLAB_ 46
curved prism definition 40
curved wall definition 48
CUSTOM 181
CUTFORM 107
CUTPLANE 101
CUTPOLY 103
CUTPOLYA 105
CUTSHAPE 107
cutting polygon definition 103, 105
cutting shape definition 107
CWALL_ 46
CYLIND 33
cylinder definition 33

D

database definition 177
DATABASE_SET 177
DEFINE FILL 175
DEFINE FILL_A 175
DEFINE FILLA 167
DEFINE LINE_TYPE 170, 175
DEFINE MATERIAL 158, 175
DEFINE STYLE 171
DEFINE SYMBOL_FILL 169, 175
DEFINE SYMBOL_LINE 171, 175
DEL 31
DEL TOP 31
DESCRIPTOR 178
descriptors 14
 ~ definition 178
 ~ reference 178
DIALOG 288
DIM 189
DO 204
draw order definition 157
DRAWINDEX 157
DRAWING 180
drawing
 ~ definition for Element Lists 128
 ~ reference in 2D script 180

DRAWING2 128
DRAWING3 128
DRAWING3{2} 129

E

EDGE 96
edge definition 96
edges 94
ELBOW 59
ELLIPS 34
ELSE 206
END 23, 208
end of script definition 208
ENDGROUP 112
ENDIF 206
ENDWHILE 204
entry level commands 15
exclamation marks in GDL scripts 23
EXIT 23, 208
EXOR 192
EXP 194
expert level scripting 19
exporting values to a file 214
EXTRUDE 65
extruded prism generation 65

F

file operations 213
FILL 157, 164, 212
fill pattern definition
 simple ~ 164
fill pattern setting
 ~ for Section/Elevation windows 155
 ~ in 2D views 157
FILLA 167
flow control statements 18
FOR 203
FPRISM_ 41
FRA 193
FRAGMENT2 13, 126
FULLPATH 288

G

geometric primitives 19
GET 208
global origin 20
global variables 21, 24

GOSUB 23, 206, 207
GOTO 23, 206, 207
GROUP 112

H

half ellipsoid definition 34
HIDEPARAMETER 183
HOTSPOT 131
HOTSPOT2 117, 131
HPRISM_ 43
hybrid 111

I

Identifiers 24
IF 206
importing values from a file 214
IND 196, 238
INPUT 214
INT 193
intermediate commands 16
ISECTGROUP 109, 113
ISECTLINES 110, 113

K

KILLGROUP 113

L

labels 23
LET 149
LGT 194
LIBRARY 288
library parts 13
LIGHT 90
light source definition 90
line definition 117
line type
 ~ definition 170
 ~ setting 158
Line Types 170
LINE_TYPE 158, 170, 212
LINE2 117
lines in scripts 23
local coordinate system 20
local variables 24
LOCK 183
LOG 195

log wall parameters 50
loops 203
Ludolphs constant 194

M

macro calls 19
 ~ definition 212
 ~ for doors/windows 244
macro objects 212
mask values 138
masking rules
 ~ for meshes 57
 ~ for prisms 137
MASS 88
master coordinate system 20
Master Script 13
MASTER_GDL 21, 25, 158, 180
MASTEREND_GDL 21
MATERIAL 154, 158, 212
material definition 158
material setting 154
MAX 195
MESH 57
mesh definition
 equidistant ~ 57
mesh generation 88
MIN 195
MOD 192
MODE 288
MODEL 153, 212
MODEL SURFACE 112
modeling mode setting 153
moving the local coordinate system 28
MUL 29
MUL2 27
MULX 29
MULY 29
MULZ 29

N

natural logarithm 195
NEXT 203
node definition in 3D 95
 ~ with texture origin 95
non-rectangular doors/windows
 ~ in curved walls 253
 ~ in straight walls 246

NOT 195
 NSP 209
 NTR 32
 Numeric expressions 25
O
 OPEN 214
 opening a file 114
 OR 192
 OUTPUT 214
P
 Parameter Script 14
 PARAMETERS 182, 212
 parameters 14, 25
 ~ buffer 208
 ~ in GDL scripts 19
 derived types 25
 locking ~ values 183
 modifying ~ values in GDL 182
 simple types 25
 PEN 212
 pen color setting 152
 PGON 96
 PI 194
 PICTURE 14
 picture element definition 92, 125
 picture polygon definition 97
 PICTURE2 14, 124
 PICTURE2{2} 125
 PIPG 97
 PLACEGROUP 113
 planar polylines
 circle using centerpoint and radius 145
 segment by relative endpoint 140
 POLY2 118
 POLY2_ 119
 POLY2_A 119
 POLY2_B 120
 polygon definition 96, 118
 advanced ~ 120
 polygons 94
 POSITION 179
 preview picture 14
 PRINT 213
 PRISM 35
 prism definition 35

 ~ with hill 41
 ~ with non-parallel upper polygon 43
 extended ~ 39
 extended oblique ~ 45
 extruded general ~ 65
 oblique ~ 45
 PRISM_ 36
 programming language 13
 PROJECT2 126
 PROJECT2{2} 127
 projection of 3D script into 2D symbol 127
 prompt 26
 Properties Script 14
 PYRAMID 68
 pyramid definition 68

Q

quotation marks in GDL scripts 24

R

RADIUS 150, 212
 RANGE 181
 RECT2 118
 rectangle definition 118
 rectangular doors/windows
 ~ in straight walls 244
 REF 178
 REQ 196, 233
 REQUEST 196, 234
 request calls 196
 resetting counter
 ~ for primitive elements 101
 RESOL 212
 RETURN 207
 revolved surface definition 70
 RND 195
 RO 288
 roof definition
 sloped ~ 54
 ROT 30
 ROT2 28
 rotating the coordinate system 30
 ROTX 30
 ROTY 30
 ROTZ 30
 RULED 73
 RULED{2} 74

S

scaling the local coordinate system 29
 script types 13
 SECT_FILL 155
 SEPARATOR 288
 154, 158
 SET FILL 157
 SET LINE_TYPE 158
 SET MATERIAL 154
 SET STYLE 153
 SGN 193
 SHADOW 156, 212
 shadow casting control 156
 simple shapes 15
 SIN 194
 smoothness definition for cylindrical elements
 ~ by approximation 152
 ~ by radius 150
 ~ by resolution 151
 SOLID 153
 solid 111
 solid base 111
 special characters 26
 SPHERE 34
 sphere definition 34
 spline definition 122
 Bézier type ~ 124
 SPLINE2 122
 SPLINE2_A 123
 SPLIT 199
 SPRISM_ 43
 SQR 193
 square brackets in GDL scripts 25
 statements 23
 STEP 181, 203
 storing values in parameter buffer 208
 STR 196
 STR{2} 196
 string expressions
 creating ~ from numeric expressions 196
 format string 197
 length of ~ 200
 position of ~ in each other 200
 splitting ~ 199
 substrings 201
 width of ~ 200
 STRLEN 200

STRSTR 200
STRSUB 201
STW 200
STYLE 153, 171, 212
SUBGROUP 109, 112
subroutines 206, 207
SURFACE 153
surface 111
surface base 111
surface generated by polyline
 ~ from planar curve and space curve 74
 ~ sweeping along space curve path 79, 83
 ~ sweeping space curve path 76
surface of the 3D shape of the object 179
SURFACE3D 179
SWEEP 76
SWEEPGROUP 110, 115

T

TAN 194
TEVE 95
TEXT 93
text definition
 ~ in 2D 125
 ~ in 3D 93
Text Extension 280
text style
 ~ setting 153
TEXT2 125
TEXTURE 162
texture definition 162
THEN 206
TO 203

TOLER 152, 212
transformations
 deleting ~ 31
TUBE 79
tube definition
 ~ starting from another tube 58
 ~ starting from ellipsoid 59
 bending ~ 59
TUBEA 83

U

UI_BUTTON 184
UI_DIALOG 183
UI_GROUPBOX 184
UI_INFIELD 185
UI_NEXT 184
UI_OUTFIELD 185
UI_PAGE 183
UI_PREV 184
UI_SEPARATOR 184
UI_STYLE 185
UNTIL 205
USE 209
user global variables 24
User Interface script 14

V

value assignment 149
value lists 21, 25
VALUES 181
VARDIM1 190
VARDIM2 190

variable 26
variables 24
VARTYPE 214
VECT 95
vectorial hatch 166
vectors 94
 ~ definition 95
VERT 95
vertices 94
volume of the 3D shape of the object 179

W

WA 288
wall definition 46
 curved ~ 48
 extended ~ 50
Wall End 219
WALLHOLE 246
WALLNICHE 250
WHILE 204
WIRE 153
wireframe 111
wireframe base 111
wireframe modeling mode 153
WO 288
writing out arguments 213

X

XFORM 30
XWALL_{2} 52