

Exit

# Contents

## **General**

[Introduction](#)

[Geometrical Representation](#)

[Matrices and Vectors](#)

## **Data Representation**

[Constants](#)

[Units](#)

[Data Types >](#)

[Entities and Data Blocks >](#)

## **Structure of TVG 4.2**

[File Header](#)

[File Body of Drawings >](#)

[File Body of Libraries >](#)



# Contents

## Data Types

Data Types <

Overview

char

short

long

double

MATRIX (Matrix)

DPOINT (Point Coordinates)

DRECT (Rectangle)

COLORREF (Color Description)

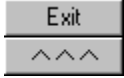
PROPERTY (Properties)

XPROPERTY (Extended Properties)

FONTDEF (Font Description)

TEXT (ANSI Text)

BINARY (Binary Data)



# Contents

## **Entities and Data Blocks**

Entities and Data Blocks <

Overview

Generic Data Blocks >

Native Data Blocks

Attribute Data Blocks

Standard Data Blocks >

Entity "Object" >

Entity "Instance"

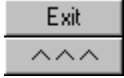
Entity "Block"

Entity "Group"

Entity "Position Number"

Entity "Custom-Defined"

Entity "End of List"



# Contents

## **Generic Data Blocks**

Entities and Data Blocks <

Generic Data Blocks <

Overview

Base Type "long"

Base Type "double"

Base Type "DPOINT"

Base Type "COLORREF"

Base Type "PROPERTY"

Base Type "XPROPERTY"

Base Type "FONTDEF"

Base Type "TEXT"

Base Type "BINARY"



# Contents

## Standard Data Blocks

[Entities and Data Blocks <](#)

[Standard Data Blocks <](#)

[Overview](#)

[Data Block Type 000 \(General Point\)](#)

[Data Block Type 001 \(Start-Point\)](#)

[Data Block Type 002 \(End-Point\)](#)

[Data Block Type 003 \(Center-Point\)](#)

[Data Block Type 004 \(Radius Definition Point\)](#)

[Data Block Type 005 \(Angle Definition Point\)](#)

[Data Block Type 006 \(Vector Definition Point\)](#)

[Data Block Type 007 \(First Pivot Point\)](#)

[Data Block Type 008 \(Second Pivot Point\)](#)

[Data Block Type 009 \(Arc End-Point\)](#)

[Data Block Type 010 \(Marking\)](#)

[Data Block Type 100 \(Constant\)](#)

[Data Block Type 101 \(Arc\)](#)

[Data Block Type 102 \(Curve\)](#)

[Data Block Type 110 \(Text\)](#)

[Data Block Type 220 \(Dimension Line\)](#)

[Data Block Type 225 \(Large Dimension\)](#)

[Data Block Type 230 \(Small Dimension\)](#)

[Data Block Type 235 \(Standard Text\)](#)

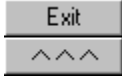
[Data Block Type 236 \(Frame Text\)](#)

[Data Block Type 237 \(Reference Text\)](#)

[Data Block Type 242 \(Clipping Surface\)](#)

[Data Block Type 243 \(Bitmap Reference\)](#)

[Data Block Type 999 \(End of List\)](#)



# Contents

## Entity "Object"

Entities and Data Blocks <

Entity "Object" <

Overview

Object 00 "Line"

Object 01 "Hatching"

Object 05 "Circle"

Object 06 "Circular Arc"

Object 07 "Circular Sector"

Object 08 "Circular Segment"

Object 10 "Zigzag Line"

Object 11 "Spline"

Object 12 "Curve"

Object 13 "Surface"

Object 15 "Ellipse"

Object 16 "Elliptical Arc"

Object 17 "Elliptical Sector"

Object 18 "Elliptical Segment"

Object 20 "Dimension Line Straight"

Object 21 "Dimension Line Curved"

Object 25 "Dimension Distance"

Object 26 "Dimension Radius"

Object 27 "Dimension Diameter"

Object 28 "Dimension Angle"

Object 29 "Dimension Arc Length"

Object 30 "Dimension Coordinate"

Object 31 "Dimension Area"

Object 32 "Dimension Perimeter"

Object 35 "Standard Text"

Object 36 "Frame Text"

Object 37 "Reference Text"

Object 40 "Comment"

Object 41 "Marking"

Object 42 "Clipping Surface"

Object 43 "Bitmap Reference"

Object 45 "Geometry Line"

Object 46 "Geometry Circle"

Object 47 "Geometry Ellipse"



# Contents

## File Body of Drawings

File Body of Drawings <

Overview

Section =DRAWING= (Drawing Information)

Section =TOOLBOX= (Toolbox)

Section =SYMBOL= (Symbol Window)

Section =KEYBOARD= (Keyboard)

Section =DEFAULT= (Defaults)

Section =USER= (Settings)

Section =MODULE= (Plug-In Settings)

Section =PAGE= (Page Format)

Section =COLOR= (Color Definitions)

Section =HATCH= (Hatching Types)

Section =MULTILINE= (Line Sequences)

Section =SYSTEM= (Coordinate Systems)

Section =PEN= (Pens)

Section =LINE= (Line Patterns)

Section =LAYER= (Layers)

Section =WINDOW= (Window Settings)

Section =BITMAP= (Embedded Bitmaps)

Section =BLOCK= (Block Definitions)

Section =OBJECT= (Object)

Section =EXIT= (End of File)

Minimal Drawing

 Contents

 Contents

## **File Body of Libraries**

[File Body of Libraries <](#)

[Overview](#)

[Section =LIBRARY= \(Library Information\)](#)

[Section =ATTRIB= \(Standard Attributes\)](#)

[Section =BLOCK= \(Block Definitions\)](#)

[Section =EXIT= \(End of File\)](#)

[Minimal Library](#)

[Standard Libraries](#)

[Font Libraries](#)



This command or function is (in this form) only available in Version 4.1 or higher of the TommySoftware® TVG File Format.

This command or function is (in this form) only available in Version 4.2 or higher of the TommySoftware® TVG File Format.

## Introduction (General)

The file format TVG 4.2 has been developed in order to store drawings, libraries and fonts created by TommySoftware® CAD applications. Its design allows other programs to read the data in such files easily.

In addition to the basic geometrical and logical drawing information, TVG files may contain settings and default values that will control the behavior of the CAD application after being loaded. Those settings and default values are very specific and will change frequently, so they should be ignored when reading TVG 4.2 files, and should not be written when creating a new or editing an existing TVG 4.2 file.

Files using the TVG 4.2 format should have a standard extension to be loaded easily by the application. The standard extension for drawings is `*.T4G`, the standard extension for libraries and fonts is `*.T4L`.

In this description of the TVG 4.2 file format, most data types will be defined and explained by simply stating their notation in the file. Such areas will be displayed in blue, using the font `Courier`. When referencing variables or data types, they will be displayed in *italic* and colored. Data types will appear in *red*, variables in *violet*.

The file format described in this documentation has been designed to be extendable. In addition, it should allow a good upward- and downward-compatibility to other file formats used by TommySoftware® products. In order to maintain this high compatibility, it is important that you do not make any presumptions that are not stated explicitly! If something has to be in a specific order, it will be stated. If not, the order is free! The same applies to the existence of all other file elements. They are only required if it is said so.

If, e.g., a data block is described with currently 4 elements, it may have 6 or even 100 elements in the next version - but the first 4 elements will have the same meaning. In general: any element number stated may grow in future versions. But it will never get smaller. The same applies to data block sequences. If an object currently starts with the data block sequence B001, B002, B102, B225 in this order, it will always start with that data block sequence. But in future versions, other data blocks may follow.

And finally, it applies to the file sections. Even though only a limited number of sections is described in this documentation, future versions may feature several new sections. Those new sections should be ignored by programs that do not know them. The only legal assumption about sections is that the first section of each TVG file will be an information section (like `=DRAWING=` or `=LIBRARY=`), showing which type of data the file contains, and the last section will be `=EXIT=`. But again - even the number of file types may grow, i.e. future versions may have additional information sections. In this case, ignore the complete file, as its content may be completely unknown.

## About Compressed Drawings

Some applications using the TVG 4.2 file format offer the option to save "Compressed TVG 4.2 Drawings". Such files are industry-standard ZIP archives that contain a single compressed TVG 4.2 drawing. The drawing is compressed using the TSZIP utility delivered with the application, and can be extracted manually by means of the TSUNZIP utility. Compressed drawings can directly be opened in the application.

If you want to edit such a file manually, or if you are using an older release of the application which is not able to load compressed drawings directly, you will have to extract it using either TSUNZIP (if available) or using any standard ZIP tool. Please note that compressed drawings may be scrambled using a password! You will have to know the password to be able to access the drawing data.

## Technical Support and Information

The documentation of the TVG 4.2 file format and of the *Toso Interface* are supplied "as is", and they are both *only* available in electronic form and *only* in English. Developer support, however, is available in German as well as in English, see [TOSOAPI4.HLP - Getting Your Private Owner ID.](#)

For technical support please send an e-mail to "support@tommysoftware.com". If you have any other questions please send your e-mail to "sales@tommysoftware.com".



On our World Wide Web site (<http://www.tommysoftware.com>) you can find the latest versions of the TVG 4.2 file format documentation and of the *Toso Interface* documentation as well as the latest program versions and upgrades, a collection of import/export filters and converters, the CAD/DRAW Tutorial, the CAD/DRAW Tour, additional documentation, utilities, and current information on our products. And please also try out our special Web offers. Don't miss this opportunity to make the most of the Internet!

**TOMMY SOFTWARE®**

**North America, Inc.**

1843 10th Avenue  
San Francisco, CA 94122  
U. S. A.

Phone (415) 566 6118  
Fax (415) 566 6589

**Internet**

sales@tommysoftware.com (Sales)  
support@tommysoftware.com (Technical Support)  
<http://www.tommysoftware.com> (World Wide Web)

**Germany**

Selchower Straße 32  
D-12049 Berlin  
Germany

Phone +49 30 621 5931  
Fax +49 30 621 4064

## Geometrical Representation (General)

A TVG file usually contains a number of objects. These objects are entities that contain the data of a geometrical representation of a drawing element like a "line" or a "circle". The geometrical data consists of a number of point coordinates and some additional data. A "line" e.g. is made up of two point coordinates ("start-point" and "end-point") plus some additional data like line width, line color, layer number etc.

All point coordinates are defined in a cartesian coordinate system, having its origin in the center of the page. Its axes increment to the left and upward. The coordinates are always stored in *physical* millimeters, i.e. they determine the *printed* size of the object. A square having a *physical* side length of 10 mm will always be printed, plotted or exported with a side length of these 10 mm (exception: explicit scaling of the print-out).

If the user creates a square with a *logical* side length of 2 meters using a scale of 1:100, the square will have a *physical* side length of 20 mm, as it will be printed with that side length of 20 mm.

Each point in the coordinate system is determined by a X- and Y-coordinate, e.g. (0.0,1.0). The valid *physical* coordinate area is -1e100 mm to 1e100 mm. No coordinate may ever have the value 1e300 mm, as this value is defined as "invalid / undefined value" and may cause undefined behavior or even a system crash.

Angles are always stored in radiant [rad]. Typical values are -pi up to pi, values from -2pi up to 4pi are allowed.

The maximum page size is  $4000 \times 4000$  mm or approx.  $157 \times 157$  inch. The user may work on an area of up to  $1e100 \times 1e100$  mm, having its origin in the center of the page. All definition points of the objects have to lie completely inside this area.

## Matrices and Vectors (General)

Most objects are defined by a number of definition points lying in a cartesian coordinate system. Each definition point consists of two coordinates. If a calculation is made with such a definition point, it should be interpreted as a position vector relative to the internal origin.

If a calculation is made with the point P, the calculation is meant to be done with the vector [ x(P), y(P) ], where x(P) is the X-coordinate of P and y(P) the Y-coordinate of P. A calculation instruction like:

$$P = P1 + 0.5 \times P2$$

has to be read like:

$$\begin{aligned}x(P) &= x(P1) + 0.5 \times x(P2) \\y(P) &= y(P1) + 0.5 \times y(P2)\end{aligned}$$

Some entities contain matrices that represent transformations to be performed before displaying the entity. This is especially required when referencing external entities like blocks or characters. For such entities, the display matrix determines the position, size, rotation and shearing of the referenced entity.

Conjugated matrices, i.e. matrices reflected at their diagonal, will be marked by a 'T' at their upper right edge.

## Operations

If a referenced entity is to be displayed, it has to be multiplied with the display matrix stored in the referencing entity. Usually, this can be done by simply multiplying all of the entity's definition points with this display matrix. Such a multiplication is handled like follows:

$$\begin{bmatrix} x \\ y \\ 1.0 \end{bmatrix}^T \times \begin{bmatrix} m11 & m12 & 0.0 \\ m21 & m22 & 0.0 \\ m31 & m32 & 1.0 \end{bmatrix} = \begin{bmatrix} x \times m11 + y \times m21 + m31 \\ x \times m12 + y \times m22 + m32 \\ 1.0 \end{bmatrix}^T$$

In complex entities, there might be the need to perform additional adaption of object-specific values like the arc direction of Object 06 "Circular Arc".

The great advantage of matrices is the fact that all types of transformations can be handled equally as they can all be stored in the same type of matrix. So, any combination of transformation can be performed by a single matrix multiplication.

For the most important transformation types, their corresponding matrices will be listed below. If multiple transformations shall be combined, simply multiply the corresponding matrices and apply the resulting matrix to the entities. The matrices' multiplication will be easier if, instead of always multiplying two matrices, the new transformation is applied to an existing matrix. To do so, always start with the identity matrix (ident[]):

$$\text{ident[]} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The following descriptions of transformations do always show the matrices OLD and NEW from the equation  $\text{NEW} = \text{OLD} \times \text{transformation-matrix}$ . The transformation matrix itself is not stated as it is not relevant.

## Moving

A movement of vx and vy (move[vx,vy]) leads to the following result:

```
[ m11 m12 0.0 ]   move[vx,vy]   [  m11    m12  0.0 ]
[ m21 m22 0.0 ] -----> [  m21    m22  0.0 ]
[ m31 m32 1.0 ]           [ m31+vx m32+vy 1.0 ]
```

## Scaling

A scaling of factor sx and sy (scale[sx,sy]) leads to the following result:

```
[ m11 m12 0.0 ]   scale[sx,sy]   [ m11*sx m12*sy 0.0 ]
[ m21 m22 0.0 ] -----> [ m21*sx m22*sy 0.0 ]
[ m31 m32 1.0 ]           [ m31*sx m32*sy 1.0 ]
```

## Rotating

A rotation of angle  $\beta$  (rot[ $\beta$ ]) leads to the following result:

```
[ m11 m12 0.0 ]   rot[ $\beta$ ]   [ m11*cos( $\beta$ )-m12*sin( $\beta$ ) m11*sin( $\beta$ )+m12*cos( $\beta$ ) 0.0 ]
[ m21 m22 0.0 ] -----> [ m21*cos( $\beta$ )-m22*sin( $\beta$ ) m21*sin( $\beta$ )+m22*cos( $\beta$ ) 0.0 ]
[ m31 m32 1.0 ]           [ m31*cos( $\beta$ )-m32*sin( $\beta$ ) m31*sin( $\beta$ )+m32*cos( $\beta$ ) 1.0 ]
```

## Shearing

A horizontal shearing with slope s (hshear[s]) leads to the following result:

```
[ m11 m12 0.0 ]   hshear[s]   [ m11 m12+m11*s 0.0 ]
[ m21 m22 0.0 ] -----> [ m21 m22+m21*s 0.0 ]
[ m31 m32 1.0 ]           [ m31 m32+m31*s 1.0 ]
```

A vertical shearing with slope s (vshear[s]) leads to the following result:

```
[ m11 m12 0.0 ]   vshear[s]   [ m11+m12*s m12 0.0 ]
[ m21 m22 0.0 ] -----> [ m21+m22*s m22 0.0 ]
[ m31 m32 1.0 ]           [ m31+m32*s m32 1.0 ]
```

## Reflecting

A reflection along the X-axis (xref[]) leads to the following result:

```
[ m11 m12 0.0 ]   xref[]   [ m11 -m12 0.0 ]
[ m21 m22 0.0 ] -----> [ m21 -m22 0.0 ]
[ m31 m32 1.0 ]           [ m31 -m32 1.0 ]
```

A reflection along the Y-axis (yref[]) leads to the following result:

```
[ m11 m12 0.0 ]   yref[]   [ -m11 m12 0.0 ]
[ m21 m22 0.0 ] -----> [ -m21 m22 0.0 ]
[ m31 m32 1.0 ]           [ -m31 m32 1.0 ]
```

Reflections can also be implemented as a scaling of (-1.0,1.0) or (1.0,-1.0) respectively:

```
xref[] = scale[ 1.0,-1.0 ]
```

```
yref[] = scale[ -1.0,1.0 ]
```

## Inverting

In order to invert the effect of the transformations stored in a matrix, this matrix can be inverted directly. Due to the simplified representation of the 3×3 matrix, this inversion (inv[]) can be done as follows:

```
det = m11×m22 - m12×m21
```

```
[ m11 m12 0.0 ]   inv[]   [           m22/det           -m12/det           0.0 ]
[ m21 m22 0.0 ] -----> [           -m21/det           m11/det           0.0 ]
[ m31 m32 1.0 ]           [ (m21×m32-m22×m31)/det (m12×m31-m11×m32)/det 1.0 ]
```

This inversion is obviously only possible if `det` is nonzero.

## Extraction

If a matrix shall be resolved into its basic transformational elements, this can be done by means of the following calculation steps:

```
len1 = sqrt( m11×m11+m12×m12 )
len2 = sqrt( m21×m21+m22×m22 )
det   = m11×m22-m12×m21
```

```
          ident [] ×
[ m11 m12 0.0 ]   scale [ len1, abs(det/len2) ] ×
[ m21 m22 0.0 ] = hshear[ tan(atan2(m22,m21) - atan2(m12,m11)) ] ×
[ m31 m32 1.0 ]   rot   [ atan2(m12,m11) ] ×
                  move  [ m31,m32 ]
```

This extraction is obviously only possible if `det` is nonzero.



## Constants (Data Representation)

Numeric and other values will be shown in one of the following ways in this documentation:

86	An integer value in decimal notation.
0x56	The same integer value in hexadecimal notation.
105.6	A floating point value in its normal notation.
1.056e2	The same floating point value in exponential notation.
'A'	A single character. Characters may also be shown as a numeric value stating the index of the character in the ANSI character set. The character 'A', e.g., has the index 65 or 0x41.
"Abcd"	A text.

Inside TVG 4.2 files, integer values are *always* used in decimal notation. Floating point values may either be used in normal or in exponential notation.

## Units (Data Representation)

Most values in the TVG file are depending on a unit. This unit can either be a "length unit" (when used for scale-dependent values like coordinates), a "line unit" (when used for scale-independent values like line width or text size) or a "angle unit" (when used for angular values). Even though most of the values are *stored* unit-independent (i.e. always in millimeters based on 1:1-scale), they have to be *interpreted* unit-dependent.

The following units are available:

### Length Units

Micrometer	[µm]	0.001 mm	1/1000000 m
Millimeter	[mm]	1.0 mm	1/1000 m
Centimeter	[cm]	10.0 mm	1/100 m
Decimeter	[dm]	100.0 mm	1/10 m
Meter	[m]	1000.0 mm	1 m
Kilometer	[km]	1000000.0 mm	1000 m
Mil	[mil]	0.0254 mm	1/1000 Inch
Inch	[inch]	25.4 mm	1 Inch
Foot	[foot]	304.8 mm	12 Inch
Yard	[yard]	914.4 mm	3 Foot
Mile	[mile]	1609344 mm	1760 Yard
Decipoint	[dp]	0.0352777777777777 mm	1/720 Inch
Point	[pt]	0.3527777777777777 mm	1/72 Inch
Didot Point	[bp]	0.3759398496241 mm	1/2660 m
Cicero	[cic]	4.5112781954892 mm	12/2660 m

### Line Units

Micrometer	[µm]	0.001 mm	1/1000000 m
Millimeter	[mm]	1.0 mm	1/1000 mm
Centimeter	[cm]	10.0 mm	1/100 mm
Mil	[mil]	0.0254 mm	1/1000 Inch
Inch	[inch]	25.4 mm	1 Inch
Decipoint	[dp]	0.0352777777777777 mm	1/720 Inch
Point	[pt]	0.3527777777777777 mm	1/72 Inch
Didot Point	[bp]	0.3759398496241 mm	1/2660 m
Cicero	[cic]	4.5112781954892 mm	12/2660 m

### Angle Units

Degree	[deg]	360.0 = 1 Rotation	
New Degree/Gon	[gra]	400.0 = 1 Rotation	
Radian	[rad]	2pi = 6.283185307179586 = 1 Rotation	
Relative	[rel]	1.0 = 1 Rotation	

## Data Types Overview (Data Representation)

TVG 4.2 uses four standard data types similar to data types used in ANSI C - *char*, *short*, *long* and *double*. All further data types are based on these four standard data types. To indicate the logical relation between the data types defined in TVG 4.2 and similar data types in the *Toso Interface* (see [TOSOAPI4.HLP](#)), they both use the same name even though they are defined in a different manor (textual representation vs. binary representation).

All data types are displayed in their "natural" form, i.e. numbers in decimal or exponential notation, and texts as a character string delimited by the character " (Ansi 34). Single values are separated by the character , (Ansi 44), data blocks, entities and sections are terminated by the character ; (Ansi 59).

Comments may occur between values and their separators. A comment is a character string delimited by the character | (Ansi 124). Comments may not be placed *inside* a numeric value or inside a text (of course the character | may occur inside a text, but it will not be interpreted as a comment delimiter in this case). Control characters like line-feed, carriage-return, tab, space and all other characters between Ansi 0 and Ansi 32 are ignored if they do not occur inside a text.

In order to keep the resulting file as small as possible, numeric values may be "empty", i.e. there is no character between the separators. In this case, the value is defined to be zero. Example:

```
,,8|Data Type Text|,250|Max. 250 Bytes|,"Hello";
```

This line contains 5 values and some comments. The data line uses "empty" values and is equal to the following line:

```
0,0,8 |Data Type Text|,250 |Max. 250 Bytes|,"Hello";
```

As comments and spaces are ignored when reading the file, the data line looks to the reading application as follows:

```
0,0,8,250,"Hello";
```

When showing element sequences in the descriptions, some values will be underlined. Those "values" are only names of placeholders that have to be replaced with an explicit value in the TVG file. For example:

```
BlockOwner,0,3,1,Coordinates;
```

In this example, the values *BlockOwner* and *Coordinates* are placeholders for explicit values. A detailed description on those values will follow in the text.

Line-feeds, comments and spaces shown in examples are only a *suggestion*, they are not required. Usually, lines in TVG files should not be longer than 80 to 100 characters, as some text editors have limitations in line length.

## Standard Data Types

Standard data types are native ANSI C data types, written to the file using standard *printf()* format strings. An example of a suitable format string is shown in the description of each data type.

char

short

long

double

## **Extended Data Types**

Extended data types are based on the four standard data types listed above. The order of the elements in such a data type is equal to the order in the file. Usually, the single values are separated by commas.

MATRIX (Matrix)

DPOINT (Point Coordinates)

DRECT (Rectangle)

COLORREF (Color Description)

PROPERTY (Properties)

XPROPERTY (Extended Properties)

FONTDEF (Font Description)

TEXT (ANSI Text)

BINARY (Binary Data)

## **char** (Standard Data Types)

A *char* is an unsigned integer value. Its *allowed* value range is 0 to 255 (8-bit value). A *char* is directly written to the file, the according format string is "%c" .

Inside a TVG file, single characters (or non-delimited character strings respectively) do only occur in the identification at the beginning of the file. Otherwise, characters occur inside delimited character strings (see TEXT (ANSI Text)). Characters use the complete ANSI character set. Characters from Ansi 0 to Ansi 31 are usually ignored or replaced by spaces.

## **short** (Standard Data Types)

A *short* is a signed integer value. Its *allowed* value range is -32766 to 32767 (16-bit value). A *short* is written to the file in decimal notation, the according format string is `"%d"` or `"%hd"` respectively (depending on the operating system). Be sure not to produce any useless spaces or leading zeros. Keep the file compact!

## **long** (Standard Data Types)

A *long* is a signed integer value. Its *allowed* value range is -2147483646 to 2147483647 (32-bit value). A *long* is written to the file in decimal notation, the according format string is `"%d"` or `"%ld"` respectively (depending on the operating system). Be sure not to produce any useless spaces or leading zeros. Keep the file compact!

## **double** (Standard Data Types)

A *double* is a signed floating point value. Its *allowed* value range is  $-1e100$  to  $1e100$  (64-bit value). A *double* is written to the file either in normal or in exponential notation, the according format string is `"%.13lg"`. This ensures the output of at least 13 valid fractional digits and uses the shortest of the two possible notations (normal or exponential). Be sure not to produce any useless spaces or leading zeros. Keep the file compact!



## MATRIX (Matrix) (Extended Data Types)

The data type *MATRIX* is used to store a  $3 \times 3$  matrix. Such matrices are used to store a combination of one or more of the following operations: translation (moving), scaling, rotation, shearing and reflection. To store all of these operations, a  $3 \times 3$  matrix is required.  $3 \times 3$  matrices use the following representation:

```
[ m11 m12 m13 ]  
[ m21 m22 m23 ]  
[ m31 m32 m33 ]
```

As the application only allows two-dimensional data and therefore only two-dimensional operations, all resulting matrices have the following, simplified form:

```
[ m11 m12 0.0 ]  
[ m21 m22 0.0 ]  
[ m31 m32 1.0 ]
```

Due to this simplification, only the first two columns of the matrix are stored. The third column is set to  $[0.0 \ 0.0 \ 1.0]_T$  implicitly.

### ▪ Element Sequence

[m11, m12, m21, m22, m31, m32](#)

## Contents **Element Description**

*m11, m12*

[*double*] First line of a  $3 \times 2$  matrix.

*m21, m22*

[*double*] Second line of a  $3 \times 2$  matrix.

*m31, m32*

[*double*] Third line of a  $3 \times 2$  matrix.

## Contents **Example**

[1.0, 0.0, 0.0, 1.0, 0.0, 0.0](#)

These six values describe the first two columns of the  $3 \times 3$  identity matrix (see [Matrices and Vectors](#)). The third column is to be set to  $[0.0 \ 0.0 \ 1.0]_T$ , so the resulting matrix is:

```
[ 1.0 0.0 0.0 ]  
[ 0.0 1.0 0.0 ]  
[ 0.0 0.0 1.0 ]
```

## DPOINT (Point Coordinates) (Extended Data Types)

The data type *DPOINT* is used to store the coordinates of a point. Each *DPOINT* consists of a X- and a Y-coordinate.

 Contents **Element Sequence**  
 $x, y$


 Contents **Element Description**

$x, y$  [*double*] X- and Y-Coordinate of the point. The coordinates are in [mm] relative to the page center.

## DRECT (Rectangle) (Extended Data Types)

The data type *DRECT* is used to store the extents of a rectangular frame. Such frames are usually used to store the surrounding frames of blocks or characters.

 **Contents** Element Sequence  
 $x_1$ ,  $y_1$ ,  $x_2$ ,  $y_2$

 **Contents** Element Description

$x_1, y_1$

[*double*] X- and Y-Coordinate of the first corner-point of the frame. The coordinates are in [mm] relative to the page center.

$x_2, y_2$

[*double*] X- and Y-Coordinate of the second corner-point (on the same diagonal as  $x_1, y_1$ ) of the frame. The coordinates are in [mm] relative to the page center.

In most cases, the values in this data type have to be sorted, i.e.  $x_1 \leq x_2$  and  $y_1 \leq y_2$ . This speeds up further calculations with these frames.

## COLORREF (Color Description) (Extended Data Types)

The data type *COLORREF* is used to store a color definition. Color definitions are based on the RGB color model, i.e. they consist of three components (Red, Green, Blue). These three components are separated by the character / (Ansi 47). Each component may have values between 0.0 and 1.0 (including).

 **Contents** **Element Sequence**  
RValue/GValue/BValue

 **Contents** **Element Description**

*RValue* [double] Red component of the color.

*GValue* [double] Green component of the color.

*BValue* [double] Blue component of the color.

 **Contents** **Examples**

Black	0/0/0	White	1/1/1
Gray 90%	0.1/0.1/0.1	Gray 80%	0.2/0.2/0.2
Gray 70%	0.3/0.3/0.3	Gray 60%	0.4/0.4/0.4
Gray 50%	0.5/0.5/0.5	Gray 40%	0.6/0.6/0.6
Gray 30%	0.7/0.7/0.7	Gray 20%	0.8/0.8/0.8
Gray 10%	0.9/0.9/0.9		
Blue	0/0/1	Dark Blue	0/0/0.5
Green	0/1/0	Dark Green	0/0.5/0
Cyan	0/1/1	Dark Cyan	0/0.5/0.5
Red	1/0/0	Dark Red	0.5/0/0
Magenta	1/0/1	Dark Magenta	0.5/0/0.5
Yellow	1/1/0	Dark Yellow	0.5/0.5/0

Please note that values of type *double* may be empty, i.e. the value // is a correct value, it is equal to 0/0/0 and represents the color black! All fractional values will usually be stated without a leading zero, i.e. normally, the color "Gray 50%" will be stored as .5/.5/.5 instead of 0.5/0.5/0.5.

Depending on the accuracy of the creating program, the component values of the examples above might not be exactly reproduced. If each component is stored in a 8 bit field, the value .5 might be exported as .498 or .502. If no exact color match is needed, the program may round these values to 2 fractional digits.

When exporting a color definition, the format string "%.3lg/%.3lg/%.3lg" should be used to achieve an accuracy that will allow to handle 8 bit components without errors.

## PROPERTY (Properties) (Extended Data Types) Version 4.2

The data type *PROPERTY* is used to store the properties of an object. These properties consist of some non-geometrical informations that are needed to draw objects. They determine the line width, line color, line pattern etc.

### Contents **Element Sequence**

[FillMode](#), [FillColor](#), [LineColor](#), [LineWidth](#), [LineType](#), [LineCaps](#)

### Contents **Element Description**

#### *FillMode*

[*long*] The value *FillMode* determines, which parts of an object are to be drawn. Following values are defined:

- 0x0000 The outline of the object is drawn.
- 0x0001 The object is filled.
- 0x0002 The object is filled and its outline is drawn.
- 0x0003 The object is erased (i.e. filled in background color).
- 0x0004 The object is erased (i.e. filled in background color) and its outline is drawn.

Some objects do not have a surface (e.g. a line or a circular arc). If such an object is drawn using a *FillMode* of 0x0001 or 0x0003, it will be invisible.

#### *FillColor*

[*COLORREF*] Color of the object's surface in RGB notation.

#### *LineColor*

[*COLORREF*] Color of the object's outline in RGB notation.

#### *LineWidth*

[*double*] Width of the object's outline in [mm] between 0.0 and 100.0 (including). A width of 0.0 always results in a line of the minimum width possible on the respective device (one pixel).

#### *LineType*

[*int*] Index of the line pattern used to draw the outline (see [Section =LINE= \(Line patterns\)](#)).

#### *LineCaps* Version 4.2

[*int*] Line cap and line join mode. This value is a bitwise-or combination of two values. The first one determines the form of line end caps, and can be one of the following values:

- 0x0000 End caps are round.
- 0x0100 End caps are square.
- 0x0200 End caps are flat.

The second one determines the line join mode, and can be one of the following values:

- 0x0000 Joins are round.
- 0x1000 Joins are beveled.
- 0x2000 Joins are mitered when they are within the current limit. If it exceeds this limit, the join is beveled.

Both values are defined according to the Win32 definitions used in `ExtCreatePen()` calls.

### Contents **Example**

2,1/0/0,0/0/0,0.0,0

If, e.g., a circle is drawn with this properties, it will result in a red filled circle, outlined with a minimum-

width solid (Line pattern 0 is predefined by default as solid, see Section =LINE= (Line patterns).) black line.

## XPROPERTY (Extended Properties) (Extended Data Types) Version 4.2

The data type *XPROPERTY* is used to store the extended properties of an object.

As described in the chapter PROPERTY (Properties), the display of each object is determined by a set of five properties. This "direct" assignment of properties is only suitable for simple, independent objects.

If objects are combined to blocks (see Entity "Block") problems occur. Each separate object inside the block may have for example a different outline color. If such a block is referenced multiple times, all references will use the same outline color for all corresponding objects. Now imagine you want to set the outline color of *just one* referenced block to red, without modifying the other references.

The solution is provided by a transmission process. Each instance (i.e. reference) of a block contains its own extended set of properties, plus the information which of these properties shall be transmitted to the referenced block's objects. If the instance contains the information "*transmit the outline color red to all objects*", all objects of this instance will be drawn with red outlines - without any change in the block's definition. All other instances will still be drawn in their object's original outline colors.

Such a transmission is valid for *all* subordinate objects and instances, i.e. usually *all* objects of an instance will be modified by transmission. Now imagine a block consisting of several objects, some of which have to maintain a specific property, some not. For example, some objects have to be only outlined (as they are lines), some may either be outlined or filled, depending on the intended use of that block. To achieve this difference, objects themselves have to contain transmission information - saying that they do not want to receive a special transmitted property, i.e. that this special object property is fixed. In the preceding example, the transmission information for those objects that should always be outlined would be "*I do not accept the transmission of a filling mode because this property is fixed*".

Both types of transmission information are similar, as for each single property there is one flag saying either "transmit" or "do not accept transmission" (or in other words: "fix property"). As a result, the data type used to store the properties and the transmission information is equal for objects, instances, and blocks, only the interpretation is different.

In entities of type "Instance" or "Block" the transmission flags indicate whether to transmit a specific property or not. In entities of type "Object" they indicate whether to accept the transmission or not, i.e. whether to fix a specific property or not.

To store the properties and the transmission information the following structure is used. For further description of the transmission process itself, see the reference of the application.



### Contents **Element Sequence**

[Flag](#), [Pen](#), [FillMode](#), [FillColor](#), [LineColor](#), [LineWidth](#), [LineType](#), [LineCaps](#), [Layer](#)



### Contents **Element Description**

*Flag*

[*long*] In entities of type "Instance" or "Block", the transmission flags indicate whether to transmit a specific property or not. This is stored in the value *Flag*, which is a bitwise OR combination of one or more of the following values:

```

#define USE_NULL          0x0000
#define USE_PEN           0x0001
#define USE_FILLMODE     0x0002
#define USE_FILLCOLOR    0x0004
#define USE_LINECOLOR    0x0008
#define USE_LINEWIDTH    0x0010
#define USE_LINETYPE     0x0020
#define USE_LINECAPS     0x0080
#define USE_LAYER        0x0040

```

The value `USE_PEN` has the lowest priority, the value `USE_LAYER` has the highest priority. Assuming that both a pen index (and consequently the pen's line width) and an explicit line width are transmitted, the explicit line width (transmitted by setting the `USE_LINEWIDTH` flag) has a higher priority than the transmitted pen's line width (transmitted by setting the `USE_PEN` flag). In entities of type "Object", the transmission flags indicate whether to accept the transmission or not, i.e. whether to fix specific properties or not. This is stored in the value *Flag*, which is a bitwise OR combination of one or more of the same values as stated above. If one of these flags is set a transmission of the corresponding property is not accepted, i.e. the property is fixed.

### Pen

[*long*] Index of the pen (see [Section =PEN= \(Pen\)](#)).

### FillMode

[*long*] The value *FillMode* determines, which parts of an object are to be drawn. Following values are defined:

0x0000 The outline of the object is drawn.  
 0x0001 The object is filled.  
 0x0002 The object is filled and its outline is drawn.  
 0x0003 The object is erased (i.e. filled in background color).  
 0x0004 The object is erased (i.e. filled in background color) and its outline is drawn.

Some objects do not have a surface (e.g. a line or a circular arc). If such an object is drawn using a *FillMode* of 0x0001 or 0x0003, it will be invisible.

### FillColor

[*COLORREF*] Color of the object's surface in RGB notation.

### LineColor

[*COLORREF*] Color of the object's outline in RGB notation.

### LineWidth

[*double*] Width of the object's outline in [mm] between 0.0 and 100.0 (including). A width of 0.0 always results in a line of the minimum width possible on the respective device (one pixel).

### LineStyle

[*int*] Index of the line pattern used to draw the outline (see [Section =LINE= \(Line patterns\)](#)).

### LineCaps **Version 4.2**

[*int*] Line cap and line join mode. This value is a bitwise-or combination of two values. The first one determines the form of line end caps, and can be one of the following values:

0x0000 End caps are round.  
 0x0100 End caps are square.  
 0x0200 End caps are flat.

The second one determines the line join mode, and can be one of the following values:

0x0000 Joins are round.  
 0x1000 Joins are beveled.  
 0x2000 Joins are mitered when they are within the current limit. If it exceeds this limit, the join is beveled.

Both values are defined according to the Win32 definitions used in `ExtCreatePen()` calls.

### Layer



[*long*] Index of the layer (see [Section =LAYER= \(Layers\)](#)).

## Contents **Examples**

0,0,2,0/0/1,0/0/0,0.5,1,0,0

An object with this property set is assigned to layer 0 (layer 0 transmits no properties, see [Section =LAYER= \(Layers\)](#).) and it uses pen 0 (pen 0 transmits no properties, see [Section =PEN= \(Pen\)](#)). There are no fixed properties. Because layer and pen are equal to 0 the object is drawn with its own properties: a blue filling and a 0.5 mm wide black line using line pattern 1.

8,1,2,0/0/1,0/0/0,0.5,0,0,100

An object with this property set is assigned to layer 100 and it uses pen 1. Its line color is fixed. So how this object is drawn also depends on the transmitted properties of layer 100 and pen 1. However a fixed property is always drawn no matter whether the corresponding layer or pen property is transmitted. A transmitted layer property has a higher priority (see [Section =LAYER= \(Layers\)](#)) than a transmitted pen property. Because pen properties have the lowest priority (see [Section =PEN= \(Pen\)](#)) a pen property is drawn only if the corresponding layer and object property is not transmitted and not fixed respectively.

## FONTDEF (Font Description) (Extended Data Types)

The data type *FONTDEF* is used to store a font's description. In TVG files, three font types are supported: TrueType, device (esp. PostScript) and internal fonts.

### Contents **Element Sequence**

Type, Style, Weight, Name

### Contents **Element Description**

#### Type

[*long*] Type of the font. This value determines, if the specified font is an internal font, a TrueType font or a device font (esp. PostScript). Possible values are:

0x0000 Internal font (e.g. "DINDRAFT").  
0x0001 TrueType font (e.g. "Times New Roman")  
0x0002 PostScript or device font (e.g. "Palatino")

#### Style

[*long*] Style of the font. This value is a bitwise OR combination of several of the following styles:

0x0000 No special style.  
0x0001 Italic. If this bit is set the font will be displayed in italic (if possible).  
0x0002 Underline. If this bit is set the font will be displayed underlined (if possible).  
0x0004 Strikeout. If this bit is set the font will be displayed striked out (if possible). This bit is ignored at the moment!  
0x0100 Symbol font. This bit has to be set if the font is a symbol font (e.g. "Symbol" or "Wingdings").

For internal fonts, this value should be set to 0x0000, as it has no effect in this case.

#### Weight

[*long*] Weight of the font. The weight of the font is defined analogous to the weight definition of TrueType fonts. Possible values are:

0 Undefined  
100 Thin  
200 Extra Light  
300 Light  
400 Regular  
500 Medium  
600 Semibold  
700 Bold  
800 Extrabold  
900 Black

For internal fonts, this value should be set to 400 (Regular), as it has no effect in this case.

#### Name

[*TEXT64*] Name of the font, up to 63 characters. Names of TrueType and PostScript fonts may be up to 31 characters long.

### Contents **Examples**

1, 1, 600, "Arial"

This structure describes the font "Arial Bold Italic", it is a TrueType font. If this font is not available, the system will try to find a similar font instead.

0, 0, 400, "DINDRAFT"

This structure describes the font "DINDRAFT", it is an internal font. The values of *Style* and *Weight* are set to their default values 0 and 400, as they have no effect.

## **TEXT (ANSI Text) (Extended Data Types)**

The data type *TEXT* is used to store texts of variable length. Inside this documentation, a numeric value will usually be appended to the data types' name, indicating the maximum length allowed, e.g. *TEXT64* for a text that may be up to 64 characters long including the terminating null character (0x00). Typical lengths are 32, 64 und 8000 characters, the maximum length of a text is 32,000 characters.

Texts in TVG 4.2 files do always use the 8-bit ANSI character set. Usually, texts are represented as a character string delimited by the character " (Ansi 34). To be able to use the character "" inside of texts, the standard C text encoding is used: The character " is to be replaced by \" (Ansi 92 34), a single \ (Ansi 92) by \\ (Ansi 92 92). This has to be decoded when reading texts from TVG 4.2 files! The according format string is "%s", where the characters " and \ have to be encoded before.

## **BINARY (Binary Data) (Extended Data Types) Version 4.2**

The data type *BINARY* is used to store binary data of variable length. Inside this documentation, a numeric value will usually be appended to the data types' name, indicating the maximum length allowed, e.g. *BINARY16* for binary data that may be up to 16 bytes long. The maximum length of binary data is 24,000 bytes.

Binary area uses a simple character encoding. The resulting strings are handled like *TEXT* data, i.e. the character sequence is delimited by the character " (Ansi 34).

Each three bytes are encoded in four characters, using the characters " " (Ansi 32) to "\_ " (Ansi 95), where the characters 'A' (Ansi 65) to 'Z' (Ansi 90) are case-independent, i.e they may be replaced by 'a' (Ansi 97) to 'z' (Ansi 122). Due to a better readability, the lower-case characters should be preferred. Together, these four characters encode 24 bits, making 3 bytes. The first character encodes the highest bits, the fourth character the lowest bits. The string "adm+" decodes to the numeric value 5655391, resulting in the bytes sequence 56 4b 5f . If the binary string does not include a number of bytes dividable by three, the encoding assumes the missing one or two bytes to be zero, i.e. a single last byte 5f will be encoded equal to 5f 00 00 , resulting in the string "7p " (two spaces at the end). Please note that the strings used to encode binary data may contain the characters "" (Ansi 34) and \" (Ansi 92). As a result, those strings must be encoded like standard text strings (see TEXT (ANSI Text)).

# Entities and Data Blocks Overview (Data Representation)

An entity consists of a header, determining the entity's type and containing some entity-specific data, and a sequence of data blocks. Each data block itself consists of a header and a variable data section.

For a description of the different entity header types, see [Entity "Object"](#) and the following chapters.

The entity header is followed by data blocks (not to be confused with the entity type "Block" !!!). Data blocks are the smallest information elements inside entities. Data blocks can either be [Generic Data Blocks](#) that contain simple data types and that will be used in all types of entities or [Standard Data Blocks](#) that contain complex data structures and that will be used only in a few special entities.

Each data block starts with a sequence of three values: The data block owner identification, the data block identification number and the data block type. The values inside a data block are separated by a comma, the data block is ended with a semicolon.

The number of values inside a data block is variable. Even if a special data block is currently defined with a fixed number of values, it may grow in future. If so, new values will be appended to the data block, i.e. the currently defined values will remain at the same location. As a result, do always read all defined values of each data block, and then skip until the next semicolon to overread unknown values.

Each entity may contain a variable number of data blocks of different type and length. This data block sequence is terminated by a data block of type 999. Data blocks of the same type do not always have the same length. Especially text and binary data varies in length!

If the description of an entity type states a specific data block sequence, this sequence must be maintained in any case. A general rule is: First all generic and standard data blocks (type 0 to 199), then all native data block (type 200 to 299), followed by local and global attributes (type 300 to 499), and terminated by a data block of type 999. If this sequence is not maintained, this can lead to an application crash, as this sequence is not explicitly tested during file load.

Data blocks of all types use the same element sequence that determines the owner of the data block, the data block type and the type and number of data values stored in that data block.

## Contents **Element Sequence**

[BlockOwner](#), [BlockType](#), [ElemType](#), [ElemCount](#), [Data\(1\)](#), . . . , [Data\(ElemCount\)](#);

## Contents **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *BlockType*

[*short*] This value is an internal identification of the data block. For data blocks created by TommySoftware® (where *BlockOwner* is 0), the following values are currently defined:

000      see [Data Block Type 000 \(General Point\)](#)

001	see <a href="#">Data Block Type 001 (Start-Point)</a>
002	see <a href="#">Data Block Type 002 (End-Point)</a>
003	see <a href="#">Data Block Type 003 (Center-Point)</a>
004	see <a href="#">Data Block Type 004 (Radius Definition Point)</a>
005	see <a href="#">Data Block Type 005 (Angle Definition Point)</a>
006	see <a href="#">Data Block Type 006 (Vector Definition Point)</a>
007	see <a href="#">Data Block Type 007 (First Pivot Point)</a>
008	see <a href="#">Data Block Type 008 (Second Pivot Point)</a>
009	see <a href="#">Data Block Type 009 (Arc End-Point)</a>
010	see <a href="#">Data Block Type 010 (Marking)</a>
100	see <a href="#">Data Block Type 100 (Constant)</a>
101	see <a href="#">Data Block Type 101 (Arc)</a>
102	see <a href="#">Data Block Type 102 (Curve)</a>
110	see <a href="#">Data Block Type 110 (Text)</a>
220	see <a href="#">Data Block Type 220 (Dimension Line)</a>
225	see <a href="#">Data Block Type 225 (Large Dimension)</a>
230	see <a href="#">Data Block Type 230 (Small Dimension)</a>
235	see <a href="#">Data Block Type 235 (Standard Text)</a>
236	see <a href="#">Data Block Type 236 (Frame Text)</a>
237	see <a href="#">Data Block Type 237 (Reference Text)</a>
242	see <a href="#">Data Block Type 242 (Clipping Surface)</a>
243	see <a href="#">Data Block Type 243 (Bitmap Reference)</a>
300 - 499	see <a href="#">Attribute Data Blocks</a>
999	see <a href="#">Data Block Type 999 (End of List)</a>

For data blocks created by third-party plug-in creators, this value is defined by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined data blocks used.

Non-standard data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

### *ElemType*

[*short*] Type of the data elements that are stored in this data block. The following data types are currently defined:

0x0000	Native data block containing a predefined structure, see <a href="#">Native Data Blocks</a>
0x0001	<i>long</i> , see <a href="#">Base Type "long"</a>
0x0002	<i>double</i> , see <a href="#">Base Type "double"</a>
0x0003	<i>DPOINT</i> , see <a href="#">Base Type "DPOINT"</a>
0x0004	<i>COLORREF</i> , see <a href="#">Base Type "COLORREF"</a>
0x0005	<i>PROPERTY</i> , see <a href="#">Base Type "PROPERTY"</a>
0x0006	<i>XPROPERTY</i> , see <a href="#">Base Type "XPROPERTY"</a>
0x0007	<i>FONTDEF</i> , see <a href="#">Base Type "FONTDEF"</a>
0x0008	<i>TEXT</i> , see <a href="#">Base Type "TEXT"</a>
0x0009	<i>BINARY</i> , see <a href="#">Base Type "BINARY"</a>

### *ElemCount*

[*short*] Number of values that are stored in this data block.

Inside native data blocks (where *ElemType* is 0), this value is usually ignored and set to 0. The only exception are [Attribute Data Blocks](#) which store the maximum attribute length here.

### *Data(x)*

[*???*] List of values, separated by commas, ended by a semicolon.

## **Data Blocks**

Generic Data Blocks

Native Data Blocks

Attribute Data Blocks

Standard Data Blocks

## **Entities**

Entity "Object"

Entity "Instance"

Entity "Block"

Entity "Group"

Entity "Position Number"

Entity "Custom-Defined"

Entity "End of List"



## Generic Data Blocks Overview (Entities and Data Blocks)

Generic data blocks are simple data blocks that are used to store values of a single, explicitly stated data type and can be handled without knowledge of their purpose. These data blocks can be used by third-party plug-ins to store all types of data.

Base Type "long"

Base Type "double"

Base Type "DPOINT"

Base Type "COLORREF"

Base Type "PROPERTY"

Base Type "XPROPERTY"

Base Type "FONTDEF"

Base Type "TEXT"

Base Type "BINARY"

## Base Type "long" (Generic Data Blocks)

This data block is used to store a given number of *long* values.

### Contents **Element Sequence**

[BlockOwner](#), [BlockType](#), [1](#), [ElemCount](#), [Data\(1\)](#), . . . , [Data\(ElemCount\)](#);

### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *BlockType*

[*short*] This value is an internal identification of the data block. This value is used by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined objects used.

Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

#### *ElemCount*

[*short*] Number of values that are stored in this data block. The value must be between 1 and 16000 (inclusive).

#### *Data(x)*

[*long*] List of values, separated by commas, ended by a semicolon.

## Base Type "double" (Generic Data Blocks)

This data block is used to store a given number of *double* values.



## Contents

### Element Sequence

[BlockOwner](#), [BlockType](#), [2](#), [ElemCount](#), [Data\(1\)](#), ..., [Data\(ElemCount\)](#);



## Contents

### Element Description

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *BlockType*

[*short*] This value is an internal identification of the data block. This value is used by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined objects used.

Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

#### *ElemCount*

[*short*] Number of values that are stored in this data block. The value must be between 1 and 8000 (inclusive).

#### *Data(x)*

[*double*] List of values, separated by commas, ended by a semicolon.

## Base Type "DPOINT" (Generic Data Blocks)

This data block is used to store a given number of *DPOINT* values.

### Contents **Element Sequence**

[BlockOwner](#), [BlockType](#), 3, [ElemCount](#), [Data\(1\)](#), . . . , [Data\(ElemCount\)](#);

### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *BlockType*

[*short*] This value is an internal identification of the data block. This value is used by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined objects used.

Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

#### *ElemCount*

[*short*] Number of values that are stored in this data block. The value must be between 1 and 4000 (inclusive).

#### *Data(x)*

[[DPOINT](#)] List of values, separated by commas, ended by a semicolon.

## Base Type "COLORREF" (Generic Data Blocks)

This data block is used to store a given number of *COLORREF* values.



### Contents

#### Element Sequence

[BlockOwner](#), [BlockType](#), 4, [ElemCount](#), [Data\(1\)](#), . . . , [Data\(ElemCount\)](#);



### Contents

#### Element Description

##### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

##### *BlockType*

[*short*] This value is an internal identification of the data block. This value is used by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined objects used.

Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

##### *ElemCount*

[*short*] Number of values that are stored in this data block. The value must be between 1 and 16000 (inclusive).

##### *Data(x)*

[*COLORREF*] List of values, separated by commas, ended by a semicolon.

## Base Type "PROPERTY" (Generic Data Blocks)

This data block is used to store a given number of *PROPERTY* values.



### Contents **Element Sequence**

[BlockOwner](#), [BlockType](#), [5](#), [ElemCount](#), [Data\(1\)](#), . . . , [Data\(ElemCount\)](#);



### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *BlockType*

[*short*] This value is an internal identification of the data block. This value is used by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined objects used.

Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

#### *ElemCount*

[*short*] Number of values that are stored in this data block. The value must be between 1 and 1000 (inclusive).

#### *Data(x)*

[[PROPERTY](#)] List of values, separated by commas, ended by a semicolon.

## Base Type "XPROPERTY" (Generic Data Blocks)

This data block is used to store a given number of *XPROPERTY* values.



### Contents

#### Element Sequence

[BlockOwner](#), [BlockType](#), [6](#), [ElemCount](#), [Data\(1\)](#), ..., [Data\(ElemCount\)](#);



### Contents

#### Element Description

##### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

##### *BlockType*

[*short*] This value is an internal identification of the data block. This value is used by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined objects used.

Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

##### *ElemCount*

[*short*] Number of values that are stored in this data block. The value must be between 1 and 1000 (inclusive).

##### *Data(x)*

[[XPROPERTY](#)] List of values, separated by commas, ended by a semicolon.

## Base Type "FONTDEF" (Generic Data Blocks)

This data block is used to store a given number of *FONTDEF* values.



## Contents

### Element Sequence

[BlockOwner](#), [BlockType](#), [7](#), [ElemCount](#), [Data\(1\)](#), ..., [Data\(ElemCount\)](#);



## Contents

### Element Description

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *BlockType*

[*short*] This value is an internal identification of the data block. This value is used by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined objects used.

Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

#### *ElemCount*

[*short*] Number of values that are stored in this data block. The value must be between 1 and 1000 (inclusive).

#### *Data(x)*

[*FONTDEF*] List of values, separated by commas, ended by a semicolon.



## Base Type "TEXT" (Generic Data Blocks)

This data block is used to store one value of type *TEXT*.



### Contents

**Element Sequence**  
`BlockOwner, BlockType, 8, ElemCount, Text;`



### Contents

**Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *BlockType*

[*short*] This value is an internal identification of the data block. This value is used by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined objects used.

Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

#### *ElemCount*

[*short*] Maximum text length to be stored in this data block. If the value is positive, only the number bytes actually needed to store the current text will be allocated. This saves memory, but requires a reorganisation of the data block structure each time the text is altered.

If the value is negative, the amount of bytes given by the absolute of *ElemCount* will be allocated statically in memory, allowing a direct modification of the text.

Both types of texts must be supported. The static allocation (negative *ElemCount*) is currently only used for texts containing dimension numbers.

The absolute value must be between 1 and 32000 (inclusive).

#### *Text*

[*TEXT*] Character string. The length of the string may be less or equal to *ElemCount*, including 0.

## Base Type "BINARY" (Generic Data Blocks)

This data block is used to store one value of type *BINARY*.



### Contents

**Element Sequence**  
[BlockOwner](#), [BlockType](#), [9](#), [ElemCount](#), [Binary](#);



### Contents

**Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *BlockType*

[*short*] This value is an internal identification of the data block. This value is used by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined objects used.

Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

#### *ElemCount*

[*short*] Maximum length of the binary data in this data block. The number of bytes given by *ElemCount* will be allocated statically in memory, allowing a direct modification of the binary data. The value must be between 1 and 24000 (inclusive).

#### *Binary*

[*BINARY*] Hexadecimal character string. The length of the string may be less or equal to  $1.5 \times \text{ElemCount}$  (since three bytes are always represented by four characters), including 0. The number of characters in the hexadecimal string must be dividable by four!

## Native Data Blocks (Entities and Data Blocks)

Native data blocks are complex data blocks that are used to store native data of the application. Every native data block has its own structure and must be handled separately. Third-party plug-ins are not allowed to define their own native data blocks, they have to use [Generic Data Blocks](#) instead.



### Contents

#### Element Sequence

[BlockOwner](#), [BlockType](#), 0, [ElemCount](#), [Data](#);



### Contents

#### Element Description

##### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

##### *BlockType*

[*short*] This value is an internal identification of the data block. This value is used by the plug-in that created this object. For information about this value see the documentation of the plug-in which should include a description of all custom-defined objects used.

Data blocks created by third-party vendors should use values between 1000 and 29999 (inclusive), the other values are reserved for direct use by TommySoftware®.

##### *ElemCount*

[*short*] Usually number of values that are stored in this data block. Inside native data blocks, this value is usually ignored and set to 0. The only exception are [Attribute Data Blocks](#) which store the maximum attribute length here.

##### *Data*

[*???*] List of values, depending on *BlockType*. For a list of all native block types used, see [Standard Data Blocks](#) and [Attribute Data Blocks](#).

## Attribute Data Blocks (Entities and Data Blocks)

Some entity types can contain explicit attributes. An explicit attribute is a named text containing information and is assigned to an entity to be processed later on, e.g. to create a parts list or to create any other kind of statistic.

Explicit attributes can be either "global" or "local". Global attributes can be assigned to block definitions and are valid for all instances of that block, i.e. they are *equal* for all instances. Local attributes can be assigned to block definitions and to instances. Inside a block definition, a local attribute is only a "recommandation", that means such a local attribute indicates that it could be assigned to any instance of this very block. Inside the instance local attributes are valid only for this single instance. This can lead to different values of a local attribute in several instances of the same block.

Both local and global attributes may either be of type "Text" or "Number". Attributes of type "Text" may contain any text of up to 250 characters in length, whereas attributes of type "Number" must contain a valid numeric value.

Each entity may contain up to 200 attributes. As attributes are referenced to by their names, these names have to be unique.

Explicit attributes are stored in Native Data Blocks, using the following data structure:

 **Contents** **Element Sequence**  
*BlockOwner, BlockType, 0, ElemCount, Name, Text;*

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *BlockType*

[*short*] This value indicates the type of attribute that is stored in the data block. The following value are currently defined:

300	Global attribute, containing text
301	Global attribute, containing a number
400	Local attribute, containing text
401	Local attribute, containing a number

### *ElemCount*

[*short*] Maximum attribute length to be stored in this data block. If the value is positive, only the number bytes actually needed to store the current attribute will be allocated. This saves memory, but requires a reorganisation of the data block structure each time the attribute is altered.

If the value is negative, the amount of bytes given by the absolute of *ElemCount* will be allocated statically in memory, allowing a direct modification of the attribute.

Both types of attributes must be supported, even though the static allocation (negative *ElemCount*) is currently not used.

The absolute value must be between 1 and 32000 (inclusive).

*Name*

[TEXT32] Name of the attribute, up to 32 bytes.

*Text*

[TEXT] Attribute to be stored in form of a ANSI text, maximum length according to *ElemCount*. Attributes of entities owned by TommySoftware® may not be longer than 250 bytes, i.e. *ElemCount* should be set to 250.

## Standard Data Blocks (Entities and Data Blocks)

All kinds of data blocks have a unique identification stored in the value *BlockType*. The application uses some standard data blocks for building its entities. These standard data blocks include generic as well native data blocks. Generic data blocks are normally used for general purposes, whereas native data blocks are used for a few number number of object types.

Data Block Type 000 (General Point)

Data Block Type 001 (Start-Point)

Data Block Type 002 (End-Point)

Data Block Type 003 (Center-Point)

Data Block Type 004 (Radius Definition Point)

Data Block Type 005 (Angle Definition Point)

Data Block Type 006 (Vector Definition Point)

Data Block Type 007 (First Pivot Point)

Data Block Type 008 (Second Pivot Point)

Data Block Type 009 (Arc End-Point)

Data Block Type 010 (Marking)

Data Block Type 100 (Constant)

Data Block Type 101 (Arc)

Data Block Type 102 (Curve)

Data Block Type 110 (Text)

Data Block Type 220 (Dimension Line)

Data Block Type 225 (Large Dimension)

Data Block Type 230 (Small Dimension)

Data Block Type 235 (Standard Text)

Data Block Type 236 (Frame Text)

Data Block Type 237 (Reference Text)

Data Block Type 242 (Clipping Surface)

Data Block Type 243 (Bitmap Reference)

Data Block Type 999 (End of List)

## Data Block Type 000 (General Point) (Standard Data Blocks)

A data block of type 000 is used to store the coordinates of a single, general-purpose point. This may be, e.g., the position of the dimension number.

 **Contents** **Element Sequence**  
BlockOwner, 0, 3, 1, Coordinates;

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Coordinates*

[DPOINT] Coordinates of the point.

## Data Block Type 001 (Start-Point) (Standard Data Blocks)

A data block of type 001 is used to store the coordinates of a single start-point. This may be, e.g., the start-point of a line or Bézier curve.

 **Contents** **Element Sequence**  
BlockOwner, 1, 3, 1, Coordinates;

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Coordinates*

[*DPOINT*] Coordinates of the point.



## Data Block Type 002 (End-Point) (Standard Data Blocks)

A data block of type 002 is used to store the coordinates of a single end-point. This may be, e.g., the end-point of a line or a Bézier curve.

 **Contents** **Element Sequence**  
BlockOwner, 2, 3, 1, Coordinates;

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Coordinates*

[*DPOINT*] Coordinates of the point.

## Data Block Type 003 (Center-Point) (Standard Data Blocks)

A data block of type 003 is used to store the coordinates of a single center-point. This may be, e.g., the center of a circle or an ellipse.

 **Contents** **Element Sequence**  
BlockOwner, 3, 3, 1, Coordinates;

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Coordinates*

[DPOINT] Coordinates of the point.

## Data Block Type 004 (Radius Definition Point) (Standard Data Blocks)

A data block of type 004 is used to store the coordinates of a single radius definition point. This may be, e.g., the radius definition of a circle or a curved dimension line.

### Contents **Element Sequence**

BlockOwner, 4, 3, 1, Coordinates;

### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *Coordinates*

[DPOINT] Coordinates of the point.

## Data Block Type 005 (Angle Definition Point) (Standard Data Blocks)

A data block of type 005 is used to store the coordinates of a single angle definition point. This may be, e.g., the start-angle or end-angle of an arc.

 **Contents** **Element Sequence**  
BlockOwner, 5, 3, 1, Coordinates;

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Coordinates*

[*DPOINT*] Coordinates of the point.

## Data Block Type 006 (Vector Definition Point) (Standard Data Blocks)

A data block of type 006 is used to store the coordinates of a single vector definition point. This may be, e.g., the spreading vector of an ellipse.

 **Contents** **Element Sequence**  
BlockOwner, 6, 3, 1, Coordinates;

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Coordinates*

[*DPOINT*] Coordinates of the point.

## Data Block Type 007 (First Pivot Point) (Standard Data Blocks)

A data block of type 007 is used to store the coordinates of the first pivot point of a Bézier curve.

### Contents **Element Sequence**

BlockOwner, 7, 3, 1, Coordinates;

### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *Coordinates*

[*DPOINT*] Coordinates of the point.

## Data Block Type 008 (Second Pivot Point) (Standard Data Blocks)

A data block of type 008 is used to store the coordinates of the second pivot point of a Bézier curve.

 **Contents** **Element Sequence**  
BlockOwner, 8, 3, 1, Coordinates;

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.


### *Coordinates*

[*DPOINT*] Coordinates of the point.

## Data Block Type 009 (Arc End-Point) (Standard Data Blocks)

A data block of type 009 is used to store the coordinates of the end-point of an arc inside a curve.

 **Contents** **Element Sequence**  
BlockOwner, 9, 3, 1, Coordinates;

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Coordinates*

[*DPOINT*] Coordinates of the point.



## Data Block Type 010 (Marking) (Standard Data Blocks)

A data block of type 010 is used to store the coordinates of a single marking.

 **Contents** **Element Sequence**  
BlockOwner, 10, 3, 1, Coordinates;

 **Contents** **Element Description**

*BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

*Coordinates*

[DPOINT] Coordinates of the point.

## Data Block Type 100 (Constant) (Standard Data Blocks)

A data block of type 100 is used to store a single *double*, mainly in dimensions.

 **Contents** **Element Sequence**  
BlockOwner, 100, 2, 1, Constant;

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.


Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Constant*

[*double*] Constant value.

## Data Block Type 101 (Arc) (Standard Data Blocks)

A data block of type 101 is used to store the direction of a circular or elliptical arc.

 **Contents** **Element Sequence**  
[BlockOwner, 101, 2, 1, Orientation;](#)

 **Contents** **Element Description**

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Orientation*

[*double*] Orientation of the arc. Positive values indicate counter-clockwise direction, negative values indicate clockwise direction.

## Data Block Type 102 (Curve) (Standard Data Blocks)

A data block of type 102 is used to store the direction and the curvature of a circular arc inside a curve.

	<b>Contents</b>	<b>Element Sequence</b>
		<u><a href="#">BlockOwner</a></u> , <u><a href="#">102</a></u> , <u><a href="#">2</a></u> , <u><a href="#">1</a></u> , <u><a href="#">Orientation</a></u> , <u><a href="#">Curvature</a></u> ;

	<b>Contents</b>	<b>Element Description</b>
---	-----------------	----------------------------

### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Orientation*

[*double*] Orientation of the arc. Positive values indicate counter-clockwise direction, negative values indicate clockwise direction.

### *Curvature*

[*double*] Curvature of the circular arc.

## Data Block Type 110 (Text) (Standard Data Blocks)

A data block of type 110 is used to store all kinds of ANSI texts.



### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *Text*

[*TEXT*] Text to be stored, consisting of ANSI characters, delimited by the characters " (Ansi 34). The maximum size of texts is 8,000 characters, inside of dimensions or comments it is 250 characters.

The text may contain variables. Variables are text sections delimited by the character ~ (Ansi 126). A possible text including a variable is:

```
Serial Number: ~SerNum~
```

If such a variable is found inside a text, the program determines whether the text object resides inside a block. If so, an attribute having the same name as the variable (in this example: *SerNum*) is searched. If it does exist, the current value of that attribute is displayed instead of the variable's name. Supposing the value of the attribute *SerNum* is "DT3507", the displayed text would be:

```
Serial Number: DT3507
```

A text may contain more than one variable. If the attribute of the given name is not found, the text "(Undefined)" will be display instead of the attribute's value. If the text object is *not* inside a block, the text is displayed unmodified.

## Data Block Type 220 (Dimension Line) (Standard Data Blocks)

A data block of type 220 is used to store the parameters for dimension lines (straight and curved).

### Contents **Element Sequence**

[BlockOwner](#), 220, 0, 0, [ArrowStartForm](#), [ArrowStartMode](#), [ArrowEndForm](#), [ArrowEndMode](#);

### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *ArrowStartForm*

[*long*] The value *ArrowStartForm* determines the form of the dimension arrow at the start-point of the dimension line. Possible values are:

- |        |  |
|--------|--|
| 0x0000 | No dimension arrow.  |
| 0x0001 | Filled triangular arrow. The triangle has an opening angle of 20° and a side length of 10.0 times the dimension line's width.        |
| 0x0002 | Non-filled triangular arrow. The triangle has an opening angle of 20° and a side length of 10.0 times the dimension line's width.    |
| 0x0003 | Open triangular arrow. The triangle has an opening angle of 60° and a side length of 10.0 times the dimension line's width.          |
| 0x0004 | Diagonal stroke. The stroke has a relative angle of 45° to the dimension line and a length of 12.0 times the dimension line's width. |
| 0x0005 | Filled circle. The filled circle has a radius of 1.5 times the dimension line's width.   |
| 0x0006 | Non-filled circle. The filled circle has a radius of 2.5 times the dimension line's width.   |

#### *ArrowStartMode*

[*long*] The values *ArrowStartMode* determines whether the dimension ends at the dimension's start-point or whether it is extended (which would result in rotated dimension arrows). Possible values are:

- |        |   |
|--------|---|
| 0x0000 | The dimension line ends at the dimension's start-point, the dimension arrows will not be rotated.   |
| 0x0001 | The dimension line is extended, the dimension arrows will be rotated.   |
| 0x0002 | Automatic length detection. If the dimension is less than 30.0 times the dimension line's width, the dimension lines will be extended and the dimension arrows will be rotated. |

The extension of the dimension depends on the dimension arrow type. For types 0x0001, 0x0002 and 0x0003, it is 30.0 times the dimension line's width, for types 0x0004, 0x0005 and 0x0006, it is 5.0 times the dimension line's width. Type 0x0000 has no extension.

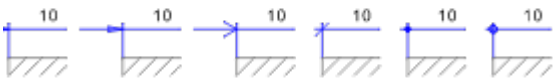
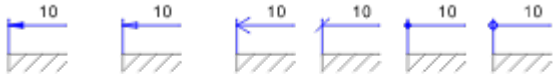
#### *ArrowEndForm*

[*long*] Equivalent to *ArrowStartForm*, but referring to the end-point.

#### *ArrowEndMode*

[*long*] Equivalent to *ArrowStartMode*, but referring to the end-point.

The following image shows all types of dimension arrows, once in normal presentation (upper row), once in rotated presentation with extended dimension line (lower row):



## Data Block Type 225 (Large Dimension) (Standard Data Blocks)

Version 4.2

A data block of type 225 is used to store the parameters for dimensions that include dimension lines (like distance, radius, diameter and angle).

### Contents **Element Sequence**

[BlockOwner](#), 225, 0, 0, [TextFont](#), [TextXProperty](#),  
[TextSize1](#), [TextSize2](#), [CharDistance](#), [TabDistance](#), [TextMode](#),  
[NumAccuracy](#), [NumRefresh](#), [NumCentered](#), [NumTight](#), [NumRotate](#),  
[ArrowStartForm](#), [ArrowStartMode](#), [ArrowEndForm](#), [ArrowEndMode](#),  
[ExtStartDisplay](#), [ExtEndDisplay](#), [LineDisplay](#),  
[LineOrientation](#), [LineType](#), [LineDistMode](#), [LineDistance](#), [LineOffset](#),  
[System](#);

### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *TextFont*

[*FONTDEF*] Description of the font to be used for the dimension texts (dimension and tolerances).

#### *TextXProperty*

[*XPROPERTY*] Properties for the dimension texts (dimension and tolerances). The properties of the entity itself are only valid for dimension line and dimension arrow.

#### *TextSize1*

[*double*] Font size of the dimension in mm.

#### *TextSize2*

[*double*] Font size of the tolerances in mm.

### Contents

#### *CharDistance*

[*double*] The value *CharDistance* determines the gap between two characters. This gap is stated relative to the font size. A value of 0.1 at a font size of 10pt will result in a character gap of 1pt. Allowed values are -10.0 to +10.0. The default value for TrueType and device fonts should be 0.0, for internal fonts 0.125.

### Contents

#### *TabDistance*

[*double*] The value *TabDistance* determines the distance between two tabulators. This distance is stated relative to the font size. A value of 4.0 at a font size of 5 mm will result in a tabulator distance of 20 mm. Allowed values are -100.0 to 100.0. The default value is 4.0.

### Contents

#### *TextMode*

[*long*] The value *TextMode* states the position of the text relative to the insertion point. It can be one of the following values:



- 0x0000 The insertion point defines the left end-point of the text's baseline, i.e. the text will be displayed left-aligned.
- 0x0001 The insertion point defines the center-point of the text's baseline, i.e. the text will be displayed centered.
- 0x0002 The insertion point defines the right end-point of the text's baseline, i.e. the text will be displayed right-aligned.

### *NumAccuracy*

[*long*] The value *NumAccuracy* determines the accuracy of the dimension's display. If numeric values are displayed as decimal numbers, this value determines the number of fractional digits. The value may be between 0 (no fractional digit) and 9 (nine fractional digits). Whether trailing zeros will be displayed or not depends of user-dependent settings in the application. If numeric values are displayed as fractional numbers, this value determines the maximum power of two that the denominator will have. The value may be between 0 (no fraction) and 9 (maximum denominator 512). The resulting fraction will be reduced. If the numeric value is 2.1, the resulting fraction will be  $2 \frac{3}{32}$  for a *NumAccuracy* of 6 and  $2 \frac{51}{512}$  for a *NumAccuracy* of 9.

### *NumRefresh*

[*long*] The value *NumRefresh* determines whether the dimension shall be recalculated after each modification or not. Possible values are:

- 0x0000 The dimension will only be recalculated on demand.
- 0x0001 The dimension will be recalculated after each modification.

### *NumCentered*

[*long*] The value *NumCentered* determines whether the dimension number shall always be placed centered to the dimension line or not. Possible values are:

- 0x0000 The dimension number can be placed anywhere. In this case, (nx1,ny1) determines the center of the base line of the dimension number.
- 0x0001 The dimension number is always placed centered to the dimension line. The resulting position is the base point of a perpendicular dropped from (nx1,ny1) onto the mid-perpendicular of the dimension line.

A rotation angle defined by the points (nx1,ny1) and (nx2,ny2) remains unchanged.

If both *NumTight* and *NumCentered* are non-zero, first the calculation of *NumCentered* is executed, the calculation of *NumTight*.

## *NumTight* Contents

[*long*] The value *NumTight* determines whether the dimension number shall always be placed tight to the dimension line or not. If so, the distance between the text's base line and the dimension line is one-quarter of the dimension's font size. Possible values are:

- 0x0000 The dimension number can be placed anywhere. In this case, (nx1,ny1) determines the center of the base line of the dimension number.
- 0x0001 The dimension number is always placed tight to the dimension line. The resulting position is on the perpendicular dropped from (nx1,ny1) onto the dimension line, having a distance of one-quarter of the dimension's font size to the dimension line.  
A rotation angle defined by the points (nx1,ny1) and (nx2,ny2) remains unchanged.
- 0x0002 The dimension number is always placed inside the dimension line. The resulting position is on the perpendicular dropped from (zx1,zy1) onto the dimension line, having a distance of 40% to the dimension line. The dimension line will be interrupted if this mode is active!  
A rotation angle defined by the points (zx1,zy1) and (zx2,zy2) remains unchanged.

If both *NumTight* and *NumCentered* are non-zero, first the calculation of *NumCentered* is executed, the calculation of *NumTight*.

### *NumRotate*

[long] The value *NumRotate* determines, how the dimension number shall be rotated. Possible values are:

- 0x0000 The dimension number is parallel to the dimension line, with an angle of its base line bewteen  $-90^\circ$  and  $+90^\circ$ , i.e. the text can either be read from below or from the right.
- 0x0001 The dimension number is parallel to a line running through the points (nx1,ny1) und (nx2,ny2).

#### ArrowStartForm

[long] The value *ArrowStartForm* determines the form of the dimension arrow at the start-point of the dimension line. Possible values are:

- 0x0000 No dimension arrow.
- 0x0001 Filled triangular arrow. The triangle has an opening angle of  $20^\circ$  and a side length of 10.0 times the dimension line's width.
- 0x0002 Non-filled triangular arrow. The triangle has an opening angle of  $20^\circ$  and a side length of 10.0 times the dimension line's width.
- 0x0003 Open triangular arrow. The triangle has an opening angle of  $60^\circ$  and a side length of 10.0 times the dimension line's width.
- 0x0004 Diagonal stroke. The stroke has a relative angle of  $45^\circ$  to the dimension line and a length of 12.0 times the dimension line's width.
- 0x0005 Filled circle. The filled circle has a radius of 1.5 times the dimension line's width.
- 0x0006 Non-filled circle. The filled circle has a radius of 2.5 times the dimension line's width.

#### ArrowStartMode

[long] The values *ArrowStartMode* determines whether the dimension ends at the dimension's start-point or whether it is extended (which would result in rotated dimension arrows). Possible values are:

- 0x0000 The dimension line ends at the dimension's start-point, the dimension arrows will not be rotated.
- 0x0001 The dimension line is extended, the dimension arrows will be rotated.
- 0x0002 Automatic length detection. If the dimension is less than 30.0 times the dimension line's width, the dimension lines will be extended and the dimension arrows will be rotated.

The extension of the dimension depends on the dimension arrow type. For types 0x0001, 0x0002 and 0x0003, it is 30.0 times the dimension line's width, for types 0x0004, 0x0005 and 0x0006, it is 5.0 times the dimension line's width. Type 0x0000 has no extension.

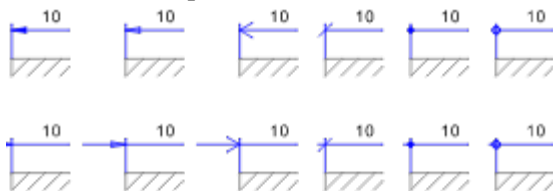
#### ArrowEndForm

[long] Equivalent to *ArrowStartForm*, but referring to the end-point.

#### ArrowEndMode

[long] Equivalent to *ArrowStartMode*, but referring to the end-point.

The following image shows all types of dimension arrows, once in normal presentation (upper row), once in rotated presentation with extended dimension line (lower row):



#### ExtStartDisplay

[long] Determines whether the dimension extension line at the start-point of the dimension shall be drawn or not. Possible values are:

- 0x0000 The dimension extension line at the start-point will not be drawn.
- 0x0001 The dimension extension line at the start-point will be drawn.

#### ExtEndDisplay

[long] Determines whether the dimension extension line at the end-point of the dimension shall be drawn or not. Possible values are:

0x0000 The dimension extension line at the end-point will not be drawn.  
0x0001 The dimension extension line at the end-point will be drawn.

#### *LineDisplay*

[*long*] Determines whether to display dimension line and dimension extension line at all or not.  
Possible values are:

0x0000 The dimension line and optionally the dimension extension lines will not be drawn.  
0x0001 The dimension line and optionally the dimension extension lines will be drawn.

#### *LineOrientation,*

#### *LineType,*

#### *LineDistMode*

[*long*] The usage of these values depends on the object type the data block is defined in. Please refer to the description of the corresponding object type for detailed information.

#### *LineDistance*

[*double*] This value determines the distance between the dimension line and corresponding object in mm. It will only be used if *LineDistMode* is set to a non-zero value.

## *LineOffset* Contents

[*double*] This value determines the offset between the dimension extension lines and corresponding object in mm.

#### *System*

[*long*] Index of the coordinate system the dimension shall be based on. If this coordinate system is set to a distorting display (isometric or dimetric) the dimension will be calculated accordingly.

## Data Block Type 230 (Small Dimension) (Standard Data Blocks)

### Contents

A data block of type 230 is used to store the parameters for dimensions that do not include dimension lines (like coordinate or perimeter).

### Contents **Element Sequence**

[BlockOwner](#), [230](#), [0](#), [0](#), [TextFont](#), [TextXProperty](#),  
[TextSize1](#), [TextSize2](#), [CharDistance](#), [TabDistance](#), [TextMode](#),  
[NumAccuracy](#), [NumRefresh](#), [NumRotate](#),  
[System](#);

### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *TextFont*

[*FONTDEF*] Description of the font to be used for the dimension texts (dimension and tolerances).

#### *TextXProperty*

[*XPROPERTY*] Properties for the dimension texts (dimension and tolerances). The properties of the entity itself are only valid for dimension line and dimension arrow.

#### *TextSize1*

[*double*] Font size of the dimension in mm.

#### *TextSize2*

[*double*] Font size of the tolerances in mm.

### Contents

#### *CharDistance*

[*double*] The value *CharDistance* determines the gap between two characters. This gap is stated relative to the font size. A value of 0.1 at a font size of 10pt will result in a character gap of 1pt. Allowed values are -10.0 to +10.0. The default value for TrueType and device fonts should be 0.0, for internal fonts 0.125.

### Contents

#### *TabDistance*

[*double*] The value *TabDistance* determines the distance between two tabulators. This distance is stated relative to the font size. A value of 4.0 at a font size of 5 mm will result in a tabulator distance of 20 mm. Allowed values are -100.0 to 100.0. The default value is 4.0.

### Contents

#### *TextMode*

[*long*] The value *TextMode* states the position of the text relative to the insertion point. It can be one of the following values:

0x0000 The insertion point defines the left end-point of the text's baseline, i.e. the text will be

- displayed left-aligned.
- 0x0001 The insertion point defines the center-point of the text's baseline, i.e. the text will be displayed centered.
- 0x0002 The insertion point defines the right end-point of the text's baseline, i.e. the text will be displayed right-aligned.

### *NumAccuracy*

[*long*] The value *NumAccuracy* determines the accuracy of the dimension's display. If numeric values are displayed as decimal numbers, this value determines the number of fractional digits. The value may be between 0 (no fractional digit) and 9 (nine fractional digits). Whether trailing zeros will be displayed or not depends of user-dependent settings in the application. If numeric values are displayed as fractional numbers, this value determines the maximum power of two that the denominator will have. The value may be between 0 (no fraction) and 9 (maximum denominator 512). The resulting fraction will be reduced. If the numeric value is 2.1, the resulting fraction will be  $2 \frac{3}{32}$  for a *NumAccuracy* of 6 and  $2 \frac{51}{512}$  for a *NumAccuracy* of 9.

### *NumRefresh*

[*long*] The value *NumRefresh* determines whether the dimension shall be recalculated after each modification or not. Possible values are:

- 0x0000 The dimension will only be recalculated on demand.
- 0x0001 The dimension will be recalculated after each modification.

### *NumRotate*

[*long*] The value *NumRotate* determines, how the dimension number shall be rotated. Possible values are:

- 0x0000 The dimension number is parallel to the dimension line with an angle of its base line bewteen  $-90^\circ$  and  $+90^\circ$ , i.e. the text can either be read from below or from the right.
- 0x0001 The dimension number is parallel to a line running throught the points (nx1,ny1) und (nx2,ny2).

### *System*

[*long*] Index of the coordinate system the dimension shall be based on. If this coordinate system is set to a distorting display (isometric or dimetric), the dimension will be calculated accordingly.

## Data Block Type 235 (Standard Text) (Standard Data Blocks)

A data block of type 235 is used to store the parameters for standard texts, i.e. for texts that have an insertion point, but no surrounding frame.

### Contents **Element Sequence**

[BlockOwner](#), 235, 0, 0, [TextFont](#), [TextXProperty](#), [TextMatrix](#),  
[CharDistance](#), [TabDistance](#), [LineDistance](#), [TextMode](#);

### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *TextFont*

[*FONTDEF*] Description of the font to be used for this text.

#### *TextXProperty*

[*XPROPERTY*] Properties of the text. The properties of the entity itself are not used for texts.

#### *TextMatrix*

[*MATRIX*] Display matrix of the text. The text will be multiplied with this matrix before display. The matrix contains all operations like translation, rotation, scaling and shearing. The font size is coded as scaling relative to 1 mm.

All transformations apply to the *complete* text, not to single characters, i.e. a rotation will rotate the complete text, not the single characters.

#### *CharDistance*

[*double*] The value *CharDistance* determines the gap between two characters. This gap is stated relative to the font size. A value of 0.1 at a font size of 10pt will result in a character gap of 1pt. Allowed values are -10.0 to +10.0. The default value for TrueType and device fonts should be 0.0, for internal fonts 0.125.

#### *TabDistance*

[*double*] The value *TabDistance* determines the distance between two tabulators. This distance is stated relative to the font size. A value of 4.0 at a font size of 5 mm will result in a tabulator distance of 20 mm. Allowed values are -100.0 to 100.0. The default value is 4.0.

#### *LineDistance*

[*double*] The value *LineDistance* determines the offset between two lines of text, measured from baseline to baseline. This offset is stated relative to the font size. A value of 1.2 at a font size of 10pt will lead to a line offset of 12pt. Allowed values are -100.0 to 100.0. The default value is 1.0.

#### *TextMode*

[*long*] The value *TextMode* states the position of the text relative to the insertion point. It can be one of the following values:

- 0x0000 The insertion point defines the left end-point of the text's baseline, i.e. the text will be displayed left-aligned.
- 0x0001 The insertion point defines the center-point of the text's baseline, i.e. the text will be displayed centered.
- 0x0002 The insertion point defines the right end-point of the text's baseline, i.e. the text will be displayed right-aligned.

# Contents **Example**

```
0,235,0,0, |BlockType|
1,1,400,"Times New Roman", |TextFont|
64, |TextXProperty.Flag|
0,1,0.0/0.0/0.0,0.0/0.0/0.0,0.0,0,0, | ... |
1, |TextXProperty.Layer|
10.0,0.0,0.0,10.0,7.0,-7.0, |TextMatrix|
0.0,4.0,1.0, |CharDistance,TabDistance,LineDistance|
0; |TextMode|
```

The text will be displayed using the TrueType font "Times New Roman Italic" with a text size of 10 mm. It will be non-rotated and is aligned left to the position (7.0,-7.0). The characters will be assigned to layer 1 (*TextXProperty.Flag* = USE\_LAYER, *TextXProperty.Layer* = 1), they are drawn filled in black. The character distance will be 0.0 mm (font size 10 mm  $\times$  0.0 = 0.0 mm), the tabulator distance will be 40 mm (font size 10 mm  $\times$  4.0 = 40 mm) and the line distance 10 mm (font size 10 mm  $\times$  1.0 = 10.0 mm).

## Data Block Type 236 (Frame Text) (Standard Data Blocks)

A data block of type 236 is used to store the parameters for frame texts, i.e. for texts that have a surrounding frame, but no insertion point.

### Contents **Element Sequence**

[BlockOwner](#), 236, 0, 0, [TextFont](#), [TextXProperty](#), [TextSize](#),  
[CharDistance](#), [TabDistance](#), [LineDistance](#), [TextMode](#);

### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *TextFont*

[*FONTDEF*] Description of the font to be used for this text.

#### *TextXProperty*

[*XPROPERTY*] Properties of the text. The properties of the entity itself are not used for texts.

#### *TextSize*

[*double*] Size of the font. This size is handled differently depending of the font's type. If the font is a TrueType or device font, this value determines the typographical font size, i.e. the minimum offset between two lines of texts.

If the font is internal, this value determines the actual character height. Usually, an internal font will be displayed about 25% larger with the same value of *TextSize*.

#### *CharDistance*

[*double*] The value *CharDistance* determines the gap between two characters. This gap is stated relative to the font size. A value of 0.1 at a font size of 10pt will result in a character gap of 1pt. Allowed values are -10.0 to +10.0. The default value for TrueType and device fonts should be 0.0, for internal fonts 0.125.

#### *TabDistance*

[*double*] The value *TabDistance* determines the distance between two tabulators. This distance is stated relative to the font size. A value of 4.0 at a font size of 5 mm will result in a tabulator distance of 20 mm. Allowed values are -100.0 to 100.0. The default value is 4.0.

#### *LineDistance*

[*double*] The value *LineDistance* determines the offset between two lines of text, measured from baseline to baseline. This offset is stated relative to the font size. A value of 1.2 at a font size of 10pt will lead to a line offset of 12pt. Allowed values are -100.0 to 100.0. The default value is 1.0.

#### *TextMode*

[*long*] The value *TextMode* states the position of the text relative to the surrounding frame. It can be one of the following values:

0x0000 The text will be displayed left-aligned inside the surrounding frame.

0x0001 The text will be displayed centered inside the surrounding frame.

0x0002 The text will be displayed right-aligned inside the surrounding frame.

0x0003 The text will be displayed justified. The last line of each paragraph and lines containing a single word will be display left-aligned.

For justification, the word gaps made up by the character Ansi 32 will be enlarged. Word



gaps made up by the character Ansi 160 will *not* be enlarged, i.e. Ansi 160 is a "fixed space".

## Contents **Example**

```
0,236,0,0, |BlockType|
0,1,400,"TIMES", |TextFont|
64, |TextXProperty.Flag|
0,1,0.0/0.0/0.0,0.0/0.0/0.0,0.0,0,0, | ... |
1, |TextXProperty.Layer|
10.0, |TextSize|
0.05,4.0,1.2, |CharDistance,Tabdistance,LineDistance|
0; |TextMode|
```

The text will be displayed using the font "TIMES" with a text size of 10 mm. The characters will be assigned to layer 1 (*TextXProperty.Flag* = USE\_LAYER, *TextXProperty.Layer* = 1), they are drawn filled in black.

The character distance will be 0.5 mm (font size 10 mm  $\times$  0.05 = 0.5 mm), the tabulator distance will be 40 mm (font size 10 mm  $\times$  4.0 = 40 mm) and the line distance 12 mm (font size 10 mm  $\times$  1.2 = 12.0 mm).

## Data Block Type 237 (Reference Text) (Standard Data Blocks)

A data block of type 237 is used to store the additional parameters for reference texts, i.e. for texts that are used for numeration, referencing etc.

### Contents **Element Sequence**

[BlockOwner](#), 237, 0, 0, [ArrowForm](#), [ArrowMode](#), [FrameForm](#), [FrameOffset](#);

### Contents **Element Description**

#### *BlockOwner*

*[short]* This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *ArrowForm*

*[long]* The value *ArrowForm* determines the form of the arrow at the start-point (x,y) of the reference line. Possible values are:

- 0x0000 No dimension arrow.
- 0x0001 Filled triangular arrow. The triangle has an opening angle of 20° and a side length of 10.0 times the dimension line's width.
- 0x0002 Non-filled triangular arrow. The triangle has an opening angle of 20° and a side length of 10.0 times the dimension line's width.
- 0x0003 Open triangular arrow. The triangle has an opening angle of 60° and a side length of 10.0 times the dimension line's width.
- 0x0004 Diagonal stroke. The stroke has a relative angle of 45° to the dimension line and a length of 12.0 times the dimension line's width.
- 0x0005 Filled circle. The filled circle has a radius of 1.5 times the dimension line's width.
- 0x0006 Non-filled circle. The filled circle has a radius of 2.5 times the dimension line's width.

#### *ArrowMode*

*[long]* The value *ArrowMode* determines the form of the reference line. Allowed values are:

- 0x0000 Straight
- 0x0001 Bend, 45° angle at the arrow's side
- 0x0002 Bend, 45° angle at the text's side
- 0x0003 Bend, 90° angle at the arrow's side
- 0x0004 Bend, 90° angle at the text's side
- 0x0005 Horizontal
- 0x0006 Vertical

#### *FrameForm*

*[long]* The value *FrameForm* determines, which form the surrounding frame of the text shall have. Allowed values are:

- 0x0000 Rectangle
- 0x0001 Rhomb
- 0x0002 Circle
- 0x0003 Ellipse

#### *FrameOffset*

*[double]* The value *FrameOffset* determines the minimum distance between text itself and its surrounding frame.

# Contents **Example**

```
0,237,0,0, |BlockType|  
1,1,0,2; |  
ArrowForm,ArrowMode,FrameForm,FrameOffset|
```

The text will be displayed inside a circle, whose radius will be 2 mm larger than the minimum circle surrounding the text. The reference line will have a bend of 45° at the arrow's end, the arrow will be a filled triangle.

## Data Block Type 242 (Clipping Surface) (Standard Data Blocks)

A data block of type 242 is used to store the parameters for clipping surfaces, i.e. for surface that are used to clip other objects.

### Contents **Element Sequence**

[BlockOwner](#), 242, 0, 0, [XProperty](#),  
[LibraryName](#), [BlockName](#), [DisplayMatrix](#), [IgnoreBlock](#);

### Contents **Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *XProperty*

[*XPROPERTY*] Properties of the instance including transmission.

#### *LibraryName*

[*TEXT64*] Name of the library containing the desired block, maximum 63 characters. If the instance references a block located in the same drawing / library as the instance, set this name to "\*".

#### *BlockName*

[*TEXT64*] Name of the block, maximum 63 characters. If the first character is 0x00, this instance is invalid and will neither be loaded nor stored!

If *LibraryName* is set to "\*", the content of *BlockName* may have a special form. If the first character is # (Ansi 35), the block is a special, internally used and handled block (see [Entity "Group"](#) and [Entity "Position Number"](#)). The function of this block then depends on the character following the # sign. Such instances are only allowed inside drawings, *not* inside libraries!

#### *DisplayMatrix*

[*MATRIX*] Display matrix of the block. All entities stored in the block have to be multiplied with this matrix before display. It contains translation, rotation, scaling and shearing.

#### *IgnoreBlock*

[*long*] This value determines, how a clipping surface behaves during user interaction. Allowed values are:

- 0x0000 The block referenced by the clipping surface will be used during all operations. In this case, a clipping surface will behave like a surface plus an instance.
- 0x0001 The block referenced by the clipping surface is ignored during operations like object selection. This can be used to "hide" the internal structure of the clipping surface from the user. In this case, a clipping surface will behave like a standard filled surface, i.e. it can only be identified by clicking onto its outline.

### Contents **Example**

```
0, 242, 0, 0, |BlockType|
0, 0, 0, 0/0/0, 0/0/0, 0.0, 0, 0, |HeaderType, XProperty|
" * ", |LibraryName|
" MyBlock ", |BlockName|
1.0, 0.0, 0.0, 1.0, 10.0, -7.0, |DisplayMatrix|
```


```
0;
```

```
|IgnoreBlock|
```

This clipping surface displays the block "MyBlock". The entities of the block will be displayed with their own properties (*XProperty.Flag* = USE\_NULL). The block will be displayed at the location (10.0,-7.0), non-rotated and non-scaled. The referenced block will be fully operational.

## Data Block Type 243 (Bitmap Reference) (Standard Data Blocks)

A data block of type 243 is used to store the parameters for bitmap references, i.e. for objects that are used to display bitmap files in the drawing.



### Contents

**Element Sequence**

```
BlockOwner, 243, 0, 0, BitmapName,  
DisplayMatrix;
```



### Contents

**Element Description**

#### *BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *BitmapName*

[*TEXT256*] Name of the bitmap to be displayed. If this name starts with the prefix character '#' (Ansi 35), this name is a reference to an internal bitmap listed in the Section =BITMAP= (Embedded Bitmaps) of this file. Else, it is a file name of a valid Windows-style bitmap file. The maximum size of such a bitmap is 32,000 by 32,000 pixels, using color depths of 1 bit, 4 bit, 8 bit or 24 bit.

#### *DisplayMatrix*

[*MATRIX*] Display matrix of the bitmap. The position stored in the matrix places the lower left corner of the bitmap, the scaling and rotation information determines the size and orientation of the display. The size is relative to the default bitmap resolution stored in the bitmap's header. If the bitmap does not contain a valid resolution information, its resolution is assumed to be 300 dpi.



### Contents

**Example**

```
0, 243, 0, 0,                               |BlockType|  
"C:\\BITMAPS\\MYIMAGE.BMP",                 |BitmapName|  
2.0, 0.0, 0.0, 2.0, 10.0, -7.0;             |DisplayMatrix|
```

This bitmap referende displays the bitmap C:\BITMAPS\MYIMAGE.BMP. It will be located at (10.0, 7.0) and will be displayed in twice its original size.

## Data Block Type 999 (End of List) (Standard Data Blocks)

A data block of type 999 is used to end the list of data blocks inside an entity's data area.

 **Contents** **Element Sequence**  
BlockOwner, 999, 0, 0;

 **Contents** **Element Description**

*BlockOwner*

[*short*] This value is a unique identification of the plug-in that created the data block. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

## Entity "Object"

This entity represents the smallest display element of drawings. An object may be a line, a circle, a text etc. It consists of a header, geometrical data and some object-specific data. An entity of type "Object" has the following structure:



### Contents

**Element Sequence**  
UnitOwner, 0, XProperty, ObjectType;  
-Data Section-



### Contents

**Element Description**

#### *UnitOwner*

[*short*] This value is a unique identification of the plug-in that created the entity. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *XProperty*

[*XPROPERTY*] Properties of the object including transmission.

#### *ObjectType*

[*long*] Type of the object. Determines whether this entity contains a line, a circle or another object type. For a description of all object types see below.

Following is the data section of this entity. This section contains a list of data blocks (see [Entities and Data Blocks](#)). Currently, objects do not contain attributes. This may change in further releases, so be prepared to find attributes and other data blocks in this section.



### Contents

**Example**

0,0,0,1,0,0/0/0,0/0/0,0.0,0,0,0;	UnitOwner,XProperty,ObjectType
0,1,2,1,-10.0,7.0;	B001
0,2,2,1,10.0,-7.0;	B002
0,999,0,0;	B999

This entity is an object of type "Line". It represents a line starting at (-10.0, 7.0) and ending at (10.0, -7.0). This line will be displayed using pen 1 and is assigned to layer 0.

For each object type, the following descriptions will list all required data blocks in their correct order. In addition, the object type's definition will be described. If object-specific data blocks are used, their specific values will be described, too.

## Standard Objects

Standard objects are "simple" objects that can be assembled of lines, circle parts and ellipse parts.

Object 00 "Line"

Object 01 "Hatching"



Object 05 "Circle"  
Object 06 "Circular Arc"  
Object 07 "Circular Sector"  
Object 08 "Circular Segment"  
Object 15 "Ellipse"  
Object 16 "Elliptical Arc"  
Object 17 "Elliptical Sector"  
Object 18 "Elliptical Segment"

## Extended Objects

Extended objects are more complex than standard objects. Such object cannot directly be handled by trimming and other modifications. Instead, they will have to be resolved in polylines before. This requires additional processing time and leads to reduced accuracy.

Object 10 "Zigzag Line"  
Object 11 "Spline"  
Object 12 "Curve"  
Object 13 "Surface"  
  
Object 40 "Comment"  
Object 41 "Marking"  
Object 42 "Clipping Surface"  
Object 43 "Bitmap Reference"

## Dimension Objects

Dimension objects are usually used to add dimensions to a drawing. In addition, they can be used to create simple arrows. Dimensions are very complex and should be handled carefully.

The following image shows the different elements of a distance dimension and their names. These names are also equally valid for other dimension forms:



Dimensions are always referring to a coordinate system. This coordinate system contains all required information about the current entities, the scale and the desired number display modes.

**Note:** The examples shown in this section are mainly not according to DIN rules. Instead, they shall display all possible forms of dimension available in compact form.

Object 20 "Dimension Line Straight"  
Object 21 "Dimension Line Curved"  
Object 25 "Dimension Distance"  
Object 26 "Dimension Radius"  
Object 27 "Dimension Diameter"  
Object 28 "Dimension Angle"  
Object 29 "Dimension Arc Length"  
Object 30 "Dimension Coordinate"  
Object 31 "Dimension Area"

Object 32 "Dimension Perimeter"

## **Text Objects**

Text objects are used for the lettering of the drawing and are basically implemented as a multi-instance of characters.

Object 35 "Standard Text"

Object 36 "Frame Text"

Object 37 "Reference Text"

## **Geometry Objects**

Geometry objects are used to create an auxiliary geometry for easy and fast construction. They are not relevant for printer or plotter output (except if explicitly activated by the user) and will only be displayed on the screen. Geometry objects can usually be ignored during file conversion.

Object 45 "Geometry Line"

Object 46 "Geometry Circle"

Object 47 "Geometry Ellipse"

## Object 00 "Line"



# Contents

### Data Block Sequence

Data Block 001( x1, y1 )      - Start-point  
Data Block 002( x2, y2 )      - End-point

The points (x1,y1) and (x2,y2) determine the start-point and end-point of the line. The line pattern has to start at the point (x1,y1). When drawing lines to raster devices the final pixel has to be drawn in any case! In Windows this has to be done explicitly.

A line is an "open" object, i.e. it has a non-closed outline and cannot be filled.

# Object 01 "Hatching"



## Contents

### Data Block Sequence

<u>Data Block 001</u> ( x1, y1 )	- Start-point of a curve
...	
<u>Data Block 002</u> ( x?, y? )	- End-point of a line
...	
<u>Data Block 007</u> ( x?, y? )	- Pivot point 1
<u>Data Block 008</u> ( x?, y? )	- Pivot point 2
<u>Data Block 002</u> ( x?, y? )	- End-point of a Bézier curve
...	
<u>Data Block 009</u> ( x?, y? )	- End-point of the circular arc
<u>Data Block 102</u> ( Orientation, Curvature )	- Orientation of the circular arc - Curvature of the circular arc

A hatching is a collection of several curves. A curve starts with a start-point in a data block of type 001 followed by a sequence of curve elements. Three types of curve elements are available:

#### Line

A line is simply defined by stating its end-point in a data block of type 002. The line is drawn from the curve's current end-point to the line's end-point. The line's end-point then becomes the curve's current end-point.

#### Bézier curve

A Bézier curve is defined by stating two pivot points in data blocks of type 007 and 008, and an end-point in a data block of type 002. The Bézier curve is drawn from the curve's current end-point (named 'S') to the Bézier curve's end-point (named 'E'), influenced by the two pivot points (named P1 and P2). The Bézier curve's end-point then becomes the curve's current end-point.

The points P of such a Bézier curve are calculated using the following equation:

$$P = (1-t)^3 \times S + 3t(1-t)^2 \times P1 + 3t^2(1-t) \times P2 + t^3 \times E \quad (0 \leq t \leq 1)$$

#### Circular arc

A circular arc is defined by its end-point in a data block of type 009, and its orientation and curvature in a data block of type 102. The arc is drawn from the curve's current end-point (named 'S') to the arc's end-point (named 'E'), influenced by its orientation and curvature. The arc's end-point then becomes the curve's current end-point.

The value *Orientation* determines whether to draw the arc in clockwise direction (*Orientation* < 0) or in counter-clockwise direction (*Orientation* >= 0).

The value *Curvature* determines the radius of the arc in indirect manner. It can be handled in two ways. In geometrical view, the absolute value of *Curvature* is  $1/(2 \tan(\beta/2))$ , where  $\beta$  is the arc-angle of the circular arc. The sign of *Curvature* determines, which of the two possible arcs to use.

In practical use, this definition is not very handy. Instead, *Curvature* should be used to perform a simple vector calculation to obtain the arc's defining center-point M:

$$\begin{aligned} x(M) &= 0.5 * ( x(S) + x(E) ) - Curvature * ( y(E) - y(S) ) \\ y(M) &= 0.5 * ( y(S) + y(E) ) + Curvature * ( x(E) - x(S) ) \end{aligned}$$

After calculating the center-point, all points required to draw a standard circular arc are known. The allowed value range of *Curvature* is  $\pm 1e100$ .

Each curve ends if either a new curve is started by a new start-point (in a data block of type 001) or the complete surface is ended (by a data block of type 999). A hatching is an "open" object, i.e. it has a non-closed outline and cannot be filled.

If a hatching is drawn with a non-solid line pattern, this line pattern has to be continued during each curve. A hatching may contain up to 1000 nested curves made of up to 2000 data blocks in total.

## Object 05 "Circle"



# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 004</u> ( rx, ry )	- Radius definition

The point (cx,cy) determines the circle's center-point, the point (rx,ry) is a point on the circle's outline and thus defines the radius.

## Object 06 "Circular Arc"



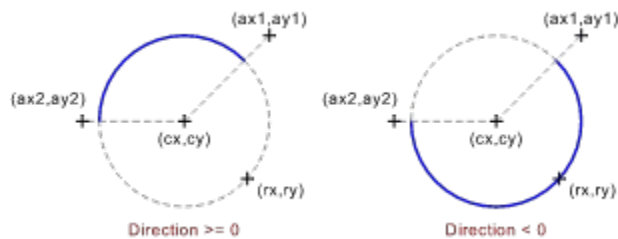
# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 004</u> ( rx, ry )	- Radius definition
<u>Data Block 005</u> ( ax1, ay1 )	- Start-angle definition
<u>Data Block 005</u> ( ax2, ay2 )	- End-angle definition
<u>Data Block 101</u> ( Orientation )	- Arc direction

The point (cx,cy) determines the circle's center-point, the point (rx,ry) is a point on the circle's outline and thus defines the radius. The points (ax1,ay1) and (ax2,ay2), in relation to the circle's center-point (cx,cy), determine the start- and end-angle of the arc. If *Orientation*  $\geq 0$ , the arc is drawn counter-clockwise from the start-angle to the end-angle. If *Orientation*  $< 0$ , the arc is drawn clockwise.

A circular arc is an "open" object, i.e. it has a non-closed outline and cannot be filled.



## Object 07 "Circular Sector"

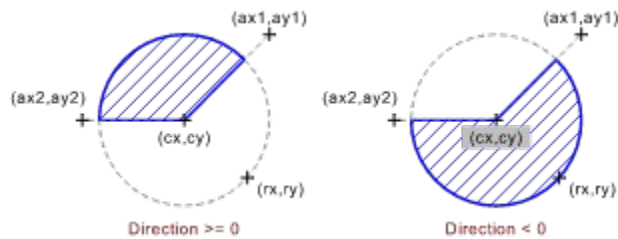


# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 004</u> ( rx, ry )	- Radius definition
<u>Data Block 005</u> ( ax1, ay1 )	- Start-angle definition
<u>Data Block 005</u> ( ax2, ay2 )	- End-angle definition
<u>Data Block 101</u> ( Orientation )	- Arc direction

The point (cx,cy) determines the circle's center-point, the point (rx,ry) is a point on the circle's outline and thus defines the radius. The points (ax1,ay1) and (ax2,ay2), in relation to the circle's center-point (cx,cy), determine the start- and end-angle of the sector. If *Orientation*  $\geq 0$ , the sector is drawn counter-clockwise from the start-angle to the end-angle. If *Orientation*  $< 0$ , the sector is drawn clockwise.





## Object 08 "Circular Segment"

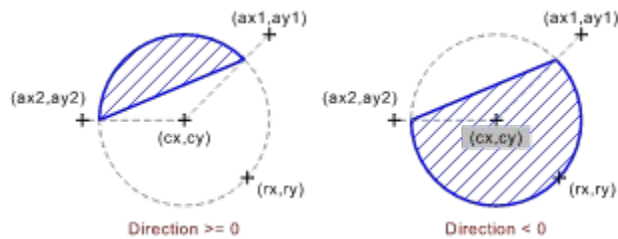


# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 004</u> ( rx, ry )	- Radius definition
<u>Data Block 005</u> ( ax1, ay1 )	- Start-angle definition
<u>Data Block 005</u> ( ax2, ay2 )	- End-angle definition
<u>Data Block 101</u> ( Orientation )	- Arc direction

The point (cx,cy) determines the circle's center-point, the point (rx,ry) is a point on the circle's outline and thus defines the radius. The points (ax1,ay1) and (ax2,ay2), in relation to the circle's center-point (cx,cy), determine the start- and end-angle of the segment. If *Orientation*  $\geq 0$ , the segment is drawn counter-clockwise from the start-angle to the end-angle. If *Orientation*  $< 0$ , the segment is drawn clockwise.



## Object 10 "Zigzag Line"



# Contents

### Data Block Sequence

<u>Data Block 001</u> ( x1, y1 )	- Start-point
<u>Data Block 002</u> ( x2, y2 )	- End-point
<u>Data Block 100</u> ( Distance )	- Distance between two indents in mm

The points (x1,y1) and (x2,y2) determine the start-point and end-point of the line. The value *Distance* determines the distance between two indents. Each indent has an opening angle of 30°, its total height (both sides together) is 20 times the line's width.

If a zigzag line is drawn with a non-solid line pattern, this line pattern has to be continued along the complete line. A zigzag line is an "open" object, i.e. it has a non-closed outline and cannot be filled.



## Object 11 "Spline"



# Contents

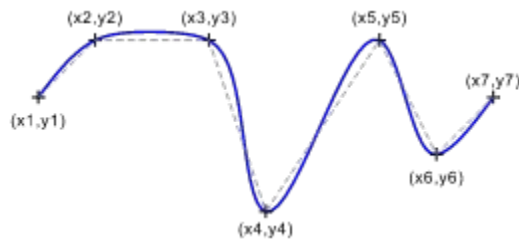
### Data Block Sequence

<u>Data Block 001</u> ( x1, y1 )	- Start-point
<u>Data Block 002</u> ( x2, y2 )	- Definition point
...	
<u>Data Block 002</u> ( x?, y? )	- End-point

A spline is a curve that connects a sequence of points with a curved line as "smooth" as possible, i.e. the connecting line has no sharp bends and a minimum deflection. The theory of splines covers several types of splines, basically differing in the way that "smoothness" is defined.

The most usual spline form in technical design is a "cubic spline", so this form is used here. Since the calculation effort for a plain cubic spline is enormous, the application uses an interpolation method that first calculates a short cubic spline for each point triple, resulting in two different curve segments for each point-to-point segment. The resulting curve is then calculated as the connection of those two curve segments.

If a spline is drawn with a non-solid line pattern, this line pattern has to be continued along the complete spline. A spline may contain up to 2000 data blocks in total. A spline is an "open" object, i.e. it has a non-closed outline and cannot be filled.



## Object 12 "Curve"



# Contents

### Data Block Sequence

<u>Data Block 001</u> ( x1, y1 )	- Start-point of a curve
...	
<u>Data Block 002</u> ( x?, y? )	- End-point of a line
...	
<u>Data Block 007</u> ( x?, y? )	- Pivot point 1
<u>Data Block 008</u> ( x?, y? )	- Pivot point 2
<u>Data Block 002</u> ( x?, y? )	- End-point of a Bézier curve
...	
<u>Data Block 009</u> ( x?, y? )	- End-point of the circular arc
<u>Data Block 102</u> ( Orientation,	- Orientation of the circular arc
Curvature )	- Curvature of the circular arc

A curve describes a complex object made up of several elements. A curve starts with a start-point in a data block of type 001 followed by a sequence of curve elements. Three types of curve elements are available:

#### Line

A line is simply defined by stating its end-point in a data block of type 002. The line is drawn from the curve's current end-point to the line's end-point. The line's end-point then becomes the curve's current end-point.

#### Bézier curve

A Bézier curve is defined by stating two pivot points in data blocks of type 007 and 008, and an end-point in a data block of type 002. The Bézier curve is drawn from the curve's current end-point (named 'S') to the Bézier curve's end-point (named 'E'), influenced by the two pivot points (named P1 and P2). The Bézier curve's end-point then becomes the curve's current end-point.

The points P of such a Bézier curve are calculated using the following equation:

$$P = (1-t)^3 \times S + 3t(1-t)^2 \times P1 + 3t^2(1-t) \times P2 + t^3 \times E \quad (0 \leq t \leq 1)$$

#### Circular arc

A circular arc is defined by its end-point in a data block of type 009, and its orientation and curvature in a data block of type 102. The arc is drawn from the curve's current end-point (named 'S') to the arc's end-point (named 'E'), influenced by its orientation and curvature. The arc's end-point then becomes the curve's current end-point.

The value *Orientation* determines whether to draw the arc in clockwise direction (*Orientation* < 0) or in counter-clockwise direction (*Orientation* >= 0).

The value *Curvature* determines the radius of the arc in indirect manner. It can be handled in two ways. In geometrical view, the absolute value of *Curvature* is  $1/(2 \tan(\beta/2))$ , where  $\beta$  is the arc-angle of the circular arc. The sign of *Curvature* determines, which of the two possible arcs to use.

In practical use, this definition is not very handy. Instead, *Curvature* should be used to perform a simple vector calculation to obtain the arc's defining center-point M:

$$\begin{aligned} x(M) &= 0.5 * ( x(S) + x(E) ) - Curvature * ( y(E) - y(S) ) \\ y(M) &= 0.5 * ( y(S) + y(E) ) + Curvature * ( x(E) - x(S) ) \end{aligned}$$

After calculating the center-point, all points required to draw a standard circular arc are known. The allowed value range of *Curvature* is  $\pm 1e100$ .

The curve ends if the complete curve is ended (by a data block of type 999). A curve is an "open" object, i.e. it has a non-closed outline and cannot be filled.

If a curve is drawn with a non-solid line pattern, this line pattern has to be continued along the complete curve. A curve may contain up to 2000 data blocks in total.

## Object 13 "Surface"



# Contents

### Data Block Sequence

<u>Data Block 001</u> ( x1, y1 )	- Start-point of a curve
...	
<u>Data Block 002</u> ( x?, y? )	- End-point of a line
...	
<u>Data Block 007</u> ( x?, y? )	- Pivot point 1
<u>Data Block 008</u> ( x?, y? )	- Pivot point 2
<u>Data Block 002</u> ( x?, y? )	- End-point of a Bézier curve
...	
<u>Data Block 009</u> ( x?, y? )	- End-point of the circular arc
<u>Data Block 102</u> ( Orientation, Curvature )	- Orientation of the circular arc - Curvature of the circular arc

A surface is a collection of several curves, each defining a closed area. A curve starts with a start-point in a data block of type 001 followed by a sequence of curve elements. Three types of curve elements are available:

#### Line

A line is simply defined by stating its end-point in a data block of type 002. The line is drawn from the curve's current end-point to the line's end-point. The line's end-point then becomes the curve's current end-point.

#### Bézier curve

A Bézier curve is defined by stating two pivot points in data blocks of type 007 and 008, and an end-point in a data block of type 002. The Bézier curve is drawn from the curve's current end-point (named 'S') to the Bézier curve's end-point (named 'E'), influenced by the two pivot points (named P1 and P2). The Bézier curve's end-point then becomes the curve's current end-point.

The points P of such a Bézier curve are calculated using the following equation:

$$P = (1-t)^3 \times S + 3t(1-t)^2 \times P1 + 3t^2(1-t) \times P2 + t^3 \times E \quad (0 \leq t \leq 1)$$

#### Circular arc

A circular arc is defined by two data blocks. The first is of type 009 and contains the end-point of the arc, the second is of type 102 and contains both its orientation and its curvature. The arc is drawn from the curve's current end-point (named 'S') to the arc's end-point (named 'E'), influenced by its orientation and curvature. The arc's end-point then becomes the curve's current end-point.

The value *Orientation* determines whether to draw the arc in clockwise direction (*Orientation* < 0) or in counter-clockwise direction (*Orientation* >= 0).

The value *Curvature* determines the radius of the arc in indirect manner. It can be handled in two ways. In geometrical view, the absolute value of *Curvature* is  $1/(2 \tan(\beta/2))$ , where  $\beta$  is the arc-angle of the circular arc. The sign of *Curvature* determines, which of the two possible arcs to use.

In practical use, this definition is not very handy. Instead, *Curvature* should be used to perform a simple vector calculation to obtain the arc's defining center-point M:

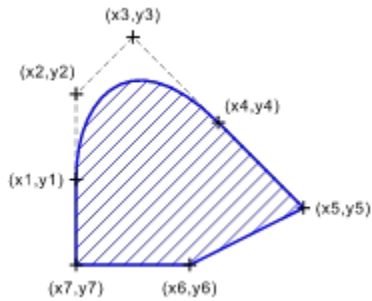
$$\begin{aligned} x(M) &= 0.5 * ( x(S) + x(E) ) - Curvature * ( y(E) - y(S) ) \\ y(M) &= 0.5 * ( y(S) + y(E) ) + Curvature * ( x(E) - x(S) ) \end{aligned}$$

After calculating the center-point, all points required to draw a standard circular arc are known. The allowed value range of *Curvature* is  $\pm 1e100$ .

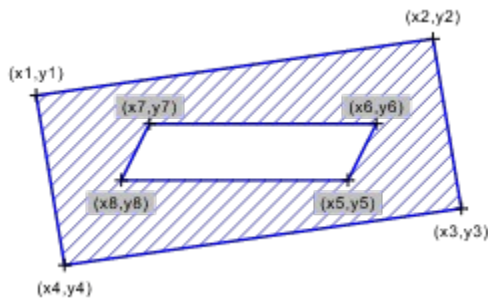
Each curve ends if either a new curve is started by a new start-point (in a data block of type 001) or the complete surface is ended (by a data block of type 999). Each curve has to be closed, i.e. its start-point

and end-point have to be connected with a line.

If a surface consists of only one curve simply the inside area of that curve will be filled:



If a surface consists of multiple curves they will be filled alternately, i.e. only areas overlapped by an *odd* number of curve's insides will be filled. If, e.g., a small round curve lies inside a large one, this small round curve will be transparent, as its inside is overlapped by two curves (*even* number!):



If a surface is drawn with a non-solid line pattern, this line pattern has to be continued during each curve. A surface may contain up to 1000 nested curves made of up to 2000 data blocks in total.

## Object 15 "Ellipse"



# Contents

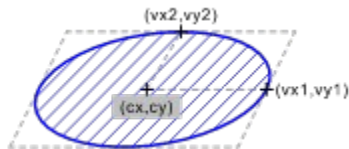
### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 006</u> ( vx1, vy1 )	- Vector end-point 1
<u>Data Block 006</u> ( vx2, vy2 )	- Vector end-point 2

The point (cx,cy) determines the ellipse's center-point, the points (vx1,vy1) and (vx2,vy2) are end-points of vectors that define the ellipse. Having the center-point C and the two vectors V1 and V2 (vector from the center-point to the respective vector end-point), the points P on such an ellipse are determined by the following equation:

$$P = C + V1 \times \sin(\beta) + V2 \times \cos(\beta) \quad (0 \leq \beta < 2\pi)$$

The resulting ellipse can be any type of ellipse, rectangular as well as arbitrary.





## Object 16 "Elliptical Arc"



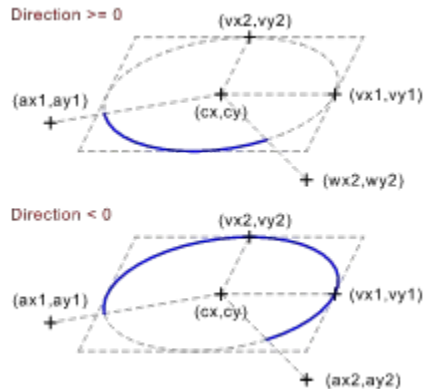
# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 006</u> ( vx1, vy1 )	- Vector end-point 1
<u>Data Block 006</u> ( vx2, vy2 )	- Vector end-point 2
<u>Data Block 005</u> ( ax1, ay1 )	- Start-angle definition
<u>Data Block 005</u> ( ax2, ay2 )	- End-angle definition
<u>Data Block 101</u> ( Orientation )	- Arc direction

The point (cx,cy) determines the ellipse's center-point, the points (vx1,vy1) and (vx2,vy2) are end-points of vectors that define the ellipse. The points (ax1,ay1) and (ax2,ay2), in relation to the ellipse's center-point (cx,cy), determine the start- and end-angle of the arc. If *Orientation*  $\geq 0$ , the arc is drawn counter-clockwise from the start-angle to the end-angle. If *Orientation*  $< 0$ , the arc is drawn clockwise.

An elliptical arc is an "open" object, i.e. it has a non-closed outline and cannot be filled.



## Object 17 "Elliptical Sector"

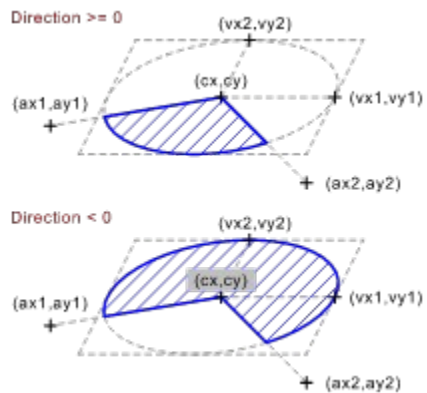


# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 006</u> ( vx1, vy1 )	- Vector end-point 1
<u>Data Block 006</u> ( vx2, vy2 )	- Vector end-point 2
<u>Data Block 005</u> ( ax1, ay1 )	- Start-angle definition
<u>Data Block 005</u> ( ax2, ay2 )	- End-angle definition
<u>Data Block 101</u> ( Orientation )	- Arc direction

The point (cx,cy) determines the ellipse's center-point, the points (vx1,vy1) and (vx2,vy2) are end-points of vectors that define the ellipse. The points (ax1,ay1) and (ax2,ay2), in relation to the ellipse's center-point (cx,cy), determine the start- and end-angle of the sector. If *Orientation*  $\geq 0$ , the sector is drawn counter-clockwise from the start-angle to the end-angle. If *Orientation*  $< 0$ , the sector is drawn clockwise.



# Object 18 "Elliptical Segment"

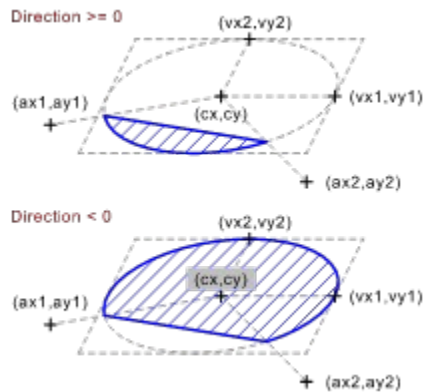


## Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 006</u> ( vx1, vy1 )	- Vector end-point 1
<u>Data Block 006</u> ( vx2, vy2 )	- Vector end-point 2
<u>Data Block 005</u> ( ax1, ay1 )	- Start-angle definition
<u>Data Block 005</u> ( ax2, ay2 )	- End-angle definition
<u>Data Block 101</u> ( Orientation )	- Arc direction

The point (cx,cy) determines the ellipse's center-point, the points (vx1,vy1) and (vx2,vy2) are end-points of vectors that define the ellipse. The points (ax1,ay1) and (ax2,ay2), in relation to the ellipse's center-point (cx,cy), determine the start- and end-angle of the segment. If *Orientation*  $\geq 0$ , the segment is drawn counter-clockwise from the start-angle to the end-angle. If *Orientation*  $< 0$ , the segment is drawn clockwise.



## Object 20 "Dimension Line Straight"



# Contents

### Data Block Sequence

<u>Data Block 001</u> ( x1, y1 )	- Start-point
<u>Data Block 002</u> ( x2, y2 )	- End-point
<u>Data Block 220</u> ( ... )	- Dimension line parameters

The points (x1,y1) and (x2,y2) determine the start-point and end-point of the dimension. These points are always the arrow's end-points, but not necessarily the line's end-points. If the arrows are rotated, the dimension line will overlap the end-points.

A dimension line is an "open" object, i.e. it has a non-closed outline and cannot be filled. Nevertheless, the dimension arrows will be filled if required. This is independent from the object's properties!

## Object 21 "Dimension Line Curved"



# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 004</u> ( rx, ry )	- Radius definition
<u>Data Block 005</u> ( ax1, ay1 )	- Start-angle definition
<u>Data Block 005</u> ( ax2, ay2 )	- End-angle definition
<u>Data Block 101</u> ( Orientation )	- Arc direction
<u>Data Block 220</u> ( ... )	- Dimension line parameters

The point (cx,cy) determines the circle's center-point, the point (rx,ry) is a point on the circle's outline and thus defines the radius. The points (ax1,ay1) and (ax2,ay2), in relation to the circle's center-point (cx,cy), determine the start- and end-angle of the arc. If *Orientation*  $\geq 0$ , the arc is drawn counter-clockwise from the start-angle to the end-angle. If *Orientation*  $< 0$ , the arc is drawn clockwise.

The two angles determine the arrow's end-points, but not necessarily the line's end-points. If the arrows are rotated, the dimension line will overlap the given angles.

A dimension line is an "open" object, i.e. it has a non-closed outline and cannot be filled. Nevertheless, the dimension arrows will be filled if required. This is independent from the object's properties!

## Object 25 "Dimension Distance"



# Contents

### Data Block Sequence

<u>Data Block 001</u> ( x1, y1 )	- Start-point
<u>Data Block 002</u> ( x2, y2 )	- End-point
<u>Data Block 000</u> ( dx1, dy1 )	- First dimension line definition
<u>Data Block 000</u> ( dx2, dy2 )	- Second dimension line definition
<u>Data Block 000</u> ( dx3, dy3 )	- Third dimension line definition
<u>Data Block 000</u> ( nx1, ny1 )	- First dimension number definition
<u>Data Block 000</u> ( nx2, ny2 )	- Second dimension number definition
<u>Data Block 110</u> ( PreText )	- Text in front of number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( NumText )	- Dimension itself ( <i>ElemCount</i> = -250)
<u>Data Block 110</u> ( PostText )	- Text behind the number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance1 )	- Upper tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance2 )	- Lower tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 225</u> ( ... )	- Distance dimension parameters

The points (x1,y1) and (x2,y2) determine the distance to be measured. The point (dx1,dy1) determines the dimension's direction and thus the dimension line's direction. The point (dx2,dy2) defines the distance between the dimension line and the line to be measured. Finally, the point (dx3,dy3) determined the end-point of the dimension line if uses a short form. The points (nx1,ny1) and (nx2,ny2) determine the position and, if applicable, the orientation of the dimension number.

The following texts make up the dimension texts to be displayed, where *PreText*, *NumText* and *PostText* will be connected to one text ("Main dimension text") without separating characters.

The name "Dimension Distance" does not mean that this object can only be used to dimension distances, but that the reference is given by the distance of two points. As a result, this dimension can also be used to dimension a diameter or radius in a plain view.



# Contents

### Specific Usage of Data Block 225

#### *LineOrientation*

[*long*] The value *LineOrientation* determines the dimension's orientation, i.e. the direction into which the dimension is to be calculated as well as the dimension line's orientation. This allows e.g. to dimension a diagonal line vertically or horizontally. Possible values are:

- 0x0000 The dimension (and thus the dimension line) are parallel to the line stored in the object.
- 0x0001 The dimension (and thus the dimension line) are parallel to a line passing through the line's start-point (x1,y1) and the first dimension line definition point (dx1,dy1).
- 0x0002 Identical to 0x0001. This setting is used inside the application to create a horizontal dimension. After creating the object, it behaves like with value 0x0001.
- 0x0003 Identical to 0x0001. This setting is used inside the application to create a vertical dimension. After creating the object, it behaves like with value 0x0001.

#### *LineType*

[*long*] The value *LineType* determines whether the dimension line has full size with two arrows or only a partial length with one arrow. Additionally, it determines whether the dimension extension lines shall be perpendicular to the dimension line or not. Possible values are:

- 0x0000 Full-length dimension line with two arrows. The dimension line is placed so that probable dimension extension lines would be perpendicular to it (even if they are not visible).

- 0x0001 Full-length dimension line with two arrows. The dimension line starts at a line passing through the points (x2,y2) and (dx2,dy2) and ends at a line parallel to it, passing through the point (x1,y1).
- 0x0002 Partial-length dimension line with one arrow. A probable dimension extension line would be perpendicular to the dimension, starting at the end-point being closer to (dx2,dy2). At this extension line starts the dimension line, having an arrow and extension line according to *ArrowStartForm*, *ArrowStartMode* and *ExtStartDisplay*. The dimension line ends at a line parallel to the dimension extension line, passing through the point (dx3,dy3).

*LineDistMode*

[long] The value *LineDistMode* determines how the dimension line's distance to the dimension is calculated. The distance is always measured perpendicularly to the dimension line, even if it is rotated (see *LineOrientation*). Possible values are:

- 0x0000 The dimension line passes through the point (dx2,dy2).
- 0x0001 The dimension line has exactly the distance stated in *LineDistance* to the nearest end of the dimension. The position of (dx2,dy2) determines on what side it lies.
- 0x0002 The dimension line's distance to the nearest end of the dimension is a multiple of the value stated in *LineDistance*, while being as close as possible to (dx2,dy2).

## Contents **Example**

```

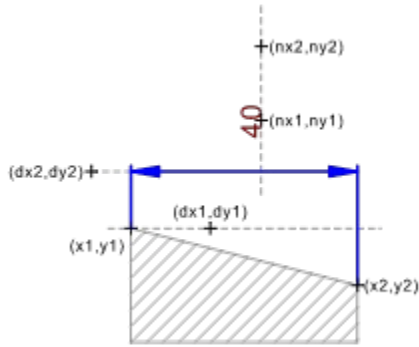
0,225,0,0, |BlockType|
0,0,400,"DINLQ", |TextFont|
1, |TextXProperty.Flag|
1,0,0.0/0.0/0.0,0.0/0.0/0.0,0.0,0,0, |...|
0, |TextXProperty.Layer|
5.0,3.5, |TextSize1,TextSize2|
0,4,1, |CharDistance,TabDistance,TextMode|
2,0,0,0,1, |Accuracy,Refresh,Centered,Tight,Rotate|
1,0,1,0, |ArrowStart...,ArrowEnd...|
1,1,1, |ExtStart/EndDisplay,LineDisplay|
1,0,0,10.0,0.0, |Orientation,Type,DistMode,Distance|
0; |System|

```

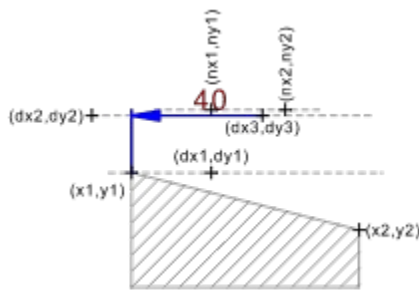
This distance dimension will be lettered with the font "DINLQ", using pen 1. The main dimension has a text size of 5 mm, the tolerances have a text size of 3.5 mm. The dimension number can be placed anywhere, even be rotated, and it will not be updated automatically.

The dimension and thus the dimension line does not necessarily lie parallel to the dimension. The dimension line is full-length and its distance to the dimension is arbitrary. Both dimension extension lines will be visible, they are perpendicular to the dimension line. At both ends of the dimension line, a filled, unrotated, triangular arrow will be drawn.

The dimension will be calculated according to the settings in the coordinate system "00: \*Standard".



The image above shows a distance dimension created using the settings described above. The points  $(x1,y1)$  and  $(x2,y2)$  define the distance to be dimensioned. The dimension shall be calculated horizontally, so the first dimension line definition point  $(dx1,dy1)$  is placed besides the dimension's start-point  $(x1,y1)$ . The dimension line is parallel to a line passing through  $(dx1,dy1)$  and  $(x1,y1)$ , and passes through the second dimension line definition point  $(dx2,dy2)$ . It is full-length and has an arrow at each end. As *LineStyle* is set to 0x0000, the dimension extension lines are perpendicular. The dimension number was placed by setting the point  $(nx1,ny1)$  and rotated by  $90^\circ$  by placing the point  $(nx2,ny2)$  above it.



The image above shows another distance dimension. It was created using settings similar to the settings described above but *LineStyle* was set to 0x0002, so the dimension line has only partial length and a single arrow. The dimension line starts at the point  $(dx1,dy1)$  and ends at the point  $(dx3,dy3)$ . The dimension number was placed non-rotated by placing  $(nx2,ny2)$  right to  $(nx1,ny1)$ .



## Object 26 "Dimension Radius"



# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 004</u> ( rx, ry )	- Radius definition
<u>Data Block 000</u> ( dx1, dy1 )	- First dimension line definition
<u>Data Block 000</u> ( dx2, dy2 )	- Second dimension line definition
<u>Data Block 000</u> ( nx1, ny1 )	- First dimension number definition
<u>Data Block 000</u> ( nx2, ny2 )	- Second dimension number definition
<u>Data Block 110</u> ( PreText )	- Text in front of number ( <i>ElemCount</i> =250)
<u>Data Block 110</u> ( NumText )	- Dimension itself ( <i>ElemCount</i> = -250)
<u>Data Block 110</u> ( PostText )	- Text behind the number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance1 )	- Upper tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance2 )	- Lower tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 225</u> ( ... )	- Radius dimension parameters

The point (cx,cy) is the center-point of the circle element whose radius is to be measured, the point (rx,ry) determines the radius. The point (dx1,dy1) determines the dimension's direction and thus the dimension line's direction, the point (dx2,dy2) defines the distance between the dimension line and the radius to be measured. The points (nx1,ny1) and (nx2,ny2) determine the position and, if applicable, the orientation of the dimension number.

The following texts make up the dimension texts to be displayed, where *PreText*, *NumText* and *PostText* will be connected to one text ("Main dimension text") without separating characters.

The name "Dimension Radius" does not mean that this object can only be used to dimension a radius, but that the reference is given by the radius of a circle element.



# Contents

### Specific Usage of Data Block 225

#### *LineOrientation*

[*long*] The value *LineOrientation* determines the dimension's orientation, i.e. the direction into which the dimension is to be calculated as well as the dimension line's orientation. This allows to dimension a radius in a specific direction. Possible values are:

0x0000	The dimension (and thus the dimension line) are parallel to the radius stored in the object.
0x0001	The dimension (and thus the dimension line) are parallel to a line passing through the center-point (cx,cy) and the first dimension line definition point (dx1,dy1).
0x0002	Identical to 0x0001. This setting is used inside the application to create a horizontal dimension. After creating the object, it behaves like with value 0x0001.
0x0003	Identical to 0x0001. This setting is used inside the application to create a vertical dimension. After creating the object, it behaves like with value 0x0001.

#### *LineType*

[*long*] The value *LineType* determines whether the dimension line has full size with two arrows or only a partial length with one arrow. Additionally, it determines whether the dimension extension lines shall be perpendicular to the dimension line or not. Possible values are:

0x0000	Full-length dimension line with two arrows. The dimension line is placed so that probable dimension extension lines would be perpendicular to it (even if they are not visible). Seen from the circle's center, it starts at the same side that (dx1,dy1) lies at.
0x0001	Full-length dimension line with two arrows. The dimension line starts at a line passing

through the radius's start-point and (dx2,dy2) and ends at a line parallel to it, passing through the point (cx,cy). Probable dimension extension lines are not necessarily perpendicular to the dimension line.

0x0002 Partial-length dimension line with one arrow. A probable dimension extension line would be perpendicular to the dimension, starting at the radius' end-point being closer to (dx1,dy1). At this extension line starts the dimension line, having an arrow and extension line according to *ArrowStartForm*, *ArrowStartMode* and *ExtStartDisplay*. The dimension line ends at a line parallel to the dimension extension line, passing through the point (dx2,dy2).

### *LineDistMode*

[*long*] The value *LineDistMode* determines how the dimension line's distance to the dimension is calculated. The distance is always measured perpendicularly to the dimension line, even if it is rotated (see *LineOrientation*). Possible values are:

0x0000 The dimension line passes through the point (dx2,dy2).

0x0001 The dimension line has exactly the distance stated in *LineDistance* to the radius or the circle. The position of (dx2,dy2) determines on what side it lies.

0x0002 The dimension line's distance to the radius or circle is a multiple of the value stated in *LineDistance*, while being as close as possible to (dx2,dy2).

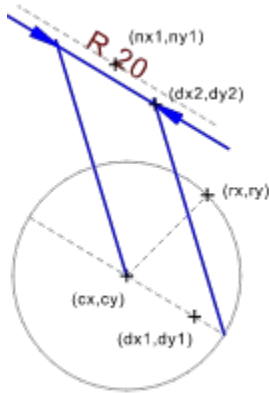
## Contents **Example**

```
0,225,0,0, |BlockType|
0,0,400,"DINDRAFT", |TextFont|
1, |TextXProperty.Flag|
1,0,0.0/0.0/0.0,0.0/0.0/0.0,0.0,0,0, | ... |
0, |TextXProperty.Layer|
5.0,3.5, |TextSize1,TextSize2|
0,4,1, |CharDistance,TabDistance,TextMode|
2,0,0,0,0, |Accuracy,Refresh,Centered,Tight,Rotate|
1,0,1,0, |ArrowStart...,ArrowEnd...|
1,1,1, |
ExtStartDisplay,ExtEndDisplay,LineDisplay|
1,1,0,10.0,0.0, |Orientation,Type,DistMode,Distance|
0; |System|
```

This radius dimension will be lettered with the font "DINDRAFT", using pen 1. The main dimension has a text size of 5 mm, the tolerances have a text size of 3.5 mm. The dimension number can be placed anywhere, but not rotated, and it will not be updated automatically.

The dimension and thus the dimension line does not necessarily lie parallel to the dimension. The dimension line is full-length and its distance to the dimension is arbitrary. Both dimension extension lines will be visible, they are not perpendicular to the dimension line. At both ends of the dimension line, a filled, unrotated, triangular arrow will be drawn.

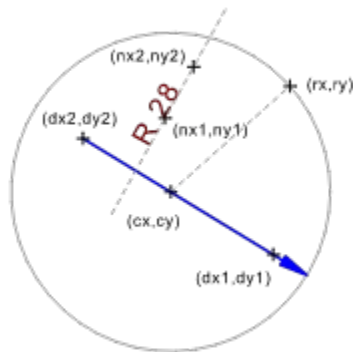
The dimension will be calculated according to the settings in the coordinate system "00: \*Standard".



The image above shows a radius dimension created using the settings described above. The points  $(cx, cy)$  and  $(rx, ry)$  define the radius to be dimensioned. The dimension's orientation is determined by the first dimension line definition point  $(dx1, dy1)$ .

The dimension line is parallel to a line passing through  $(dx1, dy1)$  and  $(cx, cy)$ , and passes through the second dimension line definition point  $(dx2, dy2)$ . It is full-length and has an arrow at each end. As *LineStyle* is set to 0x0001 the dimension extension lines are not perpendicular, so the dimension line starts exactly at  $(dx2, dy2)$ .

The dimension number was placed by setting the point  $(nx1, ny1)$  and is not rotated because *NumRotate* is set to 0x0000. The point  $(nx2, ny2)$  is not used. By setting *PreText* to the text "R " the dimension number was equipped with a preceding radius symbol.



The image above shows another radius dimension. It was created using settings similar to the settings described above but *LineStyle* was set to 0x0002, so the dimension line has only partial length and a single arrow.

The dimension line starts at the right as  $(dx1, dy1)$  lies right to the circle's center-point  $(cx, cy)$  and ends at the point  $(dx2, dy2)$ .

Additionally, *NumRotate* was set to 0x0001, i.e. the dimension number is rotated. Its position and orientation is determined by the points  $(nx1, ny1)$  and  $(nx2, ny2)$ . By setting *PreText* to the text "R " the dimension number was equipped with a preceding radius symbol.

## Object 27 "Dimension Diameter"



# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 004</u> ( rx, ry )	- Radius definition
<u>Data Block 000</u> ( dx1, dy1 )	- First dimension line definition
<u>Data Block 000</u> ( dx2, dy2 )	- Second dimension line definition
<u>Data Block 000</u> ( nx1, ny1 )	- First dimension number definition
<u>Data Block 000</u> ( nx2, ny2 )	- Second dimension number definition
<u>Data Block 110</u> ( PreText )	- Text in front of number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( NumText )	- Dimension itself ( <i>ElemCount</i> = -250)
<u>Data Block 110</u> ( PostText )	- Text behind the number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance1 )	- Upper tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance2 )	- Lower tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 225</u> ( ... )	- Diameter dimension parameters

The point (cx,cy) is the center-point of the circle element whose diameter is to be measured, the point (rx,ry) determines the radius. The point (dx1,dy1) determines the dimension's direction and thus the dimension line's direction, the point (dx2,dy2) defines the distance between the dimension line and the diameter to be measured. The points (nx1,ny1) and (nx2,ny2) determine the position and, if applicable, the orientation of the dimension number.

The following texts make up the dimension texts to be displayed, where *PreText*, *NumText* and *PostText* will be connected to one text ("Main dimension text") without separating characters.

The name "Dimension Diameter" does not mean that this object can only be used to dimension a diameter, but that the reference is given by the diameter of a circle element.



# Contents

### Specific Usage of Data Block 225

#### *LineOrientation*

[*long*] The value *LineOrientation* determines the dimension's orientation, i.e. the direction into which the dimension is to be calculated as well as the dimension line's orientation. This allows to dimension a diameter in a specific direction. Possible values are:

- 0x0000 The dimension (and thus the dimension line) are parallel to the diameter stored in the object.
- 0x0001 The dimension (and thus the dimension line) are parallel to a line passing through the center-point (cx,cy) and the first dimension line definition point (dx1,dy1).
- 0x0002 Identical to 0x0001. This setting is used inside the application to create a horizontal dimension. After creating the object, it behaves like with value 0x0001.
- 0x0003 Identical to 0x0001. This setting is used inside the application to create a vertical dimension. After creating the object, it behaves like with value 0x0001.

#### *LineType*

[*long*] The value *LineType* determines whether the dimension line has full size with two arrows or only a partial length with one arrow. Additionally, it determines whether the dimension extension lines shall be perpendicular to the dimension line or not. Possible values are:

- 0x0000 Full-length dimension line with two arrows. The dimension line is placed so that probable dimension extension lines would be perpendicular to it (even if they are not visible). Seen from the circle's center, it starts at the same side that (dx1,dy1) lies at.

- 0x0001 Full-length dimension line with two arrows. The dimension line starts at a line passing through the diameter's start-point and (dx2,dy2) and ends at a line parallel to it, passing through the corresponding diameter end-point at the other side of the circle. Probable dimension extension lines are not necessarily perpendicular to the dimension line.
- 0x0002 Partial-length dimension line with one arrow. A probable dimension extension line would be perpendicular to the dimension, starting at the diameter's end-point being closer to (dx1,dy1). At this extension line starts the dimension line, having an arrow and extension line according to *ArrowStartForm*, *ArrowStartMode* and *ExtStartDisplay*. The dimension line ends at a line parallel to the dimension extension line, passing through the point (dx2,dy2).

*LineDistMode*

[long] The value *LineDistMode* determines how the dimension line's distance to the dimension is calculated. The distance is always measured perpendicularly to the dimension line, even if it is rotated (see *LineOrientation*). Possible values are:

- 0x0000 The dimension line passes through the point (dx2,dy2).
- 0x0001 The dimension line has exactly the distance stated in *LineDistance* to the radius or the circle. The position of (dx2,dy2) determines on what side it lies.
- 0x0002 The dimension line's distance to the radius or circle is a multiple of the value stated in *LineDistance*, while being as close as possible to (dx2,dy2).

## Contents **Example**

```

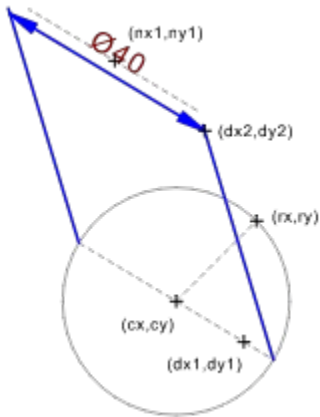
0,225,0,0, |BlockType|
0,0,400,"DINDRAFT", |TextFont|
1, |TextXProperty.Flag|
1,0,0.0/0.0/0.0,0.0/0.0/0.0,0.0,0,0, |...|
0, |TextXProperty.Layer|
5.0,3.5, |TextSize1,TextSize2|
0,4,1, |CharDistance,TabDistance,TextMode|
2,0,0,0,0, |Accuracy,Refresh,Centered,Tight,Rotate|
1,0,1,0, |ArrowStart...,ArrowEnd...|
1,1,1, |
ExtStartDisplay,ExtEndDisplay,LineDisplay|
1,1,0,10.0,0.0, |Orientation,Type,DistMode,Distance|
0; |System|

```

This radius dimension will be lettered with the font "DINDRAFT", using pen 1. The main dimension has a text size of 5 mm, the tolerances have a text size of 3.5 mm. The dimension number can be placed anywhere, but not rotated, and it will not be updated automatically.

The dimension and thus the dimension line does not necessarily lie parallel to the dimension. The dimension line is full-length and its distance to the dimension is arbitrary. Both dimension extension lines will be visible, they are not perpendicular to the dimension line. At both ends of the dimension line, a filled, unrotated, triangular arrow will be drawn.

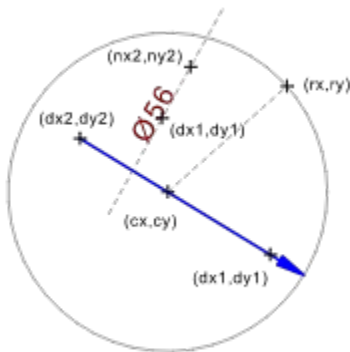
The dimension will be calculated according to the settings in the coordinate system "00: \*Standard".



The image above shows a diameter dimension created using the settings described above. The points  $(cx, cy)$  and  $(rx, ry)$  define the diameter to be dimensioned. The dimension's orientation is determined by the first dimension line definition point  $(dx1, dy1)$ .

The dimension line is parallel to a line passing through  $(dx1, dy1)$  and  $(cx, cy)$ , and passes through the second dimension line definition point  $(dx2, dy2)$ . It is full-length and has an arrow at each end. As *LineType* is set to 0x0001 the dimension extension lines are not perpendicular, so the dimension line starts exactly at  $(dx2, dy2)$ .

The dimension number was placed by setting the point  $(nx1, ny1)$  and is not rotated because *NumRotate* is set to 0x0000. The point  $(nx2, ny2)$  is not used. By setting *PreText* to the text "Ø" the dimension number was equipped with a preceding diameter symbol.



The image above shows another diameter dimension. It was created using settings similar to the settings described above, but *LineType* was set to 0x0002, so the dimension line has only partial length and a single arrow.

The dimension line starts at the right as  $(dx1, dy1)$  lies right to the circle's center-point  $(cx, cy)$  and ends at the point  $(dx2, dy2)$ .

Additionally, *NumRotate* was set to 0x0001, i.e. the dimension number is rotated. Its position and orientation is determined by the points  $(nx1, ny1)$  and  $(nx2, ny2)$ . By setting *PreText* to the text "Ø" the dimension number was equipped with a preceding diameter symbol.

## Object 28 "Dimension Angle"



# Contents

### Data Block Sequence

<u>Data Block 001</u> ( x1, y1 )	- Start-point first edge
<u>Data Block 002</u> ( x2, y2 )	- End-point first edge
<u>Data Block 001</u> ( x3, y3 )	- Start-point second edge
<u>Data Block 002</u> ( x4, y4 )	- End-point second edge
<u>Data Block 000</u> ( dx2, dy2 )	- Dimension line definition
<u>Data Block 000</u> ( nx1, ny1 )	- First dimension number definition
<u>Data Block 000</u> ( nx2, ny2 )	- Second dimension number definition
<u>Data Block 110</u> ( PreText )	- Text in front of number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( NumText )	- Dimension itself ( <i>ElemCount</i> = -250)
<u>Data Block 110</u> ( PostText )	- Text behind the number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance1 )	- Upper tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance2 )	- Lower tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 225</u> ( ... )	- Angle dimension parameters

The points (x1,y1) and (x2,y2) define the first edge of the angle, the points (x3,y3) and (x4,y4) define the second edge. The point (dx2,dy2) defines the position of the dimension line, including which partial angle shall be measured. The dimension line is a circular arc and has its center-point in the virtual (or real) intersection point of the two edges. The points (nx1,ny1) and (nx2,ny2) determine the position and, if applicable, the orientation of the dimension number.

The following texts make up the dimension texts to be displayed, where *PreText*, *NumText* and *PostText* will be connected to one text ("Main dimension text") without separating characters.



# Contents

### Specific Usage of Data Block 225

*LineOrientation*,

*LineType*

[*long*] Unused.

*LineDistMode*

[*long*] The value *LineDistMode* determines how the dimension line's distance to the dimension is calculated. The distance is always measured relative to the one of the edge's end-points that is farthest away from the virtual (or real) intersection point of the two edges. Possible values are:

0x0000 The dimension line passes through the point (dx2,dy2).

0x0001 The dimension line has exactly the distance stated in *LineDistance* to the edge's end-point. The position of (dx2,dy2) determines what partial angle to use.

0x0002 The dimension line's distance is a multiple of the value stated in *LineDistance*, while being as close as possible to (dx2,dy2). The position of (dx2,dy2) determines what partial angle to use.



# Contents

### Example

```
0,225,0,0, |BlockType|
0,0,400,"DINLQ", |TextFont|
1, |TextXProperty.Flag|
1,0,0.0/0.0/0.0,0.0/0.0/0.0,0.0,0, | ... |
0, |TextXProperty.Layer|
```

```

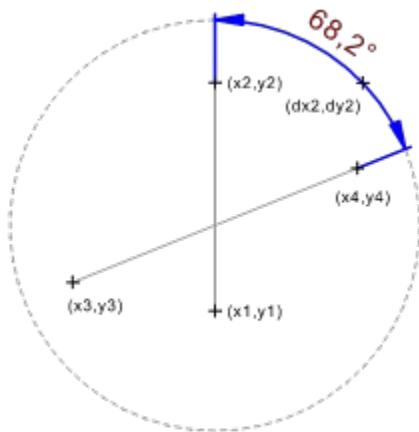
5.0,3.5,                                     |TextSize1,TextSize2|
0,4,1,                                       |CharDistance,TabDistance,TextMode|
2,1,1,1,0,                                   |Accuracy,Refresh,Centered,Tight,Rotate|
1,0,1,0,                                     |ArrowStart...,ArrowEnd...|
1,1,1,                                       |
ExtStartDisplay,ExtEndDisplay,LineDisplay|
1,1,1,10.0,0.0,                             |Orientation,Type,DistMode,Distance|
0;                                           |System|

```

This angle dimension will be lettered with the font "DINLQ", using pen 1. The main dimension has a text size of 5 mm, the tolerances have a text size of 3.5 mm. The dimension number will automatically placed centered and tight to the dimension line, it will be updated automatically.

The dimension line has a fixed distance of 10 mm to the edge's end-point. Both dimension extension lines will be visible, they are drawn radially from an edge's end-point to the dimension line. At both ends of the dimension line, a filled, unrotated, triangular arrow will be drawn.

The dimension will be calculated according to the settings in the coordinate system "00: \*Standard".

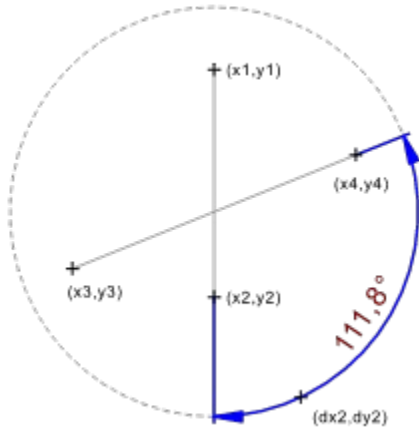


The image above shows an angle dimension created using the settings described above. The points  $(x1,y1)$ ,  $(x2,y2)$ ,  $(x3,y3)$  and  $(x4,y4)$  define the two edges of the angle. The dimension line has a distance of 10 mm to the point  $(x4,y4)$ .

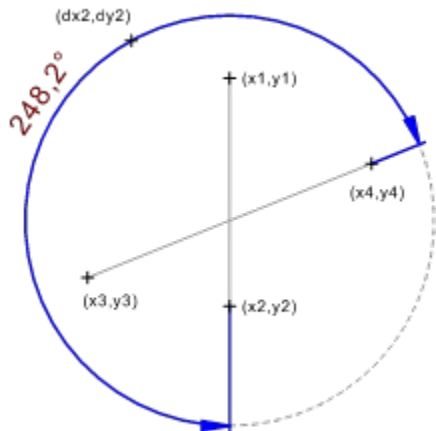
The dimension number was centered and set tight automatically, so the point  $(nx1,ny1)$  is irrelevant. As the number is not rotated the point  $(nx2,ny2)$  is not used.

The orientation of the two edges together with the dimension line definition point  $(dx1,dy1)$  determines which of the possible eight angles is to be measured. As  $(x2,y2)$  is above  $(x1,y1)$ , the dimension starts above the virtual intersection point. As  $(x4,y4)$  is right to  $(x3,y3)$ , the dimension ends right to the virtual intersection point. This constellation still allows two angles - one running clockwise from top to right, one running counter-clockwise. Now the dimension line definition point  $(dx2,dy2)$  determines which one to choose. In this case it is the angle running clockwise, i.e. the smaller angle at the upper right.





The image above shows a similar angle dimension, but the end-points of the first edge are placed vice versa. As  $(x_2,y_2)$  now lies below  $(x_1,y_1)$ , the dimension starts below the virtual intersection point. According to the dimension line definition point  $(dx_2,dy_2)$ , the lower right angle is to be measured.



The image above shows another similar angle dimension but now dimension line definition point  $(dx_2,dy_2)$  was placed somewhere else, so now the upper left angle is measured.

**Note:** The two edges of the angle do not have to intersect directly even though they do in all examples shown here. In any case a "virtual" intersection point is calculated, i.e. the intersection of the edges extended to endless lines. If both edges are parallel no dimension is displayed!

## Object 29 "Dimension Arc Length"



# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 004</u> ( rx, ry )	- Radius definition
<u>Data Block 005</u> ( ax1, ay1 )	- Start-angle definition
<u>Data Block 005</u> ( ax2, ay2 )	- End-angle definition
<u>Data Block 101</u> ( Orientation )	- Arc direction
<u>Data Block 000</u> ( dx2, dy2 )	- Dimension line definition
<u>Data Block 000</u> ( nx1, ny1 )	- First dimension number definition
<u>Data Block 000</u> ( nx2, ny2 )	- Second dimension number definition
<u>Data Block 110</u> ( PreText )	- Text in front of number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( NumText )	- Dimension itself ( <i>ElemCount</i> = -250)
<u>Data Block 110</u> ( PostText )	- Text behind the number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance1 )	- Upper tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance2 )	- Lower tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 225</u> ( ... )	- Arc length dimension parameters

The point (cx,cy) determines the circle's center-point, the point (rx,ry) is a point on the circle's outline and thus defines the radius. The points (ax1,ay1) and (ax2,ay2), in relation to the circle's center-point (cx,cy), determine the start- and end-angle of the arc. If *Orientation*  $\geq 0$ , the arc is drawn counter-clockwise from the start-angle to the end-angle. If *Orientation*  $< 0$ , the arc is drawn clockwise.

The point (dx2,dy2) determines the radius of the dimension line. The dimension line is a circular arc and has the same center-point as the circle element to be measured. The points (nx1,ny1) and (nx2,ny2) determine the position and, if applicable, the orientation of the dimension number.

The following texts make up the dimension texts to be displayed, where *PreText*, *NumText* and *PostText* will be connected to one text ("Main dimension text") without separating characters.



# Contents

### Specific Usage of Data Block 225

*LineOrientation*,

*LineType*

[*long*] Unused.

*LineDistMode*

[*long*] The value *LineDistMode* determines how the dimension line's distance to the dimension is calculated. The distance is the difference of the circle's and the dimension line's radius. Possible values are:

0x0000 The dimension line passes through the point (dx2,dy2).

0x0001 The dimension line has exactly the distance stated in *LineDistance*. The position of (dx2,dy2) determines on what side it lies.

0x0002 The dimension line's distance is a multiple of the value stated in *LineDistance*, while being as close as possible to (dx2,dy2).



# Contents

### Example

```
0,225,0,0,  
0,0,400,"DINDRAFT",
```

```
|BlockType|  
|TextFont|
```

```

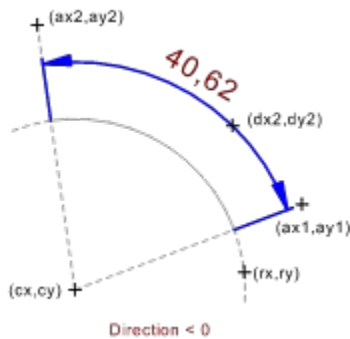
1, |TextXProperty.Flag|
1,0,0.0/0.0/0.0,0.0/0.0/0.0,0.0,0, | ... |
0, |TextXProperty.Layer|
5.0,3.5, |TextSize1,TextSize2|
0,4,1, |CharDistance,TabDistance,TextMode|
2,1,1,1,0, |Accuracy,Refresh,Centered,Tight,Rotate|
1,0,1,0, |ArrowStart...,ArrowEnd...|
1,1,1, |
ExtStartDisplay,ExtEndDisplay,LineDisplay|
1,1,0,10.0,0.0, |Orientation,Type,DistMode,Distance|
0; |System|

```

This arc length dimension will be lettered with the font "DINLQ", using pen 1. The main dimension has a text size of 5 mm, the tolerances have a text size of 3.5 mm. The dimension number will automatically placed centered and tight to the dimension line, it will be updated automatically.

The dimension line has a fixed distance of 10 mm to the circular arc. Both dimension extension lines will be visible, they are either drawn radially from an edge's end-point to the dimension line or parallel (this depends on settings in the application). At both ends of the dimension line, a filled, unrotated, triangular arrow will be drawn.

The dimension will be calculated according to the settings in the coordinate system "00: \*Standard".



The image above shows an arc length dimension created using the settings described above. The points (cx,cy), (rx,ry), (ax1,ay1) and (ax2,ay2) define the circular arc to be measured. The dimension line has a distance of 10 mm, the point (dx2,dy2) determines its position.

The dimension number was centered and set tight automatically, so the point (nx1,ny1) is irrelevant. As the number is not rotated the point (nx2,ny2) is not used.

## Object 30 "Dimension Coordinate"



# Contents

### Data Block Sequence

<u>Data Block 000</u> ( x1, y1 )	- Coordinate to be dimensioned
<u>Data Block 000</u> ( nx1, ny1 )	- First dimension number definition
<u>Data Block 000</u> ( nx2, ny2 )	- Second dimension number definition
<u>Data Block 110</u> ( PreText )	- Text in front of coordinates ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Coord1Text )	- X-coordinate ( <i>ElemCount</i> = -250)
<u>Data Block 110</u> ( MiddleText )	- Text between coordinates ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Coord2Text )	- Y-coordinate ( <i>ElemCount</i> = -250)
<u>Data Block 110</u> ( PostText )	- Text behind coordinates ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance1 )	- Upper tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance2 )	- Lower tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 230</u> ( ... )	- Coordinate dimension parameters

The point (x1,y1) is the point whose coordinates shall be measured. The points (nx1,ny1) and (nx2,ny2) determine the position and, if applicable, the orientation of the dimension number.

The following texts make up the dimension texts to be displayed, where *PreText*, *Coord1Text*, *MiddleText*, *Coord2Text* and *PostText* will be connected to one text ("Main dimension text") without separating characters.



# Contents

### Example

0,230,0,0,	BlockType
0,0,400,"DINLQ",	TextFont
1,	TextXProperty.Flag
1,0,0.0/0.0/0.0,0.0/0.0/0.0,0.0,0,	...
0,	TextXProperty.Layer
5.0,3.5,	TextSize1,TextSize2
0,4,1,	CharDistance,TabDistance,TextMode
2,1,0,	Accuracy,Refresh,Rotate
0;	System

This coordinate dimension will be lettered with the font "DINLQ", using pen 1. The main dimension has a text size of 5 mm, the tolerances have a text size of 3.5 mm. The dimension number will not be rotated, but automatically be updated.

The dimension will be calculated according to the settings in the coordinate system "00: \*Standard".

(120,194)  
(nx1,ny1)  
 + (x1,y1)

The image above shows a coordinate dimension created using the settings described above. The point (x1,y1) defines the coordinate, the point (nx1,ny1) places the dimension number. As the number is not rotated, the point (nx2,ny2) is not used.

*PreText* is set to "(" by default, *MiddleText* is set to "/" and *PostText* is set to ")".



## Object 31 "Dimension Area"



# Contents

### Data Block Sequence

<u>Data Block 000</u> ( nx1, ny1 )	- First dimension number definition
<u>Data Block 000</u> ( nx2, ny2 )	- Second dimension number definition
<u>Data Block 100</u> ( Area )	- Area in mm <sup>2</sup>
<u>Data Block 110</u> ( PreText )	- Text in front of number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( NumText )	- Area as a text ( <i>ElemCount</i> = -250)
<u>Data Block 110</u> ( PostText )	- Text behind the number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance1 )	- Upper tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance2 )	- Lower tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 230</u> ( ... )	- Area dimension parameters

The points (nx1,ny1) and (nx2,ny2) determine the position and, if applicable, the orientation of the dimension number. The area value is explicitly stored in *Area*, using internal square millimeters. When updating this dimension, the value stored in *Area* is written to *NumText*, considering the current scale and length unit. The value of *Area* itself remains unchanged!

The following texts make up the dimension texts to be displayed, where *PreText*, *NumText* and *PostText* will be connected to one text ("Main dimension text") without separating characters.



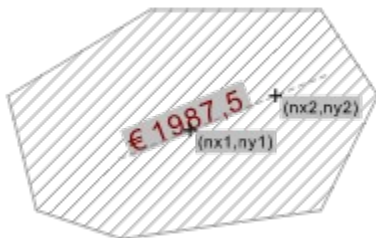
# Contents

### Example

```
0,230,0,0,          |BlockType|
0,0,400,"DINDRAFT", |TextFont|
1,                  |TextXProperty.Flag|
1,0,0.0/0.0/0.0,0.0/0.0/0.0,0.0,0, | ... |
0,                  |TextXProperty.Layer|
5.0,3.5,           |TextSize1,TextSize2|
0,4,1,             |CharDistance,TabDistance,TextMode|
2,1,1,             |Accuracy,Refresh,Rotate|
0;                 |System|
```

This area dimension will be lettered with the font "DINDRAFT", using pen 1. The main dimension has a text size of 5 mm, the tolerances have a text size of 3.5 mm. The dimension number is rotated and will automatically be updated.

The dimension will be calculated according to the settings in the coordinate system "00: \*Standard".



The image above shows an area dimension created using the settings described above. The points (nx1,ny1) and (nx2,ny2) place and rotate the dimension number.

*PreText* is set to "€" (Ansi 128) by default.



## Object 32 "Dimension Perimeter"



# Contents

### Data Block Sequence

<u>Data Block 000</u> ( nx1, ny1 )	- First dimension number definition
<u>Data Block 000</u> ( nx2, ny2 )	- Second dimension number definition
<u>Data Block 100</u> ( Perimeter )	- Perimeter in mm
<u>Data Block 110</u> ( PreText )	- Text in front of number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( NumText )	- Perimeter as a text ( <i>ElemCount</i> = -250)
<u>Data Block 110</u> ( PostText )	- Text behind the number ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance1 )	- Upper tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 110</u> ( Tolerance2 )	- Lower tolerance ( <i>ElemCount</i> = 250)
<u>Data Block 230</u> ( ... )	- Perimeter dimension parameters

The points (nx1,ny1) and (nx2,ny2) determine the position and, if applicable, the orientation of the dimension number. The perimeter value is explicitly stored in *Perimeter*, using internal millimeters. When updating this dimension, the value stored in *Perimeter* is written to *NumText*, considering the current scale and length unit. The value of *Perimeter* itself remains unchanged!

The following texts make up the dimension texts to be displayed, where *PreText*, *NumText* and *PostText* will be connected to one text ("Main dimension text") without separating characters.



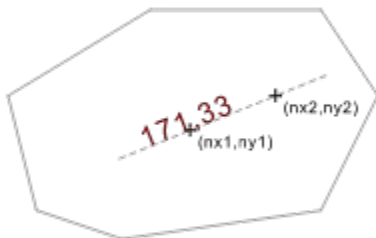
# Contents

### Example

```
0,230,0,0,          |BlockType|
0,0,400,"DINDRAFT",|TextFont|
1,                  |TextXProperty.Flag|
1,0,0.0/0.0/0.0,0.0/0.0/0.0,0.0,0, | ... |
0,                  |TextXProperty.Layer|
5.0,3.5,           |TextSize1,TextSize2|
0,4,1,             |CharDistance,TabDistance,TextMode|
2,0,1,             |Accuracy,Refresh,Rotate|
0;                 |System|
```

This perimeter dimension will be lettered with the font "DINDRAFT", using pen 1. The main dimension has a text size of 5 mm, the tolerances have a text size of 3.5 mm. The dimension number is rotated and will not automatically be updated.

The dimension will be calculated according to the settings in the coordinate system "00: \*Standard".



The image above shows a perimeter dimension created using the settings described above. The points (nx1,ny1) and (nx2,ny2) place and rotate the dimension number.





## Object 35 "Standard Text"



# Contents

### Data Block Sequence

<u>Data Block 110</u> ( Text )	- Text ( <i>ElemCount</i> = 8000)
<u>Data Block 235</u> ( ... )	- Standard text parameters

The value *Text* contains the text to be displayed. Its length may be up to 8000 characters including the terminating null character (0x00).

In order to achieve a tabulator, use the character  $\backslash$  (Ansi 172). For a line feed, use the character ¶ (Ansi 182). All other characters between Ansi 32 and Ansi 255 inclusive will be displayed using the given font. Characters below Ansi 32 will be ignored.

## Object 36 "Frame Text"



# Contents

### Data Block Sequence

<u>Data Block 000</u> ( x1, y1 )	- Corner-point
<u>Data Block 000</u> ( x2, y2 )	- First edge's end-point
<u>Data Block 000</u> ( x3, y3 )	- Second edge's end-point
<u>Data Block 110</u> ( Text )	- Text ( <i>ElemCount</i> = 8000)
<u>Data Block 236</u> ( ... )	- Frame text parameters

The three points define a parallelogram ("frame") that surrounds the frame text. The point (x1,y1) is the starting corner of the text, the point (x2,y2) in relation to the first point defines the direction and length of each text line, and the point (x3,y3) in relation to the first point defines the process direction, i.e. the direction into which the base line is moved after each line.

The value *Text* contains the text to be displayed. Its length may be up to 8000 characters including the terminating null character (0x00).

In order to achieve a tabulator, use the character `␣` (Ansi 172). For a line feed, use the character `␣` (Ansi 182). All other characters between Ansi 32 and Ansi 255 inclusive will be displayed using the given font. Characters below Ansi 32 will be ignored.

Long lines will be broken either at the characters Ansi 32, - (Ansi 45) or `␣` (Ansi 172). The characters Ansi 160 and `␣` (Ansi 172) will not be used as word-breaks. A single word will never be broken, even if it might be larger than the text frame!

## Object 37 "Reference Text"



# Contents

### Data Block Sequence

<u>Data Block 000</u> ( x, y )	- Reference point
<u>Data Block 110</u> ( Text )	- Text ( <i>ElemCount</i> = 8000)
<u>Data Block 235</u> ( ... )	- Standard text parameters
<u>Data Block 237</u> ( ... )	- Reference text parameters

The value *Text* contains the text to be displayed. Its length may be up to 8000 characters including the terminating null character (0x00).

In order to achieve a tabulator, use the character  $\text{␣}$  (Ansi 172). For a line feed, use the character  $\text{␣}$  (Ansi 182). All other characters between Ansi 32 and Ansi 255 inclusive will be displayed using the given font. Characters below Ansi 32 will be ignored.

## Object 40 "Comment"

### Contents **Data Block Sequence**

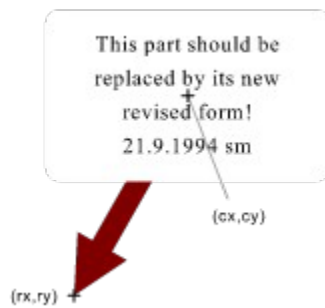
<u>Data Block 000</u> ( rx, ry )	- Reference position
<u>Data Block 000</u> ( cx, cy )	- Position of the comment's center
<u>Data Block 110</u> ( Text )	- Comment text ( <i>ElemCount</i> = 250)

This object type is used to apply a comment text to any object, area or location inside a drawing. This is used for information interchange of several engineers working on the same drawing.

The point (rx,ry) is the reference position, i.e. the end-point of the comment's arrow. The point (cx,cy) determines the comment's center-point, i.e. the center of a rectangular region containing the comment text. The value *Text* contains the comment text to be displayed. Its length may be up to 250 characters including the terminating null character (0x00).

Long lines will be broken either at the characters Ansi 32 or - (Ansi 45). The characters Ansi 160 and " Ansi 172 will not be used as word-breaks.

Comments are displayed in a filled rectangle with rounded corners. The exact appearance of comments (size, color etc.) is determined by the user inside the application.



## Object 41 "Marking"



# Contents

### Data Block Sequence

Data Block 010( x1, y1 ) - Marking

...  
Data Block 010( x?, y? ) - Marking

This object type contains a collection of "markings". A marking is used to store and display specific positions in the drawing. They are especially used during construction and to save positions for further reference. Markings will usually not be output to printer or clipboard.

A marking object may contain up to 2000 markings in total.

## Object 42 "Clipping Surface"



# Contents

### Data Block Sequence

<u>Data Block 242</u> ( ... )	- Instance information
<u>Data Block 001</u> ( x1, y1 )	- Start-point of a curve
...	
<u>Data Block 002</u> ( x?, y? )	- End-point of a line
...	
<u>Data Block 007</u> ( x?, y? )	- Pivot point 1
<u>Data Block 008</u> ( x?, y? )	- Pivot point 2
<u>Data Block 002</u> ( x?, y? )	- End-point of a Bézier curve
...	
<u>Data Block 009</u> ( x?, y? )	- End-point of the circular arc
<u>Data Block 102</u> ( Orientation, Curvature )	- Orientation of the circular arc - Curvature of the circular arc

A clipping surface is a collection of several curves, each defining a closed area. A curve starts with a start-point in a data block of type 001 followed by a sequence of curve elements. Three types of curve elements are available:

#### Line

A line is simply defined by stating its end-point in a data block of type 002. The line is drawn from the curve's current end-point to the line's end-point. The line's end-point then becomes the curve's current end-point.

#### Bézier curve

A Bézier curve is defined by stating two pivot points in data blocks of type 007 and 008, and an end-point in a data block of type 002. The Bézier curve is drawn from the curve's current end-point (named 'S') to the Bézier curve's end-point (named 'E'), influenced by the two pivot points (named P1 and P2). The Bézier curve's end-point then becomes the curve's current end-point.

The points P of such a Bézier curve are calculated using the following equation:

$$P = (1-t)^3 \times S + 3t(1-t)^2 \times P1 + 3t^2(1-t) \times P2 + t^3 \times E \quad (0 \leq t \leq 1)$$

#### Circular arc

A circular arc is defined by its end-point in a data block of type 009, and its orientation and curvature in a data block of type 102. The arc is drawn from the curve's current end-point (named 'S') to the arc's end-point (named 'E'), influenced by its orientation and curvature. The arc's end-point then becomes the curve's current end-point.

The value *Orientation* determines whether to draw the arc in clockwise direction (*Orientation* < 0) or in counter-clockwise direction (*Orientation* >= 0).

The value *Curvature* determines the radius of the arc in indirect manner. It can be handled in two ways. In geometrical view, the absolute value of *Curvature* is  $1/(2 \tan(\beta/2))$ , where  $\beta$  is the arc-angle of the circular arc. The sign of *Curvature* determines, which of the two possible arcs to use.

In practical use, this definition is not very handy. Instead, *Curvature* should be used to perform a simple vector calculation to obtain the arc's defining center-point M:

$$\begin{aligned}x(M) &= 0.5 * ( x(S) + x(E) ) - Curvature * ( y(E) - y(S) ) \\y(M) &= 0.5 * ( y(S) + y(E) ) + Curvature * ( x(E) - x(S) )\end{aligned}$$

After calculating the center-point, all points required to draw a standard circular arc are known. The allowed value range of *Curvature* is  $\pm 1e100$ .

Each curve ends if either a new curve is started by a new start-point (in a data block of type 001) or the

complete surface is ended (by a data block of type 999). Each curve has to be closed, i.e. its start-point and end-point have to be connected with a line.

If a surface consists of only one curve simply the inside area of that curve will be filled:

## Contents

If a surface consists of multiple curves they will be filled alternately, i.e. only areas overlapped by an *odd* number of curve's insides will be filled. If, e.g., a small round curve lies inside a large one, this small round curve will be transparent, as its inside is overlapped by two curves (*even* number!):

## Contents

If a surface is drawn with a non-solid line pattern, this line pattern has to be continued during each curve. A surface may contain up to 1000 nested curves made of up to 2000 data blocks in total.



## Object 43 "Bitmap Reference"

### Contents **Data Block Sequence**

Data Block 243( ... ) - Bitmap information

A bitmap reference is a reference to a bitmap file featuring a bitmap's filename and a display matrix. The display matrix determines the location, size and orientation of the bitmap display.

If the bitmap referenced by this object is a monochrome bitmap, the line color and fill color of this objects are used as the foreground and background color of the bitmap. If the bitmap if a non-monochrome bitmap, the object's properties have no direct influence in the bitmap's display. Anyway, the layer information is checked, i.e. bitmap references in invisible layers will not be visible and so on.

## Object 45 "Geometry Line"



# Contents

### Data Block Sequence

Data Block 001( x1, y1 )      - Start-point  
Data Block 002( x2, y2 )      - End-point

The points (x1,y1) and (x2,y2) determine two points the line is passing through. The line is endless, i.e. it starts at one end of the coordinate range and ends at the other end. Geometry lines will usually not be output to printer or clipboard.

A geometry line is an "open" object, i.e. it has a non-closed outline and cannot be filled.

## Object 46 "Geometry Circle"



# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 004</u> ( rx, ry )	- Radius definition

The point (cx,cy) determines the circle's center-point, the point (rx,ry) is a point on the circle's outline and thus defines the radius. Geometry circles will usually not be output to printer or clipboard.

## Object 47 "Geometry Ellipse"



# Contents

### Data Block Sequence

<u>Data Block 003</u> ( cx, cy )	- Center-point
<u>Data Block 006</u> ( vx1, vy1 )	- Vector end-point 1
<u>Data Block 006</u> ( vx2, vy2 )	- Vector end-point 2

The point (cx,cy) determines the ellipse's center-point, the points (vx1,vy1) and (vx2,vy2) are end-points of vectors that define the ellipse. Having the center-point C and the two vectors V1 and V2 (vector from the center-point to the respective vector end-point), the points P on such an ellipse are determined by the following equation:

$$P = C + V1 \times \sin(\beta) + V2 \times \cos(\beta) \quad (0 \leq \beta < 2\pi)$$

The resulting ellipse can be any type of ellipse, rectangular as well as arbitrary. Geometry ellipses will usually not be output to printer or clipboard.



# Contents

## Entity "Instance"

This entity represents an instance (internal or external reference) of a block. It is used to reference and display a block definition. An entity of type "Instance" has the following structure:

 **Contents** **Element Sequence**  
UnitOwner, 1, XProperty,  
LibraryName, BlockName, DisplayMatrix;  
-Data Section-

 **Contents** **Element Description**

### *UnitOwner*

[*short*] This value is a unique identification of the plug-in that created the entity. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *XProperty*

[XPROPERTY] Properties of the instance including transmission.

### *LibraryName*

[TEXT64] Name of the library containing the desired block, maximum 63 characters. If the instance references a block located in the same drawing / library as the instance, set this name to "\*".

### *BlockName*

[TEXT64] Name of the block, maximum 63 characters. If the first character is 0x00, this instance is invalid and will neither be loaded nor stored!

If *LibraryName* is set to "\*", the content of *BlockName* may have a special form. If the first character is # (Ansi 35), the block is a special, internally used and handled block (see Entity "Group" and Entity "Position Number"). The function of this block then depends on the character following the # sign.

Such instances are only allowed inside drawings, *not* inside libraries!

### *DisplayMatrix*

[MATRIX] Display matrix of the block. All entities stored in the block have to be multiplied with this matrix before display. It contains translation, rotation, scaling and shearing.

Following is the data section of this entity. This section contains a list of data blocks (see Entities and Data Blocks). Currently, instances do only contain local attributes. This may change in further releases, so be prepared to find global attributes and other data blocks in this section.

 **Contents** **Examples**

```
0, 3, 64, 0, 0, 0/0/0, 0/0/0, 0.0, 0, 0, 1, |HeaderType, XProperty|
" *", |LibraryName|
"Furniture\Living Room\Sideboard", |BlockName|
2.0, 0.0, 0.0, 2.0, 0.0, 0.0; |DisplayMatrix|
0, 400, 0, 250, "Material", "Pine", |B400|
0, 999, 0, 0; |B999|
```

This instance inserts the block "Furniture\Living Room\Sideboard" that has been introduced in the chapter Entity "Block". As the block is an internal block of the drawing, and not an external block in a

library, *LibraryName* is set to "\*". The layer index 1 is transmitted to all entities of the block, the block will be displayed at location (0.0,0.0), scaled by factor 2.0.

The instance has received the local attribute "Material" from the block's definition, its content was changed from "Oak" to "Pine" either during creation of the instance or later by explicit editing of the user.

```
0,3,0,0,0,0/0/0,0/0/0,0.0,0,0,0,    |HeaderType,XProperty|
"Screws",                               |LibraryName|
"Cylinder Screws\DIN 912\M8",          |BlockName|
-1.0,0.0,0.0,-1.0,10.0,-7.0;          |DisplayMatrix|
0,999,0,0;                             |B999|
```

This instance inserts the external block "M8" of the library "Screws". It is placed in the subfolder "DIN 912" which is located in the folder "Cylinder Screws". The entities of the block will be displayed with their own properties (*XProperty.Flag* = USE\_NULL). The block will be displayed at the location (10.0,-7.0), rotated by 180°. This instance does not own any attribute.

## Entity "Block"

This entity represents either an internal (inside a drawing file) or an external (inside a library file) block definition. Both definitions are handled similar, they both are a named collection of multiple entities, being referenced by instances (see [Entity "Instance"](#)).

All entities inside a block are moved in a way that their origin (coordinate (0.0,0.0) ) is the desired "insertion point" of the block. The insertion point is entered by the user when creating a block, it determines the location he wants to enter later when inserting the block into a drawing.

Blocks are referred to by their names. Such a block name consists of folder names and the block name, assembled similar to file names. The folder names and the final block names are separated by a back-slash \ (Ansi 92). Each name element may be up to 63 characters long, the assembled block name may also be up to 63 characters long. A possible block name would be e.g. "Cylinder Screws\DIN 912\M8". This name describes a block named "M8", being located in the folder "DIN 912", which is a sub-folder of "Cylinder Screws".

An entity of type "Block" has the following structure:

 **Contents** **Element Sequence**  
UnitOwner, 2, XProperty, BlockName, BlockRect;  
-Data Section-

 **Contents** **Element Description**

### *UnitOwner*

[*short*] This value is a unique identification of the plug-in that created the entity. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

### *XProperty*

[*XPROPERTY*] Properties of the block including transmission (a block does usually not contain any transmission information unless explicitly enforced by the user).

### *BlockName*

[*TEXT64*] Name of the block, up to 63 characters. If the first character of the block's name is 0x00, the block is invalid and will neither be loaded nor saved!

If the first character of the block's name is # (Ansi 35), the block is a special, internally used and handled block (see [Entity "Group"](#) and [Entity "Position Number"](#)). The function of this block then depends on the character following the # sign. Such blocks are only allowed inside drawings, *not* inside libraries!

### *BlockRect*

[*DRECT*] Rectangular frame surrounding the character cell, valid only if the block is a character inside a font library (see [Font Libraries](#)). In this case, ( *BlockRect.x1* / *BlockRect.y1* ) is the lower left corner, ( *BlockRect.x2* / *BlockRect.y2* ) is the upper right corner of the cell.

Following is the data section of this entity. This section contains a list of data blocks (see [Entities and Data Blocks](#)). Currently, blocks do only contain global and local attributes. This may change in further releases, so be prepared to find other data blocks in this section.

# Contents **Example**

```
0,2,0,0,0,0/0/0,0/0/0,0.0,0,0,0,    |HeaderType,XProperty|
"Furniture\Living Room\Sideboard",    |BlockName|
-10.0,-5.0,10.0,5.0;                  |BlockRect|
0,301,0,250,"Price","349.00";         |B301|
0,400,0,250,"Material","Oak";         |B400|
0,999,0,0;                             |B999|
```

This defines a block named "Sideboard", residing in the sub-folder "Living Room" of the folder "Furniture". The entities of this block will be displayed unmodified (*XProperty.Flag* = USE\_NULL). The surrounding rectangle of the block (as calculated by the application when creating this block) is (-10.0, -5.0, 10.0, 5.0).

The block owns a global numeric attribute named "Price", its value is "135.00", and a local attribute named "Material" with a predefined value of "Oak".

Behind this entity, a sequence of all entities of the blocks will follow, ended by an entity of type "End of List".



## Entity "Group"

A "group" is not a new entity type, but a special form of the Entity "Block". This block form is indicated by a special content of the values *LibraryName* and *BlockName*:

### Contents **Element Sequence**

UnitOwner, 2, XProperty, BlockName, BlockRect;  
-Data Section-

### Contents **Element Description**

#### *UnitOwner*

[*short*] This value is a unique identification of the plug-in that created the entity. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *XProperty*

[XPROPERTY] Properties of the block including transmission (a block does usually not contain any transmission information unless explicitly enforced by the user).

#### *LibraryName*

[TEXT64] This value has to be set to "\*", i.e. groups can only occur inside drawings, not inside libraries!

#### *BlockName*

[TEXT64] The name of a group block must have the following form:

"#G\ ..." (Ansi 35 71 92 ...)

where "..." is a character sequence that identifies each group unequivocally. Usually, the application creates group names of the following form:

"#G\sssss-sss-xxxxxxxxxxxxxx"

The value "xxxxxxxxxxxxxx" is the current system time in milliseconds plus random digits, "sssss-sss" is the serial number of the application. This leads to a unique name used to refer to a group.

Anyway, the user will usually have no need to know a group's name, as group are always handled directly by simply identifying them via mouse clicks.

**Important!** Groups (like all special blocks whose names start with #) have a special quality that "normal" blocks do not have: They are aware of all attributes that the entities collected in such a block do own. If a group contains an instance, the group itself knows all attributes of that instance and all attributes of the block the instance refers to.

This quality can be used to display attribute values of instances or blocks: Create a text object containing a variable stating the attribute's name (see Data Block Type 110 (Text) for a description of variables), and combine this text object together with the instance in a group.

Groups will usually be handled like "normal" blocks, i.e. they will appear in all dialogs used to select or modify blocks. The user will be able to select such a group from the drawing's local library and insert it into the drawing. But groups are automatically maintained, i.e. they will be deleted as soon as no instance of them exists any more. This deletion will take place each time the drawing is loaded or saved.

## Entity "Position Number"

A "position number" is not a new entity type, but a special form of the Entity "Block". This block form is indicated by a special content of the values *LibraryName* and *BlockName*:

### Contents **Element Sequence**

UnitOwner, 2, XProperty, BlockName, BlockRect;  
-Data Section-

### Contents **Element Description**

#### *UnitOwner*

[*short*] This value is a unique identification of the plug-in that created the entity. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement an plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *XProperty*

[XPROPERTY] Properties of the block including transmission (a block does usually not contain any transmission information unless explicitly enforced by the user).

#### *LibraryName*

[TEXT64] This value has to be set to "\*", i.e. position numbers can only occur inside drawings, not inside libraries!

#### *BlockName*

[TEXT64] The name of a position number block must have the following form:

"#P\ ..." (Ansi 35 80 92 ...)

where "..." is a character sequence that identifies each position number unequivocally. Usually, the application creates position number names of the following form:

"#P\sssss-sss-xxxxxxxxxxxxxx"

The value "xxxxxxxxxxxxxx" is the current system time in milliseconds plus random digits, "sssss-sss" is the serial number of the application. This leads to a unique name used to refer to a position number. Anyway, the user will usually have no need to know a position number's name, as position numbers are always handled directly by simple identifying them via mouse clicks.

**Important!** Position numbers (like all special blocks whose names start with #) have a special quality that "normal" blocks do not have: They are aware of all attributes that the entities collected in such a block do own. If a position number contains an instance, the position number itself knows all attributes of that instance and all attributes of the block the instance refers to.

This quality can be used to display attribute values of instances or blocks: Create a text object containing a variable stating the attribute's name (see Data Block Type 110 (Text) for a description of variables), and combine this text object together with the instance in a position number.

Position numbers will usually be handled like "normal" blocks, i.e. they will appear in all dialogs used to select or modify blocks. The user will be able to select such a position number from the drawing's local library and insert it into the drawing. But position numbers are automatically maintained, i.e. they will be deleted as soon as no instance of them exists any more. This deletion will take place each time the drawing is loaded or saved.



## Entity "Custom-Defined"

This entity represents a user-defined entity, i.e. an entity that was created by a plug-in. Such an entity can only be displayed and modified if a plug-in is present that knows how to handle such an entity. An entity of type "Custom-Defined" has the following structure:

### Contents **Element Sequence**

```
UnitOwner, 9, XProperty,  
LibraryName, BlockName, DisplayMatrix,  
UserType;  
-Data Section-
```

### Contents **Element Description**

#### *UnitOwner*

[*short*] This value is a unique identification of the plug-in that created the entity. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

#### *XProperty*

[XPROPERTY] Properties of the display instance including transmission.

#### *LibraryName*

[TEXT64] Name of the library containing the desired display block, maximum 63 characters. If the instance references a block located in the same drawing / library as the instance, set this name to "\*".

#### *BlockName*

[TEXT64] Name of the display block, maximum 63 characters. If the first character is 0x00, this instance is invalid and will neither be loaded nor stored!

If *LibraryName* is set to "\*", the content of *BlockName* may have a special form. If the first character is # (Ansi 35), the block is a special, internally used and handled block (see Entity "Group" and Entity "Position Number"). The function of this block then depends on the character following the # sign. Such instances are only allowed inside drawings, *not* inside libraries!

#### *DisplayMatrix*

[MATRIX] Display matrix of the display block. All entities stored in the block have to be multiplied with this matrix before display. It contains translation, rotation, scaling and shearing.

#### *UserType*

[*long*] Internal identification of the custom-defined object. For detailed information about this identification, see the documentation of the program or plug-in that created this entity.

Following is the data section of this entity. This section contains a list of data blocks (see Entities and Data Blocks). Custom-defined objects may contain any types of data blocks.

## Entity "End of List"

This entity marks the end of an entity sequence. Such an entity terminates each section containing entities and each block definition. An entity of type "End of List" has the following structure:

 **Contents** **Element Sequence**  
UnitOwner, 999;

 **Contents** **Element Description**

*UnitOwner*

[*short*] This value is a unique identification of the plug-in that created the entity. The value 0 is reserved for use by TommySoftware®, especially for objects and data blocks that are created and handled directly by the application.

Third-party vendors that plan to implement a plug-in that produces custom-defined objects have to contact TommySoftware® to receive their unique identification. This service is free of charge.

## File Header (Structure of TVG 4.2)

A TVG 4.2 file consists of a standard file header and a sequence of variable sections. The order of the sections is fixed, but each section may either be empty (i.e. contain only a termination) or not exist at all.

Each section starts and ends with a unique keyword. A keyword is a word delimited by the character = (Ansi 61). The word itself may be up to 31 characters long and may neither contain the character = (Ansi 61) nor a line feed. Each keyword is followed immediately by a semicolon. All section except =EXIT= are terminated by the keyword =END= .

The file header consists of a special character sequence, which has to have exactly the form described below. The character sequence has to start at the first byte of the file, as this character sequence is checked by directly reading the first 22 bytes of the file and comparing them with the expected identification! As a result, this identification may *not* be enclosed in quotes, and it must be handled case sensitive!



TommySoftware TVG 4.20;

The further file content depends on the file's type. Currently, two file types are defined: "Drawing" and "Library". In future, additional file types might be introduced. So be sure to check whether a file contains the desired data before trying to access it!

# File Body of Drawings Overview (Structure of TVG 4.2)

A TVG 4.2 file containing a drawing starts with the section =DRAWING= , followed by several sections containing settings, block definitions and drawing data.

Not every of the sections listed below has to exist in a drawing file. If it does exist, it should follow the order of the sections below (although this order is *not* definite). Section marked with a \* have to exist in any drawing file at the given position!

## Sections

Section =DRAWING= (Drawing Information)\*

Section =TOOLBOX= (Toolbox)

Section =SYMBOL= (Symbol Window)

Section =KEYBOARD= (Keyboard)

Section =DEFAULT= (Defaults)

Section =USER= (Settings)

Section =MODULE= (Plug-In Settings)

Section =PAGE= (Page Format)

Section =COLOR= (Color Definitions)

Section =HATCH= (Hatching Types)

Section =MULTILINE= (Line Sequences)

Section =SYSTEM= (Coordinate Systems)

Section =PEN= (Pens)

Section =LINE= (Line Patterns)

Section =LAYER= (Layers)

Section =WINDOW= (Window Settings)

Section =BITMAP= (Embedded Bitmaps)

Section =BLOCK= (Block Definitions)

Section =OBJECT= (Objects)

Section =EXIT= (End of File)\*

## About Drawings

Minimal Drawing

## Section =DRAWING= (Drawing Information)

### Contents **Element Sequence**

```
=DRAWING=;  
Title,  
Theme,  
Author1,  
Date1,  
Author2,  
Date2,  
Comment;  
=END=;
```

### Contents **Element Description**

#### *Title*

[TEXT64] Title of the drawing. This text usually contains a detailed description of the drawing.

#### *Theme*

[TEXT64] Theme of the drawing. This text usually contains a description of the project the drawing belongs to.

#### *Author1*

[TEXT64] Name of the user that created the drawing. This text is initialized when creating a drawing.

#### *Date1*

[TEXT64] Date and time when the drawing was created. In the English release, this text might e.g. be "Tuesday, October 15 1991, 5:15 pm".

#### *Author2*

[TEXT64] Name of the user that modified this drawing for the last time. This text is initialized each time the drawing is saved.

#### *Date2*

[TEXT64] Date and time when this drawing was saved for the last time. This time is initialized each time the drawing is saved.

#### *Comment*

[TEXT256] This text contains any further comment on the drawing.

The section =DRAWING= has to exist in every drawing file! If desired, this section may contain only the title of the drawing, set to "" (empty text). In this case, the value *Title* will be set to the drawing's file name.

In any case, empty texts at the end of the section do not have to be stated, if e.g. *Comment* is empty, the section can already be ended after *Date2*, which then has to be followed by a semicolon. When reading this section, simply read all texts available (by reading until a semicolon is found) and set all other texts to "".



## Section =TOOLBOX= (Toolbox)

The section =TOOLBOX= contains the toolbox window's assignment that was active when the drawing was saved.

### Contents Element Sequence

```
=TOOLBOX=;  
??? ... ???;  
=END=;
```

The content of the section =TOOLBOX= should be ignored. When creating a new drawing file this section should not exist.

## Section =SYMBOL= (Symbol window)

The section =SYMBOL= contains the symbol window's assignment that was active when the drawing was saved.

### Contents **Element Sequence**

```
=SYMBOL=;  
??? ... ???;  
=END=;
```

The content of the section =SYMBOL= should be ignored. When creating a new drawing file this section should not exist.

## Section =KEYBOARD= (Keyboard)

The section =KEYBOARD= contains the keyboard assignment that was active when the drawing was saved.

### Contents **Element Sequence**

```
=KEYBOARD=;  
??? ... ???;  
=END=;
```

The content of the section =KEYBOARD= should be ignored. When creating a new drawing file this section should not exist.

## Section =DEFAULT= (Defaults)

The section =DEFAULT= contains the default pens and layers that were active when the drawing was saved. These default values are used by the application to assign default pens and layers to newly created objects.

### Contents **Element Sequence**

```
=DEFAULT=;  
Pens[0], Pens[1], ... ;  
Layers[0], Layers[1], ... ;  
=END=;
```

### Contents **Element Description**

#### *Pens*

[*long*[]] Indices of pens that will automatically be assigned to objects created in some standard situations. Use the following field indices to access elements of this array:

- 0x0000 Default pen for dimension lines.
- 0x0001 Default pen for dimension texts.
- 0x0002 Default pen for standard, frame and reference texts.
- 0x0003 Default pen for additional reference text elements (frame and arrow).
- 0x0004 Default pen for geometry objects.

If any of these indices is -1, there is no default pen defined for this situation, i.e. the object will be assigned to the currently active pen.

#### *Layers*

[*long*[]] Indices of layers that will automatically be assigned to objects created in some standard situations. Use the following field indices to access elements of this array:

- 0x0000 Default layer for markings.
- 0x0001 Default layer for generated outlines (surfaces).
- 0x0002 Default layer for dimension lines and texts.
- 0x0003 Default layer for texts.
- 0x0004 Default layer for hatchings.
- 0x0005 Default layer for block instances.
- 0x0006 Default layer for geometry objects.
- 0x0007 Default layer for group instances.

If any of these indices is -1, there is no default layer defined for this situation, i.e. the object will be assigned to the currently active layer.

If the section =DEFAULT= does not exist, assume all default indices to be -1, i.e. no default pen or layer is defined. The number of indices stored in each array may vary. Do always read as many values as available and set all other values to -1.

## Section =USER= (Settings)

The section =USER= contains all user-dependent settings that were active when the drawing was saved.

## Contents Element Sequence

```
=USER=;  
??? ... ???;  
=END=;
```

The content of the section =USER= should be ignored. When creating a new drawing file this section should not exist.

## Section =MODULE= (Plug-In Settings)

The section =MODULE= contains settings that were stored in the drawing by plug-ins.

### Contents **Element Sequence**

```
=MODULE=;  
-Entity Sequence-  
=END=;
```

The entity sequence consists of entities of type "Custom-Defined". The sequence of entities is terminated by an Entity "End of List".

The content of the section =MODULE= should be ignored. When creating a new drawing file this section should not exist.

Normally, plug-ins will never read or write this section directly, instead they will use procedures of the *Toso Interface* to get or to set their private settings. Anyway, plug-ins may directly scan this section for their private settings if required.

## Section =PAGE= (Page Format)

The section =PAGE= contains a description of the page format that was active when the drawing was saved. This page format can be used as default for display and/or output.

### Contents Element Sequence

```
=PAGE=;  
PageType, PageOrient, PageXSize, PageYSize;  
=END=;
```

### Contents Element Description

#### *PageType*

[*long*] This value determines the page's size. Possible values are:

0x0000 User-defined format, i.e. the values of *PageXSize* and *PageYSize* state the page size. In this case *PageOrient* should also be 0x0000.

0x0001	DIN A4	210 × 297 mm
0x0002	DIN A3	297 × 420 mm
0x0003	DIN A3.2	297 × 594 mm
0x0004	DIN A3.1	297 × 841 mm
0x0005	DIN A3.0	297 × 1189 mm
0x0006	DIN A2	420 × 594 mm
0x0007	DIN A2.1	420 × 841 mm
0x0008	DIN A2.0	420 × 1189 mm
0x0009	DIN A1	594 × 841 mm
0x000a	DIN A1.0	594 × 1189 mm
0x000b	DIN A0	841 × 1189 mm
0x000c	DIN 2A0	1189 × 1682 mm
0x000d	ISO A4×3	297 × 630 mm
0x000e	ISO A4×4	297 × 841 mm
0x000f	ISO A4×5	297 × 1051 mm
0x0010	ISO A4×6	297 × 1261 mm
0x0011	ISO A3×3	420 × 891 mm
0x0012	ISO A3×4	420 × 1189 mm
0x0013	ISO A2×3	594 × 1261 mm
0x0014	DIN B5	176 × 250 mm
0x0015	DIN B4	250 × 353 mm
0x0016	DIN B3	353 × 500 mm
0x0017	DIN B2	500 × 707 mm
0x0018	DIN B1	707 × 1000 mm
0x0019	DIN B0	1000 × 1414 mm
0x001a	DIN C5	162 × 229 mm
0x001b	DIN C4	229 × 324 mm
0x001c	DIN C3	324 × 458 mm
0x001d	DIN C2	458 × 648 mm
0x001e	DIN C1	648 × 917 mm
0x001f	DIN C0	917 × 1297 mm
0x0020	US Half	5.5 × 8.5 inch
0x0021	ANSI A / US Letter	8.5 × 11.0 inch

0x0022	ANSI B / US Tabloid	11.0 × 17.0 inch
0x0023	ANSI C	17.0 × 22.0 inch
0x0024	ANSI D	22.0 × 34.0 inch
0x0025	ANSI E	34.0 × 44.0 inch
0x0026	US Legal	8.5 × 14.0 inch

### *PageOrient*

[*long*] This value determines whether the page shall have landscape or portrait orientation. Possible values are:

0x0000	User-defined format, i.e. the values of <i>PageXSize</i> and <i>PageYSize</i> state the page size. In this case <i>PageType</i> should also be 0x0000.
0x0001	Portrait
0x0002	Landscape

### *PageXSize*

[*double*] This value states the page's horizontal size in millimeters. It must be valid even if *PageType* states an explicit page format. The valid range is 1.0 to 4000.0 inclusive.

### *PageYSize*

[*double*] This value states the page's vertical size in millimeters. It must be valid even if *PageType* states an explicit page format. The valid range is 1.0 to 4000.0 inclusive.

If the section =PAGE= does not exist, assume either the page format DIN A3 landscape (2,2,420,297) or US D landscape (22,2,431.8,279.4). If the section does exist, it must contain all of the four values stated above.



## Contents

### Example of a Page Format Definition

6, 2, 594, 420;

|Type, Orient, XSize, YSize|

This is the definition of a page based on DIN A2 (420 × 594 mm), but in landscape orientation. The resulting page size is 594 × 420 mm.



## Section =COLOR= (Color Definitions)

The section =COLOR= contains up to 500 custom color definitions. Custom color will be displayed in most color selection dialog windows.

 **Contents** **Element Sequence**  
=COLOR=;  
ColorNum;

 **Contents** **Entry for each Color Definition**  
ColorName, ColorValue;

 **Contents** **End of Section**  
=END=;

 **Contents** **Element Description**

*ColorNum*

[*long*] Number of color definitions to follow. Valid range is 0 to 500 inclusive.

*ColorName*

[TEXT32] Name of the color, up to 31 characters.

*ColorIndex*

[COLORREF] RGB definition of the color.

If the section =COLOR= does not exist, assume no custom color to be defined (*ColorNum* = 0).

## Section =HATCH= (Hatching Types)

The section =HATCH= contains up to 100 hatching types.

### Contents **Element Sequence**

=HATCH=;  
[HatchNum](#), [HatchActive](#);

### Contents **Entry for each Hatching Type Definition**

[HatchName](#), [HatchIndex](#);  
[HatchLine1Active](#), [HatchLine1](#), [HatchLine1Rotate](#);  
[HatchLine2Active](#), [HatchLine2](#), [HatchLine2Rotate](#);  
[HatchBlockActive](#), [HatchLibraryName](#), [HatchBlockName](#),  
[HatchBlockRotate](#), [HatchBlockScale](#),  
[HatchBlockXStep1](#), [HatchBlockXStep2](#),  
[HatchBlockYStep1](#), [HatchBlockYStep2](#),  
[HatchBlockLineStep1](#), [HatchBlockLineStep2](#);  
[HatchRotate](#), [HatchOffset1](#), [HatchOffset2](#);

### Contents **End of Section**

=END=;

### Contents **Element Description**

#### *HatchNum*

[*long*] Number of hatching type definitions to follow. Valid range is 0 to 100 inclusive.

#### *HatchActive*

[*long*] Index of the hatching type that was active when the drawing was saved (referring to *HatchIndex*). Valid range is 0 to 100 inclusive. This value is optional and will be set to 0 if not existing!

#### *HatchName*

[*TEXT64*] Name of the hatching type, up to 63 characters.

#### *HatchIndex*

[*long*] Index of the hatching type (1 to 100). Each index may be assigned to one hatching type only, as this index is used internally to identify a hatching type.

#### *HatchLine1Active*

[*BOOL*] Determines whether the first line sequence is active or not.

#### *HatchLine1*

[*int*] Index of the first line sequence to be used.

#### *HatchLine1Rotate*

[*double*] Rotation angle of the first line sequence.

#### *HatchLine2Active*

[*BOOL*] Determines whether the second line sequence is active or not.

#### *HatchLine2*

[*int*] Index of the first second sequence to be used.

#### *HatchLine2Rotate*

[*double*] Rotation angle of the second line sequence.

*HatchBlockActive*

[*BOOL*] Determines whether a hatching block is to be used.

*HatchLibraryName*

[*TEXT64*] Name of the library containing the desired block, maximum 63 characters. If the desired hatching block is located inside the drawing (i.e. it is an internal block), set this name to "\*".

*HatchBlockName*

[*TEXT64*] Name of the block or block, maximum 63 characters.

*HatchBlockRotate*

[*double*] This value determines the rotation of the hatching block (based on its insertion point).

*HatchBlockScale*

[*double*] This value determines the scaling of the hatching block.

*HatchBlockXStep1*

*HatchBlockXStep2*

[*double*] These two values determine the horizontal advance of the block-based hatching.

*HatchBlockYStep1*

*HatchBlockYStep2*

[*double*] These two values determine the vertical advance of the block-based hatching.

*HatchBlockLineStep1*

*HatchBlockLineStep2*

[*double*] These two values determine the horizontal line offset of the block-based hatching.

*HatchRotate*

[*double*] This value determines the global rotation of the complete hatching.

*HatchOffset1*

*HatchOffset2*

[*double*] These two values determine the offset of the hatching.

If the section =HATCH= does not exist, assume no hatching type to be defined (*HatchNum* = 0) and hatching type 0 to be active.

## Section =MULTILINE= (Line Sequences)

The section =MULTILINE= contains up to 50 line sequences.

### Contents **Element Sequence**

```
=MULTILINE=;  
MultiLineNum;
```

### Contents **Entry for each Line Sequence Definition**

```
MultiLineName, MultiLineIndex;  
MultiLineUse[0], MultiLineDistance[0], MultiLineXProperty[0];  
...  
MultiLineUse[7], MultiLineDistance[7], MultiLineXProperty[7];
```

### Contents **End of Section**

```
=END=;
```

### Contents **Element Description**

#### *MultiLineNum*

[*long*] Number of hatching type definitions to follow. Valid range is 0 to 50 inclusive.

#### *MultiLineName*

[*TEXT32*] Name of the hatching type, up to 31 characters.

#### *MultiLineIndex*

[*long*] Index of the hatching type (1 to 50). Each index may be assigned to one hatching type only, as this index is used internally to identify a hatching type.

#### *MultiLineUse[x]*

[*BOOL*] Determines whether the corresponding line is to be used or not.

#### *MultiLineDistance[x]*

[*double*] Determines the distance between this line and the next active line in internal mm. The valid range is 1e-10 to 1e10 inclusive.

#### *MultiLineXProperty[x]*

[*XPROPERTY*] This set of properties defines single or multiple properties to be transmitted to the corresponding hatching line.

If the section =MULTILINE= does not exist, assume no hatching type to be defined (*MultiLineNum* = 0) and hatching type 0 to be active.

## Section =SYSTEM= (Coordinate Systems)

The section =SYSTEM= contains up to 50 coordinate system definitions.

A coordinate system contains a set of several settings that influence the drawing's display and output. These settings are: scaling, rotation, distortion, origin position, length unit, line unit, angle unit, number representation, position grid and display grid.

The coordinate system with the index 0 is named "\*Standard" and is predefined by default as an unrotated, unscaled coordinate system with its origin in the page's center. The length and line unit is [mm], the angle unit is [deg], both grids are initialized with 1 unit, but not active. This coordinate system is valid throughout all drawings and libraries and does not occur in any TVG 4.2 file.

### Contents **Element Sequence**

=SYSTEM=;  
[SystemNum](#);

### Contents **Entry for each Coordinate System Definition**

[SystemName](#), [SystemIndex](#),  
[SystemRotate](#), [SystemScale](#), [SystemOption](#),  
[SystemOrgMode](#), [SystemXOrg](#), [SystemYOrg](#),  
[SystemLenUnit](#), [SystemLineUnit](#), [SystemAngleUnit](#),  
[SystemFraction](#), [SystemAccuracy](#),  
[SystemGridMode](#), [SystemXGrid](#), [SystemYGrid](#),  
[SystemSnapMode](#), [SystemXSnap](#), [SystemYSnap](#);

### Contents **End of Section**

=END=;

### Contents **Element Description**

#### *SystemNum*

[*long*] Number of coordinate system definitions to follow. Valid range is 0 to 50 inclusive.

#### *SystemName*

[*TEXT32*] Name of the coordinate system, up to 31 characters.

#### *SystemIndex*

[*long*] Index of the coordinate system (1 to 50). Each index may be assigned to one coordinate system only, as this index is used internally to identify a coordinate system.

#### *SystemRotate*

[*double*] Rotation of the display in [rad]. The rotation is valid for screen display only, not during output.

#### *SystemScale*

[*double*] Drawing scale (100.0 represents a scale of 100:1, 0.02 represents a scale of 1:50 etc.). Valid range is 1e-10 to 1e10 inclusive.

#### *SystemOption*

[*long*] Distortion mode of the display. The distortion is valid for screen display only. This option

allows to distort the screen display of a drawing in a way that isometric or dimetric views inside this drawing will be displayed rectangular, plain and without distortion. By this means, isometric and dimetric drawings can be produced without having to calculate the exact angles and lengths of lines and ellipses. For an illustration of the effect of this option, have a close look at the application.

Possible values are:

0x0000	No distortion
0x0001	Left view of an isometric drawing
0x0002	Right view of an isometric drawing
0x0003	Top view of an isometric drawing
0x0004	Left view of an dimetric drawing 1 (-7°)
0x0005	Right view of an dimetric drawing 1 (-7°)
0x0006	Top view of an dimetric drawing 1 (-7°)
0x0007	Left view of an dimetric drawing 2 (+7°)
0x0008	Right view of an dimetric drawing 2 (+7°)
0x0009	Top view of an dimetric drawing 2 (+7°)

This distortion is independent from the grid settings (see *SystemGridMode* and *SystemSnapMode*).

Anyway, a distorted view should never be combined with a distorted grid - this will lead to deep confusion of the user.

### *SystemOrgMode*

[*long*] Placement of the coordinate system's origin. Possible values are:

0x0000	The origin is located at the position stated in <i>SystemXOrg</i> and <i>SystemYOrg</i> .
0x0001	The origin is located at the upper left page corner.
0x0002	The origin is located at the center of the upper page edge.
0x0003	The origin is located at the upper right page corner.
0x0004	The origin is located at the center of the left page edge.
0x0005	The origin is located at the page's center.
0x0006	The origin is located at the center of the right page edge.
0x0007	The origin is located at the lower left page corner.
0x0008	The origin is located at the center of the lower page edge.
0x0009	The origin is located at the lower right page corner.

Internal drawing coordinates are *always* relative to the page's center, independent of the origin's placement!

### *SystemXOrg*

[*double*] X-coordinate of the origin in millimeters relative to the page's center. It must be valid even if *SystemOrgMode* states an explicit origin placement. The valid range is -1e100 to 1e100.

### *SystemYOrg*

[*double*] Y-coordinate of the origin in millimeters relative to the page's center. It must be valid even if *SystemOrgMode* states an explicit origin placement. The valid range is -1e100 to 1e100.

### *SystemLenUnit*

[*long*] Length unit to be used (see Units). This unit will be used for in- and output of coordinates and all values that are depending on the drawing's scale (like dimensions, perimeter, area etc.). Possible values are:

0x0000	[ $\mu\text{m}$ ]
0x0001	[mm]
0x0002	[cm]
0x0003	[dm]
0x0004	[m]
0x0005	[km]
0x0006	[mil]
0x0007	[inch]
0x0008	[foot]

0x0009	[yard]
0x000a	[mile]
0x000b	[dp]
0x000c	[pt]
0x000d	[bp]
0x000e	[cic]

### *SystemLineUnit*

[*long*] Line unit to be used (see Units). This unit will be used for in- and output of values that are not depending on the drawing's scale (line width, font size etc.). Possible values are:

0x0000	[μm]
0x0001	[mm]
0x0002	[cm]
0x0006	[mil]
0x0007	[inch]
0x000b	[dp]
0x000c	[pt]
0x000d	[bp]
0x000e	[cic]

### *SystemAngleUnit*

[*long*] Angle unit to be used (see Units). This unit will be used for in- and output of angle values. Possible values are:

0x0000	[deg]
0x0001	[gra]
0x0002	[rad]
0x0003	[rel]

### *SystemFraction*

[*long*] This value determines how non-integer values will be displayed. It consists of two part, located in the lower 16bit section (determining the display of length values) and the upper 16bit section (determining the display of angle values). Both sections may contain one of the following values:

0x0000	The value will be displayed using floating point representation, i.e. with a decimal separator and following fractional digits, e.g. 10.75. How trailing zeros are handled depends on user settings in the application.
0x0001	The value will be displayed using a mixed fraction representation, i.e. the integer value will be followed by a fraction with a power of 2 as denominator, e.g. 10 3/4. The fraction will be reduced automatically.
0x0002	The value will be split into [foot] and [inch] elements, where the value itself is assumed to be in [inch]. The [foot] element will always be integer, the [inch] element will use floating point representation (see 0x0000). A value of 170.75 inch will be displayed as 14'2.75".
0x0003	The value will be split into [foot] and [inch] elements, where the value itself is assumed to be in [inch]. The [foot] element will always be integer, the [inch] element will use mixed fraction representation (see 0x0001). A value of 170.75 inch will be displayed as 14'2 3/4".
0x0004	The value will be split into [yard], [foot] and [inch] elements, where the value itself is assumed to be in [inch]. The [yard] and [foot] elements will always be integer, the [inch] element will use floating point representation (see 0x0000). A value of 170.75 inch will be displayed as 4yd2'2.75".
0x0005	The value will be split into [yard], [foot] and [inch] elements, where the value itself is assumed to be in [inch]. The [yard] and [foot] elements will always be integer, the [inch] element will use mixed fraction representation (see 0x0001). A value of 170.75 inch will

- be displayed as  $4\text{yd}2'2\frac{3}{4}"$ .
- 0x0006 The value will be split into [degree], [minute] and [second] elements, where the value itself is assumed to be in [degree]. The [degree] and [minute] elements will always be integer, the [second] element will use floating point representation (see 0x0000). A value of 37.331 degree will be displayed as  $37^{\circ}19'51.6"$ .
- 0x0007 The value will be split into [degree], [minute] and [second] elements, where the value itself is assumed to be in [degree]. The [degree] and [minute] elements will always be integer, the [second] element will use mixed fraction representation (see 0x0001). A value of 37.331 degree will be displayed as  $37^{\circ}19'51\frac{5}{8}"$ .
- 0x0008 The value will be displayed using floating point representation, i.e. with a decimal separator and following fractional digits, e.g. 10.75. Values below one will be multiplied by 100 before display. This setting is mainly used for architectural drawings based on meters. How trailing zeros are handled depends on user settings in the application.

**Note:** Some fraction modes make only sense for length values or coordinates, others only for angle values. Anyway, all modes are allowed in both cases.

### *SystemAccuracy*

[long] This value determines the accuracy of non-integer value output to the screen. If values are displayed using floating point representation (*SystemFraction* = 0x0000, 0x0002, 0x0004, 0x0006 or 0x0008), this value states the maximum number of fractional digits. If values are displayed using mixed fraction representation (*SystemFraction* = 0x0001, 0x0003, 0x0005 or 0x0007), this value states the maximum power of 2 the denominator will have. In both cases, the valid range is 0 to 9 inclusive.

### *SystemGridMode*

[long] Current mode of the display grid. The display grid will be displayed on the screen, it does not influence the cursor's movement. Possible values are:

- 0x0000 No grid
- 0x0001 Cartesian grid
- 0x0002 Isometric grid
- 0x0003 Dimetric grid 1 (-7°)
- 0x0004 Dimetric grid 2 (+7°)

### *SystemXGrid*

### *SystemYGrid*

[double] Distance of two display grid points in X- or Y-direction respectively in the current unit. Valid range is 0.0 to 1e10 inclusive. If *SystemXGrid* and/or *SystemYGrid* are 0.0 the grid is invalid and will not be displayed.

### *SystemSnapMode*

[long] Current mode of the position grid. The position grid influences the cursor's movement, it will not be displayed on the screen. Possible values are:

- 0x0000 No grid
- 0x0001 Cartesian grid
- 0x0002 Isometric grid
- 0x0003 Dimetric grid 1 (-7°)
- 0x0004 Dimetric grid 2 (+7°)

### *SystemXSnap*

### *SystemYSnap*

[double] Distance of two position grid points in X- or Y-direction respectively in the current unit. Valid range is 0.0 to 1e10 inclusive. If either *SystemXSnap* or *SystemYSnap* is 0.0, the cursor's movement will be restricted only in one direction. If both values are 0.0, the grid is invalid and will have no effect.



If the section =SYSTEM= does not exist, assume no coordinate system to be defined (*SystemNum* = 0).



# Contents

## Example of a Coordinate System Definition

```
"Standard, 1:50, Grid 1 m",1,      |Name,Index|
0.0,0.02,0,                        |Rotate,Scale,Option|
7,0.0,0.0,                          |OrgMode,XOrg,YOrg|
4,1,0,                               |LenUnit,LineUnit,AngleUnit|
0,4,                                  |Fraction,Accuracy|
1,1.0,1.0,                           |GridMode,XGrid,YGrid|
0,1.0,1.0;                            |SnapMode,XSnap,YSnap|
```

This is the definition of the coordinate system "Standard, 1:50, Grid 1 m" having the index 1. The origin is located in the lower left corner of the page, the display will neither be rotated nor distorted. The scale of the drawing is 1:50. The length unit is [m], the line unit is [mm] and the angle unit is [deg]. Non-integer values will be displayed in floating-point representation with up to 4 fractional digits.

Display and position grid are both set to one meter in both directions, but only the display grid is currently active. Please note that the grid point distance is always stated in the current length unit! If this length unit is changed to [cm], the grid's distances will be one centimeter.

## Section =PEN= (Pen)

The section =PEN= contains up to 500 pen definitions.

A pen basically consists of two complete property sets, one for screen display and one for output. These two property sets provide a large flexibility in property usage.

If, e.g., a drawing is created on top of a black screen background, entities could not be drawn using black lines or fillings. But later, when printing this drawing, most entities will usually be black. Using pens, this problem is easy to solve. For each line type and width to use in the drawing, define a pen that has one "colored" property set for screen display and one "black on white" property set for output.

Another advantage of pen usage is the easy modification of complete entity groups - if all entities used for outlines have to be widened from 0.25 mm to 0.35 mm, simply edit the pen used for drawing these entities.

The pen with the index 0 is named "\*Standard". It causes the usage of concrete properties instead of pen properties, i.e. pen 0 does not transmit any properties. In this case, the entities using this pen will be drawn using the properties stored in their *XProperty* field - independent of the current properties of pen 0. By this means, entities can be equipped with properties that are not stored in any pen.

### Contents **Element Sequence**

```
=PEN=;  
PenNum, PenIdentical, PenActive;  
PenZero;
```

### Contents **Entry for each Pen Definition**

```
PenName, PenIndex,  
PenIntern,  
PenExtern,  
PenLayer;
```

### Contents **End of Section**

```
=END=;
```

### Contents **Element Description**

#### *PenNum*

[*long*] Number of pen definitions to follow. Valid range is 0 to 500 inclusive.

#### *PenIdentical*

[*long*] Determines whether the two property sets of the pen are used, or only the external set. Possible values are:

0x0000     The property set *PenExtern* is used for output, the property set *PenIntern* is used for display.

0x0001     The property set *PenExtern* is used both for display and output.

#### *PenActive*

[*long*] Index of the pen that was active when the drawing was saved (referring to *PenIndex*). Valid range is 0 to 500 inclusive. This value is optional and will be set to 0 if not existing!

*PenZero*

[PROPERTY] Current properties of pen 0.

*PenName*

[TEXT64] Name of the pen, up to 31 characters.

*PenIndex*

[long] Index of the pen (1 to 500). Each index may be assigned to one pen only, as this index is used internally to identify a pen.

*PenIntern*

[PROPERTY] Property set for screen display.

*PenExtern*

[PROPERTY] Property set for output (printer, plotter, clipboard, metafile, export etc.).

*PenLayer*

[int] Index of the layer that is to be activated if this pen is activated. If *PenLayer* is -1, the current layer will not be changed. This value is optional. If it does not exist, assume it to be -1.

If the section =PEN= does not exist, assume no pen to be defined (*PenNum* = 0) and pen 0 to be active. In this case, *PenIdentical* should be initialized with 0x0000, pen 0 should be initialized with the following properties:

0,0/0/0,1/1/1,0.25,0



# Contents

## Example of a Pen Definition

```

"Black Type1 0.25mm",1,          |Name, Index|
2,0.5/0.5/0.5,1.0/1.0/1.0,0.0,0,0, |Intern|
2,0.0/0.0/0.0,0.0/0.0/0.0,0.25,0,1, |Extern|
1;                                |Layer|

```

This is the definition of the pen "Black, Type 1, 0.25mm" having the index 1. When assigning this pen to an entity, this entity will be output with a black, 0.25 mm wide frame using line pattern 1 and will be filled in black. On the screen, this entity will be drawn with a white hairline frame and it will be filled in gray. When selecting this pen in the application, layer 1 will be activated.

## Section =LINE= (Line Patterns)

The section =LINE= contains up to 100 line pattern definitions.

A line pattern defines a sequence of "lines" and "holes" creating a periodical line pattern. In addition to some standard line patterns which are defined that way, this can be used to create custom line patterns. A line pattern definition does only determine the line-hole-sequence, other properties like line width or color have to be defined in a pen (see [Section =PEN= \(Pens\)](#)).

The line pattern with the index 0 is named "\*Standard" and is predefined by default as a solid line. This line pattern is valid throughout all drawings and libraries and does not occur in any TVG 4.2 file.

### Contents **Element Sequence**

=LINE=  
[LineNum](#);

### Contents **Entry for each Line Pattern Definition**

[LineName](#), [LineIndex](#),  
[LinePairNum](#), [LineMode](#), [LineData\[0\]](#), [LineData\[1\]](#) ... ;

### Contents **End of Section**

=END=;

### Contents **Element Description**

#### *LineNum*

[*long*] Number of line pattern definitions to follow. Valid range is 0 to 100 inclusive.

#### *LineName*

[*TEXT32*] Name of the line pattern, up to 31 characters.

#### *LineIndex*

[*long*] Index of the line pattern (1 to 100). Each index may be assigned to one line pattern only, as this index is used internally to identify a line pattern.

#### *LinePairNum*

[*long*] Number of pairs of "line" and "hole". Valid range is 0 to 8 pairs inclusive.

#### *LineMode*

[*long*] Definition mode of the line pattern. Possible values are:

0x0000 The partial line's lengths will be stated in 1/100 of the line's width. If the line's width is less than 0.1 mm, the calculation will be based on a line width of 0.1 mm.

0x0001 The partial line's lengths will be stated in 1/100 mm.

#### *LineData*

[*long[]*] Partial line length pairs. Each line length may be between 100 and 10000 inclusive. The first value of each pair defines a "line", the second value defines a "hole". This list should contain exactly two times the number of line length definitions as the value stored in *LinePairNum* states.

If the section =LINE= does not exist, assume no line pattern to be defined (*LineNum* = 0).



# Contents

## Example of a Line Pattern Definition

```
"Dash-dot-dot narrow DIN 15-G", 4,      |Name, Index|  
3, 0,                                    |Num, Mode|  
4000, 500, 500, 500, 500, 500;        |Data|
```

This is the definition of the line pattern "Dash-dot-dot narrow DIN 15-G" having the index 4. It consists of three pairs of "line" and "hole", defined in 1/100 of the line's width. When drawing a 0.25 mm wide line using this line pattern, the result would be as follows:

```
xxxxxxxx x x xxxxxxxx x x xxxxxxxx  
10 5×1.25 10 5×1.25 10 mm
```

## Section =LAYER= (Layers)

The section =LAYER= contains up to 500 layer definitions.

A layer defines a transparency which is used to integrate a special type of entities in the drawing (like outlines, fillings, hatchings, texts, dimensions etc.) into a "group". They are usually used to maintain a basic structure throughout the drawing to allow easy selection and manipulation of related entities.

Especially, layers can easily be frozen or hidden by a single command.

Additionally, layers can transmit single or multiple properties to all entities assigned to them. This allows an easy way to distinguish the entities of different layers. As this transmission can be limited to screen display, no modification of the entities themselves is necessary.

The layer with the index 0 is named "\*Standard" and is predefined by default as a visible, non-frozen layer that does not transmit any properties. This layer is valid throughout all drawings and libraries and does not occur in any TVG 4.2 file.

### Contents **Element Sequence**

=LAYER=;  
LayerNum, LayerIdentical, LayerActive;

### Contents **Entry for each Layer Definition**

LayerName, LayerIndex,  
LayerIntern,  
LayerExtern, LayerMode;

### Contents **End of Section**

=END=;

### Contents **Element Description**

#### *LayerNum*

[*long*] Number of layer definitions to follow. Valid range is 0 to 500 inclusive.

#### *LayerIdentical*

[*long*] Determines whether the two extended property sets of the layer are used, or only the external set. Possible values are:

0x0000     The extended property set *LayerExtern* is used for output, the extended property set *LayerIntern* is used for display.

0x0001     The extended property set *LayerExtern* is used both for display and output.

#### *LayerActive*

[*long*] Index of the layer that was active when the drawing was saved (referring to *LayerIndex*). Valid range is 0 to 500 inclusive. This value is optional and will be set to 0 if not existing!

#### *LayerName*

[*TEXT64*] Name of the layer, up to 63 characters.

#### *LayerIndex*

[*long*] Index of the layer (1 to 500). Each index may be assigned to one layer only, as this index is used internally to identify a layer.

### LayerIntern

[XPROPERTY] This set of properties defines single or multiple properties to be transmitted to all entities assigned to this layer during display. This transmission has a higher priority than the pen transmission, i.e. it will override properties transmitted by the pen. However if an object property is fixed any transmission will be ignored. A transmission of the layer index will be ignored, of course. This extended property set will only be used if *LayerIdentical* is set to 0x0000.

### LayerExtern

[XPROPERTY] This set of properties defines single or multiple properties to be transmitted to all entities assigned to this layer during output. This transmission has a higher priority than the pen transmission, i.e. it will override properties transmitted by the pen. However if an object property is fixed any transmission will be ignored. A transmission of the layer index will be ignored, of course.

### LayerMode

[*long*] Determines whether the layer is frozen, displayed and/or idle. The display can separately be set for screen display and output. *LayerMode* is a bitwise OR combination of the following values:

```
#define LAYERMODE_DISPLAY 0x0001
#define LAYERMODE_OUTPUT 0x0002
#define LAYERMODE_FREEZE 0x0004
#define LAYERMODE_IDLE 0x0008
#define LAYERMODE_GRAY 0x0010
```

If the section =LAYER= does not exist, assume no layer to be defined (*LayerNum* = 0) and layer 0 to be active. In this case, *LayerIdentical* should be initialized with 0x0000.



## Contents

### Example of a Layer Definition

```
"Hatchings", 100, |Name, Index|
10, |Intern.Flag|
0,0,1.0/1.0/1.0, | ... |
1.0/0.0/0.0,0.0,0,0, | ... |
0, |Intern.Layer|
0, |Extern.Flag|
0,0,1.0/1.0/1.0, | ... |
0.0/0.0/0.0,0.0,0,0, | ... |
0, |Extern.Layer|
3; |Mode|
```

This is the definition of the layer "Hatchings" having the index 100. All entities assigned to this layer will be displayed framed with a red line. During output (to printer, clipboard etc.) they will keep their own properties. The entities are visible, will be output and can be modified.

## Section =WINDOW= (Window Settings)

The section =WINDOW= contains the settings of the view window and up to 4 drawing windows.

A coordinate system contains a set of several settings that influence the drawing's display and output. These settings are: scaling, rotation, distortion, origin position, length unit, line unit, angle unit, number representation, position grid and display grid.

 **Contents** **Element Sequence**  
=WINDOW=;  
[WindowNum](#);

 **Contents** **Entry for each Window Setting**  
[WindowSystem](#), [WindowGrid](#), [WindowSnap](#),  
[WindowXCenter](#), [WindowYCenter](#), [WindowZoom](#);

 **Contents** **End of Section**  
=END=;

 **Contents** **Element Description**

### *WindowNum*

[*long*] Number of window settings to follow. Valid range is 0 to 5 inclusive. The first window is the view window, followed by up to 4 drawing windows.

### *WindowSystem*

[*long*] Index of the coordinate systems that was active in the drawing window when the drawing was saved (referring to *SystemIndex*). Valid range is 0 to 50 inclusive.

### *WindowGrid*

[*BOOL*] Determines whether display grid of the drawing window was active when the drawing was saved.

### *WindowSnap*

[*BOOL*] Determines whether position grid of the drawing window was active when the drawing was saved.

### *WindowXCenter*

### *WindowYCenter*

[*double*] Coordinates of the center-point of the drawing window in millimeters relative to the page's center. The valid range is -1e100 to 1e100.

### *WindowZoom*

[*double*] View size of the drawing window. The valid range is -1e12 to 1e12. This value determines the zoom factor relative to original size.

If the section =WINDOW= does not exist, all windows will be assigned to the coordinate system 0 "\*Standard" without display and position grid, and will display a page overview.



# Section =BITMAP= (Embedded Bitmaps) Contents

The section =BITMAP= contains up to 1000 embedded bitmaps.

## Contents **Element Sequence**

```
=BITMAP=;  
BitmapNum;
```

## Contents **Entry for each Embedded Bitmap**

```
BitmapName, BitmapSize;  
BitmapDataSize[0], BitmapData[0];  
...  
BitmapDataSize[n], BitmapData[n];
```

## Contents **End of Section**

```
=END=;
```

## Contents **Element Description**

### *BitmapNum*

[*long*] Number of embedded bitmaps to follow. Valid range is 0 to 1000 inclusive. The effective maximum number of embedded bitmaps depends on user settings!

### *BitmapName*

[*TEXT256*] Name of the bitmap (including path), up to 255 characters. The first character of this name must always be '#' (Ansi 35) and should be followed by 'B' and either the original file name or any unique identifier. A valid bitmap name would be #B c:\bitmaps\test.bmp or #B 05456-324-3578a8be0002 .

### *BitmapSize*

[*long*] Total size of the bitmap data to follow in bytes. This size must be exactly the sum of all *BitmapDataSize* values to follow.

### *BitmapDataSize[x]*

[*long*] Number of bytes in the following binary data string. Valid range is 1 to 24000 inclusive. We recommend to split the bitmap data into blocks of at most 144 bytes of data each (resulting in line lengths below 255 characters). This will allow the user to load the resulting text file into most standard editors without problems.

### *BitmapData[x]*

[*BINARY*] Binary string containing the bitmap data. The string may contain up to *BitmapDataSize[x]* bytes of data. If it contains less data, the resulting bytes are to be set to zero.

If the section =BITMAP= does not exist, assume no bitmap to be defined (*BitmapNum* = 0).

## Contents **Example of an Embedded Bitmap**

```
"#B E:\\ICONS\\ERASER1.BMP",190;
```

```
14,"0dv^          #x";
24,"*  \"      @      0 !      \";
24,\"          \"      #____\\\";
24,\"jjjjjo_____^k__jj_____z_y_jk_y/_____\";
24,\"oxi^j^oy4/_^jczj_u4?_[\\j&jk_51_____\";
24,\"kzj*jj^5#_^jj*jj_] /7_zj/tjk_w]?_\";
24,\"jj_jjo_oz_^jk^jj__?u_zjgy*k_]yg_\";
24,\"jji\\j_o_y_/^jj?*j__[/_zjk*jk_____\";
8,\"jjjjjo_____\\\";
```

This is the definition of the bitmap "Eraser1.bmp".

## Section =BLOCK= (Block Definitions)

The section =BLOCK= contains block definitions. Inside a drawing file, this section may contain up to 10.000 block definitions, inside a library, it may contain up to 1.000.000 block definitions.

### Contents **Element Sequence**

```
=BLOCK=;  
-Block Definitions-  
=END=;
```

Each block definition starts with an Entity "Block", determining the name and the properties of the block. This entity is followed by a sequence of entities of type "Object" and "Instance". The sequence of block definitions is terminated by an Entity "End of List".

The order of the block definitions is not relevant. Any block reference will not be resolved until *all* blocks are loaded. If blocks contain instances, this leads to a nested structure. Be sure not to create loops where block A contains an instance of block B and block B contains a reference of block A. To avoid a deadlock due to such loops and to avoid a stack overflow during recursion the maximum nesting depth is limited to 20 levels.

## Section =OBJECT= (Objects)

The section =OBJECT= contains up to 2.000.000.000 entities.

## Contents **Element Sequence**

```
=OBJECT=;  
-Entity Sequence-  
=END=;
```

The entity sequence consists of entities of type "Object" as well as "Instance". These entities are the basic elements of each drawing. The sequence of entities is terminated by an Entity "End of List".

## Section =EXIT= (End of File)

The section =EXIT= does contain no further data. It is used to terminate a TVG 4.2 file. After reading the semicolon behind =EXIT=, the interpretation of the file is finished. This section is *not* terminated by an =END= keyword.



## Minimal Drawing (About Drawings)

The smallest valid TVG 4.2 drawing file has the following form:

### Contents **Element Sequence**

```
TommySoftware TVG 4.20;  
=DRAWING=;  
DrawingTitle;  
=END=;  
=EXIT=;
```

# **File Body of Libraries Overview (Structure of TVG 4.2)**

A TVG 4.2 file containing a library starts with the section =LIBRARY=, followed by several sections containing standard attributes and block definitions.

Not every of the sections listed below has to exist in a drawing file. If it does exist, it should follow the order of the sections below (although this order is *not* definite). Section marked with a \* have to exist in any drawing file at the given position!

## **Sections**

Section =LIBRARY= (Library Information)\*

Section =ATTRIB= (Standard Attributes)

Section =BLOCK= (Block Definitions)

Section =EXIT= (End of File)\*

## **About Libraries**

Minimal Library

Standard Libraries

Font Libraries

## Section =LIBRARY= (Library Information)

### Contents **Element Sequence**

```
=LIBRARY=;  
Title,  
Theme,  
Author1,  
Date1,  
Author2,  
Date2,  
Comment;  
=END=
```

### Contents **Element Description**

#### *Title*

[TEXT64] Title of the library. This text usually contains a detailed description of the library. This title is used to identify a library, i.e. it has to be unique over all libraries used!

#### *Theme*

[TEXT64] Theme of the library. This text usually contains a description of the project the library belongs to.

#### *Author1*

[TEXT64] Name of the user that created the library. This text is initialized when creating a library.

#### *Date1*

[TEXT64] Date and time when the library was created. In the English release this text might be, e.g., "Tuesday, October 15 1991, 5:15 pm".

#### *Author2*

[TEXT64] Name of the user that modified this library for the last time. This text is initialized each time the library is saved.

#### *Date2*

[TEXT64] Date and time when this library was saved for the last time. This time is initialized each time the library is saved.

#### *Comment*

[TEXT256] This text contains any further comment on the library.

The section =LIBRARY= has to exist in every library! If desired, this section may contain only the title of the library.

In any case, empty texts at the end of the section do not have to be stated, if e.g. *Comment* is empty, the section can already be ended after *Date2*, which then has to be followed by a semicolon. When reading this section, simply read all texts available (by reading until a semicolon is found) and set all other texts to "".



## Section =ATTRIB= (Standard Attributes)

The section =ATTRIB= contains up to 200 standard attribute definitions. These standard attributes are a default attribute set to be used when creating a new block in a library.

The use of standard attributes eases the creation of blocks that have a huge number of equal attributes, like e.g. when creating a furniture library, where each block shall have the attributes "Part No.", "Price" and "Material". In this case, these three attributes are first defined as standard attributes and then copied to every created block.

## Contents **Element Sequence**

```
=ATTRIB=;  
-Data Section-  
=END=;
```

The standard attributes are stored as a sequence of Attribute Data Blocks, terminated by a Data Block Type 999 (End of List).

Standard attributes do not *replace* the attributes assigned directly to the blocks. If a standard attribute named "Price" is defined, this attribute is *not* automatically defined for all blocks of the library. Instead, it has to be copied to a block during creation to become "active".

## Minimal Library (About Libraries)

The smallest valid TVG 4.2 library file has the following form:

### Contents **Element Sequence**

```
TommySoftware TVG 4.20;  
=LIBRARY=;  
LibraryTitle;  
=END=;  
=EXIT=;
```

## Standard Libraries (About Libraries)

Standard libraries are used to manage frequently used parts and groups, especially if they shall be used in several drawings.

A standard library may contain up to 1.000.000 block definitions having a unique name. The blocks stored in a library will be presented to the user in a hierarchical dialog box, always alphabetically sorted. So the order of the blocks in the library is not relevant.

### Limitations

Block definitions inside libraries may contain instances of either the same library or of other libraries. If an instance shall refer to a block inside the same library, the instance's *LibraryName* value must be set to "\*" .

As a result, libraries cannot contain instances of internal blocks, i.e. of blocks defined in drawing files. When trying to create an external block including an internal block's instance, a warning will appear. In this case, the internal blocks will either have to be resolved or must be transferred into the library.

If a block contains instances, this leads to a nested structure. Be sure not to create loops where block A contains an instance of block B and block B contains a reference of block A. To avoid a deadlock due to such loops and to avoid a stack overflow during recursion the maximum nesting depth is limited to 20 levels.

## Font Libraries (About Libraries)

Font libraries are used to manage font characters, i.e. to build complete ANSI fonts.

A font library should contain 224 blocks representing the characters 32 to 255 of the ANSI character set. The name of these blocks has to start with a character number, coded as a 3-digit decimal number with leading zeros, followed by either a space and additional information or immediately by the name's termination character.

A possible block name for the block representing the character "Capital A" could be "065 A", where 065 is the character index in the ANSI character set and "A" is additional information for the library's creator. The order of the blocks in the font library is not relevant.

## Limitations

Blocks in a font library have to fulfill some requirements. The first is the size and placement of a character inside the block:

### *Character Height*

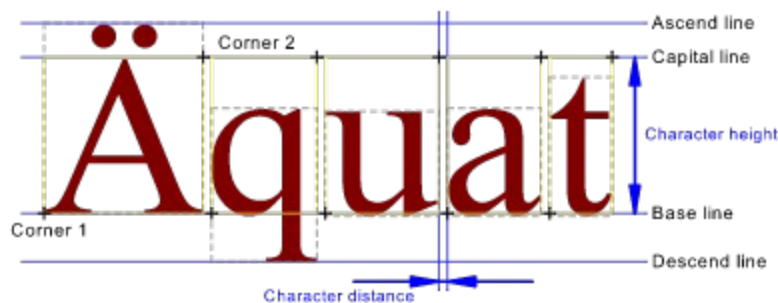
The character height in the library has to be exactly 20 mm. All other font sizes will be generated from that 20 mm character by scaling.

### *Insertion Point*

The insertion point (i.e. the internal origin of the block) has to be at the character's left edge on the baseline. If a character shall overlap the previous character (e.g. the capital W), the insertion point has to be set accordingly.

### *Surrounding Frame*

In a character block, the values *BlockRect.x1*, *BlockRect.y1*, *BlockRect.x2* and *BlockRect.y2* have to be set in a way that they lie at the lower left and the upper right corner of the character's surrounding frame. The difference *BlockRect.x2* - *BlockRect.x1* defines the character's width, i.e. the amount the drawing position is moved after the output of this character. Usually, *BlockRect.x1* should be set to 0.0.



If a font does not include the full ANSI character set (even though it is strongly recommended to implement the full set), it should contain at least the following characters:

- |          |   |
|----------|---|
| 032..126 | Standard character corresponding to ASCII / ANSI.   |
| 160      | The character Ansi 160 is similar to the space Ansi 32, but will not be broken if a word-break is performed.                        |
| 173      | The character is similar to the minus sign or hyphen respectively - (Ansi 45), but will not be broken if a word-break is performed. |
| 176      | The character ° is used to mark an angle value.   |
| 177      | The character ± is frequently used in tolerances.   |
| 216      | The character Ø is used to mark a diameter value.   |

In addition to standard ANSI characters, a font library should include the following character that is not defined by ANSI:

128            This character is used to mark an area value. It should be a framed square whose side length is equal to a non-capital character's height.

Even though font libraries can theoretically contain any types of entities, character blocks should contain only entities of type "Object". In order to allow fast display of characters, only the following object types should be used:

Object 00 "Line"

Object 01 "Hatching"

Object 05 "Circle"

Object 06 "Circular Arc"

Object 07 "Circular Sector"

Object 08 "Circular Segment"

Object 12 "Curve"

Object 13 "Surface"

Object 15 "Ellipse"

Object 16 "Elliptical Arc"

Object 17 "Elliptical Sector"

Object 18 "Elliptical Segment"

Instances and attributes inside font libraries will be ignored! All properties of character blocks will be reset during load time, i.e. their properties will not be maintained!

