17 The Tribon Geometry Macro Facility

17.1 General

The Tribon Geometry Macro Facility is a program used to create predefined geometry. The main purpose of using geometry macros in Tribon M3 is to create design standards. Example of such standards are profile cutouts in Tribon Hull and ventilation volumes in Tribon Ventilation.

Note: Previous versions of Tribon further allowed any user to define geometry macros creating drawings and sub-pictures. However, this is in Tribon M3 no longer supported. Instead Tribon Vitesse functions for Drafting should be used.

The geometry is put into the current drawing or volume.

17.2 Introduction

In interactive Tribon applications, 2D drawings are built up by different interactive functions of the applications. Similarly, 3D figures can be created by the combination of volume primitives. These operations normally work on entity level, i.e. by the addition of lines, arcs, texts, etc. or by the duplication of information of the drawing.

However, in many situations the drawing or volume to be created is parenthesized, i.e. controlled by a relatively small number of parameters and/or conditions.

The Tribon Geometry Macro Facility has been developed as a tool to create such drawings/volumes by defining a number of parameters in calls of **geometry macros**.

A geometry macro is written as a text in a format like a programming language.

The elements of the language are:

- the geometrical entities that can be created
- "program logic", like branching and loops
- basic arithmetics, logical and trigonometrical operations

Details about all these things can be found below.

17.3 Functions

17.3.1 Entities of the Geometry Macro

A geometry macro used for a drawing or a sub-picture may generate the following entities:

- point
- vector
- line
- arc
- circle
- contour
- spline
- symbol
- text
- text file
- layer
- line type
- attribute
- note
- hatch pattern

A geometry macro used for volumes may generate the following entities:

- point
- vector
- parallelepiped
- cylinder
- cone
- spherical segment
- general cylinder
- toroid
- polygon
- rotational primitive
- attribute

One geometry macro can define either a 2D drawing/sub-picture or a 3D volume. However, some of the volume primitives use 2D entities as parameters.

17.3.2 Additional Functions

The following functions can be used in any macro:

- assign
- colour
- branching: if..
  .
  .
  .
  else
  .
  .
  .
  endif
- loop
- conditional loop: while..
  .
  .
  .
  endwhile
- table (formatted output)

17.3.3 Operators and System Functions

AND	logical and
+	add
-	change sign
&	concatenate strings
%	division
******	exponentiation
==	qual
>	greater than
	greater equal
<	less than
	less equal
*****	multiplication
NOT	logical not
...	not equal
OR	logical or
-	subtraction
XOR	logical exclusive or
AB	absolute value
ACO	arcus cosine (radians)
ACOSD	arcus cosine (degrees)
ASIN	arcus sine (radians)
ASIND	arcus sine (degrees)
ATAND	arcus tangent (radians)
ATAND	arcus tangent (degrees)
BYTE	character whose ASCII code is the argument of BYTE
COS	cosine (radians)
COSD	cosine (degrees)
DEGREE	angle in degrees to angle in radians
SIN	sine (radians)
SIND	sine (degrees)
SQRT	square root
TAN	tangent (radians)
TAND	tangent (degrees)
SUBSTR	substrings
LENGTH	length of a string
INDEX	position of a specified sub-string within a string
NRCHAR	number of character with specified number of decimals

17.3.4 Creation of a Geometry Macro

The geometry macro is created by writing a geometry macro source text in a special macro language. This language is similar to other languages used for input to various Tribon applications. The syntax of the Tribon Geometry Macro Language is given in Syntax of Tribon Geometry Macro Language. The source text is stored in an ordinary text file with an arbitrary name (= the name of the macro). It can thus be created using an ordinary editor. Only capital letters, digits and _ (underscore) are allowed characters in the macro and file name.

17.3.5 Execution of a Geometry Macro

The geometry macro can be executed either from Tribon Drafting or as a stand alone program. The following is valid for both cases.

The first time the macro is to be executed, the file name must be given, including the file type. It is assumed that the geometry macro is stored on the directory given by the logical variable SBB_GEO_MACRO_SRC. The interpreter will then check the syntax of the macro in the source file (the macro is "compiled").

The result of the interpretation is given in a list file containing a list of the syntax together with possible error messages. The name of the list file is `<source_file>.LST` and the list file will be created on the directory given by the logical variable SBB_GEO_MACRO_LST.

If the interpretation was successful, a "compiled" version of the syntax will be stored on a directory given by the logical variable SBB_GEO_MACRO_BIN. The name of the "compiled" version of the syntax is `<source_file>.GLB`. At the subsequent executions of the geometry macro, it is sufficient to give the macro name and the "compiled" version will be read, after which the execution will start.

Note, the following paragraph is only valid when a project is shared between Windows and UNIX/VMS.

In case that the Tribon Geometry Macro is used in a project that is shared between platforms, and the project is located on a UNIX or VMS machine, the file handling differs a bit from what is described above. The list file emanating from a compilation on Windows (`<source_file>.LST`) will be placed in the directory indicated by the operating system environment variable TEMP, e.g. c:\Temp. Since the compiled format differs between platforms, it is necessary to have the compiled files (`<source_file>.GLB`) stored on each platform respectively. This also means that the macros have to be compiled once on each platform. The logical variable SBB_GEO_MACRO_BIN_NT is used to indicate the directory used for compiled binary files on the Windows platform. Note that the source files are still fetched from the directory indicated by SBB_GEO_MACRO_SRC, even if that is on another platform. If you do not share projects, that is, you run only on the Windows platform, you do not have to set the variable SBB_GEO_MACRO_BIN_NT, and the compiled files will end up in the directory SBB_GEO_MACRO_BIN.

Stand Alone Program

If the Geometry Macro is run as a stand alone program, the result can be presented in a number of different ways. The following activities are available:

1. Print on terminal
2. Create 2D geometry and store on data bank
3. Create 3D volume model and store on data bank
4. Create 3D volume model + picture and store on data bank

17.3.6 Parameters of a Geometry Macro

The geometry macro can be parameterised, i.e. some parameters are given values at the time of execution. A parameter can be an integer number, a decimal number, a text string or a 2D or 3D position in the drawing/volume. It is possible to define a command string to each parameter. This string will be displayed at the workstation and the operator is prompted to give the parameter value.

Submacros in a Geometry Macro

It is possible to call another geometry macro (submacro) from any macro. Data is transferred from the macro to the submacro by parameters. These parameters can have any type.

The submacro parameters will be assigned the values of the corresponding macro parameters and the submacro will be executed. When the submacro execution is finished the macro parameters will be assigned the values of the corresponding submacro parameters and the execution of the macro is continued. See also The Facility of Using Conditional Statements in a Macro.

The macro and the submacro can be the same macro so the resulting macro is recursive. See also The Possibility to Write Recursive Geometry Macros.

17.3.7 Error Handling

During the execution of a geometry macro, an error handling system is active so that abortions of Tribon Drafting due to macro programming errors will be prevented.

The programmer has the possibility to list any data at the workstation during execution. This will make it easier to detect any errors in the macro programming.

17.3.8 Execution in Batch

It is possible to execute a geometry macro in batch. All input to the macro must then be available in an ordinary text file and in exactly the same order as they are required by the macro.

The first parameter in the file must then be the activity and the second one the name of the macro.

17.4 Appendices

17.4.1 Syntax of Tribon Geometry Macro Language

Conventions Used in this Document

In this document the following conventions are used (cf. *"The Tribon Interpretative Language")*:

```
[  ]optional
<  >term
```

....preceding expression may be repeated

Statement Types

The input language contains the following different statement types:

`ARC`	The `ARC` statement defines a 2D arc. The arc can be used as input to the `PRESENT` and `CONTOUR` statements.
`ASSIGN`	The `ASSIGN` statement assigns a previously defined variable to another variable which will get the same type as the first one. However, if a 2D or a 3D point has been defined, one of these coordinates can be assigned to a variable with type decimal.
`ATTRIBUTE`	The `ATTRIBUTE` statement is used to put attributes in the object. The attributes contain non-geometrical data and it is possible to store 100 integers, 50 reals and 12 strings. Both attribute numbers and the variables can be addressed with an alias name defined in an alias file assigned to the logical variable `SBD_ALIAS`.
`CALL`	The `CALL` statement is used to call another macro. The parameter types must match and the parameters can be used for either input or output purposes. There is no limit for the number of levels of submacros.
`CHANGEDRAW`	The `CHANGEDRAW` statement is used to change the appearance of an object, a component or a subcomponent. This can be done either in one view or in all views.
`CIRCLE`	The `CIRCLE` statement defines a 2D circle. The circle can be used as input to the `PRESENT` statement.
`COLOUR`	The `COLOUR` statement sets the modal colour. Default colour is green. If the given colour is an empty string, the background colour will be used.
`CONE`	The `CONE` statement defines a 3D cone. The cone can be used as input to the `PRESENT` statement.
`CONNECTIONPOINT`	The `CONNECTIONPOINT` statement defines a 3D connection point. The connection point can be used as input to the `PRESENT` statement.
`CONTOUR`	The `CONTOUR` statement defines a 2D contour. The contour can be used as input to the `GENERALCYLINDER` and `PRESENT` statements.
`CURRENT`	The `CURRENT` statement makes it possible to structure a drawing or a volume.
`CYLINDER`	The `CYLINDER` statement defines a 3D cylinder. The cylinder can be used as input to the `PRESENT` statement.
`DECLARE`	The `DECLARE` statement allows the user to specify the type of any variable. The default type is `DECIMAL`. The `DECLARE` statements must be given directly after the `MACRO` statement.
`DELAY`	The `DELAY` statement makes it possible to introduce a delay in a macro. It has only an effect when geometry is presented at a workstation.
`DISTANCE`	The `DISTANCE` statement assigns a value or a previously defined variable to another variable. The value is scaled with the current scale.
`DRAWING_NAME`	The `DRAWING_NAME` statement assigns the name of the current drawing to it's parameter.
`ELSE`	The `ELSE` statement must be used together with the `IF` statement. If the condition given in the `IF` statement is not fulfilled, then the statements after the `ELSE` statement will be executed.
`ENDIF`	The `ENDIF` statement terminates the `IF` statement.
`ENDLOOP`	The `ENDLOOP` statement terminates the `LOOP` statement.
`ENDMACRO`	The `ENDMACRO` statement terminates the `MACRO` statement.
`ENDWHILE`	The `ENDWHILE` statement terminates the `WHILE` statement.
`EXTRACTION`	The `EXTRACTION` statement makes it possible to use Data Extraction to get any data item of a stored Tribon object. This statement is optional.

GENERALCYLINDER	The GENERALCYLINDER statement defines a 3D general cylinder. The general cylinder can be used as input to the PRESENT statement.
GET	The GET statement enables the user to get a number of variables which will be used as parameters.
HATCH	The HATCH statement defines a hatch pattern in a drawing. User defined patterns can be created. The hatch pattern may include islands. The hatch pattern is used as input to the PRESENT statement.
IF	The IF statement tests an expression and performs a specified action if the result of the test is true. All statements between the IF and ENDIF statements will be executed. If the expression is false, no action will be taken unless the ELSE statement is given. In this case, all statements between the ELSE and ENDIF statements will be executed. All statements except the MACRO and ENDMACRO statements can be given after the IF statement. The IF statements can be nested.
LAYER	The LAYER statement sets the modal layer. The default layer is 0. The layer can be given either as a number or as an alias if an alias file exists. The aliases should be defined in a file assigned to the logical variable SB_LAYER_ALIAS. In this case, geometry, text and symbols will get the same modal layer. To give them different layer values, it is possible to use layer classes. The classes are defined in a file assigned to the logical variable SBD_LAYER_CLASS.
LINE	The LINE statement defines a 2D line. The line can be used as input to the CONTOUR and PRESENT statements.
LINETYPE	The LINETYPE statement sets the modal line type. Default line type is solid. The width can also be changed. Default line width is thin.
LOOP	The LOOP statement makes it possible to execute the same statements a number of times. All statements except the MACRO and ENDMACRO statements can be given after the LOOP statement. The LOOP statements can be nested.
MACRO	The MACRO statement must be the first statement in the macro. Parameters may be given and, in that case, they will be given values before the execution of the macro.
NAME	The NAME statement is used to define a name on the drawing/subpicture/volume to be created and stored on a data bank. If the statement is omitted, the macro name will be used. It is also possible to give the form to be used, if any. If the form is omitted, a drawing will be created with no form. The scale can also be given for drawings and subpictures.
NOTE	The NOTE statement defines a note in a drawing. The note is used as input to the PRESENT statement.
PARALLELEPIPED	The PARALLELEPIPED statement defines a 3D parallelepiped. The parallelepiped can be used as input to the PRESENT statement.
POINT_2D	The POINT_2D statement defines a 2D point. The point is used as input to several of the other 2D statements. It is also possible to use this statement when any of the coordinates are to be changed.
POINT_3D	The POINT_3D statement defines a 3D point. The point is used as input to several of the other 3D statements. It is also possible to use this statement when any of the coordinates are to be changed.
POLYGON	The POLYGON statement defines a 3D polygon. The polygon is used as input to the PRESENT statement.
PRESENT	The PRESENT statement presents the geometry created by the macro.
PUT	The PUT statement makes it possible to print any variable data.
RANGE	The RANGE statement defines a range of variables. The range is used in the EXTRACTION and the LOOP statements.
ROTATIONAL	The ROTATIONAL statement defines a 3D rotational primitive. The rotational primitive is used as input to the PRESENT statement.
SPHERESEG	The SPHERESEG statement defines a 3D spherical segment. The segment is used as input to the PRESENT statement.
SPLINE	The SPLINE statement defines a 2D spline. The spline is used as input to the

	CONTOUR and PRESENT statements.
SPLIT	The SPLIT statement splits a model name into project, module and subsystems. These items can then be used in the EXTRACT statement.
SYMBOL	The SYMBOL statement defines a symbol. The symbol is used as input to the PRESENT statement.
TABLE	The TABLE statement makes it possible to present a number of variables with a format determined by the user. The table is used in the PRESENT statement.
TEXT	The TEXT statement defines a text. The text is used as input to the PRESENT statement.
TEXTFILE	The TEXTFILE statement defines a text file. The text file is used as input to the PRESENT statement.
TOROID	The TOROID statement defines a 3D toroid. The toroid is used as input to the PRESENT statement.
VECTOR_2D	The VECTOR_2D statement defines a 2D vector. The vector is used as input to other 2D statements. It is also possible to use this statement when any of the coordinates shall be changed.
VECTOR_3D	The VECTOR_3D statement defines a 3D vector. The vector is used as input to several of the other 3D statements. It is also possible to use this statement when any of the coordinates are to be changed.
VMSJOB	The VMSJOB statement is used to execute a job code or command VMS-file.
WHILE	The WHILE statement makes it possible to execute the same statements a number of times. The WHILE statement tests an expression and performs a specified action as long as the result of the test is true. All statements between the WHILE and ENDWHILE statements will be executed. If the expression is false, no action will be taken. All statements except the MACRO and ENDMACRO statements can be given after the WHILE statement. The WHILE statements can be nested.

Statement Syntax

Below, the complete syntax of each statement type is described.

The ARC Statement

```
ARC,<arc_name>,<start_pnt>
[/ARCMIDPNT=(<mid_pnt>,<end_pnt>)]
[/ARCRADIUS=(<end_pnt>,<rad>)]
[/ARCAMPLITUDE=(<end_pnt>,<ampl>)];
```

`<arc_name>` is the name of the arc and will be assigned the type ARC_2D. The maximum length of `<arc_name>` is 32 characters.

`<start_pnt>` is the starting point of the arc with type POINT_2D.

ARCMIDPNT=(<mid_pnt>,<end_pnt>)
`<mid_pnt>` is the mid point of the arc when the arc is defined by giving three points. It has the type POINT_2D.
`<end_pnt>` is the ending point of the arc with type POINT_2D.

ARCRADIUS=(<end_pnt>,<rad>)
`<end_pnt>` is the ending point of the arc with type POINT_2D.

`<rad>` is the arc radius when the arc is defined by giving two points + radius. It has the type DECIMAL.

ARCAMPLITUDE=(<end_pnt>,<ampl>)
`<end_pnt>` is the ending point of the arc with type POINT_2D.

`<ampl>` is the arc amplitude when the arc is defined by giving two points + amplitude. It has the type DECIMAL.

Structure:

NSEG		(INTEGER)
SEGPARTS(1:NSEG)		
	ENDPNT(1:2)	(DECIMAL)
	AMPLITUDE(1:2)	(DECIMAL)

Example:

```
GET/STRUCTURE=(N,<arc_name>,'NSEG')
 /STRUCTURE=(X,<arc_name>,'ENDPNT',N,1)
 /STRUCTURE=(Y,<arc_name>,'AMPLITUDE',2,'Y');
```

The ASSIGN Statement

ASSIGN,<variable_1>,<variable_2>
[/XCOORD]
[/YCOORD]
[/ZCOORD];

`<variable_1>` is assigned the same value as `<variable_2>` and will get the same type as the first one. The maximum length of `<variable_1>` and `<variable_2>` is 32 characters.
`<variable_2>` can be an expression or have any of the following types:

```
INTEGER
DECIMAL
STRING
```

```
POINT_2D
LINE_2D
ARC_2D
CONTOUR_2D
CIRCLE_2D
VECTOR_2D
SPLINE_2D
TEXTFILE_2D
TEXT_2D
SYMBOL_2D
NOTE_2D
HATCH_2D
EXTRACT
RANGE
TABLE
POINT_3D
VECTOR_3D
CONNECTIONPOINT_3D
CONE_3D
CYLINDER_3D
GENERALCYLINDER_3D
PARALLELEPIPED_3D
POLYGON_3D
SPHERESEG_3D
TOROID_3D
ROTATIONAL_3D
```

XCOORD
If `<variable2>` is of type POINT_2D or POINT_3D, then `<variable1>` can be assigned the x coordinate of `<variable2>`. This variable will get type DECIMAL.

YCOORD
If `<variable2>` is of type POINT_2D or POINT_3D, then `<variable1>` can be assigned the y coordinate of `<variable2>`. This variable will get type DECIMAL.

ZCOORD
If `<variable2>` is of type POINT_3D, then `<variable1>` can be assigned the z coordinate of `<variable2>`. This variable will get type DECIMAL.

The ATTRIBUTE Statement

ATTRIBUTE,<attr_no>
/ATTRDATA=(<variable>,<attribute>)....;

`<attr_no>` is the attribute number (> 0) and must be given within ' ' if it is addressed with an alias name. If `<attr_no>` is a true number or a variable, it shall not be given within ' '. The maximum length of `<attr_no>` as an alias is 26 characters and as a variable 32 characters.

ATTRDATA=(<variable>,<attribute>)....
`<variable>` is the variable name given as an alias or using the default names. These are I1 - I100 for the integers, R1 - R50 for the reals and S1 - S12 for the strings. `<variable>` shall always be given within ' '. The maximum length of `<variable>` as an alias is 26 characters and as a variable 32 characters.

`<attribute>` is the data to be stored in the attribute as an integer, real or string (max 26 characters). `<attribute>` must be given within ' ' if it is a string but not if it is a true number or a variable. The maximum length of `<attribute>` as a variable is 32 characters.

The CALL Statement

CALL,<macro_name>

[,<arg_1>[,<arg_2>....[,<arg_25>]....]];

`<macro_name>` is the name of the submacro. The maximum length of `<macro_name>` is 32 characters. Only upper case letters and _ are allowed characters. `<arg_1>,<arg_2>, ....,<arg_25>` are the arguments to the submacro. They cannot be expressions but must be variables. Any type is allowed. It is important to notice that if a parameter in the submacro is changed, the corresponding argument in the calling macro will also be changed. Thus, it is of great advantage if an argument is used for either input or output.

The CHANGEDRAW Statement

CHANGEDRAW,<obj_type>,<obj_name>

/COMPID=(<comp_id>)

/MARKINGCOLOUR=(<marking_colour>)

/SUBCOMPID=(<subcomp_id>)

/VIEW=(<view_id>);

`<obj_type>` is a keyword describing the type of the object. The following keywords exist:

PANEL plane and curved panel objects

CWAY cableway object

CABLE cable objects

EQUIP equipment objeACT

The object type must be a STRING constant.

`<obj_name>` is the name of the object to be changed, and has the type STRING.

`<comp_id>` is the id of a component of the type INTEGER, and is only used in the case of changing a single component.

`<marking_colour>` is the new colour of the object. It has the type STRING. The maximum length of `<marking_colour>` is 32 characters. If DEFAULT is assigned as `<marking_colour>`, the geometry will be redrawn in the default colour. In a modelling view the default colour is determined by the model and in a diagram view the default colour is determined by the General Diagram default file and, if applicable, pipe specification.

See *Tribon M3 Drafting_Appendices* for valid Tribon colours

`<subcomp_id>` is used in the same manner as `<comp_id>`, but for subcomponents. It has the type INTEGER.

`<view_id>` is the id of the view in which to change the object, and has the type INTEGER. If not given, the appearance of the object is changed in all views of the current drawing.

The CIRCLE Statement

CIRCLE,<circ_name>,<circ_cent>,<circ_rad>;

`<circ_name>` is the name of the arc and will be assigned the type `CIRCLE_2D`. The maximum length of `<circ_name>` is 32 characters.

`<circ_cent>` is the centre point of the circle with type `POINT_2D.`

`<circ_rad>` is the radius of the circle. It has the type DECIMAL.

Structure:

```
NSEG                                    (INTEGER)
SEGPARTS(1:NSEG)
                      ENDPNT(1:2)       (DECIMAL)
                      AMPLITUDE(1:2)    (DECIMAL)
```

:

Example:

```
GET/STRUCTURE=(N,<circ_name>,'NSEG')
 /STRUCTURE=(X,<circ_name>,'ENDPNT',N,1)
 /STRUCTURE=(Y,<circ_name>,'AMPLITUDE',2,'Y');
```

The COLOUR Statement

COLOUR,<col>;
`<col>` is the new modal colour. It has the type STRING. The maximum length of `<col>` is 32 characters.

See *Tribon M3 Drafting_Appendices* for valid Tribon colours.

Empty String means the background colour.

The CONE Statement

CONE,<cone_name>,<rad_1>,<rad_2>
/COORDCONE=(<cl_pnt_1>,<cl_pnt_2>);

`<cone_name>` is the name of the cone and will be assigned the type `CONE_3D.` The maximum length of <cone_name> is 32 characters.

`<rad_1>` and `<rad_2>` are the radii of the bottom and top circles respectively with type DECIMAL.

COORDCONE=(<cl_pnt_1>,<cl_pnt_2>)
`<cl_pnt_1>` and `<cl_pnt_2>` are the centre line starting and ending points, respectively. They have the type `POINT_3D`.

Structure:

`PNT(1:3)`	(POINT_3D)
`VEC(1:3)`	(VECTOR_3D)
BASE	(DECIMAL)
TOP	(DECIMAL)

Example:
```
GET/STRUCTURE=(X,<cone_name>,'PNT',1)
 /STRUCTURE=(Z,<cone_name>,'VEC',3)
 /STRUCTURE=(R1,<cone_name>,'BASE')
 /STRUCTURE=(R2,<cone_name>,'TOP');
```

The CONNECTIONPOINT Statement

CONNECTIONPOINT,<conn_name>,<conn_type>,<conn_no>,
`<conn_pnt>,<conn_vect>,<conn_desc>;`

`<conn_name>` is the name of the connection point and will be assigned the type `CONNECTIONPOINT_3D`. The maximum length of `<conn_name>` is 32 characters.

`<conn_type>` is the connection type of type INTEGER. Only types between 1 and 9 are valid.

`<conn_no>` is the connection number of type INTEGER. Only numbers between 1 and 199 are valid.

`<conn_pnt>` is the connection point defining the position. It has the type `POINT_3D`.

`<conn_vect>` is the connection vector defining the direction. It has the type `VECTOR_3D.`

`<conn_desc>` is the connection description of type STRING. Maximum length of `<conn_desc>` is 100 characters.

Structure:

`CONTYPE`	(INTEGER)
`CONNUMBER`	(INTEGER)
`PNT(1:3)`	(DECIMAL)
`VEC(1:3)`	(DECIMAL)
`DESCR`	(STRING)

Example:
```
GET/STRUCTURE=(T,<conn_name>,'CONTYPE')
 /STRUCTURE=(N,<conn_name>,'CONNUMBER')
 /STRUCTURE=(X,<conn_name>,'PNT',1)
 /STRUCTURE=(Y,<conn_name>,'VEC','Z')
 /STRUCTURE=(D,<conn_name>,'DESCR');
```

The CONTOUR Statement

```
CONTOUR,<cnt_name>[,<start_pnt>]
[/ARC=<arc_name>]
[/ARCMIDPNT=(<mid_pnt>,<end_pnt>)]
[/ARCRADIUS=(<end_pnt,<rad>)]
[/ARCAMPLITUDE=(<end_pnt,<ampl>)]
[/CONTOUR=<cnt_name>]
[/LINE=<line_name>]
[/LINEEND=<end_pnt>]
[/LINEANGLE=(<len>,<ang>)]
[/LINEOFFS=<offs>]
[/SPLINE=<spl_name>];
```

`<cnt_name>` is the name of the contour and will be assigned the type `CONTOUR_2D`. The maximum length of `<cnt_name>` is 32 characters.

`<start_pnt>` is the starting point of the contour with type `POINT_2D`.
If /ARC, /CONTOUR, /LINE or /SPLINE is used for the first segment of the contour, then `<start_pnt>` shall be omitted.

ARC=<arc_name>
`<arc_name>` is the name of the arc and has the type `ARC_2D`. If `<arc_name>` is not the first segment in the contour, the starting point of `<arc_name>` will be ignored. No check is made whether this point coincides with the ending point of the previous segment or not.

ARCMIDPNT=(<mid_pnt>,<end_pnt>)
`<mid_pnt>` and `<end_pnt>` are the mid and ending points of the arc. They have the type `POINT_2D`. The starting point is defined by `<start_pnt>` or by the ending point of the previous segment.

ARCRADIUS=(<end_pnt>,<rad>)
`<end_pnt>` is the ending point of the arc with the type `POINT_2D`. The starting point is defined by `<start_pnt>` or by the ending point of the previous segment.

`<rad>` is the arc radius when the arc is defined by giving two points + radius. It has the type DECIMAL.

ARCAMPLITUDE=(<end_pnt,<ampl>)
`<end_pnt>` is the ending point of the arc with the type `POINT_2D`. The starting point is defined by `<start_pnt>` or by the ending point of the previous segment.

`<ampl>` is the arc amplitude when the arc is defined by giving two points + amplitude. It has the type DECIMAL.

CONTOUR=<cnt_name>
`<cnt_name>` is the name of the contour and has the type `CONTOUR_2D`. If `<cnt_name>` is not

the first segment in the contour, the starting point of `<cnt_name>` will be ignored. No check is made whether this point coincides with the ending point of the previous segment.

LINE=<line_name>

`<line_name>` is the name of the line and has the type `LINE_2D`. If `<line_name>` is not the first segment in the contour the starting point of `<line_name>` will be ignored. No check is made whether this point coincides with the ending point of the previous segment or not.

LINEEND=<end_pnt>

`<end_pnt>` is the ending point of the line with the type `POINT_2D`. The starting point is defined by `<start_pnt>` or by the ending point of the previous segment.

LINEANGLE=(<len>,<ang>)

`<len>` is the length of the line and `<ang>` is the angle of the line, both with type DECIMAL. The starting point is defined by `<start_pnt>` or by the ending point of the previous segment.

LINEOFFS=<offs>

`<offs>` is the offset from the current point. It has the type STRING with at most 72 characters. The following formats are valid:

du, dv (e.g. 100,50) or

length, angle (e.g. 100,45D) or

length, verbal direction (e.g. 100,N).

Verbal directions can be North, South, West, East, Right, Left, Up or Down. Only the first letter is relevant. If the string ends with a 'U' or a 'V', the u axis or the v axis will be locked. Angles must be followed by D (degrees) or R (radians).

The strings must only contain digits, a '-' or a '.'. '-' is only valid in the first position. Between the strings, it is allowed to have a couple of ' ' and ',' but at least one, either a ' ' or a ','. If one of the letters mentioned above is at a valid position, the rest of the string is ignored. Spaces in the beginning or the end of the string are removed.

SPLINE=<spl_name>

`<spl_name>` is the name of the spline and has the type `SPLINE_2D`. If `<spl_name>` is not the first segment in the contour, the starting point of `<spl_name>` will be ignored. No check is made whether this point coincides with the ending point of the previous segment or not.

Structure:

```
NSEG                                  (INTEGER)
SEGPARTS(1:NSEG)
                   ENDPNT(1:2)        (DECIMAL)
                   AMPLITUDE(1:2)     (DECIMAL)
```

Example:

```
GET/STRUCTURE=(N,<cnt_name>,'NSEG')
 /STRUCTURE=(X,<cnt_name>,'ENDPNT',N,1)
 /STRUCTURE=(Y,<cnt_name>,'AMPLITUDE',2,'Y');
```

The CURRENT Statement

CURRENT[/SUBPICTURE=(<view_name>,<subview_name>)
[/VIEWSCALE=<view_scl>]]
[/SUBVOLUME=<subvol_no>];

SUBPICTURE=(<view_name>,<subview_name>)
[/VIEWSCALE=<view_scl>]

`<view_name>` is the name of the view. It has the type STRING. The maximum length of `<view_name>` is 26 characters. If `<view_name>` does not exist a new view and a new subview are created. The subview will get the name `<subview_name>`.

`<subview_name>` is the name of the subview. It has the type STRING. The maximum length of `<subview_name>` is 26 characters. If `<subview_name>` does not exist a new subview is created.

`<view_scl>` is the scale to be used when a new view is created. It has the type DECIMAL. If this attribute is not given then the view scale will be the same as the drawing scale (given in the NAME statement).

A new component is always created when a macro is run. This is the case even if the CURRENT statement is not present.

SUBVOLUME=<subvol_no>
`<subvol_no>` is the number of the subvolume. It has the type INTEGER. If `<subvol_no>` does not exist it is created.

The CYLINDER Statement

CYLINDER,<cyl_name>,<rad>
/COORDCYL=(<cl_pnt_1>,<cl_pnt_2>;

`<cyl_name>` is the name of the cylinder and will be assigned the type `CYLINDER_3D`. The maximum length of `<cyl_name>` is 32 characters.

`<rad>` is the radius of the cylinder with type DECIMAL.

COORDCYL=(<cl_pnt_1>,<cl_pnt_2>)
`<cl_pnt_1>` and `<cl_pnt_2>` are the centre line starting and ending points, respectively. They have the type `POINT_3D`.

Structure:

`PNT(1:3)`	(DECIMAL)
`VEC(1:3)`	(DECIMAL)
BASE	(DECIMAL)

Example:
```
GET/STRUCTURE=(X,<cyl_name>,'PNT',1)
 /STRUCTURE=(Y,<cyl_name>,'VEC','Y')
 /STRUCTURE=(R,<cyl_name>,'BASE');
```

The DECLARE Statement

DECLARE,<variable>,<type>;

`<variable>` is the name of the variable whose type is to be declared. The maximum length of `<variable>` is 32 characters.

`<type>` is the type of `<variable>`. It has the type STRING and the whole type name must be given. The following types are available:

```
INTEGER
DECIMAL
STRING
POINT_2D
LINE_2D
ARC_2D
CONTOUR_2D
CIRCLE_2D
VECTOR_2D
SPLINE_2D
TEXTFILE_2D
TEXT_2D
SYMBOL_2D
NOTE_2D
HATCH_2D
```

`EXTRACT`

`RANGE`

`TABLE`

`POINT_3D`

`VECTOR_3D`

`CONNECTIONPOINT_3D`

`CONE_3D`

`CYLINDER_3D`

`GENERALCYLINDER_3D`

`PARALLELEPIPED_3D`

`POLYGON_3D`

`SPHERESEG_3D`

`TOROID_3D`

`ROTATIONAL_3D`

The default type is DECIMAL.

The DELAY Statement

DELAY,<del_time>;
`<del_time>` is the delay time in seconds. It has the type DECIMAL.

The DISTANCE Statement

DISTANCE,<variable_1>,<variable_2>
`<variable_1>` is assigned the same value as `<variable_2>` and will get the same type as the first one. The maximum length of `<variable_1>` and `<variable_2>` is 32 characters. `<variable_2>` can be an expression, an integer constant or a decimal constant.

When the DISTANCE statement is used `<variable_1>` will be scaled with the current scale. This is not the case when the ASSIGN statement is used.

The DRAWING_NAME Statement

DRAWING_NAME, <name_drawing>;
This statement assigns the name of current drawing to `<name_drawing>` which is a string parameter. If there is no current drawing, the statement prompts the user to supply the name as text either in input window or on terminal.

The ELSE Statement

ELSE;
This statement must be used together with the IF statement.

The ENDIF Statement

ENDIF;
The ENDIF statement terminates the IF statement.

The ENDLOOP Statement

ENDLOOP;
The ENDLOOP statement terminates the LOOP statement.

The ENDMACRO Statement

ENDMACRO;
The ENDMACRO statement terminates the MACRO statement.

The ENDWHILE Statement

ENDWHILE;

The ENDWHILE statement terminates the WHILE statement.

The EXTRACTION Statement (OPTIONAL)

EXTRACTION,<dex_name>,<dex_str>;

`<dex_name>` is the name of the data extraction variable and has the type EXTRACT. The maximum length of `<dex_name>` is 32 characters.

`<dex_str>` is the data extraction string and has the type STRING. For further information, see documentation in *Tribon User's Guide Data Extraction*.

The GENERALCYLINDER Statement

GENERALCYLINDER,<gencyl_name>,<cnt_name>,<thick>, <pnt_1>,<pnt_2>,<pnt_3>;

`<gencyl_name>` is the name of the general cylinder and will be assigned the type `GENERALCYLINDER_3D`. The maximum length of `<gencyl_name>` is 32 characters.

`<cnt_name>` is the name of a 2D contour (with type `CONTOUR_2D`) which has previously been defined.

`<thick>` is the thickness of the general cylinder and has the type DECIMAL.

`<pnt_1>` is the origin of the general cylinder.

`<pnt_2>` and `<pnt_3>` defines together with `<pnt_1>` the orientation in space. The u vector is given by `<pnt_2>` and `<pnt_1>` and the v vector by `<pnt_3>` and `<pnt_1>`.

`<pnt_1>`, `<pnt_2>` and `<pnt_3>` all have the type `POINT_3D`.

Structure:

`PNT(1:3)`	(DECIMAL)	
`UVEC(1:3)`	(DECIMAL)	
`VVEC(1:3)`	(DECIMAL)	
`THICK`	(DECIMAL)	
`NSEG`	(INTEGER)	
`SEGPARTS(1:NSEG)`		
	`ENDPNT(1:2)`	(DECIMAL)
	`AMPLITUDE(1:2)`	(DECIMAL)

Example:)

```
GET/STRUCTURE=(PX,<gencyl_name>,'PNT',1
 /STRUCTURE=(N,<gencyl_name>,'NSEG')
 /STRUCTURE=(X,<gencyl_name>,'ENDPNT',N,1)
 /STRUCTURE=(Y,<gencyl_name>,'AMPLITUDE',2,'Y');
```

The GET Statement

GET

```
[/INTEGER=(<prompt>,<int>)]
[/DECIMAL=(<prompt>,<dec>)]
[/STRING=(<prompt>,<str>)]
[/DISTANCE=(<prompt>,<dist>)]
[/POINT_2D=(<prompt>,<pnt>)]
[/POINT_3D=(<prompt>,<pnt>)]
[/EXTRACT=(<variable>,<status>,<dex_name>,<arg_1>
[,<arg_2]...[,<arg_10>]..]])]
```

```
[/RANGE=(<rng_name>,<status>,<dex_name>[,<arg_1>
[,<arg_2]...[,<arg_10>]..]])]
[/STRUCTURE=(<variable>,<struct_name>,<arg_1>
[,<arg_2]...[,<arg_10>]..]])];
[/MODEL_NAME=(<prompt>, <status>, <name_model>, <name_component>)];
```

[/VIEW_ID=(<prompt>, <get_viewid>)];

`<prompt>` is a text displayed at the workstation before entering any values. It has the type STRING and the maximum length is 100 characters. `<prompt>` has the same meaning for the following attributes to the GET statement.

INTEGER=(<prompt>,<int>)

`<int>` is the integer value which has been given as input at the workstation.

DECIMAL=(<prompt>,<dec>)

`<dec>` is the decimal value which has been given as input at the workstation.

STRING=(<prompt>,<str>)

`<str>` is the string value which has been given as input at the workstation.

DISTANCE=(<prompt>,<dist>)

`<dist>` is a distance which has been given as input at the workstation. By default, the `<dist>` is given as the distance between two cursor positions (or any other point mode) but it is possible to key in `<dist>` by answering REJECT at the first cursor position. `<dist>` has the type DECIMAL.

It is important to use DISTANCE and not DECIMAL when a distance is wanted, because the current scale is taken care of in DISTANCE but not in DECIMAL.

POINT_2D=(<prompt>,<pnt>)

`<pnt>` is a point which has been given as input at the workstation. The default point mode is cursor position but it is possible to use any of the other available modes. `<pnt>` has the type `POINT_2D`.

POINT_3D=(<prompt>,<pnt>)

`<pnt>` is a point which has been given as input at the workstation. The default point mode is cursor position but it is possible to use any of the other available modes. `<pnt>` has the type `POINT_3D`.

EXTRACT=(<variable>,<status>,<dex_name>,<arg_1>[,<arg_2>]... [,<arg_10>]..]])
Optional function.

`<variable>` is the name of the variable to be assigned. It can have the type INTEGER, DECIMAL or STRING. The maximum length of `<variable>` is 32 characters.

`<status>` is a variable giving the status of `<variable>`. It can have the following INTEGER values:

`0<variable>` not defined

`1<variable>` defined

Before using `<variable>`, there must always be a test on `<status>` to ensure that `<variable>` is defined. Otherwise the macro will be aborted.

`<dex_name>` is the name of the data extraction variable and has the type EXTRACT. The maximum length of `<dex_name>` is 32 characters.

`<arg_1>[,<arg_2>]...[,<arg_10>]..]]` are the arguments corresponding to the data extraction keywords, from the top level down to the bottom level. These arguments can have the type INTEGER, DECIMAL or STRING.

RANGE=(<rng_name>,<status>,<dex_name>[,<arg_1>[,<arg_2>]... [,<arg_10>]..]])
Optional function.

`<rng_name>` is the name of the range which was the result of the extraction. It has the type RANGE. The maximum length of `<rng_name>` is 32 characters. The range can for instance contain the resulting object names.

`<status>` is a variable giving the status of `<rng_name>`. It can have the following INTEGER values:

0<rng_name> not defined

1<rng_name> defined

Before using <rng_name>, there shall always be a test on <status> to ensure that <rng_name> is defined. Otherwise the macro will be aborted.

<dex_name> is the name of the data extraction variable and has the type EXTRACT. The maximum length of <dex_name> is 32 characters.

[,<arg_1>[,<arg_2>]...[,<arg_10>]..]] are the arguments corresponding to the data extraction keywords, from the top level down to the bottom level. These arguments can have the type INTEGER, DECIMAL or STRING.

STRUCTURE=(<variable>,<struct_name>,<arg_1>[,<arg_2]...[,<arg_10>]..]])

<variable> is the name of the variable to be assigned. It can have the type INTEGER, DECIMAL or STRING. The maximum length of <variable> is 32 characters.

<struct_name> is the name of the structure from which the data shall be taken. It has the type STRING. The following structure types are available:

POINT_2D

LINE_2D

ARC_2D

CONTOUR_2D

CIRCLE_2D

VECTOR_2D

SPLINE_2D

TEXTFILE_2D

TEXT_2D

SYMBOL_2D

NOTE_2D

HATCH_2D

POINT_3D

VECTOR_3D

CONNECTIONPOINT_3D

CONE_3D

CYLINDER_3D

GENERALCYLINDER_3D

PARALLELEPIPED_3D

POLYGON_3D

SPHERESEG_3D

TOROID_3D

ROTATIONAL_3D

[<arg_1>[,<arg_2>]...[,<arg_10>]..]] are the arguments in the structure. It is thus possible for instance to get the end coordinates for the n:th segment in a certain contour or the total number of lines in the file used by the TEXTFILE statement.

The arguments are described at the respective structure statement.

MODEL_NAME=(<prompt>,<status>,<name_model>, <name_component>)

<status> is a variable giving the status of <variable>. It can have the following INTEGER values:

0<name_model> and <name_component> not defined

1<name_model> defined, <name_component> not defined

2<name_model> and <name_component> defined

Before using either of <name_model> and <name_component>, there must always be a test on <status> to ensure that the variable about to be used is defined. Otherwise the macro will be aborted. Moreover, the variable must be declared by a DECLARE statement.

`<name_model>` is a string assigned to by the statement. It is only valid if `<status>` = 1. The parameter has the name of the model indicated by user when the statement was executed.

`<name_component>` is a string assigned to by the statement. It is only valid if `<status>` = 1. The parameter has the name of the model object component indicated by user when the statement was executed.

VIEW_ID=(<prompt>,<get_viewid>)

`<get_viewid>` is the ID of a view which has been given as input at the workstation. `<get_viewid>` has the type INTEGER.

The HATCH Statement

HATCH,<hatch_name>,<hatch_cnt>,<hatch_type>

[/HATCHANGLE=<hatch_angle>]

[/HATCHDISTANCE=<hatch_dist>]

[/USERDEFINED=(<hatch_page>,<hatch_number>]

[/ISLAND=<island_cnt>];

`<hatch_name>` is the name of the hatch pattern and has the type `HATCH_2D`. The maximum length of `<hatch_name>` is 32 characters.

`<hatch_type>` is the type of hatch pattern. It has the type INTEGER. The following types are available:

1 normal, positive angle

2 normal, negative angle

3 cross-hatching

4 user defined

HATCHANGLE=<hatch_angle>

`<hatch_angle>` is the angle of the hatch pattern lines. It has the type DECIMAL. The default angle is 60 degrees.

HATCHDISTANCE=<hatch_dist>

`<hatch_dist>` is the distance between the hatch pattern lines. It has the type DECIMAL. The default distance is 5 mm.

USERDEFINED=<hatch_page>,<hatch_number>

`<hatch_page>` is the page number (1-999) in the standard book for user defined hatch patterns. It has the type INTEGER.

`<hatch_number>` is the standard number (1-8) within the page `<hatch_page>` in the standard book for user defined hatch patterns. It has the type INTEGER.

/ISLAND=<island_cnt>

`<island_cnt>` is the island contour where the hatch pattern shall be removed. It is of the type `CONTOUR_2D` or `TEXT_2D`.

Structure:

`TYPE`	(INTEGER)	
`ANGLE`	(DECIMAL)	
`DISTANCE`	(DECIMAL)	
`PAGE`	(INTEGER)	
`NUMBER`	(INTEGER)	
`NSEG`	(INTEGER)	
`ISLANDS`	(INTEGER)	
`SEGPARTS(1:NSEG)`		
	`ENDPNT(1:2)`	(DECIMAL)
	`AMPLITUDE(1:2)`	(DECIMAL)

Example:

```
GET/STRUCTURE=(T,<hatch_name>,'TYPE'
 /STRUCTURE=(A,<hatch_name>,'ANGLE')
 /STRUCTURE=(D,<hatch_name>,'DISTANCE')
 /STRUCTURE=(P,<hatch_name>,'PAGE')
 /STRUCTURE=(M,<hatch_name>,'NUMBER')
 /STRUCTURE=(N,<hatch_name>,'NSEG')
 /STRUCTURE=(I,<hatch_name>,'ISLANDS')
 /STRUCTURE=(X,<hatch_name>,'ENDPNT',N,1)
 /STRUCTURE=(Y,<hatch_name>,'AMPLITUDE',2,'Y');
```

The IF Statement

IF,<cond>;

`<cond>` is the condition to be tested. It has the type BOOLEAN.

The LAYER Statement

LAYER,<lay_no>;

`<lay_no>` is the layer number (> 0) and must be given within ' ' if it is addressed with an alias name. If `<lay_no>` is a true number or a variable, it shall not be given within ' '. If `<lay_no>` refers to a layer class, it must be preceded by # and directly followed by the class number or the class name. In this case, `<lay_no>` shall be given within ' '.

The maximum length of `<lay_no>` as an alias is 26 characters and as a variable 32 characters.

The LINE Statement

LINE,<line_name>,<stp_pnt>
[/LINEEND=<end_pnt>]
[/LINEANGLE=(<len>,<ang>)]
[/LINEOFFS=<offs>];

`<line_name>` is the name of the line and has the type `LINE_2D`. The maximum length of `<line_name>` is 32 characters.

`<stp_pnt>` is the starting point of the line with the type `POINT_2D`.

LINEEND=<end_pnt>

`<end_pnt>` is the ending point of the line with the type `POINT_2D`.

LINEANGLE=(<len>,<ang>)

`<len>` is the length of the line and <ang> is the angle of the line, both with type DECIMAL.

LINEOFFS=<offs>

`<offs>` is the offset from the current point. It has the type STRING with at most 72 characters. The following formats are valid:

du, dv (e.g. 100,50) or

length, angle (e.g. 100,45D) or

length, verbal direction (e.g. 100,N).

Verbal directions can be North, South, West, East, Right, Left, Up or Down. Only the first letter is relevant. If the string ends with a 'U' or a 'V', the u axis or the v axis will be locked. Angles must be followed by D (degrees) or R (radians).

The strings must only contain digits, a '-' or a '.'. '-' is only valid in the first position. Between the strings, it is allowed to have a couple of ' ' and ',' but at least one, either a ' ' or a ','. If one of the letters mentioned above is at a valid position, the rest of the string is ignored. Spaces in the beginning or the end of the string are removed.

Structure:

`STARTPNT(1:2)` (DECIMAL)
`ENDPNT(1:2)` (DECIMAL)

Example:

```
GET/STRUCTURE=(X,<line_name>,'STARTPNT',1)
 /STRUCTURE=(Y,<line_name>,'ENDPNT','Y'
```

The LINETYPE Statement

LINETYPE,<lin_type>
[/LINEWIDTH=<lin_wid>];

`<lin_type>` is the modal line type to be used. It has the type INTEGER. Default line type is 1 (solid). The types 1 - 5 are available.

LINEWIDTH=<lin_wid>

`<lin_wid>` is the modal line width to be used. It has the type INTEGER. Default line width is 1 (thin). The types 1 - 3 are available.

The LOOP Statement

LOOP,<loop_var>,<range>;

`<loop_var>` is the loop variable and the type is dependent on how `<range>` is given.

`<range>` can be given in three ways:

`<start_value>` : `<end_value>`

or

`<start_value> : <end_value> :: <step_value>`

or

`<range>`

where `<range>` has the type RANGE.

In the to first cases `<loop_var>` can have the type INTEGER or DECIMAL. If `<range>` has the type RANGE then `<loop_var>` can also have type STRING.

When `<step_value>` is omitted, it is by default put to 1.

`<start_value>, <end_value>` and `<step_value>` have the type DECIMAL.

The MACRO Statement

MACRO,<macro_name>
[,<arg_1>[,<arg_2>....[,<arg_25>]....]];

`<macro_name>` is the name of the macro. The maximum length of `<macro_name>` is 32 characters. Only upper case letters, digits and _ are allowed characters. `<arg_1>,<arg_2>, ....,<arg_25>` are the arguments to the macro. They cannot be expressions but must be variables. All types given in the ASSIGN statement are allowed.

The NAME Statement

NAME,<dwg_name>
[/DWG]
[/FORM=<form_name>]
[/PICT]
[/SCALE=<dwg_scale>]
[/VOLUME=(<xmax>,<ymax>)];

`<dwg_name>` is the name of the drawing/subpicture/ volume to be created and stored on a data bank. If the NAME statement is omitted, `<macro_name>` will be used. `<dwg_name>` has the type STRING and the maximum length is 32 characters.

If the drawing/subpicture/volume given by `<dwg_name>` already exists on the data bank, the geometry created by the macro will be added to the existing object.

DWG
This attribute is used to specify that a drawing is to be created.

FORM=<form_name>
`<form_name>` is the name of the form to be used, if any. `<form_name>` has the type STRING and the maximum length is 32 characters.

PICT
This attribute is used to specify that a subpicture is to be created.

SCALE=<dwg_scale>
`<dwg_scale>` is the drawing/subpicture scale to be used. The default scale is 1:50. `<dwg_scale>` has the type DECIMAL.

VOLUME=(<xmax>,<ymax>)
`<xmax>` and `<ymax>` are the x and y extensions for the volume to be created. They have the type DECIMAL.

The NOTE Statement

NOTE,<note_name>,<start_pnt>,<ref_cnt>,<note_text>
[/NOTESYMBOL=<note_symb>]
[/REFSYMBOL=<ref_symb>];

`<note_name>` is the name of the note and has the type `NOTE_2D`. The maximum length of `<note_name>` is 32 characters.

`<start_pnt>` is the starting point of the reference lines of the note. It has the type `POINT_2D`.

`<ref_cnt>` is the reference lines of the note. It has the type `CONTOUR_2D`. This contour must have been defined previously.

`<note_text>` is the text in the note (with type `TEXT_2D`) which has previously been defined.

NOTESYMBOL=<note_symb>
`<note_symb>` is the number of the note symbol. It has the type INTEGER. The default note symbol is number 31.

REFSYMBOL=<ref_symb>
`<ref_symb>` is the number of the reference symbol. It has the type INTEGER. The default reference symbol is number 21.

Structure:

`TEXT`	(STRING)	
`NOTESYMB`	(INTEGER)	
`REFSYMB`	(INTEGER)	
`PNT(1:2)`	(DECIMAL)	
`NSEG`	(INTEGER)	
`SEGPARTS(1:NSEG)`		
	`ENDPNT(1:2)`	(DECIMAL)
	`AMPLITUDE(1:2)`	(DECIMAL)

Example:
```
GET/STRUCTURE=(NS,<note_name>,'NOTESYMB')
 /STRUCTURE=(RS,<note_name>,'REFSYMB')
 /STRUCTURE=(SX,<note_name>,'PNT',1)
 /STRUCTURE=(N,<note_name>,'NSEG')
 /STRUCTURE=(X,<note_name>,'ENDPNT',N,1)
 /STRUCTURE=(Y,<note_name>,'AMPLITUDE',2,'Y');
```

The PARALLELEPIPED Statement

PARALLELEPIPED,<para_name>
[/COORDPARA=(<corn_1>,<corn_2>,<corn_3>)]
[/CENLINPARA=(<cl_pnt_1>,<cl_pnt_2>,<corn>)];

`<para_name>` is the name of the parallelepiped and has the type `PARALLELEPIPED_3D`. The maximum length of `<para_name>` is 32 characters.

COORDPARA=(<corn_1>,<corn_2>,<corn_3>)

`<corn_1>` is the origin of the parallelepiped and the lower right corner when looking in the direction of the centre line. `<corn_2>` is the upper right corner at the end surface when looking in the same direction. `<corn_3>` is then, in the same way, the upper left corner at the end surface.

`<corn_1>, <corn_2>` and `<corn_3>` all have the type `POINT_3D`.

COORDPARA=(<cl_pnt_1>,<cl_pnt_2>,<corn>)

`<cl_pnt_1>` is the starting point of the centre line and `<cl_pnt_2>` is the ending point of the centre line.

`<corn>` is the upper left corner at the end surface when looking in the direction of the centre line.

`<cl_pnt_1>, <cl_pnt_2>` and `<corn>` all have the type `POINT_3D`.

Structure:

`PNT(1:3)`	(DECIMAL)
`UVEC(1:3)`	(DECIMAL)
`VVEC(1:3)`	(DECIMAL)
`LENGTH`	(DECIMAL)

Example:

```
GET/STRUCTURE=(X,<para_name>,'PNT',1)
 /STRUCTURE=(Y,<para_name>,'UVEC',2)
 /STRUCTURE=(Z,<para_name>,'VVEC','Z')
 /STRUCTURE=(L,<para_name>,'LENGTH');
```

The POINT_2D Statement

POINT_2D,<pnt_name>,<x>,<y>;

`<pnt_name>` is the name of the point and has the type `POINT_2D`. The maximum length of `<pnt_name>` is 32 characters.

<x> and <y> are the x and y coordinates of the point, both with type DECIMAL.

If only one of the coordinates of a previously defined point are to be changed, the other one is simply omitted.

Structure:

`PNT(1:2)`	(DECIMAL)

Example:

```
GET/STRUCTURE=(X,<pnt_name>,'PNT',1)
 /STRUCTURE=(Y,<pnt_name>,'PNT','Y');
```

The POINT_3D Statement

POINT_3D,<pnt_name>,<x>,<y>,<z>;

`<pnt_name>` is the name of the point and has the type `POINT_3D`. The maximum length of `<pnt_name>` is 32 characters.

<x>, <y> and <z> are the x, y and z coordinates of the point, all with type DECIMAL.

If only one of the coordinates of a previously defined point shall be changed, the others are simply omitted.

Structure:

`PNT(1:3)`	(DECIMAL)

Example:
```
GET/STRUCTURE=(X,<pnt_name>,'PNT','X')
 /STRUCTURE=(Y,<pnt_name>,'PNT',2)
 /STRUCTURE=(Z,<pnt_name>,'PNT','Z');
```

The POLYGON Statement

POLYGON,<pol_name>,<stp_pnt>
/LINEPOLYGON=<end_pnt>....;

`<pol_name>` is the name of the polygon and has the type `POLYGON_3D`. The maximum length of `<pol_name>` is 32 characters.

`<stp_pnt>` is the starting point of the polygon with the type `POINT_3D`.

LINEPOLYGON=<end_pnt>....
`<end_pnt>` is the ending point of the polygon segment with the type `POINT_3D`.

Structure:

`NSEG`	(INTEGER)	
`SEGPARTS(1:NSEG)`		
	`ENDPNT(1:3)`	(DECIMAL)
	`AMPLITUDE(1:3)`	(DECIMAL)

Example:
```
GET/STRUCTURE=(N,<pol_name>,'NSEG')
 /STRUCTURE=(X,<pol_name>,'ENDPNT',N,1)
 /STRUCTURE=(Y,<pol_name>,'AMPLITUDE',2,'Y');
```

The PRESENT Statement

PRESENT,<var_name>;

`<var_name>` is the name of the variable which is to be presented. It can have any of the following types:

`POINT_2D`

`LINE_2D`

`ARC_2D`

`CIRCLE_2D`

`CONTOUR_2D`

`SPLINE_2D`

`SYMBOL_2D`

`TEXT_2D`

`TEXTFILE_2D`

`NOTE_2D`

`TABLE`

`CONNECTIONPOINT_3D`

`CONE_3D`

`CYLINDER_3D`

`PARALLELEPIPED_3D`

`POLYGON_3D`

`ROTATIONAL_3D`

`TOROID_3D`

`SPHERESEG_3D`

`GENERALCYLINDER_3D`

The maximum length of `<var_name>` is 32 characters.

The PUT Statement

PUT,<var_name>;

`<var_name>` is the name of the variable which contents is to be written. It can have any type. The maximum length of `<var_name>` is 32 characters.

The RANGE Statement

RANGE,<rng_name>
[/FILENAME=<file_name>];

`<rng_name>` is the name of the range and has the type RANGE. The maximum length of `<rng_name>` is 32 characters.

FILENAME=<file_name>

`<file_name>` is the file where the range is stored. It has the type STRING. The file contains a number of integer numbers OR decimal numbers OR strings, one on each line.

The ROTATIONAL Statement

ROTATIONAL,<rot_name>,<cnt_name>,<pnt>,<vec>;

`<rot_name>` is the name of the rotational primitive and will be assigned the type `ROTATIONAL_3D`. The maximum length of `<rot_name>` is 32 characters.

`<cnt_name>` is the name of a 2D contour (with type `CONTOUR_2D`) which has previously been defined.

`<pnt>` is the origin of the 2D contour in space. It has the type `POINT_3D`.

`<vec>` is the vector around which `<cnt_name>` is rotated. It has the type `VECTOR_3D`.

Structure:

`PNT(1:3)`	(DECIMAL)	
`VEC(1:3)`	(DECIMAL)	
`NSEG`	(INTEGER)	
`SEGPARTS(1:NSEG)`		
	`ENDPNT(1:2)`	(DECIMAL)
	`AMPLITUDE(1:2)`	(DECIMAL)

Example:

```
GET/STRUCTURE=(X,<rot_name>,'PNT',1)
 /STRUCTURE=(N,<rot_name>,'NSEG')
 /STRUCTURE=(Y,<rot_name>,'AMPLITUDE',N,'Y');
```

The SPHERESEG Statement

SPHERESEG,<seg_name>,<rad>
/COORDSEG=(<cl_pnt_1>,cl_pnt_2>);

`<seg_name>` is the name of the spherical segment and has the type `SPHERESEG_3D`. The maximum length of `<seg_name>` is 32 characters.

`<rad>` is the radius of the spherical segment. It has the type DECIMAL.

COORDSEG=(<cl_pnt_1>,<cl_pnt_2>)
`<cl_pnt_1>` is the starting point and `<cl_pnt_2>` is the ending point of the centre line, both with the type `POINT_3D`.

Structure:

`PNT(1:3)`	(DECIMAL)
`UVEC(1:3)`	(DECIMAL)
`RADIUS`	(DECIMAL)

Example:
```
GET/STRUCTURE=(X,<seg_name>,'PNT',1)
 /STRUCTURE=(Z,<seg_name>,'UVEC',3)
 /STRUCTURE=(R,<seg_name>,'RADIUS');
```

The SPLINE Statement

SPLINE,<spl_name>
[/SPLPNT=<spl_pnt>]
[/SPLTAN=(<spl_pnt>,<tang_ang>)];
`<spl_name>` is the name of the spline and has the type `SPLINE_2D`. The maximum length of `<spl_name>` is 32 characters.

It is possible to define the spline with or without tangent conditions in the points in the following way:

1)no tangent condition

2)tangent condition in the start point

3)tangent condition in the end point

4)tangent conditions in both start and end point

5)tangent conditions in all points

SPLPNT=<spl_pnt>
`<spl_pnt>` is the spline point with type `POINT_2D`. It is used when no tangent condition is wanted.

SPLTAN=(<spl_pnt>,<tang_ang>)
`<spl_pnt>` is the spline point with type `POINT_2D`.

`<tang_ang>` is the tangent angle to be used in 2) - 5) above. It has the type DECIMAL.

Structure:

`NSEG`	(INTEGER)	
`SEGPARTS(1:NSEG)`		
	`PNT(1:2)`	(DECIMAL)
	`VEC(1:2)`	(DECIMAL)

Example:)
```
GET/STRUCTURE=(N,<spl_name>,'NSEG'
 /STRUCTURE=(X,<spl_name>,'PNT',N,1)
 /STRUCTURE=(Y,<spl_name>,'VEC',N,2);
```

The SPLIT Statement

SPLIT,<model_name>,<delimiter>,<project>,<module>,
<subsyst_1>,<subsyst_2>,<subsyst_3>;
`<model_name>` is the name of the model. It has the type STRING. The maximum length of `<col>` is 26 characters.

`<delimiter>` is the character which is used to separate the different parts of `<model_name>`. It has the type STRING.

`<project>` is the project of `<model_name>` and has the type STRING.

`<module>` is the module of `<model_name>` and has the type STRING.

`<subsyst_1>` is the first subsystem of `<model_name>` and has the type STRING.

`<subsyst_2>` is the second subsystem of `<model_name>` and has the type STRING.

`<subsyst-3>` is the third subsystem of `<model_name>` and has the type STRING.

All three subsystems need not be present in the model name. How many subsystems that exist depends on the application.

The SYMBOL Statement

SYMBOL,<symb_name>,<symb_no>,<symb_pnt>,<symb_ang>
[/MIRRORX]
[/MIRRORY]
[/SYMBFONT=<symb_fnt>]
[/SYMBHEIGHT=<symb_hgt>]
[/SYMBWIDTH=<symb_wid>]
[/AUTOOFF];

`<symb_name>` is the name of the symbol and has the type `SYMBOL_2D`. The maximum length of `<symb_name>` is 32 characters.

`<symb_no>` is the number of the symbol. It has the type INTEGER.

`<symb_pnt>` is the origin of the symbol and has the type `POINT_2D`.

`<symb_ang>` is the symbol angle which is of type DECIMAL.

MIRRORX
With this attribute, it is possible to reflect the symbol in the x axis. It cannot be used together with `MIRRORY`.

MIRRORY
With this attribute it is possible to reflect the symbol in the y axis. It cannot be used together with `MIRRORX`.

SYMBFONT=<symb_fnt>
`<symb_fnt>` is the symbol font and has the type INTEGER. The default font is the system font number 21.

SYMBHEIGHT=<symb_hgt>
`<symb_hgt>` is the symbol height and has the type DECIMAL. The symbol is given a default height value and the symbol height can be changed with this attribute.

SYMBWIDTH=<symb_wid>
`<symb_wid>` is the symbol width and has the type DECIMAL. The symbol is given a default width value but the symbol width can be changed with this attribute.

AUTOOFF
With this attribute it is possible to switch off the default automatic positioning of the symbol.

Structure:

`SYMBNO`	(INTEGER)
`PNT(1:2)`	(DECIMAL)
`ANGLE`	(DECIMAL)
`FONTNO`	(INTEGER)
`WIDTH`	(DECIMAL)
`HEIGHT`	(DECIMAL)
`MIRR`	(INTEGER)
`AUTO`	(INTEGER)

Example:

```
GET/STRUCTURE=(S,<symb_name>,'SYMBNO')
 /STRUCTURE=(X,<symb_name>,'PNT',1)
 /STRUCTURE=(A,<symb_name>,'ANGLE')
 /STRUCTURE=(F,<symb_name>,'FONTNO')
 /STRUCTURE=(W,<symb_name>,'WIDTH')
 /STRUCTURE=(H,<symb_name>,'HEIGHT')
 /STRUCTURE=(M,<symb_name>,'MIRR')
 /STRUCTURE=(A,<symb_name>,'AUTO');
```

The TABLE Statement

TABLE,<tab_name>
[/ARGUMENTS=([<arg_1>[,<arg_2>....[,<arg_15>]....]])]
[/FORMAT=([<width_1>,<ndec_1>[,<width_2>,ndec_2>....
[,<width_15>,<ndec_15>]....]])]
[/FILENAME=<file_name>]
[/QUOTES]
[/NOAPPEND];

`<tab_name>` is the name of the table and has the type TABLE. The maximum length of `<tab_name>` is 32 characters.

ARGUMENTS=([<arg_1>[,<arg_2>.... [,<arg_15>]....]])

`<arg_1>,<arg_2>, ....,<arg_25>` are the arguments to be put in the table. They can have the type INTEGER, DECIMAL or STRING.

FORMAT=([<width_1>,<ndec_1>[,<width_2>,<ndec_2>.... [,<width_15>,<ndec_15>]....]])

This attribute describes the format of the table. `<width_1>,<width_2>, ...,<width_15>` is the width of the column for the i:th argument. It has the type integer. The default width is 8 which will be the result if FORMAT is not given or if the width for a variable is omitted (i.e. given as ,,). If the width is negative the result will be positioned left-justified, otherwise right-justified.

`<ndec_1>,<ndec_2>, ...,<ndec_15>` is the number of decimals for the i:th argument. It has the type integer. The default number of decimals is 3 which will be the result if FORMAT is not given. It is of course only necessary to give the number of decimals for arguments of type DECIMAL. In other cases it can be omitted (i.e. given as ,,) or given as 0.

The attributes ARGUMENTS and FORMAT must be given together and with the same number of parameters.

FILENAME=<file_name>

`<file_name>` is the file where the table will be presented when the PRESENT statement is used. It has the type STRING.

QUOTES

When this attribute is given in the TABLE statement all terms will be surrounded by ''. The resulting file can then directly be used as input to the Tribon Report Generator.

NOAPPEND

When this attribute is given in the TABLE statement the old contents in `<tab_name>` will be deleted and the `<tab_name>` can be used for another table. The attribute must also be given the first time `<tab_name>` is used.

The TEXT Statement

TEXT,<txt_name>,<txt_pnt>
[/TEXTLINE=<txt_str>]
[/TEXTDECNO=<txt_dec>]
[/TEXTINTNO=<txt_int>]
[/TEXTANGLE=<txt_ang>]

[/TEXTHEIGHT=<txt_hgt>]
[/TEXTFONT=<txt_fnt>];

`<txt_name>` is the name of the text line and has the type `TEXT_2D`. The maximum length of `<txt_name>` is 32 characters.

`<txt_pnt>` is the text origin which is the lower left corner of the text. It has the type `POINT_2D`.

TEXTDECNO=<txt_dec>
`<txt_dec>` is a number which shall be presented as a text. It has the type DECIMAL.

TEXTINTNO=<txt_int>
`<txt_int>` is a number which will be presented as a text. It has the type INTEGER.

TEXTLINE=<txt_str>
`<txt_str>` is a string which will be presented as a text. It has the type STRING.

TEXTANGLE=<txt_ang>
`<txt_ang>` is the text angle and has the type DECIMAL. The default angle is 0.0 but the text angle can be changed with this attribute.

TEXTHEIGHT=<txt_hgt>
`<txt_hgt>` is the text height and has the type DECIMAL. The text is given a default height value but the text height can be changed with this attribute.

TEXTFONT=<txt_fnt>
`<txt_fnt>` is the text font and has the type INTEGER. Available font numbers are 0-99. The default font number is 0.

Structure:

`TEXT`	(STRING)
`PNT(1:2)`	(DECIMAL)
`ANGLE`	(DECIMAL)
`HEIGHT`	(DECIMAL)

Example:
```
GET/STRUCTURE=(T,<txt_name>,'TEXT'
 /STRUCTURE=(X,<txt_name>,'PNT',1)
 /STRUCTURE=(A,<txt_name>,'ANGLE')
 /STRUCTURE=(H,<txt_name>,'HEIGHT');
```

The TEXTFILE Statement

TEXTFILE,<txf_name>,<file_name>
[/INTERLINESPACE=<intlin_space>]
[/POSLINES=(<start_pnt>,<first_line>,<last_line>)]
[/TEXTANGLE=<text_ang>]
[/TEXTHEIGHT=<text_hgt>]
[/TEXTFONT=<text_fnt>];

`<txf_name>` is the name of the text file variable and has the type `TEXTFILE_2D`. The maximum length of `<txf_name>` is 32 characters.

`<file_name>` is the name of the file from where the text is fetched. It has the type STRING.

INTERLINESPACE=<intlin_space>
`<intlin_space>` is the factor for the interline space. The interline space is defined as the factor * the text height. It has the type DECIMAL. The default factor is 1.5.

POSLINES=(<start_pnt>,<first_line>,<last_line>)
`<start_pnt>` is the origin of the first text line and the lower left corner of that line. It has the type `POINT_2D`.
`<first_line>` is the number of the first line to be presented. It has the type INTEGER.

`<last_line>` is the number of the last line to be presented. It has the type INTEGER.

TEXTANGLE=<text_ang>
`<text_ang>` is the text angle and has the type DECIMAL. The default angle is 0.0 but the text angle can be changed with this attribute.

TEXTHEIGHT=<text_hgt>
`<text_hgt>` is the next height and has the type DECIMAL. The text is given a default height value but the text height can be changed with this attribute.

TEXTFONT=<text_fnt>
`<text_fnt>` is the text font and has the type INTEGER. Available font numbers are 0-99. The default font number is 0.

It is possible to give both absolute and relative values of the lines to be presented. This is illustrated by the following examples:

/POSLINES=(P1,1,15) lines 1 to 15

/POSLINES=(P1,10,) next 10 lines (16 to 25)

/POSLINES=(P1,20,) next 20 lines (26 to 45)

/POSLINES=(P1,,10) previous 10 lines (16 to 25)

/POSLINES=(P1,,) the whole file

Structure:

FILENAME	(STRING)	
ANGLE	(DECIMAL)	
HEIGHT	(DECIMAL)	
ILSP	(DECIMAL)	
NDATA	(INTEGER)	
DATA(1:NDATA)		
	PNT(1:2)	(DECIMAL)
	FIRST	(INTEGER)
	LAST	(INTEGER)

Example:
```
GET/STRUCTURE=(F,<txf_name>,'FILENAME'
 /STRUCTURE=(A,<txf_name>,'ANGLE')
 /STRUCTURE=(H,<txf_name>,'HEIGHT')
 /STRUCTURE=(I,<txf_name>,'ILSP')
 /STRUCTURE=(N,<txf_name>,'NDATA')
 /STRUCTURE=(X,<txf_name>,'PNT',N,1)
 /STRUCTURE=(S,<txf_name>,'FIRST',N)
 /STRUCTURE=(E,<txf_name>,'LAST',N);
```

The TOROID Statement

TOROID,<tor_name>,<rad>
/COORDTOR=(<cl_pnt_1>,cl_pnt_2>,<cl_pnt_3>);

`<tor_name>` is the name of the toroid and has the type `TOROID_3D`. The maximum length of `<tor_name>` is 32 characters.

`<rad>` is the radius of the toroid. It has the type DECIMAL.

/COORDTOR=(<cl_pnt_1>,cl_pnt_2>,<cl_pnt_3>);
`<cl_pnt_1>` is the starting point, `<cl_pnt_2>` is the mid point and `<cl_pnt_3>` is the ending point of the toroid centre line. All have the type `POINT_3D`.

Structure:

```
STARTPNT(1:3)    (DECIMAL)
ENDPNT(1:3)      (DECIMAL)
AMPLITUDE(1:3)   (DECIMAL)
```

Example:

```
GET/STRUCTURE=(X,<tor_name>,'STARTPNT',1)
 /STRUCTURE=(Y,<tor_name>,'ENDPNT',2)
 /STRUCTURE=(Z,<tor_name>,'AMPLITUDE',3);
```

The VECTOR_2D Statement

VECTOR_2D,<vect_name>,<x>,<y>;

`<vect_name>` is the name of the vector and has the type `VECTOR_2D`. The maximum length of `<vect_name>` is 32 characters.

<x> and <y> are the x and y coordinates of the vector, both with type DECIMAL.

If only one of the coordinates of a previously defined point shall be changed, the other one is simply omitted.

Structure:

```
VEC(1:2)    (DECIMAL)
```

Example:)

```
GET/STRUCTURE=(X,<vect_name>,'VEC',1
 /STRUCTURE=(Y,<vect_name>,'VEC','Y');
```

The VECTOR_3D Statement

VECTOR_3D,<vect_name>,<x>,<y>,<z>;

`<vect_name>` is the name of the vector and has the type `VECTOR_3D`. The maximum length of `<vect_name>` is 32 characters.

<x>, <y> and <z> are the x, y and z coordinates of the vector, all with type DECIMAL.

If only one of the coordinates of a previously defined point shall be changed, the others are simply omitted.

Structure:

```
VEC(1:3)    (DECIMAL)
```

Example:)

```
GET/STRUCTURE=(X,<vect_name>,'VEC',1
 /STRUCTURE=(Y,<vect_name>,'VEC','Y')
 /STRUCTURE=(Z,<vect_name>,'VEC','Z');
```

The VMSJOB Statement

VMSJOB,<jobfilename>;

The parameter `<jobfilename>` is the name of the file to be executed. It has the type STRING. The maximum length of `<jobfilename>` is 100 characters.

The WHILE Statement

WHILE,<cond>;

`<cond>` is the condition to be tested. It has the type BOOLEAN.

17.4.2 Example of Macros Creating 2D Geometry

Example 1

This example is a macro creating some simple geometry.

```
!
! All macros begin with the MACRO-stmt
!
MACRO,RECTANGLE;
  GET/POINT_2D=('Give first corner',P1)
     /POINT_2D=('Give second corner',P3);
  ASSIGN,X1,P1/XCOORD;
  ASSIGN,Y1,P1/YCOORD;   !  The x- and y-coordinates
  ASSIGN,X2,P3/XCOORD;   !  of the 2 points are needed
  ASSIGN,Y2,P3/YCOORD;
  POINT_2D,P2,X1,Y2;     !  Create the other
  POINT_2D,P4,X2,Y1;     !  two corners
  CONTOUR,CNT,P1
    /LINEEND = P2
    /LINEEND = P3        !  Create a contour
    /LINEEND = P4
    /LINEEND = P1;
  PRESENT,CNT;           !  Display the contour
  !
  !  All macros end with the ENDMACRO-stmt
  !
ENDMACRO;
```

In the macro RECTANGLE, the contour was created using the four 2D points. In some cases, it might be more convenient to define a contour with lines instead. The macro above will then look like this:

```
MACRO,RECTANGLE;
  GET/POINT_2D=('Give first corner',P1)
     /POINT_2D=('Give second corner',P3);
  ASSIGN,X1,P1/XCOORD;
  ASSIGN,Y1,P1/YCOORD;
  ASSIGN,X2,P3/XCOORD;
  ASSIGN,Y2,P3/YCOORD;
  POINT_2D,P2,X1,Y2;
  POINT_2D,P4,X2,Y1;
  LINE,L1,P1/LINEEND=P2;
  LINE,L2,P2/LINEEND=P3;
  LINE,L3,P3/LINEEND=P4;
  LINE,L4,P4/LINEEND=P1;
  CONTOUR,CNT
    /LINE = L1
    /LINE = L2
    /LINE = L3
    /LINE = L4;
  PRESENT,CNT;
ENDMACRO;
```

Note that the CONTOUR statement looks different now. In the first case, the start point must be given but in the second case this point is the start point of the first line.

Example 2

The macro SCREW below has 4 parameters. One of them, the drawing name, is given in the MACRO statement. This parameter tells where the result of the macro shall be stored when executed by the stand alone program. A drawing will be created using the form A3 and the scale chosen is 1:1. The NAME statement has no effect when the macro is executed interactively at a workstation. The result is then put into the current drawing.

The macro can be found by following this link:

macro_screw.txt

The figure below shows the result of the execution of the macro SCREW. The current drawing contained the rectangle and the length of the screw was given by using the different point modes, in this case the node mode. The start point was given with the close point mode.

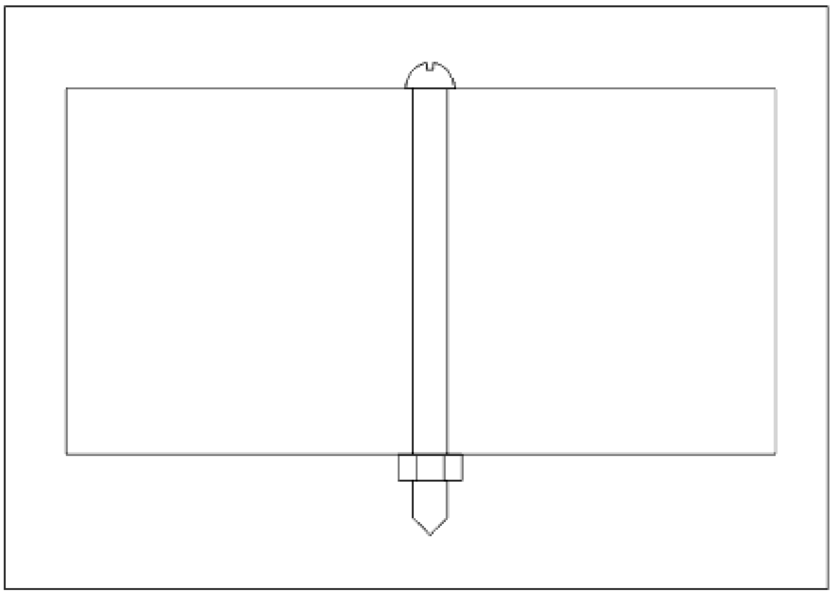


Figure 17:1. The result of the execution of the macro SCREW.

17.4.3 A Macro Generating a Volume

Example 3

The macro PUMP is an example of a geometry macro creating a volume. It uses most of the volume primitives. Note that the general cylinder has a 2D parameter but the result is 3D. If the macro is run stand alone, the NAME statement gives the name of the volume model on the data bank. The x- and y-extensions are also given.

The macro can be found by following this link:

macro_pump.txt

The figure below shows the result of the execution of the macro PUMP. The input angles were 180 degrees and 45 degrees respectively. The dimension was given as the distance between two cursor positions, both for the pump length and the pump height.

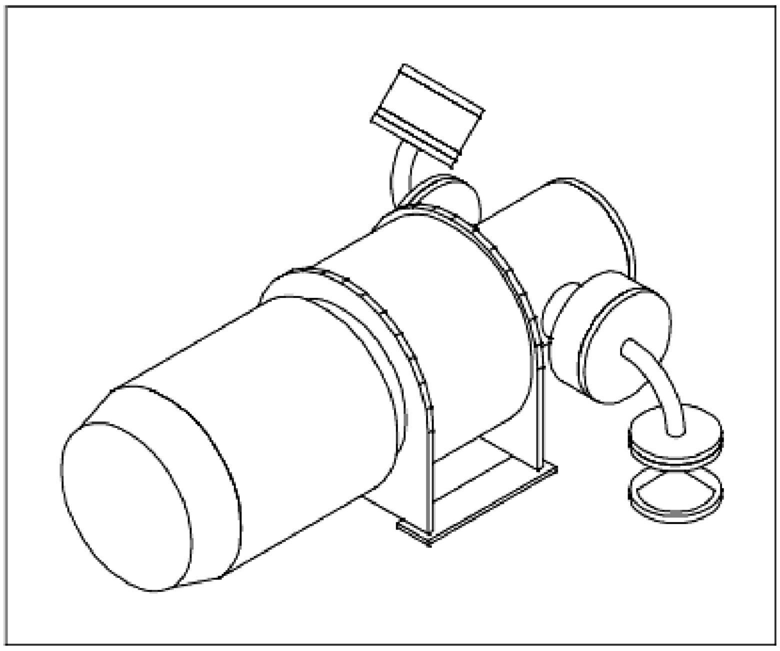


Figure 17:2. The result of the execution of the macro PUMP.

```
MACRO,PUMP;
  NAME,'MOORINGPUMP'/VOLUME=(3000,3000);
  GET/DECIMAL  =('Angle for pressure side: ',ANG1)
     /DECIMAL  =('Angle for suction side: ',ANG2)
     /POINT_3D =('Start point: ',STP)
     /DISTANCE =('Pump length: ',LEN)
     /DISTANCE =('Pump height: ',DIA);
  ASSIGN,X,STP/XCOORD;
  ASSIGN,Y,STP/YCOORD;
  ASSIGN,Z,STP/ZCOORD;
  ASSIGN,L,LEN%1250.0;
  ASSIGN,R,DIA%400.0%2;
  ASSIGN,L1,100*L;
  ASSIGN,L2,405*L;
  ASSIGN,L3,50*L;
  ASSIGN,L4,15*L;
  ASSIGN,L5,270*L;
  ASSIGN,L6,L4;
  ASSIGN,L7,L4;
  ASSIGN,L8,305*L;
  ASSIGN,L9,L4;
  ASSIGN,L10,25*L;
  ASSIGN,A11,35*L;
  ASSIGN,L12,330*L;
  ASSIGN,H12,10%330.0*L12;
  ASSIGN,W12,80%330.0*L12;
  ASSIGN,L13,L12;
  ASSIGN,H13,H12;
  ASSIGN,W13,W12;
  ASSIGN,L14,75*L;
  ASSIGN,L15,L14;
  ASSIGN,L16,L4;
  ASSIGN,L17,L4;
  ASSIGN,L18,L4;
  ASSIGN,L19,L4;
  ASSIGN,L20,125*L;
  ASSIGN,L21,L4;
  ASSIGN,L22,L4;
  ASSIGN,L23,L3;
  ASSIGN,L24,L3;
  ASSIGN,L25,L4;
  ASSIGN,L26,L20;
  ASSIGN,L27,L4;
  ASSIGN,L28,L4;
  ASSIGN,L29,L14;
  ASSIGN,L30,L4;
  ASSIGN,R1A,300*R;
  ASSIGN,R1B,350*R;
  ASSIGN,R2,R1B;
  ASSIGN,R3,R1A;
  ASSIGN,R4,400*R;
  ASSIGN,R5,375*R;
  ASSIGN,R7,R1A;
  ASSIGN,R8,225*R;
  ASSIGN,R9,R8;
  ASSIGN,R10,150*R;
  ASSIGN,R11,R10;
  ASSIGN,R14A,75*R;
  ASSIGN,R14B,100*R;
  ASSIGN,R15A,R14A;
  ASSIGN,R15B,R14B;
  ASSIGN,R16,200*R;
  ASSIGN,R17,R16;
  ASSIGN,R18,R16;
```

```
ASSIGN,R19,R16;
ASSIGN,R20,60*R;
ASSIGN,R21,R10;
ASSIGN,R22,R10;
ASSIGN,R23A,10*R;
ASSIGN,R23B,125*R;
ASSIGN,R24A,R23A;
ASSIGN,R24B,R23B;
ASSIGN,R26,R20;
ASSIGN,R25,R10;
ASSIGN,R27,R10;
ASSIGN,R28,R10;
ASSIGN,R29,R10;
ASSIGN,R30,R10;
ASSIGN,XE1,X+L1;
ASSIGN,YE1,Y;
ASSIGN,ZE1,Z;
VECTOR_3D,VEC1,1,0,0;
POINT_3D,PE1,XE1,YE1,ZE1;
CONE,CON1,R1A,R1B/COORDCONE=(P,PE1);
COLOUR,'RED';
PRESENT,CON1;
ASSIGN,XE2,L2+XE1;
POINT_3D,PE2,XE2,YE1,ZE1;
CYLINDER,CYL1,R2/COORDCYL=(PE1,PE2);
PRESENT,CYL1;
ASSIGN,XE3,L3+XE2;
POINT_3D,PE3,XE3,YE1,ZE1;
CYLINDER,CYL2,R3/COORDCYL=(PE2,PE3);
PRESENT,CYL2;
ASSIGN,XE4,L4+XE3;
POINT_3D,PE4,XE4,YE1,ZE1;
ASSIGN,R42,R4%2;
POINT_2D,PC1,R4,R42;
POINT_2D,PC2,R4,-R42;
POINT_2D,PC3,-R4,-R42;
POINT_2D,PC4,-R4,R42;
POINT_2D,PC5,0,1.5*R4;
POINT_3D,PGC1,XE3,YE1,ZE1-R42;
POINT_3D,PGC2,XE3,YE1+1,ZE1-R42;
POINT_3D,PGC3,XE3,YE1,ZE1+1-R42;
CONTOUR,CNT1,PC1
  /LINEEND=PC2
  /LINEEND=PC3
  /LINEEND=PC4
  /ARCMIDPNT=(PC5,PC1);
GENERALCYLINDER,GCL1,CNT1,L4,PGC1,PGC2,PGC3;
COLOUR,'GREEN';
PRESENT,GCL1;
ASSIGN,XE5,L5+XE4;
POINT_3D,PE5,XE5,YE1,ZE1;
CYLINDER,CYL3,R5/COORDCYL=(PE4,PE5);
COLOUR,'MAGENTA';
PRESENT,CYL3;
ASSIGN,XE6,L6+XE5;
POINT_3D,PE6,XE6,YE1,ZE1;
POINT_3D,PGC4,XE5,YE1,ZE1-R42;
POINT_3D,PGC5,XE5,YE1+1,ZE1-R42;
POINT_3D,PGC6,XE5,YE1,ZE1+1-R42;
GENERALCYLINDER,GCL2,CNT1,L6,PGC4,PGC5,PGC6;
COLOUR,'GREEN';
PRESENT,GCL2;
POINT_3D,PPA1,XE3-L12%22,YE1-R4+(W12%2-0.2*W12),
       ZE1-R4-H12%2;
```

```
POINT_3D,PPA2,XE3-L12%22+L12,YE1-R4+(W12%2-0.2*W12),
ZE1-R4-H12%2;
POINT_3D,PPA3,XE3-L12%22+L12,YE1-R4+(W12-0.2*W12),
ZE1-R4;
PARALLELEPIPED,PAR1/CENLINPARA=(PPA1,PPA2,PPA3);
PRESENT,PAR1;
  POINT_3D,PPA4,XE3-L13%22,YE1+R4-(W13%2-0.2*W13),
          ZE1-R4-H13%2;
POINT_3D,PPA5,XE3-L13%22+L13,YE1+R4-(W13%2-0.2*W13),
ZE1-R4-H13%2;
POINT_3D,PPA6,XE3-L13%22+L13,YE1+R4+0.2*W13,
ZE1-R4;
PARALLELEPIPED,PAR2/CENLINPARA=(PPA4,PPA5,PPA6);
PRESENT,PAR2;
  ASSIGN,XE7,L7+XE6;
  POINT_3D,PE7,XE7,YE1,ZE1;
  CYLINDER,CYL4,R7/COORDCYL=(PE6,PE7);
  COLOUR,'CYAN';
  PRESENT,CYL4;
  ASSIGN,XE8,L8+XE7;
  POINT_3D,PE8,XE8,YE1,ZE1;
  CYLINDER,CYL5,R8/COORDCYL=(PE7,PE8);
  PRESENT,CYL5;
  ASSIGN,XE9,L9+XE8;
  POINT_3D,PE9,XE9,YE1,ZE1;
  CYLINDER,CYL6,R9/COORDCYL=(PE8,PE9);
  PRESENT,CYL6;
  ASSIGN,XE10,L10+XE9;
  POINT_3D,PE10,XE10,YE1,ZE1;
  CYLINDER,CYL7,R10/COORDCYL=(PE9,PE10);
  COLOUR,'MAGENTA';
  PRESENT,CYL7;
  ASSIGN,XE11,A11+XE10;
  POINT_3D,PE11,XE11,YE1,ZE1;
  SPHERESEG,SPH1,R11/COORDSEG=(PE10,PE11);
  PRESENT,SPH1;
  ASSIGN,XE14,XE8-170*L;
  ASSIGN,YS14,YE1-R8;
  ASSIGN,YE14,YS14-L14;
  POINT_3D,PS14,XE14,YS14,ZE1;
  POINT_3D,PE14,XE14,YE14,ZE1;
  CONE,CON2,R14B,R14A/COORDCONE=(PS14,PE14);
  COLOUR,'YELLOW';
  PRESENT,CON2;
  ASSIGN,XE16,XE14;
  ASSIGN,YS16,YE14;
  ASSIGN,YE16,YE14-L14;
  POINT_3D,PS16,XE16,YS16,ZE1;
  POINT_3D,PE16,XE16,YE16,ZE1;
  CYLINDER,CYL8,R16/COORDCYL=(PS16,PE16);
  COLOUR,'CYAN';
  PRESENT,CYL8;
  ASSIGN,XE18,XE16;
  ASSIGN,YS18,YE16;
  ASSIGN,YE18,YE16-L16;
  POINT_3D,PS18,XE18,YS18,ZE1;
  POINT_3D,PE18,XE18,YE18,ZE1;
  CYLINDER,CYL9,R18/COORDCYL=(PS18,PE18);
  PRESENT,CYL9;
  ASSIGN,COS1,COSD(ANG1);
  ASSIGN,SIN1,SIND(ANG1);
  ASSIGN,XS20,XE18;
  ASSIGN,XE20,XE18+SIN1*L20;;
  ASSIGN,YS20,YE18;
```

```
ASSIGN,YE20,YE18-L20;
ASSIGN,ZE20,ZE1+COS1*L20;
ASSIGN,XM20,XS20+(1-COSD(45))*SIN1*L20;
ASSIGN,YM20,YE18-COSD(45)*L20;
ASSIGN,ZM20,ZE1+(1-COSD(45))*COS1*L20;
POINT_3D,PS20,XS20,YS20,ZE1;
POINT_3D,PM20,XM20,YM20,ZM20;
POINT_3D,PE20,XE20,YE20,ZE20;
TOROID,TOR1,R20/COORDTOR=(PS20,PM20,PE20);
PRESENT,TOR1;
ASSIGN,XS21,XE20;
ASSIGN,XE21,XE20+SIN1*L21;
ASSIGN,YS21,YE20;
ASSIGN,YE21,YE20;
ASSIGN,ZS21,ZE20;
ASSIGN,ZE21,ZS21+COS1*L21;
POINT_3D,PS21,XS21,YS21,ZS21;
POINT_3D,PE21,XE21,YE21,ZE21;
CYLINDER,CYL21,R21/COORDCYL=(PS21,PE21);
COLOUR,'RED';
PRESENT,CYL21;
ASSIGN,XS22,XE21;
ASSIGN,XE22,XE21+SIN1*L22;
ASSIGN,YS22,YE21;
ASSIGN,YE22,YE21;
ASSIGN,ZS22,ZE21;
ASSIGN,ZE22,ZS22+COS1*L22;
POINT_3D,PS22,XS22,YS22,ZS22;
POINT_3D,PE22,XE22,YE22,ZE22;
CYLINDER,CYL22,R22/COORDCYL=(PS22,PE22);
PRESENT,CYL22;
ASSIGN,XS23,XE22;
ASSIGN,XE23,XE22+SIN1*L23;
ASSIGN,YS23,YE22;
ASSIGN,YE23,YE22;
ASSIGN,ZS23,ZE22;
ASSIGN,ZE23,ZS23+COS1*L23;
POINT_3D,PS23,XS23,YS23,ZS23;
POINT_3D,PE23,XE23,YE23,ZE23;
CONE,CON23,R23B,R23A/COORDCONE=(PS23,PE23);
COLOUR,'WHITE';
PRESENT,CON23;
ASSIGN,XS24,XE23;
ASSIGN,XE24,XE23+SIN1*L24;
ASSIGN,YS24,YE23;
ASSIGN,YE24,YE23;
ASSIGN,ZS24,ZE23;
ASSIGN,ZE24,ZS24+COS1*L24;
POINT_3D,PS24,XS24,YS24,ZS24;
POINT_3D,PE24,XE24,YE24,ZE24;
CONE,CON24,R24A,R24B/COORDCONE=(PS24,PE24);
PRESENT,CON24;
ASSIGN,XS25,XE24;
ASSIGN,XE25,XE24+SIN1*L25;
ASSIGN,YS25,YE24;
ASSIGN,YE25,YE24;
ASSIGN,ZS25,ZE24;
ASSIGN,ZE25,ZS25+COS1*L25;
POINT_3D,PS25,XS25,YS25,ZS25;
POINT_3D,PE25,XE25,YE25,ZE25;
CYLINDER,CYL25,R25/COORDCYL=(PS25,PE25);
COLOUR,'CYAN';
PRESENT,CYL25;
ASSIGN,XE15,XE14;
```

```
ASSIGN,YS15,YE1+R8;
ASSIGN,YE15,YS15+L15;
POINT_3D,PS15,XE15,YS15,ZE1;
POINT_3D,PE15,XE15,YE15,ZE1;
CONE,CON3,R15B,R15A/COORDCONE=(PS15,PE15);
COLOUR,'YELLOW';
PRESENT,CON3;
ASSIGN,XE17,XE15;
ASSIGN,YS17,YE15;
ASSIGN,YE17,YE15+L17;
POINT_3D,PS17,XE17,YS17,ZE1;
POINT_3D,PE17,XE17,YE17,ZE1;
CYLINDER,CYL10,R17/COORDCYL=(PS17,PE17);
COLOUR,'CYAN';
PRESENT,CYL10;
ASSIGN,XE19,XE17;
ASSIGN,YS19,YE17;
ASSIGN,YE19,YE17+L19;
POINT_3D,PS19,XE19,YS19,ZE1;
POINT_3D,PE19,XE19,YE19,ZE1;
CYLINDER,CYL11,R19/COORDCYL=(PS19,PE19);
PRESENT,CYL11;
ASSIGN,COS2,COSD(ANG2);
ASSIGN,SIN2,SIND(ANG2);
ASSIGN,XS26,XE19;
ASSIGN,XE26,XE19+SIN2*L26;
ASSIGN,YS26,YE19;
ASSIGN,YE26,YE19+L26;
ASSIGN,ZE26,ZE1+COS2*L26;
ASSIGN,XM26,XS26+(1-COSD(45))*SIN2*L26;
ASSIGN,YM26,YE19+COSD(45)*L26;
ASSIGN,ZM26,ZE1+(1-COSD(45))*COS2*L26;
POINT_3D,PS26,XS26,YS26,ZE1;
POINT_3D,PM26,XM26,YM26,ZM26;
POINT_3D,PE26,XE26,YE26,ZE26;
TOROID,TOR2,R26/COORDTOR=(PS26,PM26,PE26);
PRESENT,TOR2;
ASSIGN,XS27,XE26;
ASSIGN,XE27,XE26+SIN2*L27;
ASSIGN,YS27,YE26;
ASSIGN,YE27,YE26;
ASSIGN,ZS27,ZE26;
ASSIGN,ZE27,ZS27+COS2*L27;
POINT_3D,PS27,XS27,YS27,ZS27;
POINT_3D,PE27,XE27,YE27,ZE27;
CYLINDER,CYL27,R27/COORDCYL=(PS27,PE27);
COLOUR,'RED';
PRESENT,CYL27;
ASSIGN,XS28,XE27;
ASSIGN,XE28,XE27+SIN2*L28;
ASSIGN,YS28,YE27;
ASSIGN,YE28,YE27;
ASSIGN,ZS28,ZE27;
ASSIGN,ZE28,ZS28+COS2*L28;
POINT_3D,PS28,XS28,YS28,ZS28;
POINT_3D,PE28,XE28,YE28,ZE28;
CYLINDER,CYL28,R28/COORDCYL=(PS28,PE28);
PRESENT,CYL28;
ASSIGN,XS29,XE28;
ASSIGN,XE29,XE28+SIN2*L29;
ASSIGN,YS29,YE28;
ASSIGN,YE29,YE28;
ASSIGN,ZS29,ZE28;
ASSIGN,ZE29,ZS29+COS2*L29;
```

```
  POINT_3D,PS29,XS29,YS29,ZS29;
  POINT_3D,PE29,XE29,YE29,ZE29;
  CYLINDER,CYL29,R29/COORDCYL=(PS29,PE29);
  COLOUR,'WHITE';
  PRESENT,CYL29;
  ASSIGN,XS30,XE29;
  ASSIGN,XE30,XE29+SIN2*L30;
  ASSIGN,YS30,YE29;
  ASSIGN,YE30,YE29;
  ASSIGN,ZS30,ZE29;
  ASSIGN,ZE30,ZS30+COS2*L30;
  POINT_3D,PS30,XS30,YS30,ZS30;
  POINT_3D,PE30,XE30,YE30,ZE30;
  CYLINDER,CYL30,R30/COORDCYL=(PS30,PE30);
  COLOUR,'RED';
  PRESENT,CYL30;
  COLOUR,'GREEN';
ENDMACRO;
```

17.4.4 The Possibility to Present a Text File on a Drawing

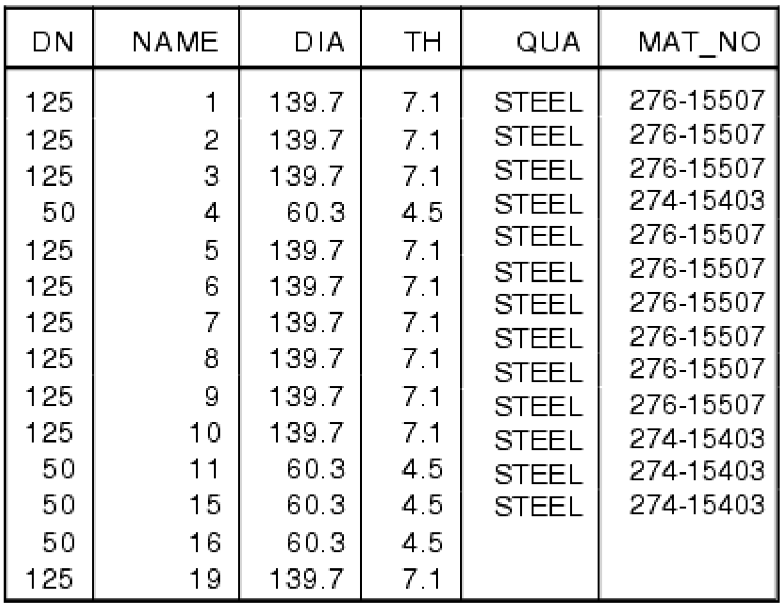
Example 4

The macro LIST uses the TEXT and TEXTFILE statements to present a table on a drawing. The data are stored on a file. A heading is added and a frame is drawn.

The macro can be found by following this link:

macro_list.txt

The result of the macro is shown in the figure below.



DN	NAME	DIA	TH	QUA	MAT_NO
125	1	139.7	7.1	STEEL	276-15507
125	2	139.7	7.1	STEEL	276-15507
125	3	139.7	7.1	STEEL	276-15507
50	4	60.3	4.5	STEEL	274-15403
125	5	139.7	7.1	STEEL	276-15507
125	6	139.7	7.1	STEEL	276-15507
125	7	139.7	7.1	STEEL	276-15507
125	8	139.7	7.1	STEEL	276-15507
125	9	139.7	7.1	STEEL	276-15507
125	10	139.7	7.1	STEEL	276-15507
50	11	60.3	4.5	STEEL	274-15403
50	15	60.3	4.5	STEEL	274-15403
50	16	60.3	4.5	STEEL	274-15403
125	19	139.7	7.1		

Figure 17:3. The result from the execution of the macro LIST.

```
MACRO,LIST;
  GET
   /POINT_2D=('Give upper left corner of frame',FRM);
  ASSIGN,X,FRM/XCOORD;
  ASSIGN,Y,FRM/YCOORD;
  ASSIGN,TXH,3.5;
  ASSIGN,ILSP,1.5*TXH;
  ASSIGN,DX,45*TXH;
!
! Give number of lines
!
  ASSIGN,N,14;
  ASSIGN,DY,(N+3)*ILSP;
!
! Create frame
!
  POINT_2D,P1,X+DX,Y;
  POINT_2D,P2,X+DX,Y-DY;
  POINT_2D,P3,X,Y-DY;
  CONTOUR,CNT,FRM/LINEEND=P1/LINEEND=P2
                  /LINEEND=P3/LINEEND=FRM;
PRESENT,CNT;
!
! Create vertical lines between the rows
!
POINT_2D,P4,X+6*TXH,Y;
POINT_2D,P5,X+6*TXH,Y-DY;
LINE,L1,P4/LINEEND=P5;
PRESENT,L1;
POINT_2D,P6,X+11*TXH,Y;
POINT_2D,P7,X+11*TXH,Y-DY;
LINE,L2,P6/LINEEND=P7;
PRESENT,L2;
POINT_2D,P8,X+19*TXH,Y;
POINT_2D,P9,X+19*TXH,Y-DY;
LINE,L3,P8/LINEEND=P9;
PRESENT,L3;
POINT_2D,P10,X+25*TXH,Y;
POINT_2D,P11,X+25*TXH,Y-DY;
LINE,L4,P10/LINEEND=P11;
PRESENT,L4;
POINT_2D,P12,X+33*TXH,Y;
POINT_2D,P13,X+33*TXH,Y-DY;
LINE,L5,P12/LINEEND=P13;
PRESENT,L5;
!
! Put a heading
!
ASSIGN,DY,ILSP;
POINT_2D,TXP,X,Y-DY;
TEXT,TXT,TXP/TEXTLINE=
'   DN  NAME   DIA     TH    QUA      MAT_NO';
PRESENT,TXT;
POINT_2D,P14,X,Y-DY-ILSP;
POINT_2D,P15,X+DX,Y-DY-ILSP;
LINE,L6,P14/LINEEND=P15;
PRESENT,L6;
!
! Put textfile
!
POINT_2D,TXP,X,Y-DY-2*ILSP;
TEXTFILE,TXF,'STEEL.DAT'/POSLINES=(TXP,1,N);
PRESENT,TXF;
ENDMACRO;
```

17.4.5 The Usage of Attributes in Geometry Macros

Example 5

The macro ATTR shows the usage of the ATTRIBUTE statement. Note that it is possible to give the attribute number as an alias, a number or a variable.

```
MACRO,ATTR;
  ASSIGN,LS,105.6;
  ASSIGN,DS,0.2543;
  ATTRIBUTE,'CHAIR'/ATTRDATA=(I2,4)
/ATTRDATA=(R1,LS)
/ATTRDATA=(R2,DS)
/ATTRDATA=(R3,35.45)
/ATTRDATA=(S2,'I2: chair no')
/ATTRDATA=(S3,'R1-R3: chair data');
ATTRIBUTE,55/ATTRDATA=(I,3)
/ATTRDATA=(R4,1.234);
ASSIGN,ATTNO,7777;
GET/STR=('Give a string: ',STR);
ATTRIBUTE,ATTNO/ATTRDATA=('S1',STR);
ENDMACRO;
```

The result of a stand alone execution of the macro ATTR is shown below.

```
RUN SB_SYSTEM:SZ006
Present Geometry Macro:
(0)  Exit
(1)  Print on terminal
(2)  Create 2D geometry and store on DB
(3)  Create 3D volume model and store on DB
(4)  Create 3D volume model + picture and store on DB
Enter activity :  1
General Component Data Bank not assigned!
Give name of macro to be run:  ATTR
ATTNO = 1111
I1                 0
I2                 4
R1         105.60000
R2           0.25430
R3          35.45000
S1
S2      I2: chair no
S3      R1-R3: chair data
ATTNO =             55
R1           0.00000
R2           0.00000
R3           0.00000
R4           1.23400
S1
S2
```

```
S3     3
Give a string:
>  Just a test string !!!
ATTNO =          7777
S1     Just a test string !!!
Once more ? N
Give name of macro to be run:
```

17.4.6 Layer Handling

Example 6

The macro LAYER shows the layer alias and layer class facilities. Note that a layer can be given as an alias or as a number. The layer class can be given either as the class name or the class number, each preceded by '#'.

```
MACRO,LAYER;
  GET/STRING=('Layer alias :  ',LALIAS)
     /STRING=('Layer class :  ',LCLASS)
     /INTEGER=('Layer number: ',LNUMBER);
  LAYER,'NOLL';
  LAYER,'FORM';
  LAYER,1001;
  LAYER,'#DRA';
  LAYER,'#1';
  LAYER,LALIAS;
  LAYER,LCLASS;
  LAYER,LNUMBER;
ENDMACRO;
```

The result of a stand alone execution of the macro LAYER is shown below.

```
Present Geometry Macro:
(0)  Exit
(1)  Print on terminal
(2)  Create 2D geometry and store on DB
(3)  Create 3D volume model and store on DB
(4)  Create 3D volume model + picture and store on DB
  Enter activity :  1
  General Component Data Bank not assigned!
  Give name of macro to be run:  LAYER
  Layer alias :
  >  FORM
  Layer class :
  >  #AAA
  Layer number:
  >  12
  LAYER & ALIAS:           0   ZERO
  LAYER & ALIAS:         300   FORM
  LAYER & ALIAS:        1001   VIEW1
  LAYER CLASS:            10   DRA
  LAYER CLASS:             1   PIPE
  LAYER & ALIAS:         300   FORM
  LAYER CLASS:            11   AAA
  LAYER & ALIAS:          12   DOZEN
  Once more ? Y
  Layer alias :
  >  FROM
  Layer class :
```

```
>  #DRAW
Layer number:
>  100
LAYER & ALIAS:            0   ZERO
LAYER & ALIAS:          300   FORM
LAYER & ALIAS:         1001   VIEW1
LAYER CLASS:             10   DRA
LAYER CLASS:              1   PIPE
Layer alias FROM not defined. Layer ignored!
Layer class DRAW not found. Layer ignored!
LAYER:     100
Once more ? N
Give name of macro to be run:
```

17.4.7 The Facility of Using Conditional Statements in a Macro

Example 7

This example illustrates the possibility of using the IF and WHILE statement to test the input data. Such a test can be useful to avoid runtime errors.

```
MACRO,INPUT;
  ASSIGN,CONT,1;
  WHILE,CONT == 1;
    GET/STR=('Key in colour',COL);
    IF,COL == 'GREEN' OR COL == 'CYAN'
      OR COL == 'BLUE' OR COL == 'MAGENTA'
      OR COL == 'RED' OR COL == 'YELLOW'
      OR COL == 'WHITE';
      ASSIGN,CONT,0;
    ENDIF;
  ENDWHILE;
  !
  ! When desired colour has been given the rest
  ! of the macro is executed.
  !
ENDMACRO;
```

It might be more convenient to have such a parameter check as a submacro which would look like follows.

```
MACRO,INPUT_COLOUR,COL;
  DECLARE,COL,STRING;
  ASSIGN,CONT,1;
  WHILE,CONT == 1;
    GET/STR=('Key in colour',COL);
    IF,COL == 'GREEN' OR COL == 'CYAN'
      OR COL == 'BLUE' OR COL == 'MAGENTA'
      OR COL == 'RED' OR COL == 'YELLOW'
      OR COL == 'WHITE';
      ASSIGN,CONT,0;
    ENDIF;
  ENDWHILE;
ENDMACRO;
The usage of INPUT_COLOUR could be like this.
MACRO,MAIN;
  DECLARE,C,STRING;
  CALL,INPUT_COLOUR,C;
  COLOUR,C;
  !
ENDIF;
```

17.4.8 The Possibility to Write Recursive Geometry Macros

Example 8

This example illustrates the possibility of writing a recursive macro, i.e. a macro which calls itself as a submacro. It uses a modified version of the macro RECTANGLE given in *Appendix 2*.

```
MACRO,RECPNT,PNT1,PNT2;
  DECLARE,PNT1,POINT_2D;
  DECLARE,PNT2,POINT_2D;
  CALL,RECTANGLE,PNT1,PNT2;
  ASSIGN,X1,PNT1/XCOORD;
  ASSIGN,Y1,PNT1/YCOORD;
  !
  !  When the condition is not fulfilled
  !  no further calls to RECPNT are made
  !  and the execution is terminated
  !
  IF,X1 < 250;
    ASSIGN,X2,PNT2/XCOORD;
    ASSIGN,Y2,PNT2/YCOORD;
    POINT_2D,PNT3,X2+X2-X1,Y2+Y2-Y1;
    CALL,RECPNT,PNT2,PNT3;
  ENDIF;
ENDMACRO;
MACRO,RECTANGLE,P1,P3;
  DECLARE,P1,POINT_2D;
  DECLARE,P3,POINT_2D;
  ASSIGN,X1,P1/XCOORD;
  ASSIGN,Y1,P1/YCOORD;
  ASSIGN,X3,P3/XCOORD;
  ASSIGN,Y3,P3/YCOORD;
  POINT_2D,P2,X1,Y3;
  POINT_2D,P4,X3,Y1;
  CONTOUR,CNT,P1
    /LINEEND = P2
    /LINEEND = P3
    /LINEEND = P4
    /LINEEND = P1;
  PRESENT,CNT;
ENDMACRO;
```

17.4.9 Change Colour in Drawing by Using Geometry Macro

Example 9

This macro is supposed to be started from the General Diagrams application.

All cables without interference class and component defined will be drawn red.

All cables with both interference class and component defined will be drawn green.

All cables with either interference class or component defined will be drawn blue.

The same function for Equipments with the data room and component.

```
MACRO, OUTF_DIAG_COLOUR;
! Declarations
DECLARE, A1,     STRING;
DECLARE, COMPNAME,       STRING;
DECLARE, DELIM,          STRING;
DECLARE, DWG,    STRING;
DECLARE, E1,     EXTRACT;
DECLARE, INDEX, INTEGER;
DECLARE, INT_C,          STRING;
DECLARE, LOOPMAX,        INTEGER;
DECLARE, MODNAME,        STRING;
DECLARE, PROJ,   STRING;
DECLARE, ROOM,   STRING;
DECLARE, STAT,   INTEGER;
DECLARE, SUB1,   STRING;
DECLARE, SUB2,   STRING;
DECLARE, SUB3,   STRING;
```

! Initiate

```
ASSIGN, DELIM,'-';
! Get the drawing name
DRAWING_NAME,DWG;
! Use data extraction to see which cables that are drawn in the diagram
ASSIGN,A1,'DRA(DWG).VIE(*).NCABLE';
EXTRACT,E1,A1;
GET/EXTRACT=(LOOPMAX,STAT,E1,DWG,,);
IF,STAT == 1;
LOOP,INDEX,1:LOOPMAX;
ASSIGN,A1,'DRA(DWG).VIE(*).CABLE(INDEX).NAME';
EXTRACT,E1,A1;
GET/EXTRACT=(MODNAME,STAT,E1,DWG,,INDEX,);
IF,STAT == 1;
! Get the cable data by using data extraction
SPLIT,MODNAME,DELIM,PROJ,CAB_SYS,CAB_NAM,SUB2,SUB3;
ASSIGN,CABLE_NAME,CAB_SYS&DELIM&CAB_NAM;
ASSIGN,A1,'CABLE(PROJ).CAB_M(CABLE_NAME). COMP_N';EXTRACT,E1,A1;
GET/EXTRACT=(COMPNAME,STAT,E1,PROJ,
CABLE_NAME,);
IF,STAT == 0;
```

```
ASSIGN,COMPNAME,'';
ENDIF;
ASSIGN,A1,
'CABLE(PROJ).CAB_M(CABLE_NAME).EL_PROP.INT_C';
EXTRACT,E1,A1;
GET/EXTRACT=(INT_C,STAT,E1,PROJ,CABLE_NAME,,);
IF,STAT == 0;
ASSIGN,INT_C,'';
ENDIF;
! Draw cables without any data red
IF,COMPNAME == '' AND INT_C == '';
CHANGEDRAW,'CABLE',MODNAME
/MARKINGCOLOUR='RED';
ELSE;
! Draw cables with all data green
IF,COMPNAME /= '' AND INT_C /= '';
CHANGEDRAW,'CABLE',MODNAME
/MARKINGCOLOUR='GREEN';
! Draw ca bles with some date blue
ELSE;
CHANGEDRAW,'CABLE',MODNAME
/MARKINGCOLOUR='BLUE';
ENDIF;
ENDIF;
ENDIF;
ENDLOOP;
ENDIF,
! Use data extraction to see which equipments that are drawn in the diagram
ASSIGN,A1,'DRA(DWG).VIE(*).NEQUIP';
EXTRACT,E1,A1;
GET/EXTRACT=(LOOPMAX,STAT,E1,DWG,,);
IF,STAT == 1;
LOOP,INDEX,1:LOOPMAX;
ASSIGN,A1,'DRA(DWG).VIE(*).EQUIP(INDEX).NAME';
EXTRACT,E1,A1;
GET/EXTRACT=(MODNAME,STAT,E1,DWG,,INDEX,);
IF,STAT == 1;
! Get the equipment data by using data extraction
SPLIT,MODNAME,DELIM,PROJ,EQ_NAME,SUB1,SUB2,SUB3;
ASSIGN,A1,'EQUIP(PROJ).ITEM(EQ_NAME).COMP_N';
EXTRACT,E1,A1;
GET/EXTRACT=(COMPNAME,STAT,E1,PROJ,EQ_NAME,);
IF,STAT == 0;
ASSIGN,COMPNAME,'';
ENDIF;
ASSIGN,A1,'EQUIP(PROJ).ITEM(EQ_NAME).ROOM';
EXTRACT,E1,A1;
GET/EXTRACT=(ROOM,STAT,E1,PROJ,EQ_NAME,);
```

```
IF,STAT == 0;
ASSIGN,ROOM,'';
ENDIF;
! Draw equipment items without any data red
IF,COMPNAME == '' AND ROOM == '';
CHANGEDRAW,'EQUIP',MODNAME/
MARKINGCOLOUR='RED';
ELSE;
! Draw equipment items with all data green
IF,COMPNAME /= '' AND ROOM /= '';
CHANGEDRAW,'EQUIP',MODNAME
/MARKINGCOLOUR='GREEN';
! Draw equipment items with some date blue
ELSE;
CHANGEDRAW,'EQUIP',MODNAME
/MARKINGCOLOUR='BLUE';
ENDIF;
ENDIF;
ENDIF;
ENDLOOP;
ENDIF,
ENDMACRO;
```

The following macro would change the colours of the equipments and cables back to be the one defined by the default file.

```
MACRO, OUTF_DIAG_DEFAULT;
! Declarations
DECLARE, A1,    STRING;
DECLARE, DWG,   STRING;
DECLARE, E1,    EXTRACT;
DECLARE, INDEX,         INTEGER;
DECLARE, LOOPMAX,       INTEGER;
DECLARE, MODNAME,       STRING;
DECLARE, STAT,  INTEGER;
! Get the drawing name
DRAWING_NAME,DWG;
! Use data extraction to see which cables that are drawn in the diagram
ASSIGN,A1,'DRA(DWG).VIE(*).NCABLE';
EXTRACT,E1,A1;
GET/EXTRACT=(LOOPMAX,STAT,E1,DWG,,);
IF,STAT == 1;
LOOP,INDEX,1:LOOPMAX;
ASSIGN,A1,'DRA(DWG).VIE(*).CABLE(INDEX).NAME';
EXTRACT,E1,A1;
GET/EXTRACT=(MODNAME,STAT,E1,DWG,,INDEX,);
IF,STAT == 1;
CHANGEDRAW,'CABLE',MODNAME
/MARKINGCOLOUR='DEFAULT';
ENDIF;
```

```
ENDLOOP;
ENDIF;
! Use data extraction to see which equipments that are drawn in the diagram
ASSIGN,A1,'DRA(DWG).VIE(*).NEQUIP';
EXTRACT,E1,A1;
GET/EXTRACT=(LOOPMAX,STAT,E1,DWG,,);
IF,STAT == 1;
LOOP,INDEX,1:LOOPMAX;
ASSIGN,A1,'DRA(DWG).VIE(*).EQUIP(INDEX).NAME';
EXTRACT,E1,A1;
GET/EXTRACT=(MODNAME,STAT,E1,DWG,,INDEX,);
IF,STAT == 1;
CHANGEDRAW,'EQUIP',MODNAME
/MARKINGCOLOUR='DEFAULT';
ENDIF;
ENDLOOP;
ENDIF;
ENDMACRO;
```