



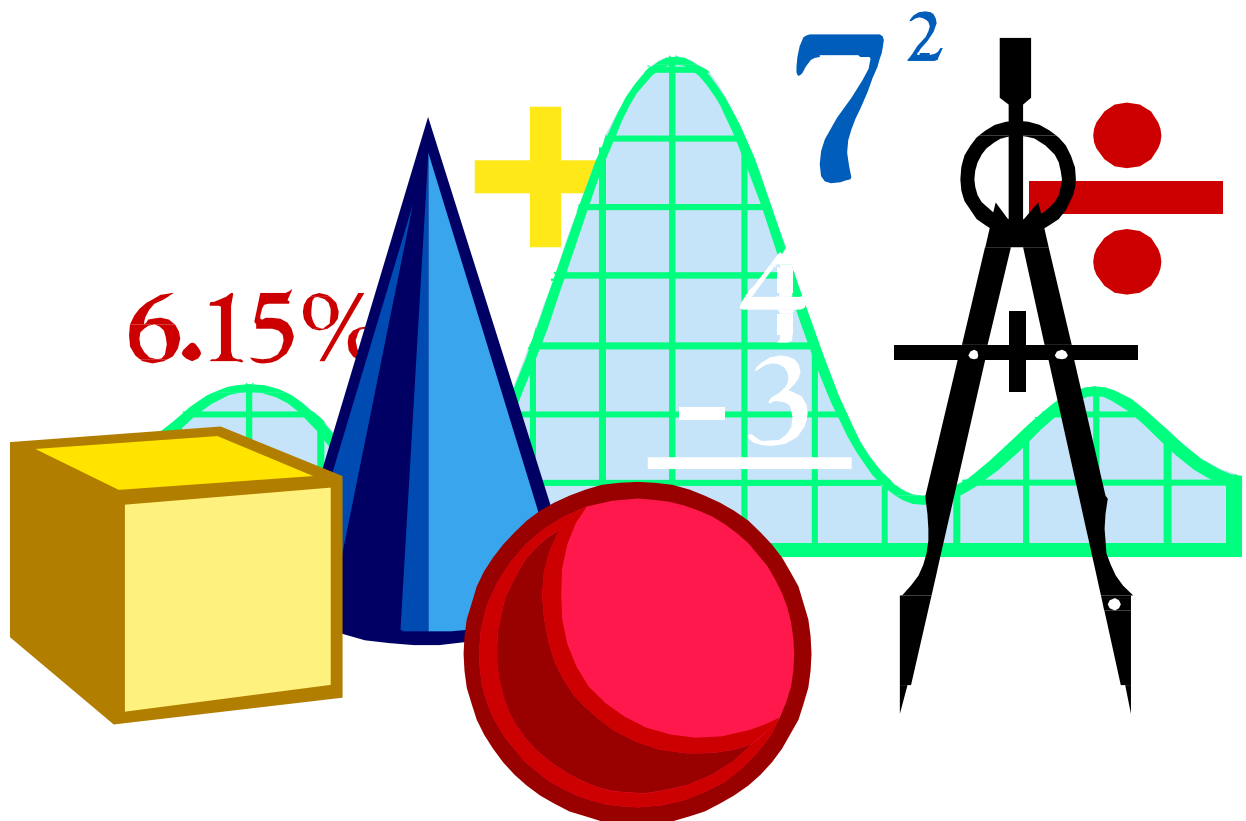
# The Gerber File Format Specification

---

A format developed by Ucamco

February 2014

Revision J1



# Contents

---

<b>Contents .....</b>	<b>i</b>
<b>Figures.....</b>	<b>v</b>
<b>Tables .....</b>	<b>vii</b>
<b>Preface.....</b>	<b>viii</b>
<b>1 Introduction .....</b>	<b>9</b>
1.1 About this Document .....	9
1.1.1 Scope .....	9
1.1.2 Formatting and Syntax Rules .....	9
1.1.3 References.....	9
1.2 Conformance.....	10
1.3 Copyright and Intellectual Property.....	11
1.4 History of the Gerber File Format .....	12
1.5 Record of Revisions .....	13
1.5.1 Revision I1 .....	13
1.5.2 Revision I2 .....	14
1.5.3 Revision I3 .....	14
1.5.4 Revision I4 .....	14
1.5.5 Revision J1.....	14
1.6 Info, Questions & Feedback .....	15
1.7 About Ucamco.....	15
<b>2 Overview of the Gerber Format .....</b>	<b>16</b>
2.1 File Structure .....	16
2.2 Graphics Objects .....	16
2.3 Operation Codes .....	17
2.4 Apertures.....	17
2.5 Stroking .....	17
2.6 Dark and Clear Polarity .....	18
2.7 Graphics State.....	19
2.8 Attributes .....	20
2.9 Example Files .....	20
2.9.1 Example 1: Two square boxes .....	21
2.9.2 Example 2: Use of levels and various apertures .....	22
2.9.3 Example 3: A drill file .....	27
2.10 Glossary.....	30

<b>3 Syntax .....</b>	<b>32</b>
3.1 Character Set .....	32
3.2 Variable Types .....	32
3.2.1 Integers .....	32
3.2.2 Decimals .....	32
3.2.3 Names .....	32
3.2.4 Strings .....	32
3.3 Data Blocks .....	33
3.4 Commands .....	34
3.4.1 Commands Overview .....	34
3.4.2 Function Codes .....	35
3.4.3 Coordinate Data Blocks .....	35
3.4.4 Parameters .....	36
 <b>4 Graphics .....</b>	 <b>38</b>
4.1 Graphics Overview .....	38
4.2 Operation Codes (D01/D02/D03) .....	41
4.3 Linear Interpolation (G01) .....	44
4.3.1 Data Block Format .....	44
4.4 Circular Interpolation (G02/G03, G74/G75) .....	45
4.4.1 Arc Overview .....	45
4.4.2 Arc Definition .....	45
4.4.3 Single Quadrant Mode .....	46
4.4.4 Multi Quadrant Mode .....	50
4.4.5 Arc Example .....	51
4.4.6 Numerical instability in multi quadrant (G75) arcs .....	52
4.4.7 Using G74 or G75 can result in a different image .....	52
4.5 Regions (G36/G37) .....	53
4.5.1 Region Overview .....	53
4.5.2 Example: a simple contour .....	54
4.5.3 Examples: how to start a single contour .....	56
4.5.4 Examples: Use D02 to start a second contour .....	57
4.5.5 Example file: Overlapping contours .....	58
4.5.6 Example file: Non-overlapping and touching .....	59
4.5.7 Example file: Overlapping and touching .....	60
4.5.8 Using levels to create holes .....	61
4.5.9 Example: a simple cut-in .....	65
4.5.10 Examples: coincident draws .....	67
4.5.11 Examples: valid and invalid cut-ins .....	69
4.6 Comment (G04) .....	74
4.7 End-of-file (M02) .....	74
4.8 FS – Format Specification .....	74
4.8.1 Coordinate Format .....	74
4.8.2 Zero Omission .....	75
4.8.3 Absolute or Incremental Notation .....	75

4.8.4 Data Block Format.....	76
4.8.5 Examples .....	76
4.9 MO – Mode.....	76
4.9.1 Data Block Format.....	76
4.9.2 Examples .....	77
4.10 AD - Aperture Definition .....	77
4.10.1 Syntax Rules.....	77
4.10.2 Standard Apertures .....	79
4.10.3 Zero-size apertures .....	84
4.10.4 Examples .....	84
4.11 AM - Aperture Macro .....	84
4.11.1 Data Block Format.....	85
4.11.2 Primitives .....	86
4.11.3 Parameter Contents .....	98
4.11.4 Syntax Rules.....	98
4.11.5 Examples .....	103
4.12 SR – Step and Repeat .....	106
4.12.1 Data Block Format.....	107
4.12.2 Examples .....	107
4.13 LP – Level Polarity .....	108
4.13.1 Data Block Format.....	108
4.13.2 Examples .....	108
<b>5 Attributes .....</b>	<b>109</b>
5.1 Attributes Overview .....	109
5.2 File attributes.....	109
5.2.1 Standard File Attributes .....	110
5.3 Aperture Attributes.....	115
5.3.1 Aperture Attributes Overview .....	115
5.3.2 Aperture Attributes Commands .....	115
5.3.3 Standard Aperture Attributes .....	117
5.3.4 Examples .....	120
<b>6 Most Common Errors &amp; Bad Practice .....</b>	<b>122</b>
6.1 Most Common Errors .....	122
6.2 Most Common Bad Practices .....	123
<b>7 Deprecated Format Elements .....</b>	<b>125</b>
7.1 Coordinate Data Blocks without Operation Code.....	125
7.2 X1200Y1000* .....	125
7.3 Open Contours .....	125
7.4 Deprecated Commands.....	126
7.4.1 AS – Axis Select.....	129
7.4.2 IN - Image Name .....	130

7.4.3 IP – Image Polarity .....	131
7.4.4 IR – Image Rotation .....	132
7.4.5 LN – Level Name.....	134
7.4.6 MI – Mirror Image .....	135
7.4.7 OF - Offset .....	136
7.4.8 SF – Scale Factor.....	137
7.5 Obsolete Standard Gerber (RS-274-D).....	138
7.5.1 Standard Gerber must not be used.....	138
7.5.2 Origin and purpose of Standard Gerber.....	138
7.5.3 Standard Gerber is a NC format, not an image format.....	140
7.5.4 A fallacy.....	140

# Figures

1.	Linear interpolation using rectangle aperture: example 1	18
2.	Linear interpolation using rectangle aperture: example 2	18
3.	Example 1: two square boxes	21
4.	Example 2: various shapes	23
5.	Example 3: drill file	27
6.	Gerber file structure	34
7.	Arc with a non-zero deviation	46
8.	Nonsensical center point	46
9.	Single quadrant mode	48
10.	Single quadrant mode example: arcs and draws	49
11.	Single quadrant mode example: resulting image	49
12.	Multi quadrant mode example: resulting image	51
13.	Simple contour example: the segments	55
14.	Simple contour example: resulting image	55
15.	Use of D02 to start an new non-overlapping contour	57
16.	Use of D02 to start an new overlapping contour	58
17.	Use of D02 to start an new non-overlapping contour	59
18.	Use of D02 to start an new overlapping and touching contour	60
19.	Resulting image: first level only	62
20.	Resulting image: first and second levels	63
21.	Resulting image: first, second and third levels	63
22.	Resulting image: all four levels	64
23.	Simple cut-in: the segments	66
24.	Simple cut-in: the image	66
25.	Coincident edges in contours, example 1	67
26.	Coincident edges in contours, example 2	68
27.	Cut-in example 2: valid, coincident segments	70
28.	Cut-in example 2: resulting image	71
29.	Cut-in example 3: invalid, overlapping segments	73
30.	Circles with different holes	79
31.	Rectangles with different holes	80
32.	Obrounds with different holes	81
33.	Polygons with different holes	83
34.	Circle primitive	88
35.	Line (vector) primitive	89
36.	Line (center) primitive	90

37.	Line (lower left) primitive	91
38.	Outline primitive	92
39.	Polygon primitive	94
40.	Moiré primitive	95
41.	Thermal primitive	97
42.	Rotated triangle	105
43.	Step and Repeat	106

# Tables

---

Document conventions	9
Graphics parameters	39
Function codes	40
Quadrant modes	45
Arithmetic operators	99
Standard file attributes	110
.FileFunction file attribute values	112
.Part file attribute values	113
Aperture attribute parameters	115
.AperFunction aperture attribute values	119
Reported Common Errors	123
Common poor/good practices	124
Deprecated codes	126
Deprecated parameters	127
Deprecated graphics state variables	128



# Preface

---

The Gerber file format is the de facto standard for printed circuit board (PCB) image data transfer. Every PCB design system outputs Gerber files and every PCB front-end engineering system inputs them. Implementations are thoroughly field-tested and debugged. Its widespread availability allows PCB professionals to exchange image, drill and route data securely and efficiently. It has been called “the backbone of the electronics manufacturing industry”.

The Gerber file format is simple, compact and unequivocal. It describes an image with very high precision, up to 1 nm. It is complete: one single file describes one single image. It is portable and easy to debug by its use of printable 7-bit ASCII characters. A well-constructed Gerber file precisely defines the PCB image and the functions of the different image elements. This results in a safe, reliable and productive transfer of PCB data from design to manufacturing.

Unfortunately, some applications generate poorly constructed Gerber files. Especially troublesome is the use of painting or stroking to create pads and copper areas. Poorly constructed files take longer to process, require more manual work and increase the risk of errors. Such problems are sometimes incorrectly blamed on the Gerber file format itself. They may result from misunderstanding the specification or the capabilities of the format. With more than 25 years of experience in CAM software we at Ucamco know which areas are most often misunderstood. We continuously refine and extend the format specification to make Gerber files safer and more efficient, making fabrication more reliable, faster and cheaper.

The current Gerber file format is RS-274X or Extended Gerber. Standard Gerber or RS-274-D is deprecated and obsolete. It is not an image description format but an NC format constrained by the technology from the 1960s and 1970s. Standard Gerber does not have a single advantage over Extended Gerber. It has many disadvantages. It is simply not suited for reliable automatic image data transfer. Do not use Standard Gerber any longer. Using Standard Gerber rather than Extended Gerber is a self-inflicted competitive disadvantage.

Although other data transfer formats have come into the market, they have not displaced the Gerber file format. The reason is simple. Most of the problems in data transfer are due not to limitations in the Gerber file format but to poor practices. To quote a PCB manufacturer: “If we would only receive proper Gerber files, it would be a perfect world.” The new formats are more complex and less transparent to the user. New implementations inevitably have bugs. Common poor practices in more complex formats make matters worse, not better. The industry has not adopted new formats. Gerber remains the standard.

The emergence of Gerber as a standard for image exchange is the result of efforts by many individuals who developed outstanding software for Gerber files. Without their dedication the widespread acceptance of a de-facto standard could not have been achieved. Ucamco thanks these dedicated individuals.

Karel Tavernier  
Managing Director,  
Ucamco

# 1 Introduction

## 1.1 About this Document

### 1.1.1 Scope

This document specifies the Gerber file format, a 2D vector format for representing a binary image. The specification is intended for the developers and users of Gerber software.

The Gerber format is widely used in the printed circuit board (PCB) industry, but also in other industries. The specification sometimes uses the terminology of PCB computer-aided manufacturing (CAM) because it is tightly related to the semantics and interpretation of the Gerber file format. A basic knowledge about PCB CAM is helpful in understanding this specification.

### 1.1.2 Formatting and Syntax Rules





The following font formatting rules are used in this specification:

- Examples of Gerber file content are written with mono-spaced font, e.g. `X0Y0D02*`
- Syntax rules are written with bold font, e.g. **<Elements set>: {<Elements>}**

The syntax rules are described using the following conventions:

- Optional items enclosed in square brackets, e.g. **[<Optional element>]**
- Items repeating zero or more times are enclosed in braces, e.g. **{<Element>}**
- **<Elements set>: <Element>{<Element>}**
- Alternative choices are separated by the '|' character, e.g. **<Option A>|<Option B>**  
Grouped items are enclosed in regular parentheses, e.g. **(A|B)(C|D)**

The following conventions are used:

 <b>Note:</b>	Provides essential extra information.
 <b>Tip:</b>	Provides useful extra information.
 <b>Example:</b>	Contains examples of file syntax and semantics.
 <b>Warning:</b>	Contains an important warning.

*Document conventions*

### 1.1.3 References

*American National Standard for Information Systems — Coded Character Sets — 7-Bit  
American National Standard Code for Information Interchange (7-Bit ASCII), ANSI X3.4-1986*

Bible, Mark 7:35

## 1.2 Conformance

A Gerber file must comply with all requirements of the specification. A file violating any requirement of the specification or containing any invalid part is wholly invalid. If the interpretation of a construct is not specified or not obvious then that construct is invalid. An invalid Gerber file is meaningless and does not represent an image.

A Gerber file reader must render a valid file according to this specification. There is no mandatory behavior on reading an invalid Gerber file, except that a warning must be given on unknown codes and parameters to prepare for future extensions of the Gerber format. Consequently, it is not mandatory to report any other error in the Gerber file as this imposes an unreasonable burden on readers. A reader is allowed to generate an image on an invalid file, e.g. as a diagnostic help or in an attempt to guess the intended image by 'reading between the lines'. However, as an invalid Gerber file is meaningless, such an image is neither correct nor incorrect.

A Gerber file writer must write files according to this specification. All the non-deprecated elements can be used. The writer is not required to take into account limitations or errors in particular readers. The writer may assume that a valid file will be processed correctly.

The responsibilities are obvious and plain. Writers must write valid files and readers must process such files correctly. Writers are not responsible to navigate around problems in the readers, nor are readers responsible to solve problems in the writers.

Current Gerber file writers must not use deprecated constructs. Current Gerber file readers may support or not support deprecated constructs as they may be present in legacy files.

Standard Gerber (RS-274-D) is obsolete and therefore non-conforming. The responsibility for misunderstandings of its non-standardized wheel file rests solely with the party that decided to use Standard Gerber rather than Extended Gerber where coordinate format and apertures are unequivocally and formally standardized.

This document is the sole specification of the Gerber format. Gerber viewers, however useful, are not the reference and do not overrule this document.

## 1.3 Copyright and Intellectual Property

© Copyright Ucamco NV, Gent, Belgium

All rights reserved. No part of this document or its content may be re-distributed, reproduced or published, modified or not, in any form or in any way, electronically, mechanically, by print or any other means without prior written permission from Ucamco.

The information contained herein is subject to change without prior notice. Revisions may be issued from time to time. This document supersedes all previous versions. Users of the Gerber Format, especially software developers, must in all cases consult [www.ucamco.com](http://www.ucamco.com) to determine whether any changes have been made.

Ucamco developed the Gerber file format. The Gerber file format, this document and all intellectual property contained in it are solely owned by Ucamco. Gerber Format is a Ucamco registered trade mark. Ucamco does not grant a license to the intellectual property contained in this document by publishing it. Ucamco encourages users to apply for a license to develop Gerber based software.

By using this document, developing software interfaces based on this format, or using the name Gerber format, users agree not to (i) rename the Gerber Format; (ii) associate the Gerber Format with data that does not conform to the Gerber file format specification; (iii) develop derivative versions, modifications or extensions without prior written approval by Ucamco; (iv) make alternative interpretations of the data; (v) communicate that the Gerber file format is not owned by Ucamco or owned by anyone other than Ucamco. Developers of software interfaces based on this format specification commit to make all reasonable efforts to comply with the latest specification.

The material, information and instructions are provided AS IS without warranty of any kind. There are no warranties granted or extended by this document. Ucamco does not warrant, guarantee or make any representations regarding the use, or the results of the use of the information contained herein. Ucamco shall not be liable for any direct, indirect, consequential or incidental damages arising out of the use or inability to use the information contained herein. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Ucamco. All product names cited are trademarks or registered trademarks of their respective owners.

## 1.4 History of the Gerber File Format

The Gerber file format derives its name from the former Gerber Systems Corp., a leading supplier of photoplotters in its time.

Originally, Gerber used a subset of the EIA RS-274-D format as standard input format for its vector photoplotters. This subset became known as Standard Gerber. Vector photoplotters are NC machines, and Standard Gerber is an NC format to drive such machines. It is not really an image description standard: it requires external data such as aperture shapes to be converted to an image. In subsequent years, Gerber extended the input format for its range of PCB devices and it actually became a family of capable image description formats.

In 1998 Gerber Systems Corp. was taken over by Barco and incorporated in its PCB division Barco ETS, now Ucamco. The variants in the family were pulled together and standardized by the publication of the first version of this document. It has become the de-facto standard for PCB image data. It is sometimes called “the backbone of the electronics industry”. Several revisions of the specification were published over the years, clarifying it and adapting it to current needs.

The Standard Gerber or RS-274-D format, now obsolete, was deprecated. It deserves a place of honor in the Museum for the History of Computing but it does not deserve a place in modern workflows.

## 1.5 Record of Revisions

### 1.5.1 Revision I1

**General.** The entire specification has been reviewed for clarity. Existing warnings and notes were clarified and new ones added. The quality of the text and the drawings has been improved.

**Deprecated elements.** Format elements that are rarely used and superfluous or prone to misunderstanding have been deprecated. They are grouped together in the second part of this document. The first part contains the current format, which is clean and frugal. *We urge all creators of Gerber files no longer to use deprecated elements of the format.*

**Graphics state and operation codes.** The underlying concept of the *graphics state* and operation codes is now explicitly described. See section 2.3 and 2.2. *We urge all providers of Gerber software to review their implementation in the light of these sections.*

**Defaults.** In previous revisions the definitions of the default values for the modes were scattered throughout the text, or were sometimes omitted. All default values are now unequivocally specified in an easy-to-read table. See 2.2. *We urge all providers of Gerber software to review their handling of defaults.*

**Rotation of macro primitives.** The rotation center of macro primitives was clarified. See 4.11.2. *We urge providers of Gerber software to review their handling of the rotation of macro primitives.*

**G36/G37.** The whole section is now much more specific. An example was added to illustrate how to use of polarities to make holes in areas, a method superior to cut-ins. See 4.5. *We urge all providers of Gerber software to review their handling of G36/G37 and to use layers to create holes in areas rather than using cut-ins.*

**Coordinate data blocks.** Coordinate data without D01/D02/D03 in the same data block create some confusion. It therefore has been deprecated. See 3.4.3. *We urge all providers of Gerber software to review their output of coordinate data in this light.*

**Maximum aperture number (D-code).** In previous revisions the maximum aperture number was 999. This was insufficient for current needs and numerous files in the market use higher aperture numbers. We have therefore increased the limit to the largest number that fits in a signed 32 bit integer.

**Standard Gerber.** We now define Standard Gerber in relation to the current Gerber file format. Standard Gerber is deprecated because it has many disadvantages and not a single advantage. *We urge all users of Gerber software not to use Standard Gerber.*

**Incremental coordinates.** These have been deprecated. Incremental coordinates lead to rounding errors. *Do not use incremental coordinates.*

**Name change: area and contour instead of polygon.** Previous revisions contained an object called a polygon. As well as creating confusion between this object and a polygon aperture, the term is also a misnomer as the object can also contain arcs. These objects remain unchanged but are now called areas, defined by their contours. This does not alter the Gerber files.

**Name change: level instead of layer.** Previous revisions of the specification contained a construct called a layer. As these were often confused with PCB layers they have been renamed as levels. This does not alter the Gerber files.

#### 1.5.1.1 Acknowledgement

This revision of the specification was developed by Karel Tavernier and Rik Breemeersch, advised by Ludek Brukner, Artem Kostyukovich, Jiri Martinek, Adam Newington, Denis Morin, Karel Langhout and Dirk Leroy.

We thank anyone who has helped us with questions, remarks or suggestions - they are too many to mention by name. However, we explicitly thank Paul Wells-Edwards who contributed substantially with insightful comments.

### **1.5.2 Revision I2**

The “exposure on/off” modifier in macro apertures and the holes in standard apertures are sometimes incorrectly implemented. These features were explained in more detail. Readers and writers of Gerber files are urged to review their implementation in this light.

### **1.5.3 Revision I3**

Questions about the order and precise effect of the deprecated parameters MI, SF, OF, IR and AS were clarified. Coincident contour segments were explicitly defined, see 4.5.1.

### **1.5.4 Revision I4**

The parameters LN, IN and IP were deprecated.

The regions overview section 4.5.1 was expanded and examples were added different places in 4.5 to further clarify regions. The chapters on function codes and syntax were restructured. The constraints on the thermal primitive parameters were made more explicit. Wording was improved in several places.

### **1.5.5 Revision J1**

Chapter 5 Attributes was added, extending the Gerber format with attributes. This important extension is called X2 or the second extension. X2 is backward compatible: attributes do not affect image generation.

X2 was developed by Karel Tavernier, Ludek Brukner and Thomas Weyn. They were assisted by a review group consisting of Roland Polliger, Luc Samyn, Wim De Greve, Dirk Leroy and Rik Breemeersch.

## 1.6 Info, Questions & Feedback

Correspondence regarding this publication or questions about the Gerber File Format can be mailed to [gerber@ucamco.com](mailto:gerber@ucamco.com)

For more information see [www.ucamco.com](http://www.ucamco.com)

## 1.7 About Ucamco

Ucamco (former Barco ETS) is a market leader in PCB CAM software and imaging systems. We have more than 25 years of continuous experience developing and supporting leading-edge front-end tooling solutions for the global PCB industry. We help fabricators world-wide raise yields, increase factory productivity, and cut enterprise risks and costs.

Today we have more than 1000 laser photoplotters and 5000 CAM systems installed around the world with local support in every major market. Our customers include the leading PCB fabricators across the global spectrum. Many of them have been with us for more than 20 years.

Key to this success has been our uncompromising pursuit of engineering excellence in all our products. For 25 years our product goals have been best-in-class performance, long-term reliability, and continuous development to keep each user at the cutting-edge of his chosen technology.

For more information see [www.ucamco.com](http://www.ucamco.com).



## 2 Overview of the Gerber Format

### 2.1 File Structure

The Gerber file format is a vector 2D binary image file format: the image is defined by resolution-independent graphics objects.

A single Gerber file specifies a single image. A Gerber file is complete: it does not need external files or parameters to be interpreted. One Gerber file represents one image. One image needs only one file.

A Gerber file is a stream of *commands*. A command can contain *function codes*, *parameters* and/or *coordinate data*. The stream of commands generates a stream of graphics object which combined produce the final image.

A Gerber file can be processed in a single pass. This imposes constraints on the sequence in the commands. For example, *the coordinate format* and *unit* must be known to convert coordinate data to coordinates. Format and unit are set by the FS and MO parameters, which therefore must precede the first coordinate.

A Gerber file must end with the end of file data block 'M02\*'.

The preferred extension is ".gbr" or ".GBR"



#### Example

```
G04 Set coordinate format and units in the file header*  
%FSLAX25Y25*%  
%MOIN*%  
G04 From here coordinate data can be interpreted*  
...  
M02*
```

### 2.2 Graphics Objects

A Gerber file creates an ordered stream of graphics objects. A graphics object has an image (shape, size), a position in the plane (coordinates) and a polarity (dark or clear).

There are four types of graphics objects:

- ❑ A *draw* is a straight line segment with a given thickness and either round or square line endings.
- ❑ An *arc* is circular arc with a given thickness, always with round endings.
- ❑ A *flash* is a replication of an aperture at a given location. An aperture is a basic geometric shape defined earlier in the file. Apertures are typically flashed many times.
- ❑ A *region* is an area of defined by its contour. A contour is constructed with of linear and circular segments.

In PCB copper layers, draws and arcs are typically used to create tracks, flashes to create pads and regions to create copper areas.

## 2.3 Operation Codes

D01, D02 and D03 are the *operation codes*. They create the graphics objects by operating on coordinate data. A coordinate data block contains the coordinate data followed a single operation code: each operation code is associated with a single coordinate pair and vice versa.



### Example:

```
X100Y100D01*  
X200Y200D02*  
X300Y-400D03*
```

The operation codes have the following effect.

- ❑ D02 moves the current point to the coordinate pair. Nothing is created. This is also called a lights-off move.
- ❑ D01 creates a straight or circular line segment by interpolating from the current point to the coordinate pair. This is also called a lights-on move. When region mode is off these segments are converted to draw or arc objects by stroking them with the current aperture, see 2.4. When region mode is on these segments form a contour defining a region, see 4.5.
- ❑ D03 creates a flash object by replicating the current aperture at the coordinate pair.

The operation codes are controlled by the graphics state, see 2.6.

## 2.4 Apertures

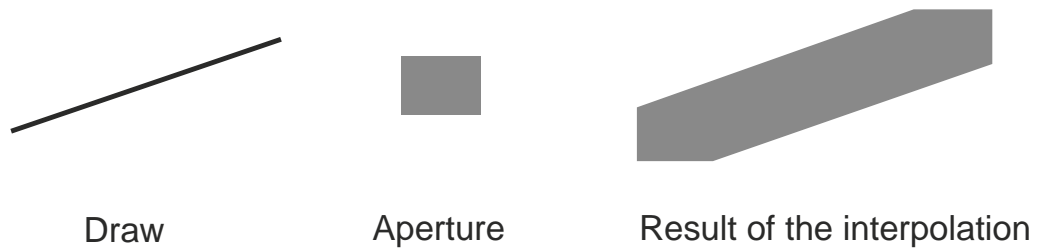
An aperture is a basic geometric shape used for flashing or stroking. There are two kinds of apertures: *standard apertures* and *special apertures*:

- Standard apertures are pre-defined: the circle, rectangle, obround and regular polygon. See 4.10. The AD parameter assigns a D-code to a standard aperture and defines its size and other parameters.
- Special apertures, also called macro apertures, are defined with an AM (Aperture Macro) parameter. They are identified by their given name. Any shape can be defined. See 4.11. The AD parameter also assigns a D-code to a special aperture and defines its size and other parameters.

An aperture has a *flash point*. When an aperture is flashed its flash point is positioned at the coordinates of the D03 data block. The flash point of a standard aperture is its geometric center. The flash point of a special aperture is the origin of the coordinates used in the AM parameter.

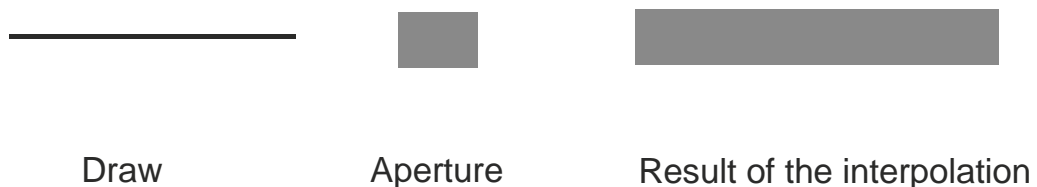
## 2.5 Stroking

A *draw object* is created by stroking a straight line segment with a solid circle or solid rectangle standard aperture. If stroked with a circle aperture the draw has round endings and its thickness is equal to the diameter of the circle. The effect of stroking a line segment with a rectangle aperture is illustrated below:



### 1. Linear interpolation using rectangle aperture: example 1

If the rectangle aperture is aligned with the draw the result is a draw with a straight line ending:



### 2. Linear interpolation using rectangle aperture: example 2



**Note:** The rectangle is *not* automatically rotated to align with the draw.

An *arc object* is created by stroking an arc segment with a solid circle standard aperture. The arc has round endings and its thickness is equal to the diameter of the circle. An arc segment cannot be stroked with a rectangle.

The only apertures allowed for stroking are the solid circle and the solid rectangle *standard* apertures (line segments only for the rectangle). Other standard apertures or special apertures, whatever their final shape, cannot be used for stroking.

A zero size circular aperture can be used for stroking. They create graphic objects without image, which can be used to transfer non-image information, e.g. an outline.

Zero-length draws and arcs are allowed. The resulting image is the same as the flashed aperture. However, the graphic object is a draw or arc, not a flash.

Any valid aperture can be flashed.

## 2.6 Dark and Clear Polarity

The final image of the Gerber file is created by superimposing the objects in the order of the stream. Objects can overlap. A dark object darkens (marks, paints, exposes) its image in the plane. A clear object clears (unmarks, rubs, erases, scratches) its image in all the lower levels. In other words, after superposing a clear object, its image is clear, whatever objects were there before. Subsequent dark objects may again darken the cleared area.

A Gerber file consists of a sequence of levels. Syntactically a level is a set of consecutive commands. For image generation a level is a consecutive set of graphics object with the same polarity. A Gerber file can be viewed as a sequence of levels superimposed in the order of appearance in the file.

The order of the objects within a level does not affect the final image. The order of the levels, however, typically affects the final image.

The LP parameter starts a new level and sets its polarity, see 4.12.

## 2.7 Graphics State

A Gerber file defines a graphics state after each command. The operation codes are controlled by the graphics state, see 2.3.

The most important graphics state variable is the *current point*. The current point is a point in the image plane. The current point is set implicitly by coordinate data blocks: after processing a coordinate data block the current point is set to the coordinate in that block.

All other graphics state variables are set by codes or parameters. They are modal, meaning that their value remains constant until explicitly changed.

The table below lists the graphics state variables. The column 'Fixed or changeable' indicates whether a variable remains fixed during the processing of a file or whether it can be changed. The column 'Value at the beginning of a file' is the default value at the beginning of each file; if the default is undefined the variable must be explicitly set before it is first used.

Graphics state variable	Value range	Fixed or changeable	At the beginning of the file
Coordinate format	See FS parameter	Fixed	Undefined
Unit	Inch or mm See MO parameter.	Fixed	Undefined
Current aperture	Standard or macro aperture. See AD and AM parameters.	Changeable	Undefined
Quadrant mode	Single-, Multi-Quadrant See G74, G75	Changeable	Undefined
Interpolation mode	See G01, G02, G03	Changeable	Undefined
Current point	Point in plane	Changeable	(0,0)
Step & Repeat	See SR parameter	Changeable	1,1,-,-
Level polarity	Dark, Clear See LP parameter.	Changeable	Dark
Region mode	On/Off. See 4.5.	Changeable	Off

Graphics state variables



**Note:** It is more robust to set the modes explicitly at the beginning of the file rather than rely on the defaults.

The graphics state determines the effect of an operation code. If a state variable is undefined when it is required to process a coordinate data block the Gerber file is *invalid*. If a graphics state variable is not needed then it can remain undefined. For example, if the interpolation mode G02 or G03 (circular interpolation) the quadrant mode is required to process coordinates and

thus must be defined; if the interpolation mode is G01 (linear interpolation) then the quadrant mode is not needed may remain undefined.

## 2.8 Attributes

Attributes add meta-information to individual objects or to the whole file. An example of metadata is the function of a flash – it is an *SMD pad*, or a *via pad*, etc. Another example is the function of the image represented by the file, making it clear if it is the *top solder mask*, or the *bottom copper layer*, etc.

Attributes do not affect the image. A Gerber reader that ignores or does not recognize attributes will generate the correct image.

The attribute syntax of the Gerber format provides a flexible and standardized way to add metadata, independent of the application and of the attribute semantics. It supports both *standard attributes* and *custom attributes*. Standard attributes and their semantics are part of this specification. Custom attributes can be created by third parties to extend the format with proprietary metadata.

Standard attributes are intended for the PCB design to fabrication workflow. Where an image simply needs to be rendered, attributes are not necessary. When the image transfers PCB data from design to fabrication the attributes are vital. Attributes convey the design intent from CAD to CAM. This is sometimes rather grandly called “adding intelligence to the image”. Without the information conveyed by attributes the PCB cannot be manufactured correctly.

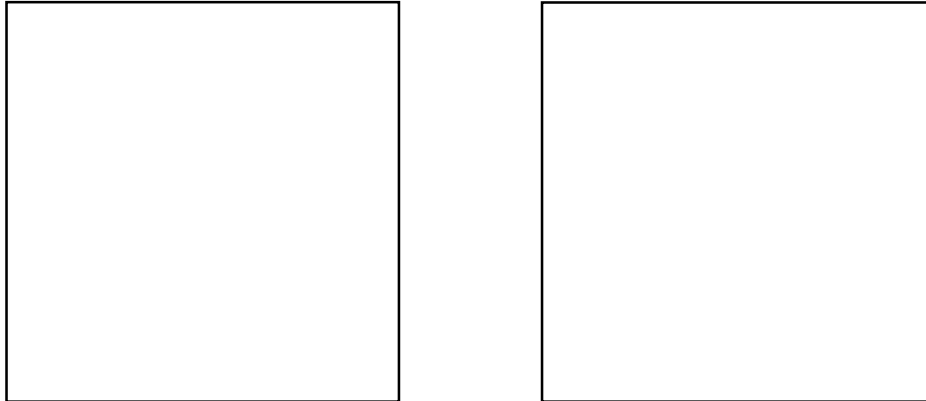
When generating data for fabrication it is therefore important to add the standard attributes. Please note that the use of standard attributes is not all “all or nothing”. When Gerber file writers cannot include all the attributes or are unsure of their use, they are encouraged to provide those attributes with which they are comfortable. Partial information is better than no information at all. However, for professional PCB production including the file function attribute is a must.

## 2.9 Example Files

These annotated samples illustrate the use of the elements of the Gerber file format. If you are not familiar with the Gerber file format they can give you a feel for it which will make it easier to read the specification.

## 2.9.1 Example 1: Two square boxes

Example 1 is a single-level image with two square boxes.

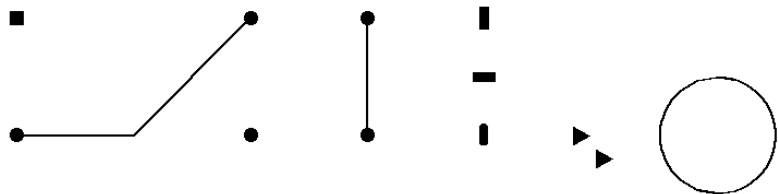
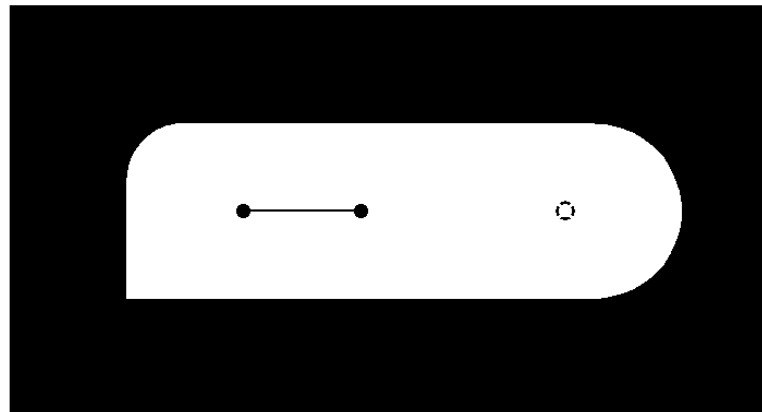


### 3. Example 1: two square boxes

G04 Ucamco ex. 1: Two square boxes*	A comment
%FSLAX25Y25*%	Coordinate format specification: Leading zero's omitted Absolute coordinates 2 digits in the integer part 5 digits in the fractional part
%MOMM*%	Unit set to mm
%LPD%	Start a new level with dark polarity
%ADD10C,0.010*%	Define aperture with D-code 10 as a 0.01 mm circle
D10*	Select aperture with D-code 10 as current aperture
X0Y0D02*	Move to (0, 0)
G01X500000Y0D01*	Linear interpolation (draw) to (5, 0) with D10
G01Y500000D01*	Draw to (5, 5) with D10
G01X0D01*	Draw to (0, 5) with D10
G01Y0D01*	Draw to (0, 0) with D10
X600000D02*	Move to (6, 0)
G01X1100000D01*	Draw to (11, 0) with D10
G01Y500000D01*	Draw to (11, 5) with D10
G01X600000D01*	Draw to (6, 5) with D10
G01Y0D01*	Draw to (6, 0) with D10
M02*	End of file

## 2.9.2 Example 2: Use of levels and various apertures

Example 2 illustrates the use of levels and various apertures.



#### 4. Example 2: various shapes

G04 Ucamco ex. 2: Shapes*	A comment statement
%TF.GerberVersion,J1*%	States the file complies with revision J1 of the spec
%TF.Part,Other*%	The file is not a layer of a known PCB part. Obviously, it is just an example.
%FSLAX23Y23*%	Format specification: Leading zero's omitted Absolute coordinates Coordinates format is 2.3
%MOIN*%	Units are inches  1/1000 inch is a very low resolution, not suited for production. It is used here to make the file easier to read by a human.
%LPD*%	Start a new level with dark polarity. This command confirms the default and makes the intention unequivocal.
%SRX1Y1I0J0*%	Set 'Step and Repeat' to 1 for both X and Y. This command confirms the default and makes the intention unequivocal.
G04 Define Apertures*	Comment
%AMTARGET125*	Aperture macro 'TARGET125'
6,0,0,0.125,.01,0.01,3,0.00 3,0.150,0*%	Moiré primitive
%AMTHERMAL80*	Aperture macro 'THERMAL80'
7,0,0,0.080,0.055,0.0125,45 *%	Thermal primitive
%ADD10C,0.01*%	Aperture definition: D10 is a circle with diameter 0.01 inch
%ADD11C,0.06*%	Aperture definition: D11 is a circle with diameter 0.06 inch
%ADD12R,0.06X0.06*%	Aperture definition: D12 is a rectangle with size 0.06 x 0.06 inch
%ADD13R,0.04X0.100*%	Aperture definition: D13 is a rectangle with size 0.04 x 0.1 inch
%ADD14R,0.100X0.04*%	Aperture definition: D14 is a rectangle with size 0.1 x 0.04 inch
%ADD15O,0.04X0.100*%	Aperture definition: D15 is an obround with size 0.04 x 0.1 inch



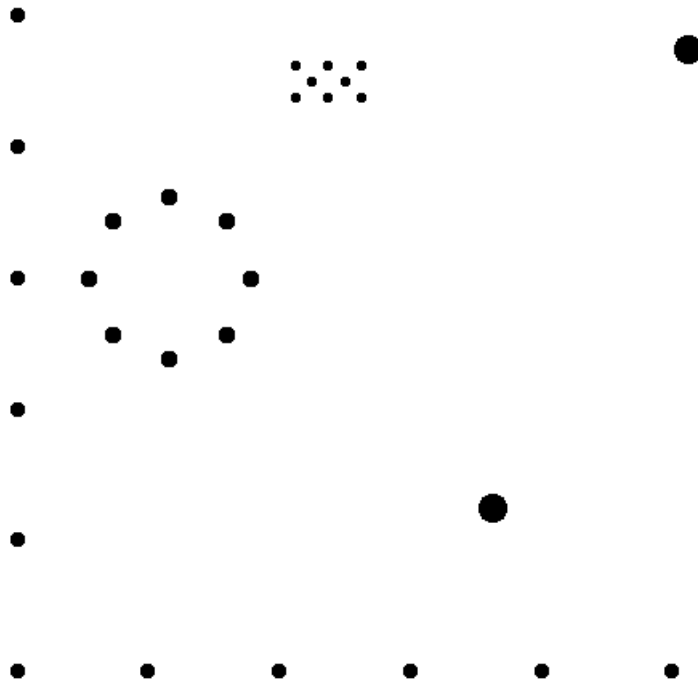
%ADD16P,0.100X3*%	Aperture definition: D16 is a polygon with 3 vertices and circumscribed circle with diameter 0.1 inch
%ADD17P,0.100X3*%	Aperture definition: D17 is a polygon with 3 vertices and circumscribed circle with diameter 0.1 inch
%ADD18TARGET125*%	Aperture definition: D18 is the special aperture called 'TARGET125' defined earlier
%ADD19THERMAL80*%	Aperture definition: D19 is the special aperture called 'THERMAL80' defined earlier
G04 Start image generation	A comment
D10*	Select aperture with D-code 10
X0Y250D02*	Move current point to (0, 0.25) inch
G01X0Y0D01*	Linear interpolation (draw)
G01X250Y0D01*	Linear interpolation (draw)
X1000Y1000D02*	Move current point
G01X1500D01*	Linear interpolation (draw)
G01X2000Y1500D01*	Linear interpolation (draw)
X2500D02*	Move current point. Since the X and Y coordinates are modal, Y is not repeated
G01Y1000D01*	Linear interpolation. The X coordinate is not repeated and thus its previous value of 2.5 inch is used
D11*	Select aperture with D-code 11
X1000Y1000D03*	Flash D11 at (1.0, 1.0). Y is modal.
X2000D03*	Flash D11 at (2.0, 1.0). Y is modal.
X2500D03*	Flash D11 at (2.5, 1.0). Y is modal.
Y1500D03*	Flash D11 at (2.5, 1.5). X is modal.
X2000D03*	Flash D11 at (2.0, 1.5). Y is modal.
D12*	Select aperture with D-code 12
X1000Y1500D03*	Move to (1.0, 1.5) and flash
D13*	Select new aperture with D-code 13
X3000Y1500D03*	Move to (3.0, 1.5) and flash
D14*	Select new aperture with D-code 14
Y1250D03*	Move to (3.0, 1.25) and flash
D15*	Select new aperture with D-code 15

Y1000D03*	Move to (3.0, 1.0) and flash
D10*	Select new aperture with D-code 10
X3750Y1000D02*	Move current point. This sets the start point for the following arc interpolation
G75*	Set multi quadrant mode
G03X3750Y1000I250J0D01*	Interpolate a complete circle
D16*	Select new aperture with D-code 16
X3400Y1000D03*	Flash D16
D17*	Select new aperture with D-code 17
X3500Y900D03*	Flash D17
D10*	Select new aperture with D-code 10
G36*	Start a region
X500Y2000D02*	Move current point to (0.5, 2.0)
G01Y3750D01*	Linear interpolation (draw)
G01X3750D01*	Linear interpolation (draw)
G01Y2000D01*	Linear interpolation (draw)
G01X500D01*	Linear interpolation (draw)
G37*	Create the region by filling the contour
D18*	Select new aperture with D-code 18
X0Y3875D03*	Flash D18
X3875Y3875D03*	Flash D18
%LPC*%	Level polarity is clear
G36*	Start a region
X1000Y2500D02*	Move current point to (1.0, 2.5)
G01Y3000D01*	Linear interpolation (draw)
G74*	Set single quadrant mode
G02X1250Y3250I250J0D01*	Clockwise arc with radius 0.25
G01X3000D01*	Linear interpolation (draw)
G75*	Set multi quadrant mode
G02X3000Y2500I0J-375D01*	Clockwise arc with radius 0.375
G01X1000D01*	Linear interpolation (draw)

G37*	Create the region by filling the contour
%LPD*%	Start a new level with dark polarity
D10*	Select new aperture with D-code 10
X1500Y2875D02*	Move current point
G01X2000D01*	Linear interpolation (draw)
D11*	Select aperture with D-code 11
X1500Y2875D03*	Flash D11
X2000D03*	Flash D11
D19*	Select aperture with D-code 19
X2875Y2875D03*	Flash D19
%TF.MD5,c637222723797acbb5d81635a1804%	The MD5 checksum of the file
M02*	End of file

### 2.9.3 Example 3: A drill file

Example 3 is a drill file.



5. Example 3: drill file

<code>%TF.GerberVersion,J1*%</code>	States the file complies with revision J1 of the spec
<code>%TF.FileFunction,PTH*%</code>	This drill file describes plated-through holes
<code>%TF.Part,Single*%</code>	The file is part of a single PCB
<code>%FSLAX26Y26*%</code>	Format specification: Leading zero's omitted Absolute coordinates Coordinate format is 2.6
<code>%MOIN*%</code>	Units are inches
<code>%LPD*%</code>	Dark level polarity
<code>%SRX1Y1I0J0*%</code>	Indicates that the following data is not stepped.
<code>%TA.DrillTolerance,0.01,0.005*%</code>	Set the drill tolerance to 10 mil in plus and 5 mil in minus in the current attribute dictionary. It will be attached to all aperture definitions until changed or deleted.
<code>%TA.AperFunction,ComponentDrill%</code>	Indicates that the following apertures define component drill holes.
<code>%ADD10C,0.014000*%</code>	Defines a drill tool that will be used to drill plated component drill holes with 10 mil positive and 5 mil negative tolerance

%TA.AperFunction,Other,MySpecialDrill*%	Indicates that the following apertures are special drill holes.
%ADD11C,0.024000*%	Defines a drill tool that will be used to drill plated special drill holes with 10 mil positive and 5 mil negative tolerance.
%TA.DrillTolerance,0.015,0.015*%	Change the drill tolerance for the following apertures to 15 mil in both directions
%TA.AperFunction,MechanicalDrill*%	Changes the tool function in the dictionary to mechanical
%ADD12C,0.043000*%	A circular aperture defining a drill tool with a tolerance of 15 mil in both directions that will be used for non-plated mechanical drill holes
%ADD13C,0.022000*%	Defines another tool with the same attributes but a smaller diameter
%TD.AperFunction*%	Removes the .AperFunction aperture attribute from the current attribute dictionary
%TD.DrillTolerance*%	Removes the .DrillTolerance aperture attribute from the current attribute dictionary
D10*	Select drill tool 10
X242000Y275000D03*	Drill plated component drill holes with diameter 14 mil at indicated coordinates
Y325000D03*	
X217000Y300000D03*	
X192000Y325000D03*	
X292000Y275000D03*	
X192000D03*	
X292000Y325000D03*	
X267000Y300000D03*	
D11*	Select drill tool 11
X124000Y0D03*	Drill plated special drill holes with diameter 24 mil at indicated coordinates.
X0Y-124000D03*	
X-124000Y0D03*	
X88000Y88000D03*	
X-88000D03*	
X0Y124000D03*	
X88000Y-88000D03*	
X-88000D03*	

D12*	Select tool 12
X792000Y350000D03*	Drill plated mechanical drill holes with diameter 43 mil at indicated coordinates
X492000Y-350000D03*	
D13*	Select tool 13
X767000Y-600000D03*	Drill plated mechanical drill holes with diameter 22 mil at indicated coordinates
X567000D03*	
X-233000Y200000D03*	
Y400000D03*	
Y0D03*	
Y-200000D03*	
Y-600000D03*	
Y-400000D03*	
X-33000Y-600000D03*	
X167000D03*	
X367000D03*	
%TF.MD5,b5d8122723797ac635a1814c04c6372b%	The MD5 checksum of the file
M02*	End of file



**Note:** One might be surprised to see drill files represented as Gerber files. Gerber is indeed not suited to drive drilling machines, but it is the best format to convey drill information from design to fabrication. After all, it defines where material must be removed, and this image information that Gerber files do perfectly. For more information, see 5.2.1.1.

## 2.10 Glossary

**ABSOLUTE POSITION:** Position expressed in Cartesian coordinates relative to the origin (0, 0).

**APERTURE:** A shape that is used for stroking or flashing. (The name is historic; vector photoplotters exposed images on lithographic film by shining light through an opening, called aperture.)

**ARC:** Either a graphic object created by stroking a circular interpolation with an aperture or a contour segment created by a circular interpolation.

**ATTRIBUTE:** Metadata association with the file as a whole or graphics objects, providing information without affecting the image.

**CIRCULAR INTERPOLATION:** Creating an arc.

**CLEAR:** Clear (unmark, rub, erase, scratch) the shape of a graphic object on the image plane.

**CONTOUR:** A closed curve defining a region.

**CURRENT POINT:** An implicit point in the plane used as a begin point of a circular or linear interpolation.

**CUSTOM ATTRIBUTE:** A third-party defined attribute to extend the format with proprietary meta-information.

**DARKEN:** Darken (mark, expose, paint) the shape of a image graphic object on the image plane

**DRAW:** Either a graphic object created by stroking a linear interpolation with an aperture or a contour segment created by a linear interpolation.

**FILE IMAGE:** The entire image defined by the file.

**FLASH:** A graphic object with the shape of an aperture.

**GRAPHICS OBJECT:** A flash, draw, arc or region. Graphics objects can be dark or clear. The image is created by darkening or clearing a stream of graphic objects on the image area.

**HEADER:** The beginning of the file until the first operation code is encountered.

**INCREMENTAL POSITION:** Position expressed as a distance in X and Y from the current point.

**INFORMATION LAYER.** See level.

**LEVEL:** A section of Gerber data. All objects in a level have the same polarity (dark or clear).

**LINEAR INTERPOLATION:** Creating a draw.

**MULTI QUADRANT MODE:** A mode defining how circular interpolation is performed. In this mode the arc is allowed extend over more than 90°. If the start point of the arc is equal to the end point the arc is a full circle of 360°.

**OPERATION CODES:** The codes D01, D02 or D03.

**PARAMETERS:** Commands that specify how the data should be processed.

**POLARITY:** When applied to the image, positive polarity means the image is positive black on white, and negative that it is negative. When applied to a level, dark means that the object exposes or marks the image area in dark and clear means that the

object clears or erases everything underneath it. See also 'transparent' and 'clear'.

**POLYGON FILL:** This is an old name for region fill. See section 4.5.

**REGION:** A graphic object with an arbitrary shape defined by its contour.

**RESOLUTION:** The distance expressed by the least significant digit of coordinate data. Thus the resolution is the step size of the grid on which all coordinates are defined.

**SINGLE QUADRANT MODE:** A mode defining how circular interpolation is performed. In this mode the arc cannot extend over more than 90°. If the start point of the arc is equal to the end point, the arc has length zero, i.e. covers 0°.

**SPECIAL APERTURE:** An aperture whose shape is defined by a macro by the AM parameter.

**STANDARD APERTURE:** An aperture with a pre-defined, standard shape.

**STANDARD ATTRIBUTE:** Pre-defined attributes conveying meta-information required for PCB data transfer from design to manufacturing.

**STEP AND REPEAT:** A method by which successive exposures of a single image block are made to produce a multiple image.

**STROKE:** To create a draw object (linear interpolation) or an arc object (circular interpolation).

**TRANSPARENT:** A property of holes in an aperture. Holes are not really part of the aperture. They have no effect on the underlying image. Holes do not clear the objects under them. Objects under a hole are visible. It is as if one can see through a hole, hence the term transparent. See also 'polarity' and 'clear'.



## 3 Syntax

### 3.1 Character Set

A Gerber file is expressed in 7-bit ASCII characters codes 32 to 126 (i.e. the printable characters in ANSI X3.4-1986) plus characters codes 10 (LF, Line Feed) and 13 (CR, Carriage Return). No other characters are valid. Gerber files are therefore printable and human readable.

Following characters are reserved:

- ❑ The asterisk '\*' is reserved as the end-of-block character.
- ❑ The percentage '%' is reserved as the parameter delimiter.
- ❑ The comma ',' is reserved as field separator.
- ❑ The character code 32 (SP, Space) is reserved for use in comments.

The line separators CR and LF have no effect; they can be ignored when processing the file. It is recommended to use line separators to improve human readability.

Gerber files are case-sensitive. Commands must be in upper case.

### 3.2 Variable Types

#### 3.2.1 Integers

Integers are a sequence of digits optionally preceded by a "+" or a "-". They must fall within the range of a 32 bit signed integer.

#### 3.2.2 Decimals

Decimals are a sequence of digits with an optional decimal point optionally preceded by a "+" or a "-", representing a decimal number.

#### 3.2.3 Names

Names are used to identify macros and variables.

Names must consist of letters (upper or lower case), an underscores ("\_") or a dollar signs ("\$\$\$") or digits. The first character cannot be a digit: `Name = [a-zA-Z_$] { [a-zA-Z_$0-9] + } .`

Names can be maximally 255 characters long.

Names are case-sensitive: `Name ≠ name`

#### 3.2.4 Strings

Strings are made up of all valid characters except the reserved characters SP, CR, LF, '%', '\*', and ';'.

`String = [a-zA-Z0-9_+~!/?<>""'(){}.\|&@# ] +`

Strings can be maximally 65535 characters long.

Strings are case-sensitive: `String ≠ string`

### 3.3 Data Blocks

Data blocks are building blocks for a Gerber file. Each data block ends with the mandatory end-of-block character asterisk '\*'. Each data block can contain one or more parameters, codes or coordinates.



**Example:**

```
X0Y0D02*
```

```
G01X50000Y0D01*
```

Data blocks are the low level syntactical elements of a Gerber file. The data blocks can be semantically interconnected so they form a group that represents a higher level element called a command.



**Tip:** It is recommended to add line separators between data blocks for readability. Do not put a line separator within a data block, except after a comma separator in long data blocks. The line separators have no effect on the image.

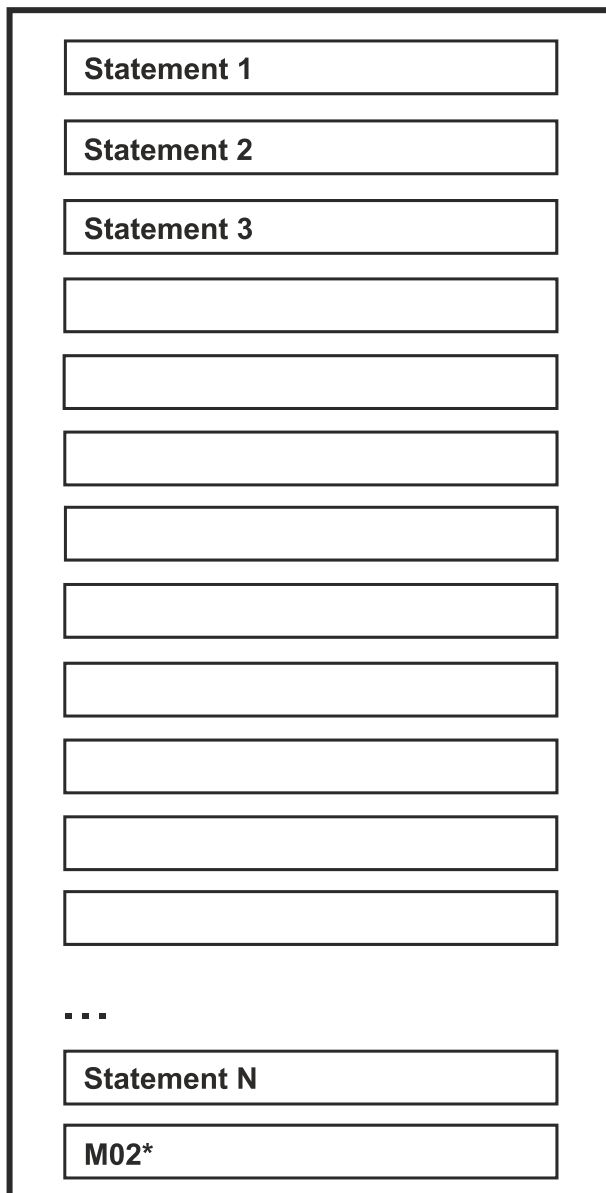
## 3.4 Commands

### 3.4.1 Commands Overview

Commands are higher level semantic elements of a Gerber file. Each command contains one or more data blocks. Many commands consist of a single data block. If a command contains parameters then it starts and ends with a '%' character. It is then called a parameter command.

A Gerber file consists of a stream of commands. There is no limitation on the number of commands in a Gerber file.

The structure of a Gerber file is shown in the picture below:



6. Gerber file structure

A Gerber file *must* end with 'M02\*', the 'end of file' data block.

The syntax of a command is as follows:

**<Command>: [%]<Data Block>{<Data Block>}[%]**

Below are examples of commands.



**Example:**

```
G02X0Y100I-400J100D01*
```

In the example above the command consists of a single data block that represents G02 and D01 function code together with a coordinate and offset in X and Y.



**Example:**

```
%AMDONUTFIX*1,1,0.100,0,0*1,0,0.080,0,0*%
```

In this example the parameter command contains an AM parameter built of three data blocks.

There are three command types:

- ❑ Function codes. See 3.4.2.
- ❑ Coordinate data. See 3.4.3.
- ❑ Parameters. See 3.4.4.

### 3.4.2 Function Codes

Function codes are either

- ❑ Operation codes, operating on coordinate data, i.e. D01, D02 or D03.
- ❑ Codes that set a graphics state variable.



**Example:**

```
G74*
```

If a code is located in the same data block as coordinate data, the graphics state is first changed before the operation code operates on the coordinate data.

In the example below there are two data blocks. In the first block the function code 'G01' is followed by coordinate data. The G01 function code means 'start linear interpolation' and the coordinate data means the starting point (300, 200) for the interpolation. In the second data block the next interpolation point (1100, 200) is specified.



**Example:**

```
G01X300Y200D02*  
G01X1100Y200D01*
```

Function codes are described in detail in chapter 0.

### 3.4.3 Coordinate Data Blocks

A coordinate data block consists of coordinate data followed by an operation code D01, D02 or D03. The code operates on the coordinate data.

A coordinate data block is expressed as follows:

**<Coordinate data>: [X<Number>][Y<Number>][I<Number>][J<Number>](D01|D02|D03)**

Syntax	Comments
X, Y	Characters indicating X or Y coordinates of a point
I, J	Characters indicating an offset in the X or Y direction
<Number>	Decimal digits, possibly with a sign defining either a coordinate (X,Y) or an offset (I,J).
D01 D02 D03	Function codes that determine the effect of the coordinate preceding it. Their meaning is explained below.

The FS and MO parameters specify how to interpret the digits following the X, Y, I, J characters.

Coordinate data define points in the plane using a right-handed ortho-normal coordinate system. The plane is infinite, but implementations can have size limitations.

Each coordinate data block must end with a one and only one operation code (D01, D02 or D03). The operation code operates on the preceding coordinate data.

Coordinates *are* modal. If an X is omitted the X coordinate of the current point is used. The same applies to Y.

Offsets *are not* modal. If I or J is omitted the default is zero (0). The offsets do not affect the current point.



#### Examples of coordinate data blocks

X200Y200D02*	point (+200, +200) and offset (0, 0) operated upon by D02
Y-300D03*	point (+200, -300) and offset (0, 0) operated upon by D03
I300J100D01*	point (+200, -300) and offset (+300, +100) operated upon by D01
Y200I50J50D01*	point (+200,+200) and offset (+50, +50) operated upon by D01
X200Y200I50J50D01*	point (+200, +200) and offset (+50, +50) operated upon by D01
X+100I-50D01*	point (+100, +200) and offset (-50, 0) operated upon by D01

As X and Y are modal in a coordinate data block, in a data block without explicit X nor Y, the previous X and Y is used. In the example below D03 operates on the current point.



#### Example

D03\*

### 3.4.4 Parameters

Parameters define characteristics of the file.



**Note:** Originally parameters were called Mass Parameters.

Parameters operating on the entire image must be placed in the header of the file. Other parameters are placed at the appropriate location in the file.

Parameters consist of a two-character parameter code followed by parameter data. The parameter code indicates which parameter is used. The structure of parameter data depends on the parameter code.

Parameters are enclosed into a pair of delimiter ‘%’ characters. Usually a parameter consists of a single data block ending with a ‘\*’. The AM parameter can include more than one data block.

The ‘%’ must immediately follow the ‘\*’ of the last data block without intervening line separators. This is an exception to the general rule that a data block can be followed by a line separator.



#### Examples:

```
%FSLAX23Y23*%
```

```
%AMDONUTFIX*1,1,0.100,0,0*1,0,0.080,0,0*%
```

Parameters may be provided single or grouped between ‘%’ delimiters, up to a maximum of 4096 characters between these delimiters.



#### Example:

```
%SFA1.0B1.0*ASAXBY*%
```

Line separators are permitted between parameters to improve readability. For a set of parameters the syntax is:

```
%<Parameter>{{<Line separator>}<Parameter>}%
```



#### Example:

```
%SFA1.0B1.0*
```

```
ASAXBY*%
```



**Tip:** For readability it is recommended to have one parameter per line.

The syntax for an individual parameter is:

**%Parameter code<required modifiers>[optional modifiers]\*%**

Syntax	Comments
Parameter code	2-character code (AD, AM, FS, etc...)
<required modifiers>	Must be entered to complete definition
<optional modifiers>	May be required depending on the required modifiers

We distinguish two classes of parameters:

- ❑ *Graphics parameters* affect image generation. They define how the function codes and coordinates are processed.
- ❑ *Attribute parameters* do not affect image generation but associate attributes with either the file as a whole or with individual graphics objects.

## 4 Graphics

---

### 4.1 Graphics Overview

The stream of graphics objects is generated by processing the stream of graphics commands. There are two types of graphics commands:

- Function codes
- Parameters

Function codes are identified by a code letter G, D or M followed by a code number, e.g. G02. Parameters are identified by two letters, e.g. MO.

The coded and code letters originates from the original EIA RS-274-D specification. Parameters are commands not present in EIA RS-274-D. This is now largely historic; please accept it as it is.

The tables below give an overview of the function codes and graphics parameters. They are explained in more details later.



#### **Example of codes and parameters:**

```
G04 Beginning of the file*  
%FSLAX25Y25*%  
%MOIN*%  
%LPD*%  
%ADD10C,0.000070*%  
X123500Y001250D02*  
...  
M02*
```

Parameter	Name	Description	Comments
FS	Format Specification	Sets the 'Coordinate format' graphics state variable	These parameters can only be used once, in the header of the file.
MO	Mode (inch or mm)	Sets the 'Unit' graphics state variable	
AD	Aperture Definition	Assigns a D code number to an aperture definition	These parameters can be used multiple times. It is recommended to put them in header of the file
AM	Aperture Macro	Defines special apertures which can be referenced from the AD parameter	
SR	Step and Repeat	Sets the 'Step and Repeat' graphics state variable	These parameters can be used multiple times over the whole file.
LP	Level Polarity	Starts a new level and sets the 'Level polarity' graphics state variable	

*Graphics parameters*



Code	Description	Comments
D01	Interpolate operation code	If region mode is off D01 creates a draw or arc using the current aperture. Only specific apertures can be used; see 2.4. When region mode is on D01 creates a contour segment. The current aperture is not used. After the object is created the current point is moved to the coordinate
D02	Move operation code	D02 does not create a graphics object but move the current point to the coordinate.
D03	Flash operation code	With region mode is off D03 flashes the current aperture. D03 is not allowed in region mode. After the flash is created the current point is moved to the coordinate
D10 and higher	Set the current aperture'	Sets the current aperture to a number defined by an AD parameter.
G01	Set the interpolation mode to linear	A modifier of the interpolation operation code D01. See 4.2 and 4.4.
G02	Set the interpolation mode to 'Clockwise circular interpolation'	
G03	Set the interpolation mode to 'Counterclockwise circular interpolation'	
G04	Ignore data block	Used for comments.
G36	Set region mode on.	Used to create regions. See 4.5.
G37	Set region mode off.	
G74	Set quadrant mode to 'Single quadrant'	A modifier of the circular interpolation mode. See dedicated section for more details.
G75	Set quadrant mode to 'Multi quadrant'	
M02	Indicates the end of the file	Every file must end in a M02. It can only occur once, at the end of the file. No data is allowed after M02.

*Function codes*

## 4.2 Operation Codes (D01/D02/D03)

D01, D02 and D03 are the *operation codes*. The operation codes create the graphics objects by operating on a coordinates.

Syntactically a coordinate data block contains the coordinate data followed its operation code. A coordinate data block must contain a single (1) operation code: each operation code is associated with a single coordinate pair and vice versa. (Coordinate data blocks without operation codes are deprecated.)



### Example:

```
X100Y100D01*
X200Y200D02*
X300Y-400D03*
```

The operation codes have the following effect.

- ❑ D01 creates a straight line segment (draw) or a circular segment (arc) by interpolating from the current point to the coordinates. This operation is called to *interpolate*, to *draw*, to *arc*. (It was also called a lights-on move in days of vector plotters.)
- ❑ D02 moves the current point to the coordinates. No graphics object is generated. This operation is called to *move*. (It was also called a lights-off move in days of vector plotters.)
- ❑ D03 creates a flash object by replicating the current aperture at the coordinate. This operation is called to *flash*.

The operation code D03 directly creates a flash object. Sequences of D01 and D02 create segments that are turned in graphics by object one of two following methods:

- ❑ Stroking. The segments are stroked with the current aperture, see 2.4.
- ❑ Region building. The segments form contour that defines a region, see 4.5.

The region mode setting determines which object generating method is used. When region mode is *off* stroking is used, when region mode is *on* region building is used.

The operation codes are controlled by the graphics state, see 2.6.

The function codes G01, G02, G03 can be put together with operation codes in the same data block. The graphics state is then modified before the operation coded is executed, whatever the order of the codes.



### Example:

```
G01X100Y100D01*
X200Y200D01*
```

G01 sets the interpolation mode to linear and this used to process the coordinate data X100Y100 from the same data block as well as the coordinate data X200Y200 from the next data block.

The syntax for G01, G02, G03, D01 and D02 is the following:

**<Interpolation>: [G(1|01|2|02|3|03)][<Coordinate data>D(1|01|2|02)]\***

The following data blocks are syntactically valid:

G01\*  
X100Y100D01\*  
G01X500Y500D01\*  
X300Y300D01\*  
G01X100Y100D01\*

A valid data block must contain at least one of the parts.

The recommended syntax for the D02 function code is the following:

**<Move current point>: [X<Coordinate>][Y<Coordinate>]D02\***

Syntax	Comments
X<Coordinate>	Defines the X coordinate of the new current point. If missing then the previous X coordinate is used.
Y<Coordinate>	Defines the Y coordinate of the new current point. If missing then the previous Y coordinate is used.
D02	Move operation code.



**Example:**

X200Y100D02\*

It is allowed but not recommended to specify G01/G02/G03 together with a D02 (move). The G01/G02/G03 is then ignored.

The syntax for the D03 function code ('Flash' mode) is:

**<Flash current aperture>: [X<Coordinate>][Y<Coordinate>]D03\***

Syntax	Comments
X<Coordinate>	Defines the X coordinate of the flash point. If missing then the previous X coordinate is used.
Y<Coordinate>	Defines the Y coordinate of the flash point. If missing then the previous Y coordinate is used.
D03	Flash operation code



**Example:**

X100Y100D03\*

An example of the use of the function codes G36,G37,G74,G75,...



**Example:**

G36\*  
X200Y1000D02\*  
G01X1200D01\*  
G01Y200D01\*  
G01X200D01\*  
G01Y600D01\*  
G01X500D01\*  
G75\*  
G03X500Y600I300J0D01\*  
G74\*  
G01X200D01\*  
G01Y1000D01\*  
G37\*

## 4.3 Linear Interpolation (G01)

Linear interpolation generates a straight line from the current point to the point with X, Y coordinates specified by the data block. The current point is then set to the X, Y coordinates specified by the data block. The resulting graphic object is called a 'draw'.

### 4.3.1 Data Block Format

The syntax for the linear interpolation code is:

**<Linear interpolation>: G(01|1)[X<Coordinate>][Y<Coordinate>][D(01|02)]\***

Syntax	Comments
G(01 1)	G01 or G1 – Sets interpolation mode to 'Linear interpolation'
X<Coordinate>	Defines the X coordinate of the draw end point. If missing then the previous X coordinate is used.
Y<Coordinate>	Defines the Y coordinate of the draw end point. If missing then the previous Y coordinate is used.
D(01 02)	Interpolate/Move operation code



**Example:**

G01X0Y250D01\*

## 4.4 Circular Interpolation (G02/G03, G74/G75)

### 4.4.1 Arc Overview

Circular interpolation generates a circular arc from the current point to the point with X, Y coordinates specified by the data block; the center of the arc is specified by the offsets I and J. The current point is then set to the X, Y coordinates specified by the data block.

There are two orientations:

- ❑ Clockwise, set by G02
- ❑ Counterclockwise, set by G03

The orientation is defined around the center of the arc, moving from begin to end.

There are two quadrant modes:

- ❑ Single quadrant mode (G74)
- ❑ Multi quadrant mode (G75)

Quadrant mode	Comments
Single quadrant (G74)	<p>In single quadrant mode the arc is not allowed to extend over more than 90°. The following relation must hold:</p> <p><math>0^\circ \leq A \leq 90^\circ</math>, where A is the arc angle</p> <p>If the start point of the arc is equal to the end point, the arc has length zero, i.e. it covers 0°. A data block is required for each quadrant. A minimum of four coordinate data blocks is required for a full circle.</p>
Multi quadrant (G75)	<p>In multi quadrant mode the arc is allowed to extend over more than 90°. To avoid ambiguity between 0° and 360° arcs the following relation must hold:</p> <p><math>0^\circ &lt; A \leq 360^\circ</math>, where A is the arc angle</p> <p>If the start point of the arc is equal to the end point, the arc is a full circle of 360°.</p>

#### *Quadrant modes*

The codes G74 and G75 allow switching between the two quadrant modes. A data block containing G75 turns on multi quadrant mode. Every block following it will be interpreted as multi quadrant, until cancelled by a G74. A data block containing G74 code turns on single quadrant mode.



**Warning:** A Gerber file containing arcs but without a preceding G74 or G75 code is invalid.

### 4.4.2 Arc Definition

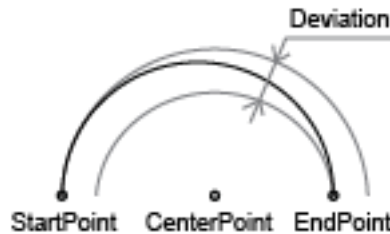
For an arc to be circular the center must be positioned at exactly the same distance - radius - from the start point and the end point. The two radii must be equal. The definition of an arc is then obvious.

However, as Gerber file has a finite resolution, the center point generally cannot be positioned such that the radii are exactly equal. Furthermore the software generating the Gerber file

unavoidably adds rounding errors of its own. The two radii are different for almost all real-life arcs, unavoidably so. We will call the difference between the radii the *arc deviation*.

This raises the question which curve is represented by a “circular arc” with a positive deviation.

The arc defined as a *continuous and monotonous curve starting at the start point and ending at the end point, approximating the ring with the given center point and radii equal to the start radius and end radius*. See figure 7. Note that this curve is only mathematically circular if the deviation is zero.



7. Arc with a non-zero deviation

The arc definition has a fuzziness of the order of magnitude of the arc deviation. The writer of the Gerber file accepts any interpretation within the fuzziness above as valid. If the writer requires a more precise interpretation of the arc he needs to write arcs with lower deviation.

It is not allowed to place the center point close to the line through begin and end point, but not in between them. Such a construct is nonsensical. See figure 8.



8. Nonsensical center point

Note that self-intersecting contours are not allowed, see 4.5. If any of the valid arc interpretations turns the contour in a self-intersecting one, the file is invalid, with unpredictable results.

Most real-life issues resulting from high deviation come from using a low coordinate resolution. Using high coordinate resolution is an obvious first step to minimize the arc deviation and potential problems. We recommend using 6 decimal place in imperial and 5 decimal places in metric.

### 4.4.3 Single Quadrant Mode

Single quadrant mode is set by a G74 code.



**Example:**

G74\*

#### 4.4.3.1 Data Block Format

The syntax in single quadrant mode is:

**<Circular interpolation>: G(02|2|03|3)[X<Coordinate>][Y<Coordinate>]  
[I<Distance>][J<Distance>][D(01|02)]\***

Syntax	Comments
G(02 2 03 3)	Sets the interpolation mode': G02 or G2 – 'Clockwise circular interpolation' G03 or G3 – 'Counterclockwise circular interpolation'

X<Coordinate>	Defines the X coordinate of the arc end point. If missing then the previous X coordinate is used.
Y<Coordinate>	Defines the Y coordinate of the arc end point. If missing then the previous Y coordinate is used.
I<Distance>	The distance between the arc start point and the center parallel to the X axis. The value is always positive. A sign is not allowed. The sign of the offset to the center is determined implicitly. If missing then a 0 distance is used.
J<Distance>	The distance between the arc start point and the center parallel to the Y axis. The value is always positive. A sign is not allowed. The sign of the offset to the center is determined implicitly. If missing then a 0 distance is used.
D(01 02)	Interpolate/Move operation code



**Note:** Because the sign in offsets is omitted, there are four candidates for the center: (<Current X> +/- <X distance>, <Current Y> +/- <Y distance>). The center is the candidate that results in an arc with the specified orientation and not greater than 90°.



**Warning:** If the center is not precisely positioned, there may be none or more than one candidate fits. In that case the arc is invalid. The creator of the file accepts any interpretation.



**Example:**

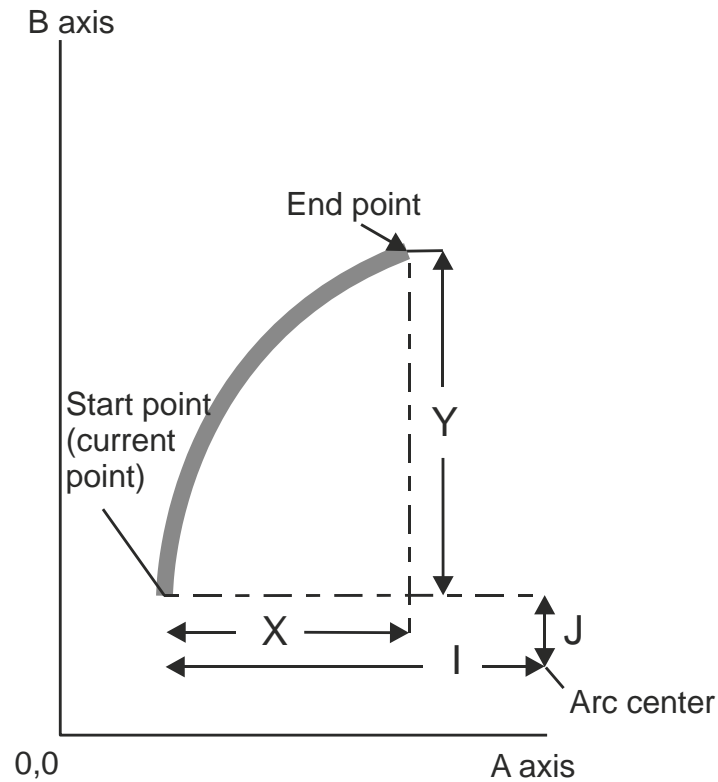
G74\*

G03X700Y1000I400J0D01\*

#### 4.4.3.2 Image

The coordinates of an arc endpoint and the center distances are interpreted according to the coordinate format specified by the FS parameter and the unit specified by the MO parameter. The following image illustrates how arcs are interpolated.

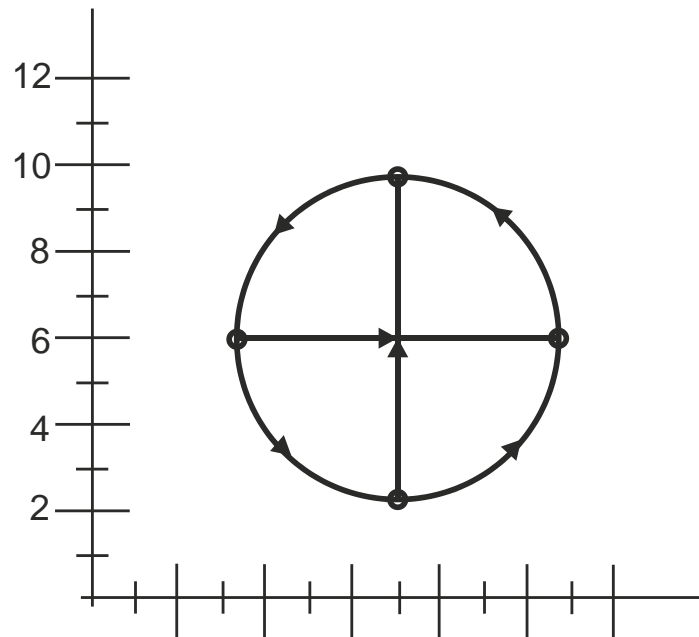




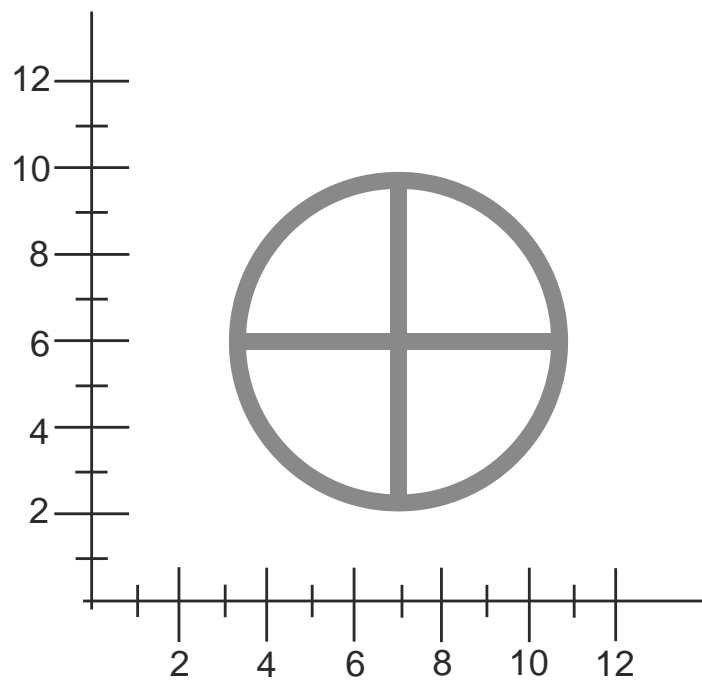
#### 9. Single quadrant mode

##### 4.4.3.3 Example

Syntax	Comments
G74*	Single quadrant mode
D10*	Use aperture D10
X1100Y600D02*	Start from (11, 6)
G03X700Y1000I400J0D01*	Quarter arc (radius 4) to (7, 10)
G03X300Y600I0J400D01*	Quarter arc (radius 4) to (3, 6)
G03X700Y200I400J0D01*	Quarter arc (radius 4) to (7, 2)
G03X1100Y600I0J400D01*	Quarter arc (radius 4) to (11, 6)
X300D02*	Start from (3 ,6)
G01X1100D01*	Draw to (11, 6)
X700Y200D02*	Start from (7, 2)
G01Y1000D01*	Draw to (7, 10)



10. Single quadrant mode example: arcs and draws



11. Single quadrant mode example: resulting image

#### 4.4.4 Multi Quadrant Mode

The multi quadrant mode is set by a G75 code.



**Example:**

G75\*

##### 4.4.4.1 Data Block Format

The syntax in multi quadrant mode is:

**<Circular interpolation>: G(02|2|03|3)[X<Coordinate>][Y<Coordinate>]  
[I<Offset>][J<Offset>][D(01|02)]\***

Syntax	Comments
G(02 2 03 3)	Sets the interpolation mode: G02 or G2 – 'Clockwise circular interpolation' G03 or G3 – 'Counterclockwise circular interpolation'
X<Coordinate>	Defines the X coordinate of the arc end point. If missing then the previous X coordinate is used.
Y<Coordinate>	Defines the Y coordinate of the arc end point. If missing then the previous Y coordinate is used.
I<Offset>	Defines the offset or signed distance between the arc start point and the center measured parallel to the X axis. If missing then a 0 offset is used.
J<Offset>	Defines the offset or signed distance between the arc start point and the center measured parallel to the X axis. If missing then a 0 offset is used.
D(01 02)	Operation code



**Note:** In multi quadrant mode the offsets in I and J are signed. If no sign is present, the offset is positive.



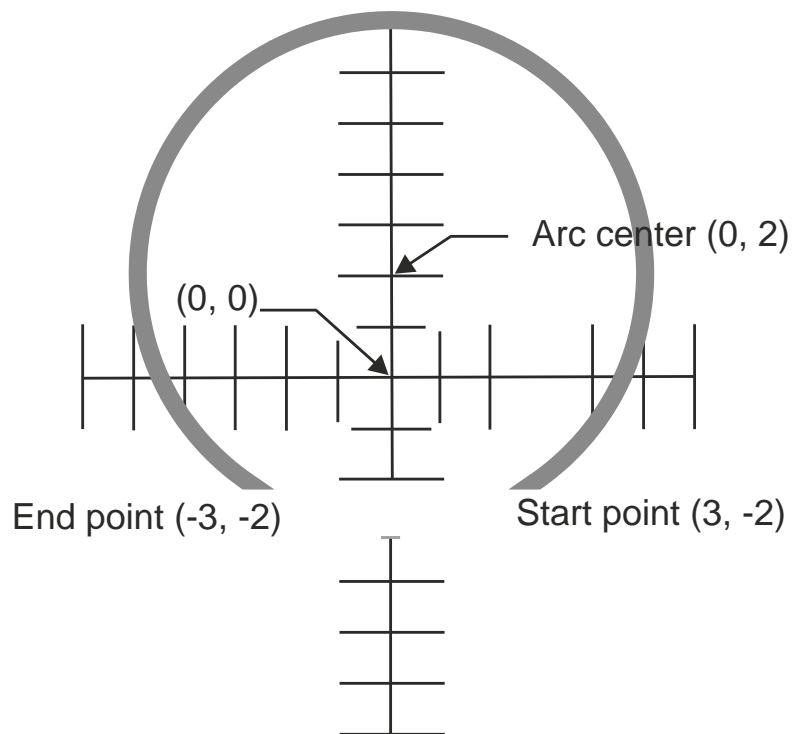
**Example:**

G75\*

G03X-300Y-200I-300J400D01\*

#### 4.4.5 Arc Example

Syntax	Comments
X300Y-200D02*	Move to (3, -2)
G75*	Set multi quadrant mode
G03X-300Y-200I-300J400D01*	Arc counterclockwise to (-3,-2); offsets from the start point to the center point are -3 for X and 4 for Y, i.e. the center point is (0, 2)
G74*G01*	Back to linear interpolation mode and G74



12. Multi quadrant mode example: resulting image

#### 4.4.6 Numerical instability in multi quadrant (G75) arcs

In G75 mode small changes in the position of center point, start point and end point can swap the large arc with the small one, dramatically changing the image.

This most frequently occurs with very small arcs. Start point and end point are close together. If the end point is slightly moved it can end on top of the start point. Under G75, if the start point of the arc is equal to the end point, the arc is a full circle of 360°, see 4.4.1. A small change in the position of the end point has changed the very small arc to a full circle.

Under G75 rounding must be done carefully. Using high resolution is an obvious prerequisite. We recommend using 6 decimal places in imperial and 5 decimal places in metric.

The Gerber writer must also consider that the reader unavoidably has rounding errors. Perfectly exact numerical calculation cannot be assumed. It is the responsibility of the writer to avoid unstable arcs.

Under G74 arcs are always less than 90° and this numerical instability does not exist. G74 is intrinsically stable. Another option is not to use very small arcs, e.g. by replacing them with draws - the error is very small and draws are stable.

#### 4.4.7 Using G74 or G75 can result in a different image

An arc command can define a completely different image under G74 and G75. The two sample files below differ only in G74/G75, but they define a dramatically different image.

Syntax	Comments
D10*	Use aperture D10
G01X0Y600D02*	Start from (0, 6)
G74*	Single quadrant mode
G02X0Y600I500J0D01*	Arc to (0, 6) with radius 5

The resulting image is small dot, an instance of the aperture at position (0, 6)

Syntax	Comments
D10*	Use aperture D10
G01X0Y600D02*	Start from (0, 6)
G75*	Multi quadrant mode
G02X0Y600I500J0D01*	Arc to (0, 6) with center (5,6)

The image is a full circle.



**Warning:** It is mandatory to always specify G74 or G75 if arcs are used.

## 4.5 Regions (G36/G37)

### 4.5.1 Region Overview

A region is a graphics object defined by its contour(s).

A contour is a sequence of connected draw or arc segments. A pair of segments is said to connect only if they are defined consecutively, with the second segment starting where the first one ends. Thus the order in which the segments of a contour are defined is significant. Non-consecutive segments that meet or intersect fortuitously are not considered to connect. A contour is closed: the end point of the last segment must connect to the start point of the first segment.

The G36 code sets region mode *on* and G37 sets it *off*. With region mode on the operation codes D01 and D02 create the contours. The first D01 encountered in region mode starts the first contour by creating the first segment. Subsequent D01's add segments to it. When a D02 is encountered the contour is considered finished. (Note that a D02 finishes a contour even if the current point stays the same place as with the data block D02\*.) A D02 is only allowed if the preceding contour is closed. The next D01 encountered starts a new contour. In this way an unlimited number of contours can be created in the same region.

When a G37 is encountered region mode is turned off and the regions graphics object is created by filling the contours. Each contour is filled individually. Different contours can touch, overlap or intersect. The filled area is the union of the filled areas of each individual contour. A G37 is only allowed if all contours are closed.


*Self-intersecting contours are not allowed. Segments cannot cross, overlap or touch except:*


1. Consecutive segments connecting in their endpoints, needed to construct the contour
2. Horizontal or vertical coincident *draws*, used to create holes in a region with cut-ins; see 4.5.9. A pair of draws are said to be coincident if and only if the draws coincide completely, with the second draw starting where the first one ends.

Any other form of self-touching or self-intersection is *not allowed*. For the avoidance of doubt, not allowed are amongst other partially coinciding draws (not sharing both vertices), diagonal coincident draws, coincident arcs, partially coinciding arcs, arcs tangent to another arc or to a draw, vertices on a segment but not on its endpoints, vertices with more than two segments.


The segments are not graphics objects in themselves; segments are part of region which is the graphics object. The segments have no thickness. The current aperture has no effect in region mode.


D01 and D02 are the *only* D codes allowed in region mode; in other words D03 and Dnn (nn≥10) are *not* allowed. G codes are allowed. Parameters are *not* allowed.


 **Warning:** Cut-ins should only be used for the simplest configurations. It is not recommended to create holes (anti-pads) in PCB planes with cut-ins. Regions with many with cut-ins are complex. Errors in these complex constructions are the most common cause of missing clearances. It is recommended to first create the plane without holes with a simple region and then make holes in it by a subsequent clear polarity level with the holes; see 2.2. This is simple and robust. Furthermore, PCB CAM needs to know the location of the anti-pads; with flashed anti-pads their location is obvious; with cut-ins the anti-pads are hidden in a complex construction and must be recovered laboriously.

 **Warning:** Care must be taken that rounding errors do not turn a proper contour into a self-intersecting one, leading to unpredictable results. The Gerber writer must also consider that the reader unavoidably has rounding errors. Perfectly exact numerical calculation cannot be assumed. This is especially important for arcs, which are intrinsically fuzzy. Construct contours

defensively. Observe sufficient clearances between the segments of the arcs. It is the responsibility of the writer to avoid brittle contours that are only marginally valid and become self-intersecting under normal rounding. Low file coordinate resolution is the most frequent culprit for rounding problems, see 4.8. We recommend using 6 decimal places in imperial and 5 decimal places in metric.

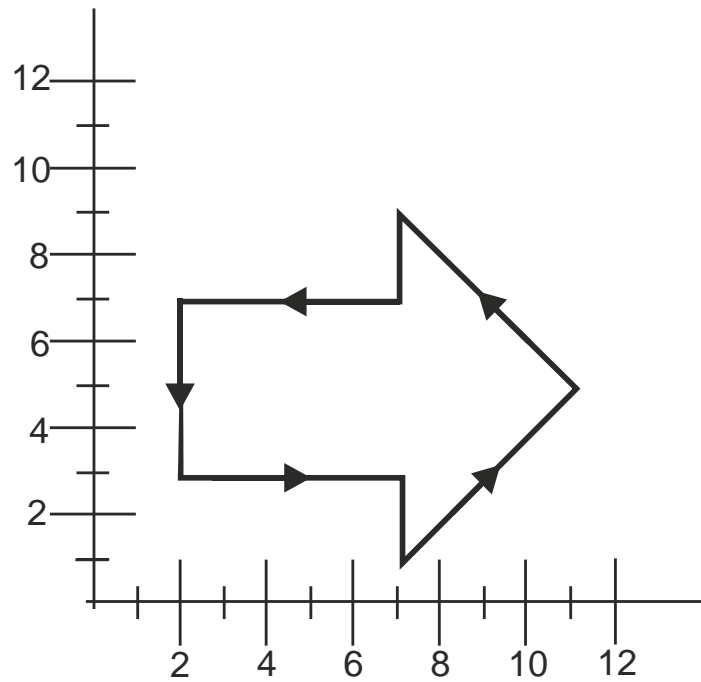
 **Warning:** An arc can be validly interpreted by any curve within a range, see 4.4. If any of these curves results in a self-intersecting contour the file is invalid and the result is unpredictable.

 **Note:** In the 1960's and 1970s, the era of vector plotters, the only way to produce a region was by painting (aka filling or stroking) it with draws. This produces the correct image. However, the file size explodes. More importantly, painted data cannot be handled properly in PCB CAM and the painting must be removed laboriously. A file with painted area's and/or painted pads is not really suitable for PCB production.

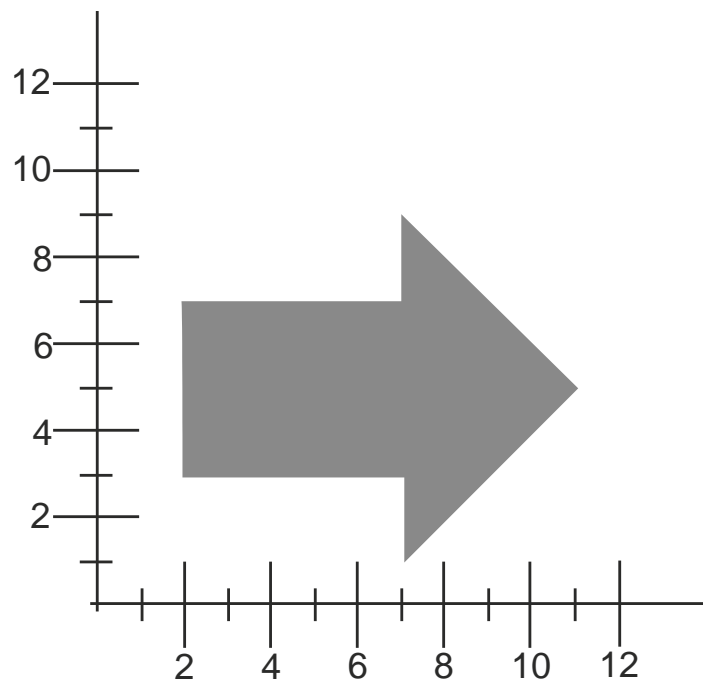
 **Note:** In previous versions of this document "contour fill" was called "polygon fill".

## 4.5.2 Example: a simple contour

Syntax	Comments
G36*	Start a region
X200Y300000D02*	Move the current point to (2, 3)
G01X700000D01*	Line segment to (7, 3)
G01Y100000D01*	Line segment to (7, 1)
G01X1100000Y500000D01*	Line segment to (11, 5)
G01X700000Y900000D01*	Line segment to (7, 9)
G01Y700000D01*	Line segment to (7, 7)
G01X200000D01*	Line segment to (2, 7)
G01Y300000D01*	Line segment to (2, 3)
G37*	Create the region by filling the contour



13. Simple contour example: the segments



14. Simple contour example: resulting image



### 4.5.3 Examples: how to start a single contour

The first D01 starts the contour at the current point, independent of how the current point is set. We give three examples of similar images; differences with the previous column are highlighted

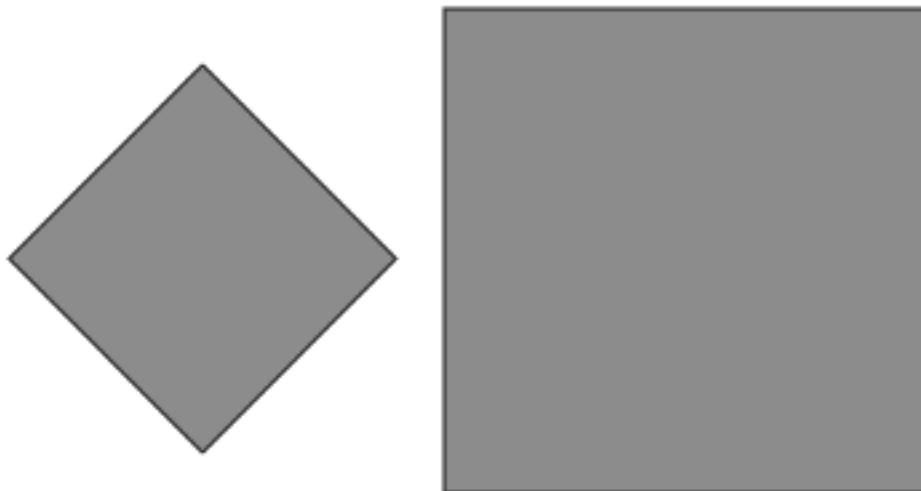
Example 1	Example 2	Example 3
... G01* D11* ... X300Y500D01* G36* X5000Y5000D02* X6000D01* Y6000D01* X5000D01* Y5000Y5000D01* G37* ...	... G01* D11* ... X300Y500D01* X5000Y5000D02* G36* X6000D01* Y6000D01* X5000D01* Y5000Y5000D01* G37* ...	... G01* D11* ... X300Y500D01* X5000Y5000D01* G36* X6000D01* Y6000D01* X5000D01* Y5000Y5000D01* G37* ...
This sequence creates a square contour after the stroked draw X300Y500D01*	Swap D02 and G336. Exactly the same image.	Replace D02 by D01. The same contour. The stroked draw X5000Y5000D01* to the image.

## 4.5.4 Examples: Use D02 to start a second contour

### Example file: Non- overlapping contours

```
G04 Non-overlapping contours*  
%FSLAX23Y23*%  
%MOMM*%  
%ADD10C,1.00000*%  
%LPD*%  
G36*  
X0Y5000D02*  
Y10000D01*  
X10000D01*  
Y0D01*  
X0D01*  
Y5000D01*  
X-1000D02*  
X-5000Y1000D01*  
X-9000Y5000D01*  
X-5000Y9000D01*  
X-1000Y5000D01*  
G37*  
M02*
```

This creates the following image:



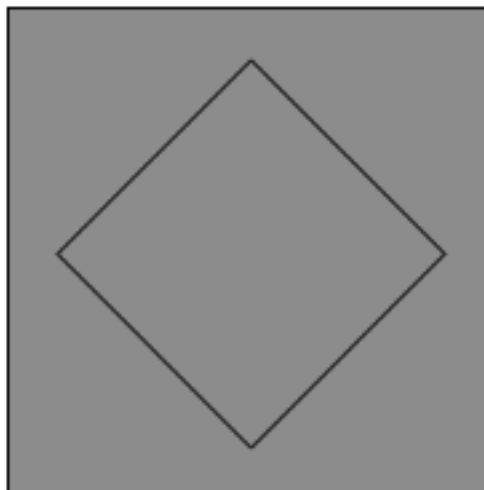
#### *15. Use of D02 to start a new non-overlapping contour*

Two different contours were created. Each contour is filled individually. The filled area is the union of the filled areas.

## 4.5.5 Example file: Overlapping contours

```
G04 Overlapping contours*  
%FSLAX23Y23*%  
%MOMM*%  
%ADD10C,1.00000*%  
%LPD*%  
G36*  
X0Y5000D02*  
Y10000D01*  
X10000D01*  
Y0D01*  
X0D01*  
Y5000D01*  
X1000D02*  
X5000Y1000D01*  
X9000Y5000D01*  
X5000Y9000D01*  
X1000Y5000D01*  
G37*  
M02*
```

This creates the following image:



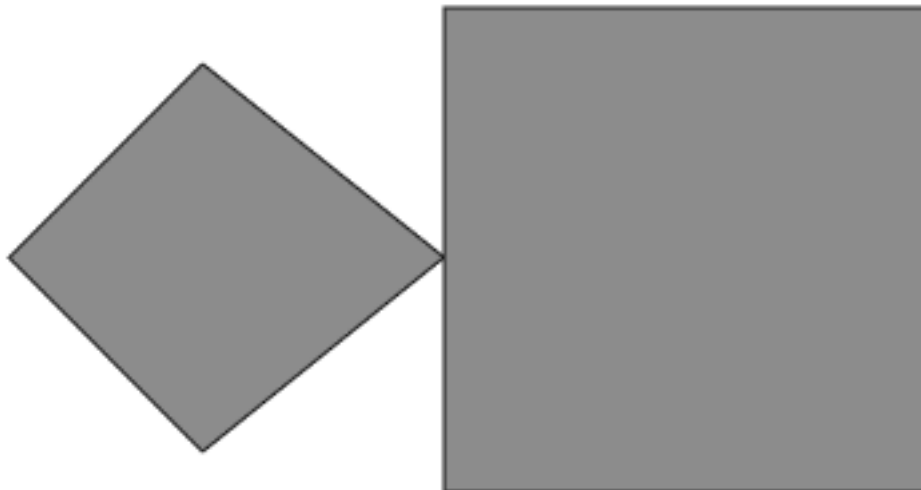
### *16. Use of D02 to start an new overlapping contour*

Two different contours were created. Each contour is filled individually. The filled area is the union of the filled areas. As the second contour is completely embedded in the first, the effective filled area is the one of the first contour.

## 4.5.6 Example file: Non-overlapping and touching

```
G04 Non-overlapping and touching*  
%FSLAX23Y23*%  
%MOMM*%  
%ADD10C,1.00000*%  
%LPD*%  
G36*  
X0Y5000D02*  
Y10000D01*  
X10000D01*  
Y0D01*  
X0D01*  
Y5000D01*  
D02*  
X-5000Y1000D01*  
X-9000Y5000D01*  
X-5000Y9000D01*  
X0Y5000D01*  
G37*  
M02*
```

This creates the following image:



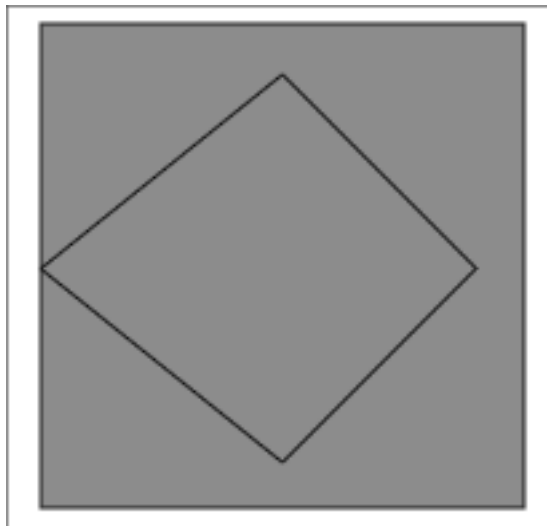
*17. Use of D02 to start an new non-overlapping contour*

As these are two different contours in the same region touching is allowed.

## 4.5.7 Example file: Overlapping and touching

```
G04 Overlapping and touching*  
%FSLAX23Y23*%  
%MOMM*%  
%ADD10C,1.00000*%  
%LPD*%  
G36*  
X0Y5000D02*  
Y10000D01*  
X10000D01*  
Y0D01*  
X0D01*  
Y5000D01*  
D02*  
X5000Y1000D01*  
X9000Y5000D01*  
X5000Y9000D01*  
X0Y5000D01*  
G37*  
M02*
```

This creates the following image:



*18. Use of D02 to start an new overlapping and touching contour*

As these are two different contours in the same region touching is allowed.

## 4.5.8 Using levels to create holes

The recommended way to create holes in regions is by using levels with alternating dark and clear polarity, as illustrated in the following example. The file has four levels. The first level has dark polarity and contains the big square region. The second level has clear polarity and contains a circular disk; the disk is cleared from the image and creates a round hole in the big square. The third level has dark polarity and contains a small square that is darkened on the image inside the hole. The fourth level has clear polarity and contains a small disk; the disk erases parts of the big and the small squares.

The file uses absolute notation with the leading zero's omitted. The units are millimeters.



### Example:

G04 This file illustrates how to use levels to create holes\*

%FSLAX27Y27\*%

%MOMM\*%

G04 First level: big square - dark polarity\*

%LPD\*%

G36\*

X2500000Y2500000D02\*

G01X17500000D01\*

G01Y17500000D01\*

G01X2500000D01\*

G01Y2500000D01\*

G37\*

G04 Second level: big circle - clear polarity\*

%LPC\*%

G36\*

G75\*

X5000000Y10000000D02\*

G03X5000000Y10000000I5000000J0D01\*

G37\*

G04 Third level: small square - dark polarity\*

%LPD\*%

G36\*

X7500000Y7500000D02\*

G01X12500000D01\*

G01Y12500000D01\*

G01X7500000D01\*

G01Y7500000D01\*

G37\*

G04 Fourth level: small circle - clear polarity\*

%LPC\*%

G36\*

G75\*

X11500000Y10000000D02\*

G03X11500000Y10000000I2500000J0D01\*

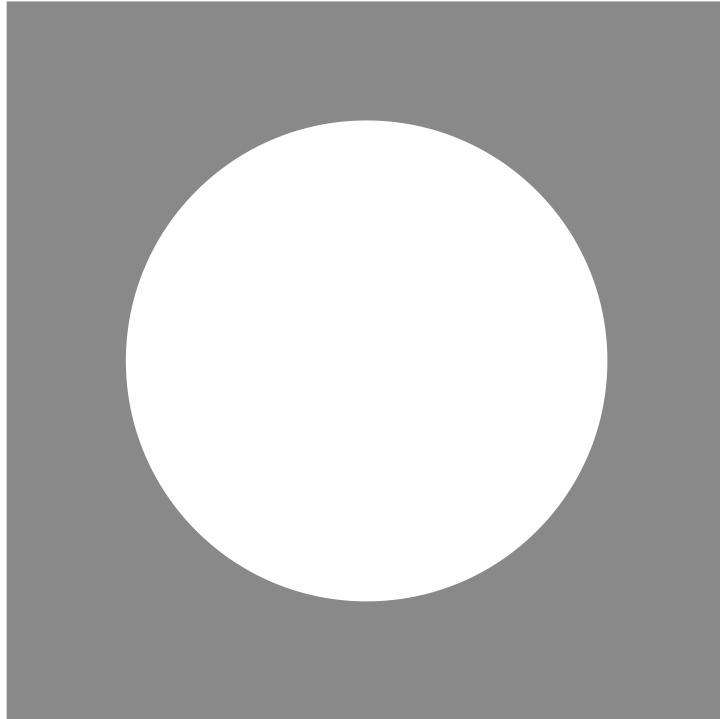
G37\*

M02\*

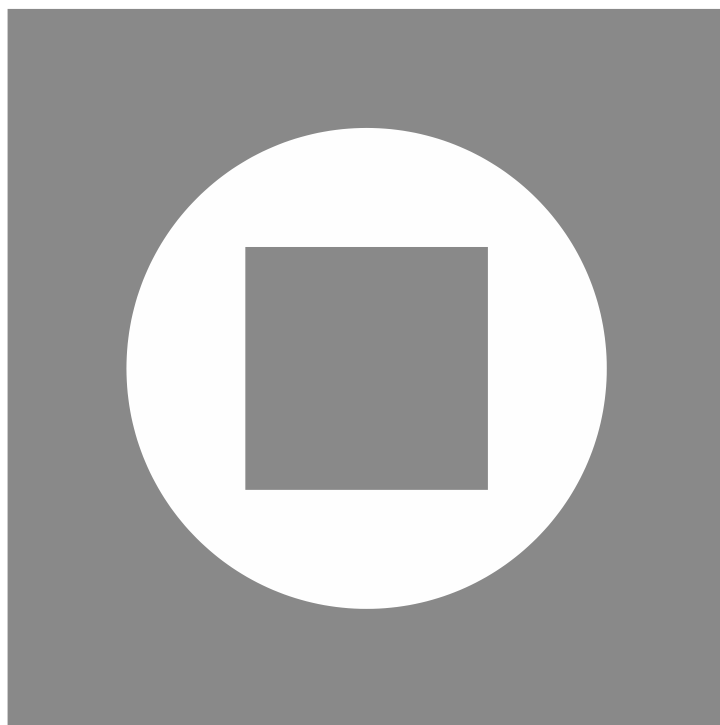
Below there are pictures which show the resulting image after adding each level.



*19. Resulting image: first level only*



*20. Resulting image: first and second levels*



*21. Resulting image: first, second and third levels*

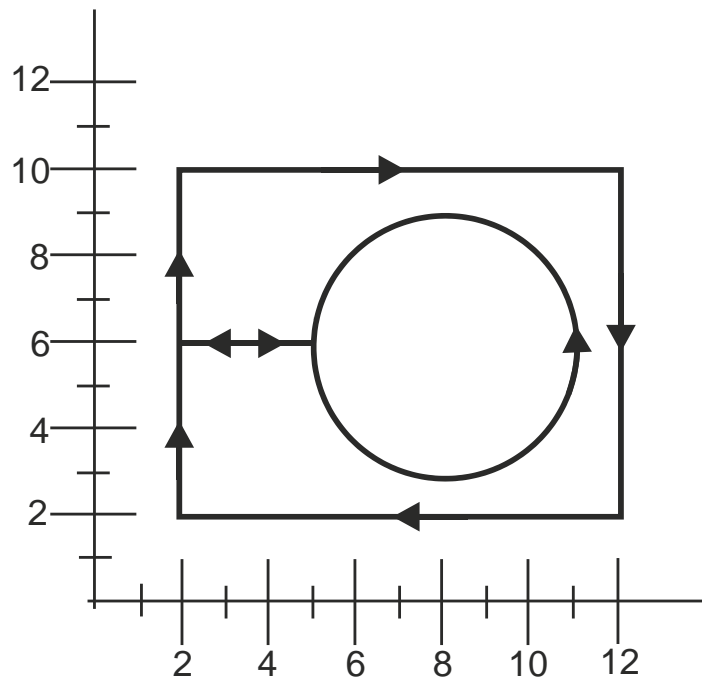




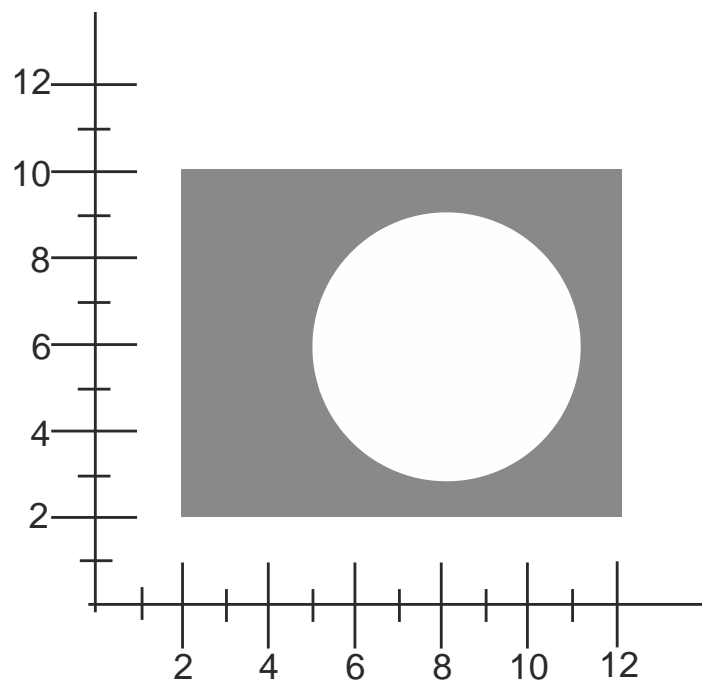
*22. Resulting image: all four levels*

## 4.5.9 Example: a simple cut-in

Syntax	Comments
G75*	Multi quadrant mode
G36*	Initiate a region
X200Y1000D02*	Move the current point to (2,10)
G01X1200D01*	Line segment to (12,10)
G01Y200D01*	Line segment to (12, 2)
G01X200D01*	Line segment to (2, 2)
G01Y600D01*	Line segment to (2, 6)
G01X500D01*	Line segment to (5, 6), coincident draw
G03X500Y600I300J0D01*	Full counterclockwise circle with radius 300
G01X200D01*	Line segment to (2, 6), coincident draw
G01Y1000D01*	Line segment to (2, 10)
G37*	Create the region by filling the contour



23. Simple cut-in: the segments



24. Simple cut-in: the image

## 4.5.10 Examples: coincident draws

### Example 1

G04 ex1: non overlapping\*

%FSLAX23Y23\*%

%MOMM\*%

%ADD10C,1.00000\*%

%LPD\*%

G36\*

X0Y5000D02\*

Y10000D01\*

X10000D01\*

Y0D01\*

X0D01\*

Y5000D01\*

X-1000D01\* First coincident draw

X-5000Y1000D01\*

X-9000Y5000D01\*

X-5000Y9000D01\*

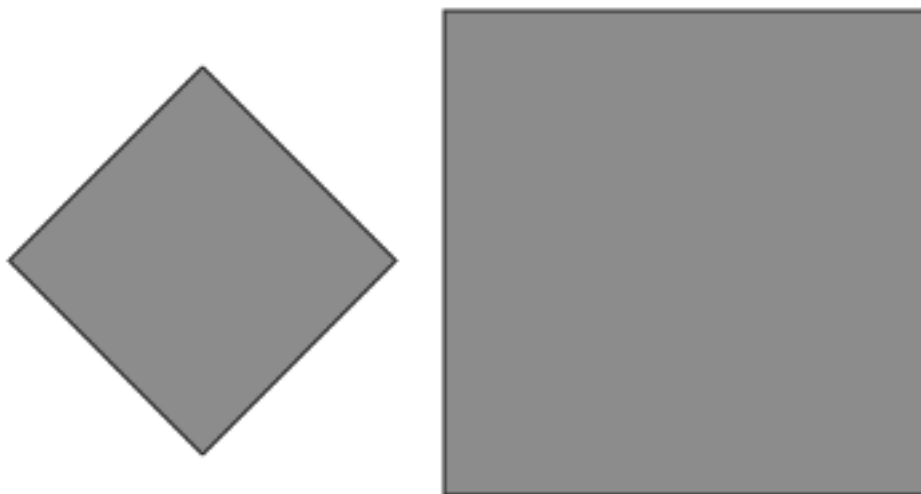
X-1000Y5000D01\*

X0D01\* Second coincident draw

G37\*

M02\*

This creates the following image:

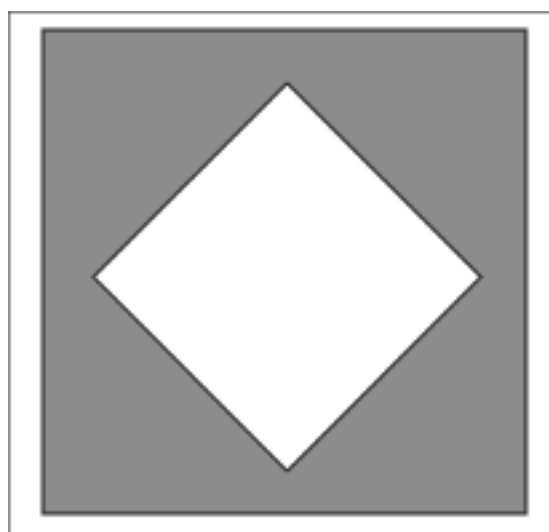


25. Coincident edges in contours, example 1

## Example 2

```
G04 ex2: overlapping*
%FSLAX23Y23*%
%MOMM*%
%SRX1Y1I0.000J0.000*%
%ADD10C,1.00000*%
%LPD*%
G36*
X0Y5000D02*
Y10000D01*
X10000D01*
Y0D01*
X0D01*
Y5000D01*
X1000D01*           First coincident draw
X5000Y1000D01*
X9000Y5000D01*
X5000Y9000D01*
X1000Y5000D01*
X0D01*             Second coincident draw
G37*
M02*
```

This creates the following image:



26. Coincident edges in contours, example 2

#### 4.5.11 Examples: valid and invalid cut-ins

Contours with cut-ins are susceptible to rounding problems: when the vertices move due to the rounding the contour may become self-intersecting. This may lead to unpredictable results. Example 2 is a cut-in with valid coincident segments, where draws which are on top of one another have the *same* end vertices. When the vertices move due to rounding, the draws will remain exactly on top of one another, and no self-intersections are created. This is a valid and robust construction.

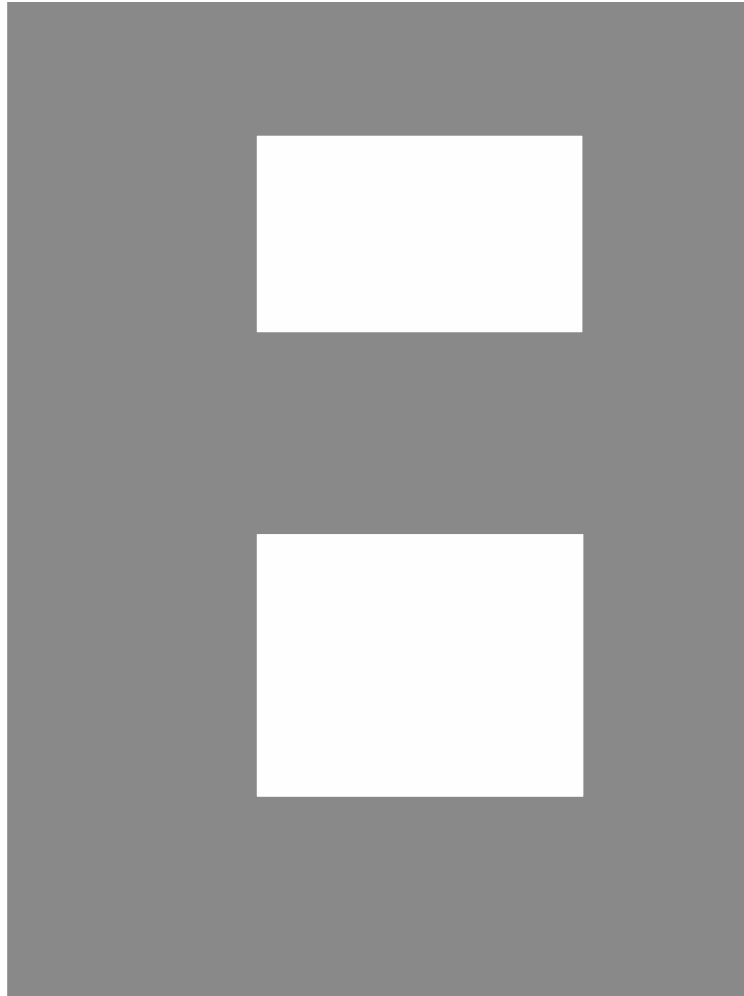
```
G36*  
X1220000Y2570000D02*  
G01Y2720000D01*  
G01X1310000D01*  
G01Y2570000D01*  
G01X1250000D01*  
G01Y2600000D01*  
G01X1290000D01*  
G01Y2640000D01*  
G01X1250000D01*  
G01Y2670000D01*  
G01X1290000D01*  
G01Y2700000D01*  
G01X1250000D01*  
G01Y2670000D01*  
G01Y2640000D01*  
G01Y2600000D01*  
G01Y2570000D01*  
G01X1220000D01*  
G37*
```

This results in the following contour:



## 27. Cut-in example 2: valid, coincident segments

This creates the following image:



*28. Cut-in example 2: resulting image*



Example 3 attempts to create the same image as example 2, but it is invalid due to the use of invalid partially coinciding segments. The number of draws has been reduced by eliminating vertices between collinear draws, creating invalid overlapping segments. This construction is invalid. It is not robust and hard to handle: when the vertices move slightly due to rounding, the draws that were on top of one another may become intersecting, with unpredictable results.

```
G36*  
X1110000Y2570000D02*  
G01Y2600000D01*  
G01X1140000D01*  
G01Y2640000D01*  
G01X1110000D01*  
G01Y2670000D01*  
G01X1140000D01*  
G01Y2700000D01*  
G01X1110000D01*  
G01Y2570000D01*  
G01X1090000D01*  
G01Y2720000D01*  
G01X1170000D01*  
G01Y2570000D01*  
G01X1110000D01*  
G37*
```

This results in the following contour:



29. Cut-in example 3: invalid, overlapping segments

## 4.6 Comment (G04)

The G04 function code is used for human readable comments. It does not affect the image.

The syntax for G04 is as follows.

**<Comment>: G(4|04)<Comment string>\***



**Example:**

G04 This is a comment\*

## 4.7 End-of-file (M02)

The M02 function code indicates the end of the file.

It is *mandatory* that the last data block in a Gerber file is the M02 function code. Readers are encouraged to report a missing M02 as this is an indication that the file has been truncated.

No data is allowed after an M02.

The syntax for M02 is as follows:

**<End of file marker>: M02\***

## 4.8 FS – Format Specification

The FS parameter specifies the format of the coordinate data. It must be used only once at the beginning of a file. It must be specified before the first use of coordinate data.



**Note:** It is recommended to put the FS parameter at the very first line, maybe after some general comments.

The FS parameter specifies the following format characteristics:

- ☐ Number of integer and decimal places in coordinate data (coordinate format)
- ☐ Zero omission (leading or trailing zero's omitted)
- ☐ Absolute or incremental coordinate notation



**Warning:** Explicit decimal points in coordinates are not allowed.

### 4.8.1 Coordinate Format

The coordinate format specifies the number of integer and decimal places in the coordinate data. For example, the “23” format specifies 2 integer and 3 decimal places. A maximum of 7 integer and 7 decimal places can be specified (nnnnnnn.nnnnnnn). The same format must be defined for X and Y. Signs are allowed. The ‘+’ sign is optional.



**Note:** In previous versions of the specification the implementation limit on integer and decimal places was 6. However, some applications started to generate 7 decimal places because they needed the accuracy. We adapted the specification to technology requirements and raised the limit to 7. However, there are probably still a number of Gerber readers in use that can only handle 6. It is therefore recommended to use 7 decimal places only if the extra accuracy is needed.



**Warning:** We strongly recommend using 6 decimal places in imperial and 5 decimal places in metric. A lower number of decimal places can lose vital precision. The option to use a lower

number of decimal places is a simplistic compression method introduced in the 1950's, when saving a few bytes was of paramount importance and computers were too feeble for proper compression algorithms. Nowadays the few bytes saved are irrelevant. Modern compression methods far outperform this simplistic method, without loss of accuracy. If the extra digits are not significant, they will be compressed away; if they are significant they should not be blindly removed. The benefits of a small number of decimal digits are long gone. The disadvantages remain. It is a source of endless confusion.

The resolution of a Gerber file is the distance expressed by the least significant digit of coordinate data. Thus the resolution is the size of grid steps of the coordinates.

The unit in which the coordinates are expressed is set by the %MO parameter. See 4.9.

## 4.8.2 Zero Omission

Zero omission allows reducing the size of data by omitting *either* leading or trailing zero's from the coordinate values.

With *leading zero omission* some or all leading zero's can be omitted but all trailing zero's are required. To interpret the coordinate string, it is first padded with zero's in front until its length fits the coordinate format. For example, with the "23" coordinate format, "015" is padded to "00015" and therefore represents 0.015.


With *trailing zero omission* some or all trailing zero's can be omitted but all leading zero's are required. To interpret the coordinate string, it is first padded with zero's at the back until its length fits the coordinate format. For example, with the "23" coordinate format, "15" is padded to "15000" and therefore represents 15.000.

Leading zero omission is easier to read.

If the coordinate data in the file does not omit zero's it is conventional to specify leading zero omission.

## 4.8.3 Absolute or Incremental Notation

Coordinate values can be expressed either as absolute coordinates (absolute notation) or as incremental distances from a previous coordinate position (incremental notation).

 **Warning:** It is recommended to use absolute notation only. With incremental notation rounding errors can accumulate, losing vital precision. With incremental notation Gerber files are no longer human readable, losing a big advantage. Incremental notation is a simplistic compression technology introduced in the 1950's, when saving a few bytes was of paramount importance and computers were too feeble for proper compression algorithms. Nowadays the few bytes saved are irrelevant. Modern compression methods far outperform this simplistic method, without any loss of accuracy. If the file size is important for you use a strong compression algorithm rather than sacrificing precision and clarity. The advantage of incremental notation is long gone. Its disadvantages remain. Incremental notation is a source of endless confusion. Always use absolute notation.

#### 4.8.4 Data Block Format

The syntax for the FS parameter is:

**<FS parameter>: FS(L|T)(A|I)X<Format>Y<Format>\***

Syntax	Comments
FS	FS for Format Specification
L T	Specifies zero omission mode: L – omit leading zero's T – omit trailing zero's
A I	Specifies coordinate values notation: A – absolute notation I – incremental notation
X<Format>Y<Format>	Specifies the format of X and Y coordinate data. The format of X and Y coordinates must be the same!  <Format> must be expressed as a number NM where N - number of integer positions in coordinate data ( $0 \leq N \leq 7$ ) M - number of decimal positions in coordinate data ( $0 \leq M \leq 7$ )

#### 4.8.5 Examples

Syntax	Comments
%FSLAX25Y25*%	Coordinate data has leading zero's omitted. Coordinates are expressed using absolute notation with 2 integer and 5 decimal positions for both axes.

### 4.9 MO – Mode

The MO parameter sets the units used for coordinate data and for sizes parameters or modifiers indicating sizes or coordinates. The units can be either inches or millimeters. This parameter must be used only once, at the beginning of the file.



**Note:** the FS parameter sets the format (i.e. number of integer and decimal positions) of the coordinate data.

#### 4.9.1 Data Block Format

The syntax for the MO parameter is:

**<MO parameter>: MO(IN|MM)\***

Syntax	Comments
MO	MO for Mode
IN MM	Units of the dimension data: IN – inches MM – millimeters

## 4.9.2 Examples

Syntax	Comments
%MOIN*%	Dimensions are expressed in inches
%MOMM*%	Dimensions are expressed in millimeters

## 4.10 AD - Aperture Definition

### 4.10.1 Syntax Rules

The AD parameter starts with 'AD', followed by 'D' and D-code number, then the aperture type and then optionally modifiers. . The AD parameter assigns a D-code number, also called aperture number, to an aperture (shape) and sets the aperture as current aperture.

The AD parameter must precede the first use of the assigned aperture. It is recommended to put all AD parameters in the beginning of the file.

The allowed range of D-code is from 10 up to 2147483647 (max int32). The D-codes 1 to 9 are reserved and *cannot* be used for apertures. Once a D-code number is assigned it cannot be re-assigned.



**Warning:** In older versions of the specification the maximum D-code was 999 and legacy applications have limitations. Use low D-codes when possible.



**Example:**

```
%ADD10C, .025*%
```



**Note:** For readability it is recommended to enclose each AD parameter into a separate pair of '%' characters.

The data block for the AD parameter is as follows:

**<AD parameter>: ADD<D-code number><Aperture type>[,<Modifiers set>]\***

**<Modifiers set>: <Modifier>{X<Modifier>}**

Syntax	Comments
--------	----------

ADD	AD for Aperture Definition and D for D-code
<D-code number>	The D-code number being defined ( $\geq 10$ )
<Aperture type>[,<Modifiers set>]	The aperture type optionally followed by modifiers

The **<Aperture type>** can be in one of two available formats:

- ❑ For a standard aperture: one of the standard aperture codes (C,R,O or P)
- ❑ For a special aperture: an aperture macro name previously defined by an AM parameter

The required number of modifiers in <Modifiers set> depends on the <Aperture type>. Modifiers are separated by the 'X' character. All sizes are decimal numbers, units follow the MO parameter. The FS parameter has no effect on aperture sizes.

Standard apertures may be solid or open. Open means there is a hole in the aperture. Holes are not part of the aperture and do not affect the image. Holes do not clear the objects under them. Objects under a hole remain and are visible. Holes are therefore called transparent. This is not the same as clear level polarity where all objects underneath are cleared.

The syntax of a hole is common for all standard apertures:

**<Hole>:    <X-axis hole size >[X<Y-axis hole size>]**

If only the **<X-axis hole size>** modifier is specified the hole is round, and the modifier specifies the diameter. If both X and Y is specified the hole is rectangular and the modifiers specify the X and Y size. If both parameters are omitted the aperture is solid. If present the sizes must be  $\geq 0$ .

The hole must fit within the aperture. It is centered on the aperture.

## 4.10.2 Standard Apertures

### 4.10.2.1 Circle

The syntax of the circle standard aperture:

**C,<Diameter>[X<Hole>]**

Syntax	Comments
C	Indicates that this is a circle aperture
<Diameter>	Circle diameter, $\geq 0$
<Hole>	Optional hole. If no hole is specified the aperture is solid.



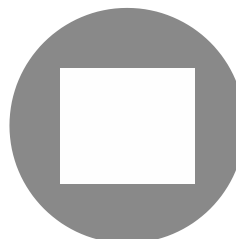
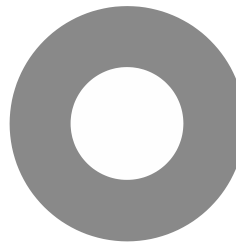
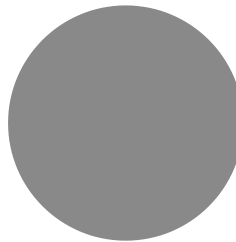
#### Examples:

These commands define the apertures below

```
%ADD10C,0.5*%
```

```
%ADD10C,0.5X0.25*%
```

```
%ADD10C,0.5X0.29X0.29*%
```



30. Circles with different holes



#### 4.10.2.2 Rectangle

The syntax of the rectangle or square standard aperture:

**R,<X size>X<Y size>[X<Hole>]**

Syntax	Comments
R	Indicates that this is a rectangle or square aperture
<X size> <Y size>	X and Y sizes of the rectangle, both must be >0. If the <X size> equals the <Y size>, the aperture is square
<Hole>	Optional hole If no hole is specified the aperture is solid



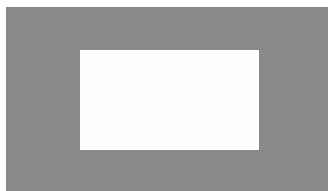
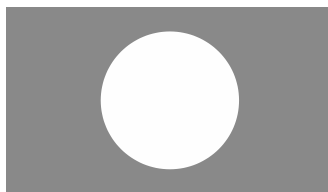
#### Examples:

These commands define the apertures below

```
%ADD22R,0.044X0.025*%
```

```
%ADD22R,0.044X0.025X0.019*%
```

```
%ADD22R,0.044X0.025X0.024X0.013*%
```



*31. Rectangles with different holes*

#### 4.10.2.3 Obround

Obround (oval) is a shape consisting of two semicircles connected by parallel lines tangent to their endpoints. The syntax of the obround standard aperture:

**O,<X size>X<Y size>[X<Hole>]**

Syntax	Comments
O	Indicates that this is an obround aperture
<X size> <Y size>	X and Y sizes of the obround sides, both must be > 0. The smallest side is terminated by half a circle. If the <X size> is larger than <Y size>, the shape is horizontal. If the <X size> is smaller than <Y size>, the shape is vertical. If the <X size> is equal to <Y size>, the shape is a circle
<Hole>	Optional hole. If no hole is specified, the aperture is solid



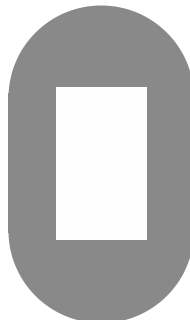
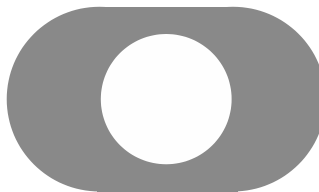
#### Example:

These commands define the apertures below

```
%ADD22O,0.046X0.026*%
```

```
%ADD22O,0.046X0.026X0.019*%
```

```
%ADD22O,0.026X0.046X0.013X0.022*%
```



32. Obrounds with different holes

#### 4.10.2.4 Regular polygon

The syntax of the polygon standard aperture:

**P,<Outer diameter>X<Number of vertices>[X<Degrees of rotation>[X<Hole>]]**

Syntax	Comments
P	Indicates that this is a polygon aperture
<Outer diameter>	Diameter of the circumscribed circle, i.e. the circle through the polygon vertices, must be > 0.
<Number of vertices>	Number of polygon vertices, ranging from 3 to 12
<Degrees of rotation>	Specifies rotation of the aperture around its center. Without rotation one vertex is on the positive X-axis through the center. Rotation angle is expressed in decimal degrees; positive value for counterclockwise rotation, negative value for clockwise rotation.
<Hole>	Optional hole. If no hole is specified the aperture is solid. The hole modifiers can be specified only after a rotation angle; set an angle of zero the aperture is not rotated.



**Note:** The orientation of the hole is not affected by the rotation angle modifier.



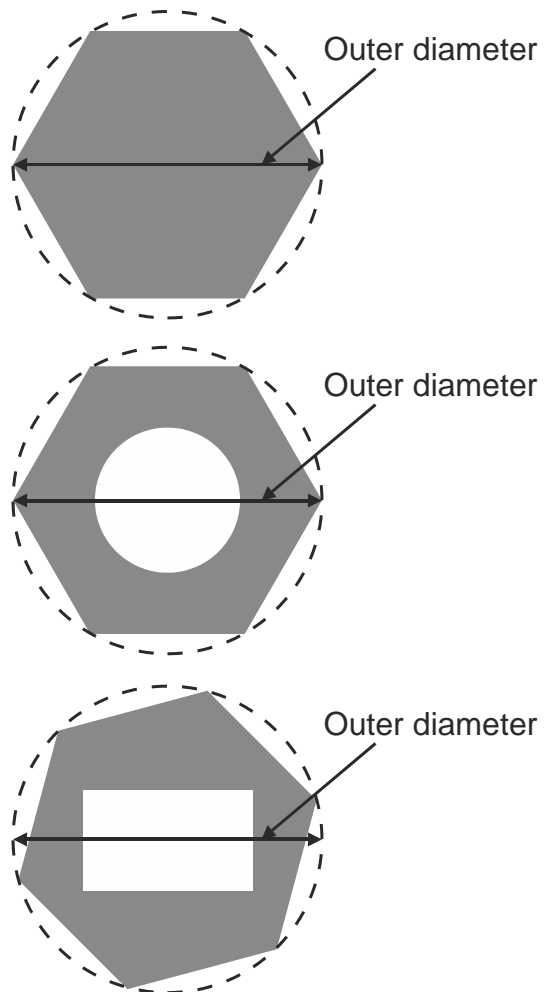
### Examples:

These commands define the apertures below

```
%ADD17P, .040X6*%
```

```
%ADD17P, .040X6X0.0X0.019*%
```

```
%ADD17P, .040X6X15.0X0.023 X0.013*%
```




33. *Polygons with different holes*

### 4.10.3 Zero-size apertures

Zero-size C standard (*circular*) apertures are allowed. No other standard or special aperture with zero size or that do not effectively represent an image is *allowed*, even if the normal shape would be a circle.

Graphics objects created with a zero-size circular aperture are valid objects. These zero-objects do not affect the image - they neither darken (mark) or clear (erase) the image plane. Attributes can be associated with zero-objects. The sole purpose of zero-objects is to provide meta-information such as reference points or an outline

 **Warning:** Only add zero-objects when needed to provide meta-information. Do not add needless zero-objects. Certainly do not abuse a zero-object to indicate the absence of an object. Needless zero-objects cause confusion and errors.

### 4.10.4 Examples

Syntax	Comments
%ADD10C,.025*%	D-code 10 is a solid circle with diameter 0.025
%ADD22R,.050X.050X.027*%	D-code 22 is a square with sides of 0.05 and with a 0.027 diameter round hole
%ADD57O,.030X.040X.015*%	D-code 57 is an obround with sizes 0.03 x 0.04 with 0.015 diameter round hole
%ADD30P,.016X6*%	D-code 30 is a solid polygon with 0.016 outer diameter and 6 vertices
%ADD15CIRC*%	D-code 15 is a special aperture described by aperture macro CIRC defined previously by an aperture macro (AM) parameter

## 4.11 AM - Aperture Macro

The AM parameter defines special apertures consisting of building blocks called primitives.

A special aperture macros defined by the AM parameter can be referenced from AD parameter definitions in the same way as the standard apertures. (One could view the standard apertures as pre-defined macro apertures.) A special aperture must be defined before a D-code number can be assigned to it.

Special apertures offer two advantages compared to standard apertures:

- ❑ Multiple shapes called primitives can be combined in a single aperture to create non-standard aperture shapes
- ❑ Aperture macro modifiers can be variable; the actual values can be defined as:
- ❑ Values provided by an AD parameter referencing the aperture macro
- ❑ Arithmetic expressions calculating the actual value based on other variables

The AM parameter can be used multiple times in a file. It must be put at the beginning of a file or level.

### 4.11.1 Data Block Format

The syntax for the AM parameter is:

**<AM parameter>:      AM<Aperture macro name>\*<Macro content>**  
**<Macro content>:      {{<Variable definition>\*<Primitive>}}**  
**<Variable definition>: \$K=<Arithmetic expression>**  
**<Primitive>:            <Primitive code>,<Modifier>{,<Modifier>}<Comment>**  
**<Modifier>:            \$M|< Arithmetic expression>**  
**<Comment>:            0 <Text>**

Syntax	Comments
AM	AM for Aperture Macro
<Aperture macro name>	Name of the aperture macro. See 3.2 for the syntax rules.
<Macro content>	Macro content describes primitives included into the aperture macro. Can also contain definitions of new variables.
<Variable definition>	Definition of a variable.
\$K=<Arithmetic expression>	Definition of the variable \$K. An arithmetic expression may use arithmetic operators (described later), constant numbers and also variables \$X where the definition of \$X precedes \$K.
<Primitive>	Individual primitive that includes primitive code and modifiers which are primitive parameters (e.g. diameter for a circle). The modifiers depend on the primitive. All primitives are described later in this document.
<Primitive code>	A code specifying the primitive (e.g. polygon).
<Modifier>	Modifier can be a decimal number (e.g. 0.050), a variable (e.g. \$1) or an arithmetic expression based on numbers and variables. The actual value for a variable is either provided by an AD parameter or defined within the AM by some previous <Variable definition>.
<Comment>	Comment does not affect the image.
<Text>	Single-line text string

Modifiers can be classified as in the following table:

Modifier type	Comments
Exposure modifier	The exposure modifier that can have the following values: 0 means exposure is 'off' 1 means exposure is 'on'
Rotation modifier	Modifier that specifies the rotation angle: positive value means counterclockwise rotation, negative - clockwise.

Geometry modifier	Modifier that specifies e.g. a diameter, X and Y positions, width, etc.
-------------------	-------------------------------------------------------------------------

Coordinates and sizes are expressed in the unit set by the MO parameter.

Exposure is set 'on' or 'off' by the exposure modifier. Exposure 'on' creates a solid part of the macro aperture. Exposure 'off' clears or erases the underlying solid part to create a hole in it. Note that the clearing action only affects the macro aperture definition itself and is not passed through to affect the image that the macro aperture is used on. Exposure off only creates a hole in the macro aperture. Holes are not part of the aperture. They have no effect on the image. Note that holes do not clear the objects under them. Objects under a hole remain and are visible. Holes are therefore called transparent. This is not the same as clear polarity where all objects underneath are cleared.

The rotation angle is expressed in degrees, by a decimal number; a positive value means counterclockwise rotation, a negative value means clockwise rotation. The pivot of the rotation of a primitive is always the origin, i.e. the point (0,0). To rotate a macro composed of several primitives it is then sufficient to rotate all primitives with the same angle. Note that for the polygon, thermal and moiré rotation is only allowed if their center is placed on the origin.

## 4.11.2 Primitives

### 4.11.2.1 Comment, primitive code 0

The comment primitive has no image meaning. It is used to include human-readable comments into the AM parameter. The comment primitive starts with the '0' code followed by a space and then a single-line text string. The text string follows the syntax rules for comments as described in section 3.1.



#### Example:

```
%AMRECTROUND CORNERS*
0 Rectangle with rounded corners. *
0 Offsets $4 and $5 are interpreted as the *
0 offset of the flash origin from the pad center. *
0 First create horizontal rectangle. *
21,1,$1,$2-$3-$3,0-$4,0-$5,0*
0 From now on, use width and height half-sizes. *
$9=$1/2*
$8=$2/2*
0 Add top and bottom rectangles. *
22,1,$1-$3-$3,$3,0-$9+$3-$4,$8-$3-$5,0*
22,1,$1-$3-$3,$3,0-$9+$3-$4,0-$8-$5,0*
0 Add circles at the corners. *
1,1,$3+$3,0-$4+$9-$3,0-$5+$8-$3*
1,1,$3+$3,0-$4-$9+$3,0-$5+$8-$3*
1,1,$3+$3,0-$4-$9+$3,0-$5-$8+$3*
1,1,$3+$3,0-$4+$9-$3,0-$5-$8+$3%
```

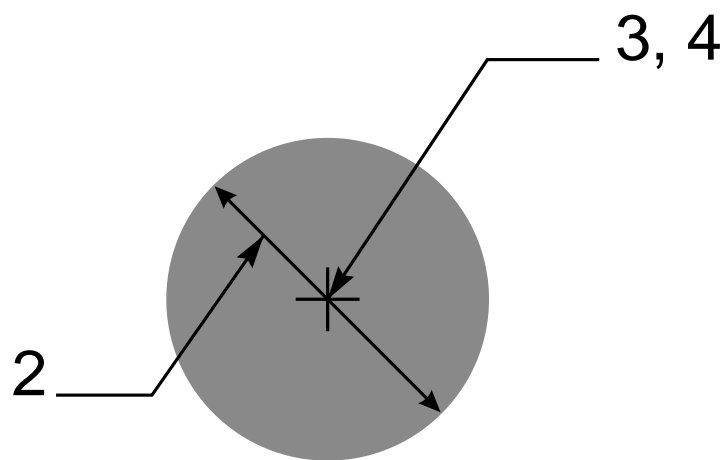
In the example above all the lines starting with 0 are comments and do not affect the image.



#### 4.11.2.2 Circle, primitive code 1

A circle primitive is defined by its center point and diameter.

Modifier number	Description
1	Exposure off/on (0/1)
2	Diameter, $\geq 0$
3	X coordinate of center position
4	Y coordinate of center position



34. Circle primitive



#### Example:

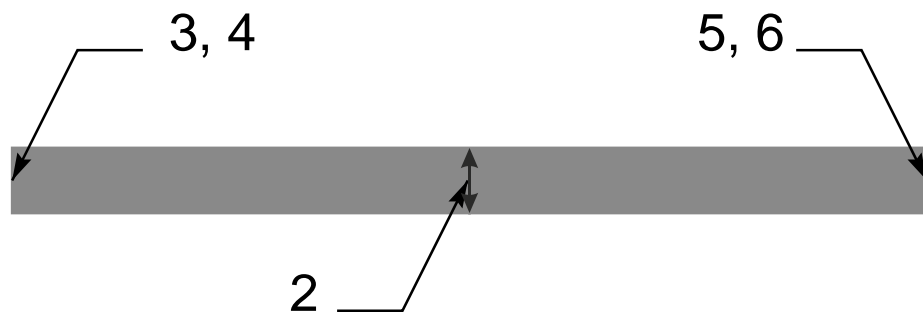
```
%AMCIRCLE*
```

```
1,1,1.5,0,0*%
```

#### 4.11.2.3 Vector Line, primitive code 2 or 20.

A vector line is a rectangle defined by its line width, start and end points. The line ends are rectangular.

Modifier number	Description
1	Exposure off/on (0/1)
2	Line width, $\geq 0$
3	X coordinate of start point
4	Y coordinate of start point
5	X coordinate of end point
6	Y coordinate of end point
7	Rotation angle around the <i>origin</i> (rotation is <i>not</i> around the center of the object)



#### 35. Line (vector) primitive



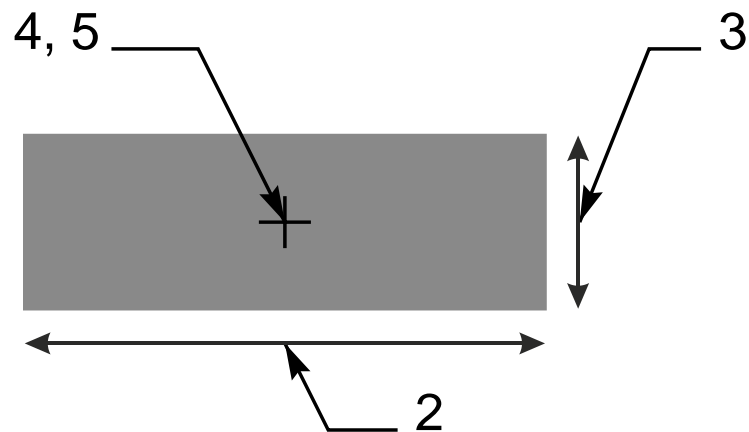
**Example:**

```
%AMLIN*20,1,0.9,0,0.45,12,0.45,0*%
```

#### 4.11.2.4 Center Line, primitive code 21

A center line primitive is a rectangle defined by its width, height, and center point..

Modifier number	Description
1	Exposure off/on (0/1))
2	Rectangle width, $\geq 0$
3	Rectangle height, $\geq 0$
4	X coordinate of center point
5	Y coordinate of center point
6	Rotation angle around the <i>origin</i> (rotation is <i>not</i> around the center of the object)



#### 36. Line (center) primitive



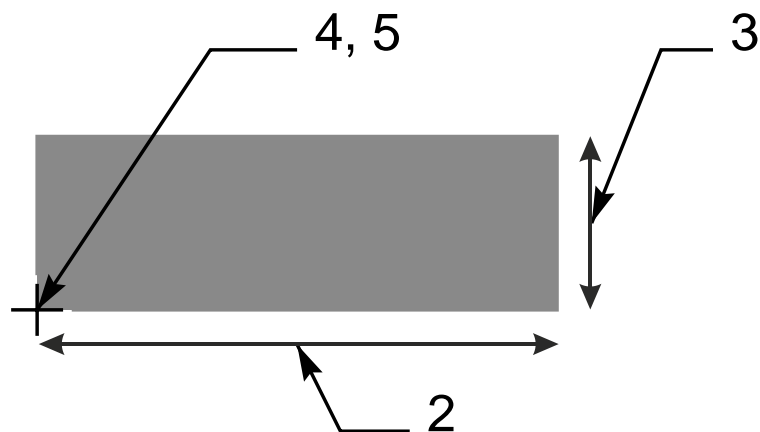
#### Example:

```
%AMRECTANGLE*21,1,6.8,1.2,3.4,0.6,0*%
```

#### 4.11.2.5 Lower Left Line, primitive code 22

A lower left line primitive is a rectangle defined by its width, height, and the lower left point.

Modifier number	Description
1	Exposure off/on (0/1))
2	Rectangle width, $\geq 0$
3	Rectangle height, $\geq 0$
4	X coordinate of lower left point
5	Y coordinate of lower left point
6	Rotation angle around the <i>origin</i> (rotation is <i>not</i> around the center of the object)



37. Line (lower left) primitive



#### Example:

```
%AMLIN2*22,1,6.8,1.2,0,0,0*%
```

#### 4.11.2.6 Outline, primitive code 4

An outline primitive is an area enclosed by an n-point polygon defined by its start point and n subsequent points. The outline must be closed, i.e. the last point must be equal to the start point. There must be at least one subsequent point (to close the outline).

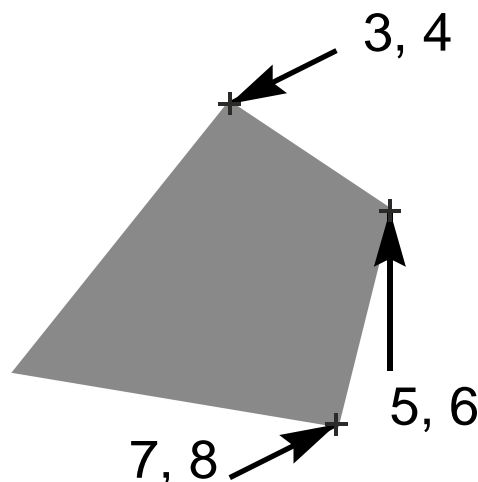
Self-intersecting outlines are *not allowed*. Segments *cannot* cross, overlap or touch except:

1. Consecutive segments connecting in their endpoints, needed to construct the outline.
2. Horizontal or vertical coincident segments, used to create holes in a region with cut-ins; see 4.5.9. A pair of segments are said to be coincident if and only if the segments coincide completely, with the second segment starting where the first one ends.

Any other form of self-touching or self-intersection is *not allowed*.

**Warning:** Make no mistake: n is the number of *subsequent* points, being the number of vertices of the outline or one less than the number of coordinate pairs.

Modifier number	Description
1	Exposure off/on (0/1)
2	The number of subsequent points n ( $n \geq 1$ )
3, 4	X and Y coordinates of the start point
5, 6	X and Y coordinates of subsequent point number 1
...	X and Y coordinates of further subsequent points
3+2n, 4+2n	X and Y coordinates of subsequent point number n. Must be equal to coordinates of start point
5+2n	Rotation angle around the <i>origin</i> (rotation is <i>not</i> around the center of the object)



38. Outline primitive

The X and Y coordinates are not modal: both the X and the Y coordinate must be specified for all points.



**Note:** Older versions of the specification defined the maximum of 50 for the number of subsequent points n. This has proven to be too restrictive, and the limit is now increased to 4000.



**Example:**

```
%AMOUTLINE*
```

```
4, 1, 4,
```

```
0.1, 0.1,
```

```
0.5, 0.1,
```

```
0.5, 0.5,
```

```
0.1, 0.5,
```

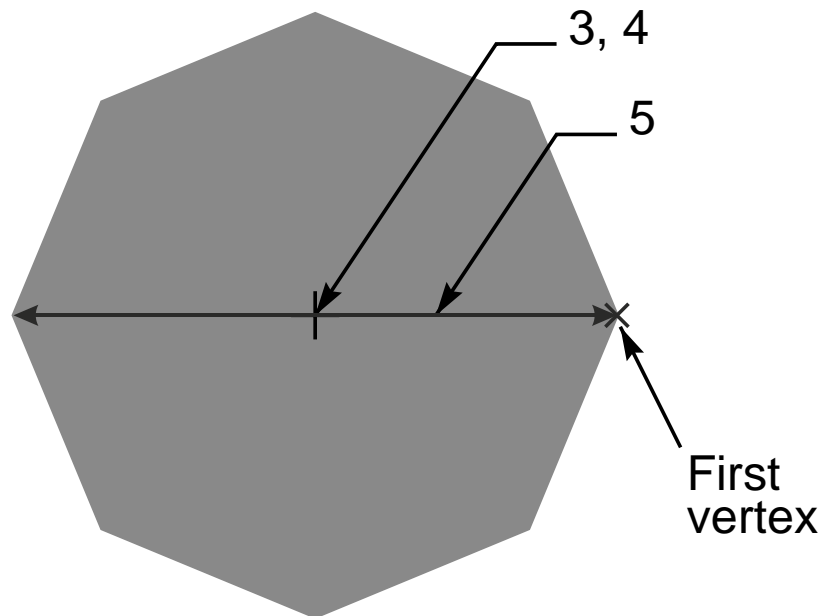
```
0.1, 0.1,
```

```
0*%
```

#### 4.11.2.8 Polygon, primitive code 5

A polygon primitive is a regular polygon defined by the number of vertices  $n$ , the center point and the diameter of the circumscribed circle.

Modifier number	Description
1	Exposure off/on (0/1)
2	Number of vertices $n$ , $3 \leq n \leq 12$
3	X coordinate of center point
4	Y coordinate of center point
5	Diameter of the circumscribed circle, $\geq 0$
6	Rotation angle around the <i>origin</i> . Rotation is only allowed if the center point is on the origin. When the rotation angle is zero, the first vertex is on the positive X-axis through the center point



#### 39. Polygon primitive



#### Example:

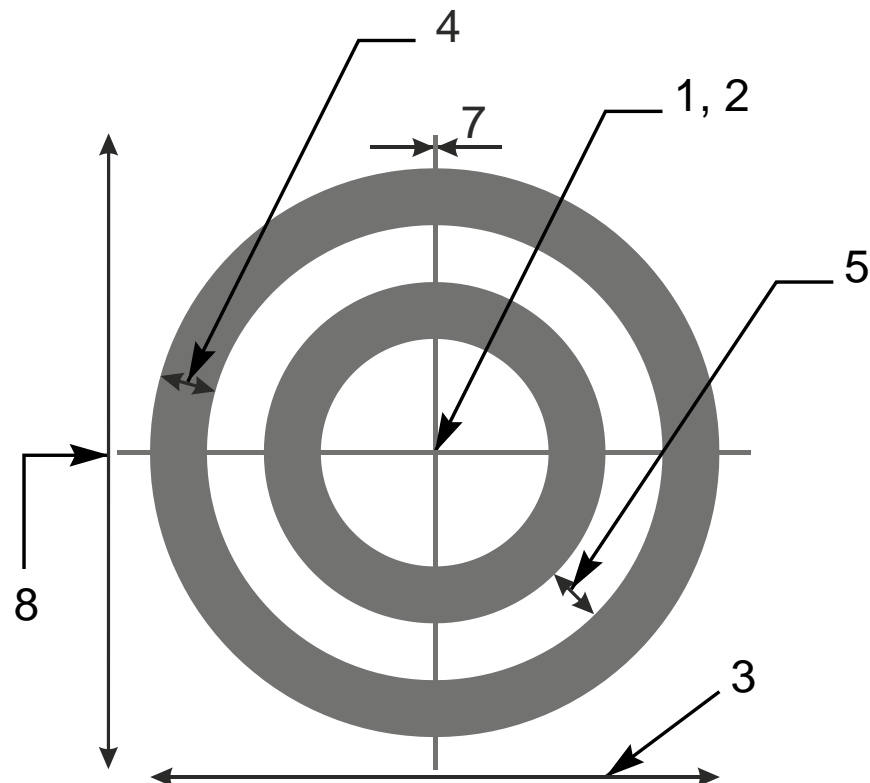
```
%AMPOLYGON*
```

```
5, 1, 8, 0, 0, 8, 0*%
```

#### 4.11.2.9 Moiré, primitive code 6

The moiré primitive is a cross hair centered on concentric rings (annuli). Exposure is always on.

Modifier number	Description
1	X coordinate of center point
2	Y coordinate of center point
3	Outer diameter of outer concentric ring
4	Ring thickness
5	Gap between rings
6	Maximum number of rings
7	Cross hair thickness
8	Cross hair length
9	Rotation angle around the <i>origin</i> . Rotation is only allowed if the center point is on the origin.



40. Moiré primitive

The outer diameter of the outer ring is specified by modifier 3. The ring has the thickness defined by modifier 4. Moving further towards the center there is a gap defined by modifier 5, and then the second ring etc. The maximum number of rings is defined by modifier 6. The



number of rings can be less if the center is reached. If there is not enough space for the last ring it becomes a full disc centered on the origin.



**Example:**

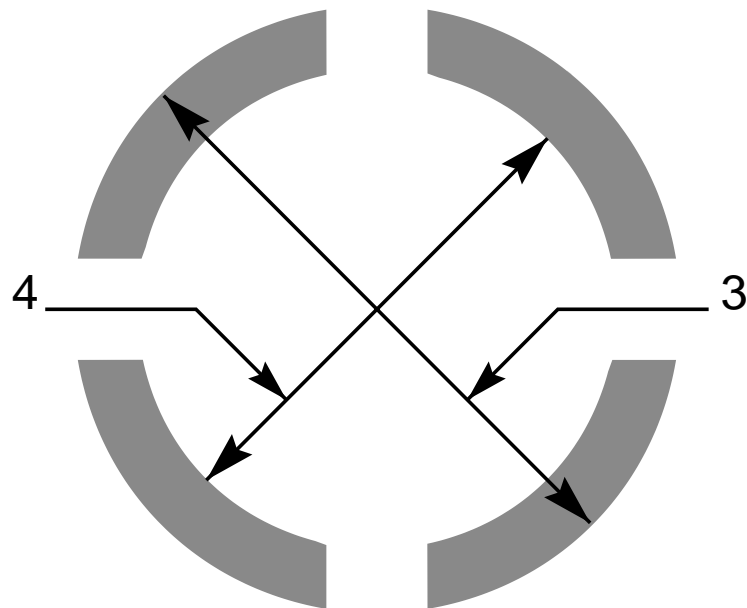
%AMMOIRE\*

6,0,0,5,0.5,0.5,2,0.1,6,0\*%

#### 4.11.2.10 Thermal, primitive code 7

The thermal primitive is a ring (annulus) interrupted by four gaps. Exposure is always on.

Modifier number	Description
1	X coordinate of center point
2	Y coordinate of center point
3	Outer diameter > Inner diameter
4	Inner diameter $\geq 0$
5	Gap thickness < Outer diameter/ $\sqrt{2}$
6	Rotation angle around the origin. Rotation is only allowed if the center point is on the origin. If the rotation angle is zero the gaps are on the X and Y axes through the center.



41. Thermal primitive

□



**Note:** Note that if the gap thickness\* $\sqrt{2} \geq$  Inner diameter the inner circle disappears. This is not invalid.

### 4.11.3 Parameter Contents

An aperture macro definition must contain an aperture macro name that later can be referenced from an AD parameter. An aperture macro definition also contains one or more of the aperture primitives described above. Each primitive (besides the special comment primitive) includes modifiers setting exposure, position, size, rotation etc. Primitive modifiers can use variables. Variables are indicated by a valid name beginning with a '\$'. The values for such variables must either be provided by an AD parameter or calculated using an arithmetic expression over other variables.

### 4.11.4 Syntax Rules

Each AM definition must be enclosed into a separate pair of '%' characters.



**Warning:** An AM definition cannot be grouped. This is different from the other parameters.

As an AM definition can be quite long, it can contain line separators to enhance readability. Line separators are ignored within an AM definition.

An AM parameter can contain the following types of data blocks:

- ❑ AM declaration
- ❑ Primitive with modifiers
- ❑ Variable definition by an arithmetic expression
- ❑ Comment (special comment primitive)

Each data block must end with the end-of-block '\*' character. Within a primitive data block each modifier must be separated by a comma. A modifier can be one of the following:

- ❑ A decimal number, such as 0, 1, 2, or 9.05
- ❑ A variable, such as \$1 or \$VAR
- ❑ An arithmetic expression including numbers and other variables.

#### 4.11.4.1 Variable values from an AD Parameter

An AM parameter can use variables, whose actual values can be provided by an AD parameter. Such variables are identified by '\$n' where n indicates the serial number of the variable value in the list provided by an AD parameter. Thus \$1 means the first value in the list, \$2 the second, and so on.



**Example:**

```
%AMDONUTVAR*1,1,$1,$2,$3*1,0,$4,$2,$3*%
```

Here the variables \$1, \$2, \$3 and \$4 are used as modifier values. The corresponding AD parameter should look like:

```
%ADD34DONUTVAR,0.100X0X0X0.080*%
```

In this case the value of variable \$1 becomes 0.100, \$2 and \$3 become 0 and \$4 becomes 0.080. These values are used as values of corresponding modifiers in the DONUTVAR macro.

#### 4.11.4.2 Arithmetic expressions

A modifier value can also be defined as an arithmetic expression that includes basic arithmetic operators such as 'add' or 'multiply', constant numbers (with or without decimal point) and other variables. The following arithmetic operators can be used:

Operator	Function
+	Add
-	Subtract
x (lowercase)	Multiply
/	Divide

*Arithmetic operators*

The result of the divide operation is decimal; it is not rounded or truncated to an integer.

The standard arithmetic precedence rules apply. Below operators are listed in order from lowest to highest priority. The brackets '(' and ')' can be used to overrule the standard precedence rules.

- ❑ Add and subtract: '+' and '-'
- ❑ Multiply and divide: 'x' and '/'
- ❑ Brackets: '(' and ')'

There is no unary minus operator. Negative values can be expressed by subtracting from zero, e.g. '0-\$1'.



#### Example:

%AMRECT\*

21, 1, \$1, \$2-\$3-\$3, 0-\$4, 0-\$5, 0\*%

Corresponding AD parameter could look like:

%ADD146RECT,0.0807087X0.1023622X0.0118110X0.5000000X0.3000000\*%

#### 4.11.4.3 Definition of a new variable

The AM parameter allows defining new variables based on previously defined variables. A new variable is defined as an arithmetic expression that follows the same rules as described in previous section. A variable definition also includes '=' sign with the meaning 'assign'.

For example, to define variable \$4 as a variable \$1 multiplied by 0.75 the following arithmetic expression can be used:

\$4=\$1x0.75



### Example:

```
%AMDONUTCAL*  
1, 1, $1, $2, $3*  
$4=$1x0.75*  
1, 0, $4, $2, $3*%
```

Local variables with names as. \$XSIZE, \$YSIZE, are allowed. Variable names are subject to the syntax rules in section 3.2 and must begin with a "\$"..



### Example:

```
%AMREC2*$XSIZE=$1*$YSIZE=$2*21, 1, $XSIZE, $YSIZE, 0, 0, 0*%
```

Here local variables \$XSIZE and \$YSIZE are defined and initialized to \$1 and \$2 values.

The values for variables in an AM parameter are determined as follows:

- ❑ All variables used in AM parameter are initialized to 0
- ❑ If an AD parameter that references the aperture macro contains N modifiers then variables \$1,\$2, ..., \$N get the values of these modifiers
- ❑ The remaining variables get their values from definitions in the AM parameter; if some variable remains undefined then its value is still 0
- ❑ The values of variables \$1, \$2, ..., \$N can also be modified by definitions in AM, i.e. the values originating from an AD parameter can be redefined



### Example:

```
%AMDONUTCAL*1, 1, $1, $2, $3*$4=$1x0.75*1, 0, $4, $2, $3*%
```

The variables \$1, \$2, \$3, \$4 are initially set to 0.

If the corresponding AD parameter contains only 2 modifiers then the value of \$3 will remain 0.

If the corresponding AD parameter contains 4 modifiers. e.g.

```
%ADD35DONUTCAL, 0.020X0X0X0.03*%
```

the variable values are calculated as follows: the AD parameter modifier values are first assigned so variable values \$1 = 0.02, \$2 = 0, \$3 = 0, \$4 = 0.03. The value of \$4 is modified by definition in AM parameter so it becomes \$4 = 0.02 x 0.75 = 0.015.

The variable definitions and primitives are handled from the left to the right in the order of AM parameter. This means a variable can be set to a value, used in a primitive, re-set to a new value, used in a next primitive etc.



**Example:**

```
%AMTARGET*1,1,$1,0,0*$1=$1x0.8*1,0,$1,0,0*$1=$1x0.8*1,1,$1,0,0*$1=$1x0.8*1,0,$1,0,0*$1=$1x0.8*1,1,$1,0,0*$1=$1x0.8*1,0,$1,0,0*%
%ADD37TARGET,0.020*%
```



Here the value of \$1 is changed by the expression '\$1=\$1x0.8' after each primitive so the value changes like the following: 0.020, 0.016, 0.0128, 0.01024, 0.008192, 0.0065536.



### Example:

```
%AMREC1*$2=$1*$1=$2*21,1,$1,$2,0,0,0*%
%AMREC2*$1=$2*$2=$1*21,1,$1,$2,0,0,0*%
%ADD51REC1,0.02X0.01*%
%ADD52REC2,0.02X0.01*%
```

Aperture 51 is the square with side 0.02 and aperture 52 is the square with side 0.01, because the variable values in AM parameters are calculated as follows:

For the aperture 51 initially \$1 is 0.02 and \$2 is 0.01. After operation '\$2=\$1' the variable values become \$2 = 0.02 and \$1 = 0.02. After the next operation '\$1=\$2' they remain \$2 = 0.02 and \$1 = 0.02 because previous operation changed \$2 to 0.02. The resulting primitive has 0.02 width and height.

For the aperture 52 initially \$1 is 0.02 and \$2 is 0.01 (the same as for aperture 51). After operation '\$1=\$2' the variable values become \$1 = 0.01 and \$2 = 0.01. After the next operation '\$2=\$1' they remain \$1 = 0.01 and \$2 = 0.01 because previous operation changed \$1 to 0.01. The resulting primitive has 0.01 width and height.

Below are some more examples of using arithmetic expressions in AM parameter. Note that some of these examples probably do not represent a reasonable aperture macro – they just illustrate the syntax that can be used for defining new variables and modifier values.



**Example:**

%AMTEST\*

1, 1, \$1, \$2, \$3\*

\$4=\$1×0.75\*

\$5=(\$2+100)×1.75\*

1, 0, \$4, \$5, \$3\*%

%AMTEST\*

\$4=\$1×0.75\*

\$5=100+\$3\*

1, 1, \$1, \$2, \$3\*

1, 0, \$4, \$2, \$5\*

\$6=\$4×0.5\*

1, 0, \$6, \$2, \$5\*%

%AMRECTROUND CORNERS\*

21, 1, \$1, \$2-\$3-\$3, 0-\$4, 0-\$5, 0\*

\$9=\$1/2\*

\$8=\$2/2\*

22, 1, \$1-\$3-\$3, \$3, 0-\$9+\$3-\$4, \$8-\$3-\$5, 0\*

22, 1, \$1-\$3-\$3, \$3, 0-\$9+\$3-\$4, 0-\$8-\$5, 0\*

1, 1, \$3+\$3, 0-\$4+\$9-\$3, 0-\$5+\$8-\$3\*

1, 1, \$3+\$3, 0-\$4-\$9+\$3, 0-\$5+\$8-\$3\*

1, 1, \$3+\$3, 0-\$4-\$9+\$3, 0-\$5-\$8+\$3\*

1, 1, \$3+\$3, 0-\$4+\$9-\$3, 0-\$5-\$8+\$3\*%

## 4.11.5 Examples

### 4.11.5.1 Fixed Modifier Values

The following AM parameter defines an aperture macro named 'DONUTFIX' consisting of two concentric circles with fixed diameter sizes:

```
%AMDONUTFIX*1,1,0.100,0,0*1,0,0.080,0,0*%
```

Syntax	Comments
AMDONUTFIX	Aperture macro name is 'DONUTFIX'
1,1,0.100,0,0	1 – Circle 1 – Exposure on 0.100 – Diameter 0 – X coordinate of the center 0 – Y coordinate of the center
1,0,0.080,0,0	1 – Circle 0 – Exposure off 0.080 – Diameter 0 – X coordinate of the center 0 – Y coordinate of the center

The AD parameter using this aperture macro can look like the following:

```
%ADD33DONUTFIX*%
```

### 4.11.5.2 Variable Modifier Values

The following AM parameter defines an aperture macro named 'DONUTVAR' consisting of two concentric circles with variable diameter sizes:

```
%AMDONUTVAR*1,1,$1,$2,$3*1,0,$4,$2,$3*%
```

Syntax	Comments
AMDONUTVAR	Aperture macro name is 'DONUTVAR'



1,1,\$1,\$2,\$3	1 – Circle 1 – Exposure on \$1 – Diameter is provided by AD parameter \$2 – X coordinate of the center is provided by AD parameter \$3 – Y coordinate of the center is provided by AD parameter
1,0,\$4,\$2,\$3	1 – Circle 0 – Exposure off \$4 – Diameter is provided by AD parameter \$2 – X coordinate of the center is provided by AD parameter (same as in first circle) \$3 – Y coordinate of the center is provided by AD parameter (same as in first circle)

The AD parameter using this aperture macro can look like the following:

```
%ADD34DONUTVAR,0.100X0X0X0.080*%
```

In this case the variable modifiers get the following values: \$1 = 0.100, \$2 = 0, \$3 = 0, \$4 = 0.080.

#### 4.11.5.3 Definition of a New Variable

The following AM parameter defines an aperture macro named 'DONUTCAL' consisting of two concentric circles with the diameter of the second circle defined as a function of the diameter of the first:

```
%AMDONUTCAL*1,1,$1,$2,$3*$4=$1x0.75*1,0,$4,$2,$3*%
```

Syntax	Comments
AMDONUTCAL	Aperture macro name is 'DONUTCAL'
1,1,\$1,\$2,\$3	1 – Circle 1 – Exposure on \$1 – Diameter is provided by AD parameter \$2 – X coordinate of the center is provided by AD parameter \$3 – Y coordinate of the center is provided by AD parameter
\$4=\$1x0.75	Defines variable \$4 to be used as the diameter of the inner circle; the diameter of this circle is 0.75 times the diameter of the outer circle
1,0,\$4,\$2,\$3	1 – Circle 0 – Exposure off \$4 – Diameter is calculated using the previous definition of this variable \$2 – X coordinate of the center is provided by AD parameter (same as in first circle) \$3 – Y coordinate of the center is provided by AD parameter (same as in first circle)

The AD parameter using this aperture macro can look like the following:

```
%ADD35DONUTCAL,0.020X0X0*%
```

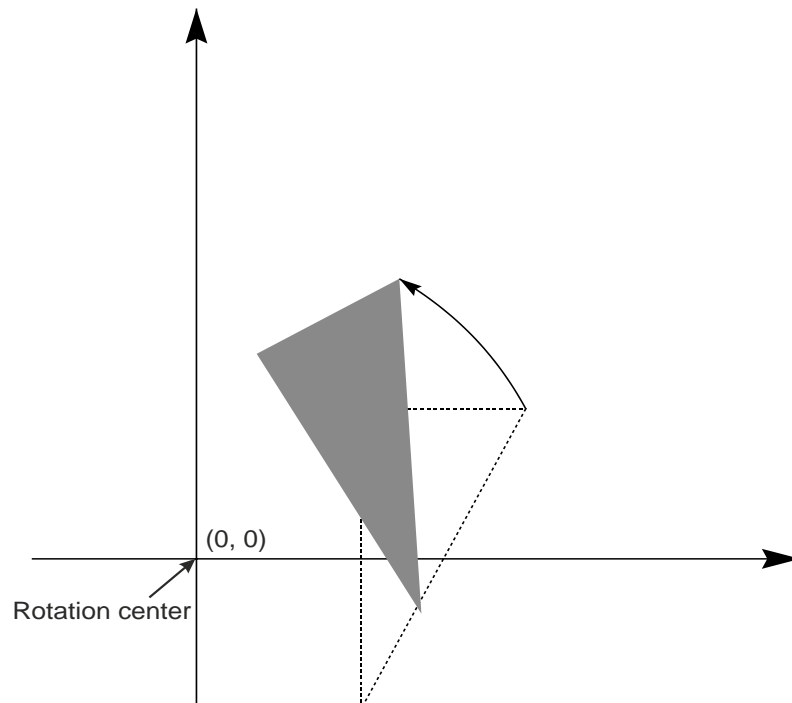
This defines a donut with outer circle diameter equal to 0.02 and inner circle diameter equal to 0.015.

#### 4.11.5.4 Rotation Modifier

The following AM parameter defines an aperture macro named 'TRIANGLE\_30'. The macro is a triangle rotated 30 degrees around the origin:

```
%AMTRIANGLE_30*4,1,3,1,-1,1,1,2,1,1,-1,30*%
```

Syntax	Comments
AMTRIANGLE_30	Aperture macro name is 'TRIANGLE_30'
4,1,3	4 – Outline 1 – Exposure on 3 – The outline has three subsequent points
1,-1	1 – X coordinate of the start point -1 – Y coordinate of the start point
1,1,2,1,1,-1	Coordinates (X, Y) of the subsequent points: (1,1), (2,1), (1,-1)
30	Rotation angle is 30 degrees counterclockwise



42. Rotated triangle

The AD parameter using this aperture macro can look like the following:

```
%ADD33AMTRIANGLE_30*%
```

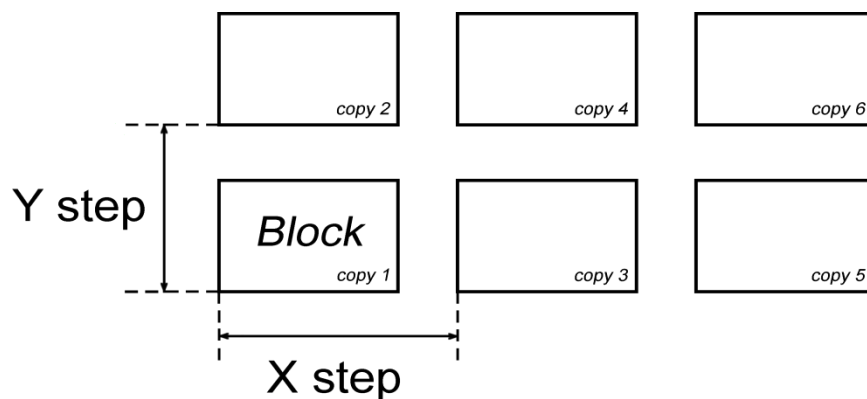
## 4.12 SR – Step and Repeat

The SR parameter sets the number of repeats and step distance along the X and Y axis in the graphics state. When the number of repeats in either X or Y is greater than 1 step & repeat mode is enabled; it is cleared when both numbers are set to 1.

In step & repeat mode the graphics objects generated in the stream of data are collected in a data set called a *block*. When another SR parameter or the end of the file is encountered the block is stepped and repeated (copied) in the image plane, first in the Y direction and then in the X direction. Each copy of the block contains identical graphics objects, put in another location. The number of the steps and the step distance is given by the graphics state.

### Example:

%SRX3Y2I5.0J4.0\*%



### 43. Step and Repeat


The block that is copied consists of *the graphics objects* generated by the Gerber source data in the SR section. It is not that the Gerber source that is copied and processed several times. The graphics objects are always identical, even if the graphics state is modified during the processing the source.

The SR parameter can be used multiple times in a file. An SR parameter remains effective till superseded by a new SR parameter.

The number of repeats and the steps can be different in X and Y.

The number of repeats along an axis can be 1, which is equivalent to no repeat. The corresponding step is then irrelevant. It is recommended to set the step to 0 in that case.

A step & repeat block can contain different polarities (LPD and LPC).

 **Warning:** It is recommended not to use overlapping steps containing clear and dark polarity. This is difficult to mentally visualize and probably not always correctly implemented by Gerber readers. Expect mistakes. A clear object in a block repeat clears *all* objects beneath it, including objects from a *different* copy of the same block. The graphics objects of each copy are identical but their effect on the image plane may be different. When repeats of blocks with both dark and clear polarity objects overlap, the step order affects the image; the correct step order must therefore be respected: step the complete block first in Y and then in X. (When all objects involved are dark or the repeats do not overlap, the step order has does not affect the image. The behavior is straightforward and safe to use.)

## 4.12.1 Data Block Format

The syntax for the SR parameter is:

**<SR parameter>: SR[X<Repeats>][Y<Repeats>][I<Step>][J<Step>]\***

Syntax	Comments
SR	SR for Step and Repeat
X<Repeats>	Defines the number of times the data is repeated along the X axis. If missing defaults to 1. If present must be a strictly positive integer.
Y<Repeats>	Defines the number of times the data is repeated along the Y axis. If missing defaults to 1. If present must be a strictly positive integer.
I<Step>	Defines the step distance along the X axis. It is mandatory if the number of repeats along X axis is >1. The step is a non-negative decimal number, expressed in the unit specified by %MO.
J<Step>	Defines the step distance along the Y axis. It is mandatory if the number of repeats along X axis is >1. The step is a non-negative decimal number, expressed in the unit specified by %MO.

## 4.12.2 Examples

Syntax	Comments
%SRX2Y3I2.0J3.0*%	Repeat the data 2 times along the X axis and 3 times along the Y axis. The step distance between X-axis repeats is 2.0 units. The step distance between Y-axis repeats is 3.0 units.
%SRX4Y1I5.0J0*%	Repeat the image 4 times along the X axis with the step distance of 5.0 units. The step distance in the J modifier is ignored because no repeats along the Y axis are specified.
%SRX1Y1I0J0*%	Repeat the data 1 time along the X and Y axes, i.e. data is not repeated.
%SRX1Y1*%	Equivalent to the above.

## 4.13 LP – Level Polarity

The LP parameter starts a new level and sets its polarity to either dark or clear. The level polarity applies to all data following the LP parameter until superseded by another LP parameter. This parameter can be used multiple times in a file. The current point is undefined after the LP parameter and must be set before use. See also 2.2.

An example can be found in 4.5.8.



**Warning:** *Level polarity* is not the same as *image polarity* (see the IP parameter for the image polarity description).

### 4.13.1 Data Block Format

The syntax for the LP parameter is:

**<LP parameter>: LP(C|D)\***

Syntax	Comments
LP	LP for Level Polarity
C D	Polarity: C – clear polarity D – dark polarity

### 4.13.2 Examples

Syntax	Comments
%LPD*%	Start a new level with dark polarity
%LPC*%	Start a new level with clear polarity

## 5 Attributes

### 5.1 Attributes Overview

Where an image simply needs to be rendered, attributes are not necessary, but where that image must be processed for PCB production, attributes are vital for the correct processing of the file and its elements. Among other applications, attributes enable design intent to accompany the images when transferring PCB design data from CAD to CAM. This is sometimes called rather grandly “adding intelligence to the image”.

Attributes do not change the image – they simply add information to the file and/or to its individual graphics elements by annotating them with metadata. Examples of this metadata might be the function of a flash – so a flash might be annotated with the command that it is an SMD pad or a via pad for example, or the whole file might be annotated with its overall function, making it clear that this is the top solder mask, or a drill map etc.

Given that attributes do not affect the image, a Gerber reader will still generate the correct image even if it ignores the attributes – which it will do when attributes are unnecessary, as in processes such as image rendering.

There are two basic types of attributes: *file attributes* associate metadata with the file as a whole, while *aperture attributes* associate metadata with individual graphics objects according to their aperture.

Each attribute consists of an *attribute name* and an optional *attribute value*. Attribute names must follow the naming syntax in section 3.2.3, while attribute values must follow the strings syntax in section 3.2.4.

The syntax of metadata parameters is similar to that of AM parameters. Each data block contains just one attribute parameter, each of which should be placed on a separate line for added clarity.

Semantically, there are *standard attributes* and *custom attributes*. Both follow the syntax and attachment rules laid out in this specification. Standard attributes and their semantics are part of this specification. Custom attributes, created with users’ proprietary metadata, extend the format further. Users are encouraged to contact Ucamco at [gerber@ucamco.com](mailto:gerber@ucamco.com) if they believe that their custom attributes could be more broadly used. Where these are included in the specification as standard attributes, authors will be properly acknowledged.

In variance with the rules of section 3.2.3, standard attribute names begin with a decimal point, while custom attribute names do not.

Standard attributes are intended for the PCB CAD/CAM workflow. Their use is neither mandatory nor is it “all or nothing”. It is possible to use just one attribute, all of them or none at all. That said, their broad use is strongly recommended, as they provide vital information in a standard way – information that must otherwise be gathered from various documents, unwritten rules, conversations or guesswork, with all the risks of error and delay that this entails. Where users cannot for some reason provide all the attributes, or are unsure of their use, they are encouraged to provide those attributes with which they are comfortable; after all, partial information is better than no information at all. In professional PCB production the bare minimum is to set the file function attribute.

### 5.2 File attributes

File attributes provide information about entire files.

The semantics of a file attribute specifies where it must be defined, typically in the header of the file. A file attribute can only be defined once. It cannot be redefined.

An attribute consists of an *attribute name* and an optional *attribute value*.

File attributes are set using the uppercase **TF** parameter using the following syntax

**<TF parameter>: TF<AttributeName>{,<AttributeValue>}**

The attribute name must follow the naming syntax in section 3.2.3, with the exception of the dot with which standard attribute names commence. The attribute value must follow the strings syntax in section 3.2.4.

## 5.2.1 Standard File Attributes

The Gerber file format specifies a number of standard file attributes. These are listed in the table below and subsequently explained in detail. All Standard aperture names and values are case-sensitive.

Name	Usage
.FileFunction	Identifies the file's function in the PCB.
.Part	Identifies the part the file represents, e.g. a single PCB
.MD5	Sets the MD5 file signature or checksum.
.GerberVersion	Identifies the version of the Gerber file format used

*Standard file attributes*

### 5.2.1.1 .FileFunction

The value of the .FileFunction file attribute identifies the function of the file in the PCB. The attribute must be defined in the file header.

Of all the attributes this is the most important.

The attribute value consists of a number of substrings separated by a “,”:

- Type. Such as copper, solder mask etc. See list below
- Position. Specifies where the file appears in the PCB. Its structure depends on the type
  - o Copper layer number: **L1, L2, L3**.... where L1 is the top copper layer
  - o Attachment: **Top** or **Bot**. For extra layers such as solder mask
  - o Span: **i,j**. These two integers are the end copper layers of a drill/route file.
- Optional index. This can be used in instances where for example there are two solder masks on the same side. The index counts from the PCB surface outwards.

The file functions are designed to support all practices and file types in current use. If a type is missing please contact us at [gerber@ucamco.com](mailto:gerber@ucamco.com).

These indications should not be taken as a suggestion that all types listed should always be included in PCB data sets. Users should include just the types that are required: no more, no less.



**Note:** Some readers, thinking that Gerber cannot define drill files, might be surprised to see these represented as Gerber image files. In fact, Gerber files convey CAD/CAM drill information perfectly well, because this is truly image information, defining where material must be removed. Of course a Gerber file cannot be sent to a drill machine, but this is not the issue

here. No manufacturer uses his client's incoming design files directly on his equipment. The design files are always read in a CAM system, and it is the CAM system that will output drill files in an NC format, including feeds and speeds. As the copper, mask, drill and route files are all image files to be read into the CAM system, it is best to use the same format for them all, thereby ensuring optimal accuracy, registration and compatibility. Mixing formats needlessly is asking for needless problems.

This specification contrasts does not differentiate between drilling and routing because these two admittedly distinct manufacturing processes are identical from the point of view of their image descriptions: the image simply represents where material is removed. Note that the choice between drilling and routing is not always obvious: a slot can be nibbled (drilled), a larger round hole can be routed.

We follow the customary practice of putting blind and buried drills in separate files. For clarity and consistency with this practice we use separate files for NPTH and PTH tools – which is also easier for systems that do not handle attributes.

.FileFunction value	Remark
Copper, L<p>[, <label>]	A conductor or copper layer. Lp – where p is an integer – indicates layer position where L1 is the top layer. The optional label can take the values Plane, Signal or Mixed; these labels are not exactly defined, the file creator adding the label as he finds appropriate.
Soldermask, (Top Bot) [, <i>]	The image represents the solder mask <i>openings</i>
Legend, (Top Bot) [, <i>]	
Goldmask, (Top Bot)	
Silvermask, (Top Bot)	
Tinmask, (Top Bot)	
Carbonmask, (Top Bot)	
Peel-off, (Top Bot)	
Viatenting, (Top Bot)	Indicates via's that must be tented
Viafill	Indicates via's that must be filled
Glue, (Top Bot)	
Keep-out, (Top Bot)	
Paste, (Top Bot)	
Heatsink, (Top Bot)	
Pads, (Top Bot)	
Scoring, (Top Bot)	



NPTH[,<label>]	Non- plated through hole drill/route. The optional label can take the values Drill, Route, Mixed; these labels are not exactly defined with the file creator adding the label as as he finds appropriate.
PTH[,<label>]	Plated through hole drill/route. See NPTH for the optional label.
Plated,i,j[,<label>]	Plated drill/rout from layer i to layer j, e.g. blind and buried vias. Not to be used for PTH as these have a separate function. See NPTH for the optional label.
Nonplated,i,j[,<label>]	Nonplated drill/rout from layer i to layer j, e.g. backdrill. Not to be used for NPTH as these have a separate function. See NPTH for the optional label.
Profile,(P NP)	Profile (outline)
Drillmap	A drill map drawing
Assembly	An assembly drawing
Mechanical	A mechanical drawing
Drawing[,<string>]	Any other drawing. The optional string informally describes the drawing subjects
Other,<string>	None of the above. The mandatory string informally indicates the file function.

*.FileFunction file attribute values*



**Example:**

Below is an example of file function attribute commands in a set of files describing a simple 4-layer PCB. Only one attribute in each file of course!

```
%TF.FileFunction,Legend,Top*%
%TF.FileFunction,Soldermask,Top*%
%TF.FileFunction,Copper,L1*%
%TF.FileFunction,Copper,L2*,Plane%
%TF.FileFunction,Copper,L3*,Plane%
%TF.FileFunction,Copper,L4*%
%TF.FileFunction,Soldermask,Bot*%
%TF.FileFunction,NPTH*,Drill%
%TF.FileFunction,NPTH*,Route%
%TF.FileFunction,PTH*%
%TF.FileFunction,Profile,NP*%
```

```
%TF.FileFunction,Drillmap*%
```

```
%TF.FileFunction,Drawing,Stackup*%
```

```
%TF.FileFunction,Drawing*%
```



**Note:** The top copper layer is expressed by L1, not Top. “Copper,Top” does not exist.



**Note:** All layers must be expressed in positive image polarity. Negative image polarity is deprecated.

### 5.2.1.2 .Part

The value of the .Part file attribute identifies which part is described. The attribute must be defined in the file header.

.Part value	Remark
Single	Single PCB
CustomerPanel	A.k.a. array or shipping panel
ProductionPanel	A.k.a. working panel, fabrication panel
Coupon	A coupon
Other,<string>	None of the above. The mandatory string informally indicates the part.

*.Part file attribute values*



**Example:**

```
%TF.Part,CustomerPanel*%
```

### 5.2.1.3 .MD5

The .MD5 file attribute, expressed in hexadecimal digits, sets a file signature (or checksum) that is calculated using the well-known MD5 algorithm.

The signature uniquely identifies the file and provides a high degree of security by allowing checks to be made for accidental modifications during storage or transmission. There is an astronomical degree of certainty that files with the same signature will be identical.

The signature is calculated over the file from the beginning till the last character before this attribute. The CR and LF are excluded from signature calculation as they do not affect the interpretation of the file but may be altered when moving platforms. By excluding them, the signature maintains portability without sacrificing security.

The signature must be put at the end of the file, just before the closing M02. Thus the file can be processed in a single pass.



#### Example:

```
%TF.MD5,e4d909c290d0fb1ca068ffaddf22cbd0*%  
M02*
```

### 5.2.1.4 .GerberVersion

The .GerberVersion file attribute identifies the version of the Gerber file format used. It has one mandatory value: the revision of the format specification to which the file is intended to conform.

When used, it must be the first block in the file so that the file reader can use this information when processing the file.



#### Example:

```
%TF.GerberVersion,J1*%
```

## 5.3 Aperture Attributes

### 5.3.1 Aperture Attributes Overview

Each *aperture attribute* is associated with an aperture and hence with all graphics objects generated with that aperture. The term *aperture attribute* is actually shorthand for *graphics object attribute defined by aperture*.

Objects are associated with apertures using the *current attribute dictionary*, which contains all current aperture attributes. It is defined after each command in the file, as follows:

- Initially the current attribute dictionary is empty.
- Aperture attributes are added with the TA parameters
- Aperture attributes are deleted from it with the TD parameter.

When an AD parameter defines an aperture, all aperture attributes in the current dictionary are associated with that aperture.



**Note:** When aperture attributes are set for an aperture, all graphics objects created with that aperture will inherit its attributes. This is why it is important that objects with different attributes, even if they are the same shape and size, must be created with different apertures. There may be a temptation to 'optimize' the file by merging apertures of the same shape and size, even if they have different functions. Resist that temptation. Use a separate aperture whenever the attributes should be different, otherwise the file is very hard to handle in CAM.

### 5.3.2 Aperture Attributes Commands

The parameters listed in the table below affect Aperture Attributes. All parameters are upper case.

Parameter	Name	Description
TA	Set aperture attribute	Enters an aperture attribute in the current dictionary.
TD	Delete attribute	Deletes a previously defined aperture attribute from the current dictionary. The deleted attribute will not be set on data following this parameter.
DR	Set region D-code	Associates regions with a D-code to associate aperture attributes with regions

*Aperture attribute parameters*



**Warning:** Aperture Attribute Parameter cannot be used in region mode.

#### 5.3.2.1 Set aperture attributes (TA)

The parameter TA enters an aperture attribute into the current dictionary. The syntax rules are the same as for the TF parameter:

**<TA parameter>: TA<AttributeName>{,<AttributeValue>}**

The attribute name must follow the naming syntax in section 3.2.3 (with the exception of the dot with which standard attribute names must start.) This name must be unique and must not already be in use for a file attribute. The value of an aperture attribute can be overruled by using the TA parameter with the same name, but a new value.

The attribute value follows the syntax of strings in section 3.2.4.



**Example: defining an aperture attribute**

```
%TA.AperFunction,ComponentPad*%
%TAMyApertureAttributeWithValue,value*%
%TAMyApertureAttributeWithoutValue*%
```



**Example: overruling the value of an aperture attribute**

```
%TA.AperFunction,ComponentPad*%
%TA.AperFunction,ViaPad*%
```

### 5.3.2.2 Delete attribute (TD)

The parameter TD deletes an attribute from the current dictionary, but the attribute remains attached to apertures and objects to which it was attached before it was deleted.

**<TD parameter>: TD<AttributeName>**

**<AttributeName>:** The name of the attribute to delete.



**Warning:** TD cannot be used on file attributes.

### 5.3.2.3 Select Region aperture (DR)

Draws, arcs and flashes are graphics objects created with apertures, so aperture attributes can be associated with them. Regions, on the other hand, are not created using apertures, so aperture attributes cannot be associated with them.

This asymmetry is not desirable. For example, it is possible to use a number of dedicated apertures with the attribute 'Conductor' to identify objects whose function is purely conductive. This works well for draws and arcs but not for regions that are also often purely conductive. With the parameters that have thus far been defined this is not possible. The parameter DR (Select Region Aperture) makes this possible by associating regions with an aperture number.

The following example shows how DR is used: %DR33\*% sets the region aperture to 33; all regions created with region aperture 33 take the attributes associated with aperture 33. Attribute-wise these regions behave as if they were created using aperture 33. The region aperture remains set to 33 until it is overruled by another %DRn\*% or cleared by %DR\*%.

When the image does not contain an aperture with the attributes needed for a certain region, one should define a *virtual aperture*.


### 5.3.2.4 Virtual aperture (AV)

A region can be associated with the attributes of an aperture by using the DR parameter. However there may not be a real aperture, (one used for flashing or stroking) with the desired

attributes for the region or painted section. This problem can be solved by defining an aperture that is not used for imaging and whose sole purpose is to carry attributes. However, this is somewhat artificial and misleading, as apertures are intended for imaging.

A cleaner solution is to use the AV (Virtual Aperture) parameter to overcome this problem. The command %AVn\* (e.g. %AV56\*) defines a virtual aperture with aperture number n, and all apertures in the current aperture dictionary are attached to it. A region can then be attached to this virtual aperture using %DRn\*, in the same way as it would be attached to a normal aperture.

A virtual aperture can only be selected using the DR parameter. It cannot be selected using %Dnn\*% (nn≥10.)

 **Warning:** It is not possible to define a virtual aperture with the same aperture number as an aperture defined by AD.

### 5.3.3 Standard Aperture Attributes

#### 5.3.3.1 .AperFunction

This aperture attribute defines the function or purpose of an aperture, or rather the graphics objects created with that aperture. Of course, functions can only be defined in a file with an applicable function. For example, an SMD pad can only be defined on an outer copper layer.

Users are encouraged to identify the function of all apertures. If this is not possible for some apertures, it is highly recommended that the aperture function is identified wherever possible – partial information is always better than no information.

The values this attribute can have are defined in the following table.

.AperFunction value(s)	Remark	Applicable File Function
ComponentDrill	A hole for component pins	NPTH, PTH, Plated, Nonplated
ViaDrill, (Filled NotFilled)	A via hole. Not used for components. Filled for filled holes, NotFilled for free holes.	
MechanicalDrill	A hole with mechanical function (registration, screw, etc.)	
BackDrill	Removes plating in a hole over a depth by drilling with a slightly larger diameter.	
OtherDrill, <string>	A hole, but none of the above. The mandatory string informally describes the type.	
ComponentPad	A component pad.	Copper
SMDPad, (SMDDef CuDef)	An SMD pad. SMDDef for solder mask defined, CuDef for copper defined.	

BGAPad, (SMDef CuDef)	A BGA pad. SMDef for solder mask defined, CuDef for copper defined.
HeatsinkPad	Heat sink pad (typically for SMDs)
TestPad	A test pad.
ConnectorPad	A connector pad.
ViaPad	A via pad. It provides a ring to attach the plating in the barrel but has no other function.
FiducialPad, (Global Local)	A fiducial pad. Local refers to a component fiducial, Global refers to an entire image or PCB fiducial.
ThermalReliefPad	A thermal relief pad, connected to the surrounding copper while restricting heat flow.
WasherPad	A pad around a tooling hole. Uses include grounding the copper using a bolt.
AntiPad	A pad with clearing polarity (LPC) creating a clearance in a plane. It makes room for a drill pass without connecting to the plane.
OtherPad, <string>	A pad not specified above. Must be accompanied with a description.
Conductor, (NotC  (ImpC, <string>))	Copper whose function is to connect pads and/or to shield. ImpC means impedance controlled, NotC means not impedance controlled. The string identifies the impedance control type, especially important if there is more than one type in the PCB.
Nonconductor	Copper that does not serve as a conductor; typically text and graphical elements without electrical function.
CopperBalancing	Copper pattern added to balance copper coverage for the plating process.
Border	Border around a panel.
Other, <string>	Indicates another function. Must be accompanied by a description.

Profile	Used to define PCB outline or profile. Profiles can be present in all PCB layers. This does not mean we recommend this, but we observe this happens and therefore enable its identification.	All
Nonmaterial	For objects that do not represent the material in the file, or objects that are not present in the PCB. For example the copper pattern is sometimes surrounded by a border containing information like a technical drawing rather than a data file. This border is not part of the copper pattern, and does not represent copper. It is strongly recommended to put this information in a separate document rather than combining it with the true copper pattern. But if you insist on mixing a drawing with a data file use the nonmaterial attribute to identify it.	
Material	Identifies the proper part of the data file, complementing the nonmaterial above. For copper and drill layers this function is split into more detailed functions.	All except drills and copper

*.AperFunction aperture attribute values*

### 5.3.3.2 Drill Tool Parameters

The aperture attributes defined in this section may only be associated with attributes in drill/rout files. They define properties for the drill holes.

#### 5.3.3.2.1 .DrillTolerance

This attribute defines the plus and minus tolerance of a drill hole. Both values are positive decimals expressed in the MO units.

*.DrillTolerance,<plus tolerance>,<minus tolerance>*



### 5.3.4 Examples



#### Example 1, simple aperture attribute:

```
%ADD13R,200X200*%      G04 this aperture has no attribute*
D13*
%TAIAmATA*%
X0Y0D03*                G04 this flash has no attribute*
%ADD11R,200X200*%      G04 this aperture now has attribute IAmATA*
%TDIAmATA*%
%ADD12C,5*%            G04 this aperture does not have attribute IAmATA*
D11*
X100Y0D03*             G04 this flash has attribute IAmATA*
X150Y50D02*
D12*
X100Y150D01*           G04 this draw has no attribute*
```



#### Example 3, aperture attribute, definition & changing value:

```
%TA.AperFunction,SMDPad*% G04 Enters attribute .AperFunction in the
                           current dictionary with value "SMDPad" to
                           identify SMD pads*

%ADD11...*%              G04 Ape. 11 gets the .Function,SMDPad
                           attribute*

%TA.AperFunction,Cond*%   G04 Changes the value of .AperFunction
                           attribute to define conductors*

%ADD20...*%              G04 Ap. 20 gets the .Function,Conductor att.*

%TACustAttr,val*%        G04 Enters attribute "CustAttr" in the current
                           dictionary and sets its value to "val".*

%AV21*%                  G04 Virtual aperture 21 is a conductor with
                           attribute CustAttr = val.*

%TD.AperFunction*%       G04 Deletes the .AperFunction attribute from
                           the current dictionary.*

%ADD22...*%              G04 Ap. 22 has no associated .AperFunction
                           attribute, but has attribute CustAttr = val.*

%TDCustAttr *%           G04 Deletes the CustAttr attribute from the
                           current dictionary.*

%ADD23...*%              G04 Ap. 21 has no aperture attribute.*
...
D11*
X1000Y1000D03*           G04 Flash an SMD pad*
D20*
X2000Y1500D01*           G04 Draw a conductor*
%DR20*%                  G04 Associate regions with ap. 20*
```

D23*	G04 Use aperture 23 for graphics objects. The region aperture is not changed.*
X2000Y3000D03*	G04 A flash without apertures*
G36*	G04 Start a conductive region. (Still associated with aperture 20)*
...	
G37*	
X1000Y1000D03*	G04 Flash an SMD pad*
%DR21*%	G04 Associate regions with virtual ap. 21*
G36*	G04 Start a conductive region with CustAttr = val.*
...	
G37*	
%DR*%	G04 Regions are no longer linked with an aperture.*
%TDSomeAttr*%	G04 Delete attribute SomeAttr*
G36*	G04 Start a region, without attributes.*
...	
G37*	

## 6 Most Common Errors & Bad Practice

### 6.1 Most Common Errors

Poor implementation of the Gerber format can give rise to invalid Gerber files or – worse – valid Gerber files that do not represent the intended image. The table below lists the most common errors.

Symptom	Cause and Correct Usage
Full circles unexpectedly appear or disappear.	The file contains arcs but no G74 or G75. This is invalid. <b>A G74 or G75 is mandatory if arcs are used.</b> See 4.4.6.
Rotating aperture macros using primitive 21 gives unexpected results.	Some CAD systems incorrectly assume that primitive 21 rotates around its center. It does not – it rotates around the origin. See 4.11.2.4.
Unexpected image after an aperture change or a D03.	Coordinates have been used without an explicit D01/D02/D03 operation code. This practice is deprecated because it leads to confusion about which operation code to use. <b>The D01/D02/D03 operation code should always be included with the coordinate data.</b> See 7.1.
Objects unexpectedly appear or disappear under holes in standard apertures.	Some CAD systems incorrectly assume the hole in an aperture <i>clears</i> (erases) the underlying objects. This is wrong, the hole is transparent and has no effect on the underlying image. See <b>Error! Reference source not found.</b>
Objects unexpectedly appear or disappear under holes in macro apertures.	Some CAD systems incorrectly assume that exposure off in a macro aperture <i>clears</i> (erases) the underlying objects under the flash. This is wrong, exposure off creates a hole in the aperture and that hole has no effect on the image. See 4.11.1.
Openings in areas disappear, typically with clearances in planes	Overlapping segments have been used to construct cut-ins. This is an error. Coincident segments should be used instead. Note that cut-ins are not intended for complex planes; use a layer in LPC to make clearances in a plane. See 4.5.

Polygons are smaller than expected.	Some CAD systems incorrectly assume the parameter of a Regular Polygon specifies the inside diameter. It does not: it specifies the outside diameter.  See 4.10.2.4.
A single Gerber file contains more than one image, separated by M00, M01 or M02	This is invalid. A Gerber file can contain only one image.  One file, one image. One image, one file.
Contour fill defined in a Level Polarity Clear (%LPC) unintentionally erases a previously defined object at that location	As for any other object in an LPC section, Contour fill does indeed clear (erase) any underlying objects. (here, clear does not mean transparent). Objects can be added to the location after the clear.  See 2.2.
The MI parameter is used to mirror a macro definition but the result is not as expected.	With the MI parameter mirroring is not applied to aperture definitions. Do not use this confusing and deprecated parameter. Apply the transformation directly in the aperture definitions and object coordinates.  See 7.4.5.

*Reported Common Errors*

## 6.2 Most Common Bad Practices

Some Gerber files are syntactically correct but are needlessly cumbersome or error-prone. The table below summarizes common poor practices and gives the corresponding good practice.

Bad Practice	Problems	Good Practice
Low resolution (numerical precision)	Poor registration of objects between PCB layers; loss of accuracy; possible self-intersecting contours; invalidated arcs; zero-arcs. These can give rise to unexpected results downstream. Note that putting the file through software processing unavoidably adds further numerical rounding, further aggravating the problem.	Use 6 decimal places in imperial and 5 decimal places in metric.  Do not sacrifice precision to save a few bytes
Multi quadrant mode and rounding errors	In G75 mode and due to rounding, a small arc suddenly becomes a full circle as the start and end points end up on top of one another.	<b>Use G74</b> single quadrant mode or take very great care when rounding on small arcs
Imprecisely positioned arc center points	An imprecisely positioned center makes the arc ambiguous and open to interpretation. This can lead to unexpected results.  See 4.4.2	<b>Always position arc center points precisely</b>

Painted or stroked pads	Painted pads produce the correct image but are very awkward and time consuming for CAM software in terms of DRC checks, electrical test and so on. Stroking was needed for vector photoplotters in the 1960s and 1970s, but these devices are as outdated as the mechanical typewriter.	<b>Always use flashed pads.</b> Define pads, including SMD pads, with the AD and AM parameters
Painted or stroked areas	As above, painted areas produce the correct image, but the files are needlessly large and the data is very confusing for CAM software.	<b>Always use contours (G36/G37) to define areas</b>
The use of cut-ins to construct clearances in planes (anti-pads)	Using cut-ins for such complex constructions can give rise to rounding errors. Furthermore, CAM systems cannot work with such constructions so must first resolve the cut-ins and recover the anti-pads.	<b>Construct planes and anti-pads using an LPD layer for the plane and an LPC layer for the holes (anti-pads)</b>
Standard Gerber or RS-274-D	Standard Gerber is deprecated. It was designed for a workflow that is as obsolete as the mechanical typewriter. It requires manual labor to process. It is not suitable for today's image exchange. Do not use it.	<b>Always use Extended Gerber.</b>

*Common poor/good practices*

## 7 Deprecated Format Elements

### 7.1 Coordinate Data Blocks without Operation Code

Previous versions of the specification allowed coordinate data *without explicit operation code* in a few situations. In the absence of an explicit operation code, a deprecated *operation mode* operates on the coordinates.

A D01 sets the operation mode to interpolate. It remains in interpolate mode till any other D code is encountered. (In older terminology, D01 turns the light on, and D02 turns it off.) This allows omitting an explicit D01 after the first coordinate data block only in sequences of D01 data blocks.



**Example:**

```
D10*  
X700Y1000D01*
```

### 7.2 X1200Y1000\*

```
X1200Y1300*  
D11*  
X1700Y2000D01*  
X2200Y2000*  
X2200Y2300*
```

This saves a few bytes. However, coordinate data blocks without explicit operation code are not intuitive and lead to errors. This risk far outweighs the meager benefit of saving a few bytes. Coordinates with operation code are therefore deprecated.

The operation mode is *only* defined after a D01 or D02. In other words the operation mode after a D03 or an aperture selection (Dnn with  $nn \geq 10$ ) is undefined. Therefore a file containing coordinates without operation code after a D03 or an aperture selection (Dnn with  $nn \geq 10$ ) is invalid.

Note that consequently a Gerber writer can neither assume that D03 behaves modally nor not modally.



**Warning:** Avoid writing coordinates without operation code like the plague. The risk of using them lies solely with the writer of the file.

### 7.3 Open Contours

Previous versions of the specification allowed leaving contours open in a region definition.

Before the region is created all open contours are closed by connecting the last point to the first with a straight draw. Closing the contour does *not* move the current point; the current remains at the last coordinate in the file.

Open contours can be misunderstood and are therefore deprecated. Contours must be explicitly closed.

Moves (D02) with zero length in open contours are *not* allowed as they are confusing.

Contours that become self-intersecting after the automatic closure are of course not allowed.

## 7.4 Deprecated Commands

The next table lists deprecated codes.

Code	Function	Comments
G54	Select aperture	This historic code optionally precedes an aperture selection D-code. It has no effect. It is superfluous and deprecated.
G55	Prepare for flash	This historic code optionally precedes D03 code. It has no effect. It is superfluous and deprecated.
G70	Set the 'Unit' to inch	These historic codes perform a function handled by the MO parameter. See 4.9. They are superfluous and deprecated.
G71	Set the 'Unit' to mm	
G90	Set the 'Coordinate format' to 'Absolute notation'	These historic codes perform a function handled by the FS parameter. See 4.8. They are superfluous and deprecated.
G91	Set the 'Coordinate format' to 'Incremental notation'	
M00	Program stop	This historic code has the same effect as M02. It is superfluous and deprecated.
M01	Optional stop	This historic code has no effect. It is superfluous and deprecated.

### *Deprecated codes*

Gerber writers can no longer use deprecated codes.

Gerber readers may implement them to support legacy applications and files. The codes G54, G70 and G71 are still found from time to time. The other codes are very rarely, if ever, found.

The table below lists the deprecated parameters. They are explained later in this chapter.

Parameter	Function	Description	Comments
AS	Axis Select	Sets the 'Axes correspondence' graphics state variable	These parameters can only be used once, at the beginning of the file.
IN	Image Name	Sets the name of the file image	
IP	Image Polarity	Sets the 'Image polarity' graphics state variable	
IR	Image Rotation	Sets 'Image rotation' graphics state variable	
MI	Mirror Image	Sets 'Image mirroring' graphics state variable	
OF	Offset	Sets 'Image offset' graphics state variable	
SF	Scale Factor	Sets 'Scale factor' graphics state variable	
LN	Layer name	LN has no effect on the image. It is no more than a comment	Can be used many times in the file.

*Deprecated parameters*

The order of execution of these parameters is always MI, SF, OF, IR and AS, independent of their order of appearance in the file.

Gerber writers (creators of Gerber files) must not use deprecated parameters.

Gerber readers may support deprecated parameters. There are a few legacy files and applications generating these deprecated parameters. If a deprecated parameters is used it is nearly always, if not always, to confirm the default value; in other words it has no effect. It is probably a waste of time to implement these parameters.



The table below contains deprecated graphics state variables.

Graphics state variable	Values range	Fixed	Value at the beginning of a file
Axes correspondence	AXBY, AYBX See AS parameter	Yes	AXBY
Image mirroring	See MI parameter	Yes	A0B0
Image offset	See OF parameter	Yes	A0B0
Image polarity	POS, NEG See IP parameter	Fixed	Positive
Image rotation	0°, 90°, 180°, 270° See IR parameter	Yes	0°
Scale factor	See SF parameter	Yes	A1B1

*Deprecated graphics state variables*

### 7.4.1 AS – Axis Select

The AS parameter is deprecated.

LN sets the correspondence between the X, Y data axes and the A, B output device axes. It does not affect the image in computer to computer data exchange. It only has an effect how the image is positioned on an output device.

This parameter affects the entire image. It can only be used once, at the beginning of the file.

#### 7.4.1.1 Data Block Format

The syntax for the AS parameter is:

**<AS parameter>: AS(AXBY|AYBX)\***

Syntax	Comments
AS	AS for Axis Select
AXBY	Assign output device axis A to data axis X, output device axis B to data axis Y
AYBX	Assign output device axis A to data axis Y, output device axis B to data axis X

#### 7.4.1.2 Examples

Syntax	Comments
%ASAXBY*%	Assign output device axis A to data axis X and output device axis B to data axis Y
%ASAYBX*%	Assign output device axis A to data axis Y and output device axis B to data axis X

## 7.4.2 IN - Image Name

The IN parameter is deprecated. Use attributes to provide meta information about the file, see chapter 5.

LN identifies the entire image contained in the Gerber file. The name must comply with the syntax rules for a string (confusingly not for a name) as described in section 3.2.4. This parameter can only be used once, at the beginning of the file.

IN has *no* effect on the image. A reader can ignore this parameter.

The informal information provide by IN can also be put a G04 comment. IN is no longer useful to man or machine and has been deprecated.

### 7.4.2.1 Data Block Format

The syntax for the IN parameter is:

**<IN parameter>: IN<Name>\***

Syntax	Comments
IN	IN for Image Name
<Name>	Image name

### 7.4.2.2 Examples

Syntax	Comments
%INPANEL_1*%	Image name is 'PANEL_1'

### **7.4.3 IP – Image Polarity**

The IP parameter is deprecated.

IP sets positive or negative polarity for the entire image. It can only be used once, at the beginning of the file.

#### **7.4.3.1 Positive image polarity**

Under *positive* image polarity, the image is generated as specified elsewhere in this document. (In other words, the image generation has been assuming positive image polarity.)

#### **7.4.3.2 Negative image polarity**

Under negative image polarity, image generation is different. Its purpose is to create a negative image, clear areas in a dark background. The entire image plane in the background is initially dark instead of clear. The effect of dark and clear polarity is toggled. The entire image is simply reversed, dark becomes white and vice versa.

Note that the first graphics object generated must have dark polarity, and therefore clears the dark background. It is not allowed to have a clear polarity first graphics object. Consequently, the first graphics object always clears the background.

#### **7.4.3.3 IP is deprecated**

Plane layers in PCB's are typically solid copper areas with holes in it, called anti-pads and thermals. In the obsolete Standard Gerber format it made sense to transfer such layers as a negative image. In Extended Gerber there is no need to transfer layers in negative. Plane layers are better described positive, using regions (G36/G37) with clear polarity levels (%LPC) to make holes. However, layers are still sometimes transferred in negative, causing some confusion. %IPNEG was a convenient way to identify such files. As the whole method is obsolete and confusing, the IP parameter is now deprecated.

#### 7.4.3.4 Data Block Format

The syntax for the IP parameter is:

**<IP parameter>: IP(POS|NEG)\***

Syntax	Comments
IP	IP for Image Polarity
POS	Image has positive polarity
NEG	Image has negative polarity

#### 7.4.3.5 Examples

Syntax	Comments
%IPPOS*%	Image has positive polarity
%IPNEG*%	Image has negative polarity

### 7.4.4 IR – Image Rotation

The IR parameter is deprecated.

IR is used to rotate the entire image counterclockwise in increments of 90° around the image (0, 0) point. All image objects are rotated.

The IR parameter affects the entire image. It must be used only once, at the beginning of the file.

#### 7.4.4.1 Data Block Format

The syntax for the IR parameter is:

**<IR parameter>: IR(0|90|180|270)\***

Syntax	Comments
IR	IR for Image Rotation
0	Image rotation is 0° counterclockwise (no rotation)
90	Image rotation is 90° counterclockwise
180	Image rotation is 180° counterclockwise
270	Image rotation is 270° counterclockwise

#### 7.4.4.2 Examples

Syntax	Comments
--------	----------

%IR0*%	No rotation
%IR90*%	Image rotation is 90° counterclockwise
%IR270*%	Image rotation is 270° counterclockwise

## 7.4.5 LN – Level Name

The LN parameter is deprecated.

LN identifies the current level. The name must comply with the syntax rules for a string (confusingly not for a name) as described in section 3.2.4. This parameter can be used multiple times in a file.

LN has *no* effect on the image. A reader can ignore this parameter. LN was intended to make the file easier to read for humans. However, this can also be done with a plain G04 comment.

### 7.4.5.1 Data Block Format

The syntax for the LN parameter is:

**<LN parameter>: LN<Name>\***

Syntax	Comments
LN	LN for Level Name
<Name>	Level name

### 7.4.5.2 Examples


Syntax	Comments
%LNVia_anti-pads*%	The name 'Via_anti-pads' is assigned to the current level.

## 7.4.6 MI – Mirror Image

The MI parameter is deprecated.

MI used to turn axis mirroring on or off. When on, all A- and/or B-axis data is mirrored. **MI does not mirror special apertures!**

This parameter affects the entire image. It can only be used once, at the beginning of the file.

 **Warning:** It is strongly recommended *not* to use the MI parameter. Avoid it like the plague. The exception for special apertures is confusing and leads to mistakes. If an image must be mirrored, write out the mirrored coordinates and apertures.

### 7.4.6.1 Data Block Format

The syntax for the MI parameter is:

**<MI parameter>: MI[A(0|1)][B(0|1)]\***

Syntax	Comments
MI	MI for Mirror image
A(0 1)	Controls mirroring of the A-axis data: A0 – disables mirroring A1 – enables mirroring (the image will be flipped over the B-axis) If the A part is missing then mirroring is disabled for the A-axis data
B(0 1)	Controls mirroring of the B-axis data: B0 – disables mirroring B1 – enables mirroring (the image will be flipped over the A-axis) If the B part is missing then mirroring is disabled for the B-axis data

### 7.4.6.2 Examples

Syntax	Comments
%MIA0B0*%	No mirroring of A- or B-axis data
%MIA0B1*%	No mirroring of A-axis data. Mirror B-axis data
%MIB1*%	No mirroring of A-axis data. Mirror B-axis data



### 7.4.7 OF - Offset

The OF parameter is deprecated.

OF moves the final image up to plus or minus 99999.99999 units from the imaging device (0,0) point. The image can be moved along the imaging device A or B axis, or both. The offset values used by OF parameter are absolute. If the A or B part is missing, the corresponding offset is 0. The offset values are expressed in units specified by MO parameter.

This parameter affects the entire image. It can only be used once, at the beginning of the file.

#### 7.4.7.1 Data Block Format

The syntax for the OF parameter is:

**<OF parameter>: OF[A<Offset>][B<Offset>]\***

Syntax	Comments
OF	OF for Offset
A<Offset>	Defines the offset along the output device A axis
B<Offset>	Defines the offset along the output device B axis

The **<Offset>** value is a decimal number n preceded by the optional sign ('+' or '-') with the following limitation:

$$0 \leq n \leq 99999.99999$$

The decimal part of n consists of not more than 5 digits.

#### 7.4.7.2 Examples

Syntax	Comments
%OFA0B0*%	No offset
%OFA1.0B-1.5*%	Defines the offset: 1 unit along the A axis, -1.5 units along the B axis
%OFB5.0*%	Defines the offset: 0 units (i.e. no offset) along the A axis, 5 units along the B axis

## 7.4.8 SF – Scale Factor

The SF parameter is deprecated.

SF sets a scale factor for the output device A- and/or B-axis data. The factor values must be between 0.0001 and 999.99999. The scale factor can be different for A and B axes. If no scale factor is set for an axis the default value '1' is used for that axis.

All the coordinate data are multiplied by the specified factor value for the corresponding axis. Note that apertures are *not* scaled.

This parameter affects the entire image. It can only be used once, at the beginning of the file.

### 7.4.8.1 Data Block Format

The syntax for the SF parameter is:

**<SF parameter>: SF[A<Factor>][B<Factor>]\***

Syntax	Comments
SF	SF for Scale Factor
A<Factor>	Defines the scale factor for the A-axis data
B<Factor>	Defines the scale factor for the B-axis data

The **<Factor>** value is an unsigned decimal number n with the following limitation:

$$0.0001 \leq n \leq 999.99999$$

The decimal part of n consists of not more than 5 digits.

### 7.4.8.2 Examples

Syntax	Comments
%SFA1B1*%	Scale factor 1
%SFA.5B3*%	Defines the scale factor: 0.5 for the A-axis data, 3 for the B-axis data

## 7.5 Obsolete Standard Gerber (RS-274-D)

The current Gerber file format is also known as RS-274X or Extended Gerber. There is also a historic format called Standard Gerber or RS-274-D format. It differs from the current Gerber file format (RS-274X), in that it:

- does not support G36 and G37 codes
- supports the deprecated codes, and
- does not support parameters; coordinate format and apertures cannot be defined

### 7.5.1 Standard Gerber must not be used

*Standard Gerber is obsolete and deprecated.* The word “standard” is misleading here. Standard Gerber is standard NC format. It is not a standard image format: image generation needs a so-called wheel file, and that wheel file is not governed by a standard. Therefore the interpretation of a wheel files, and consequently of a Standard Gerber files, is subjective. In Extended Gerber (RS-274X) image generation is fully governed by the standard. Extended Gerber is the true image standard.

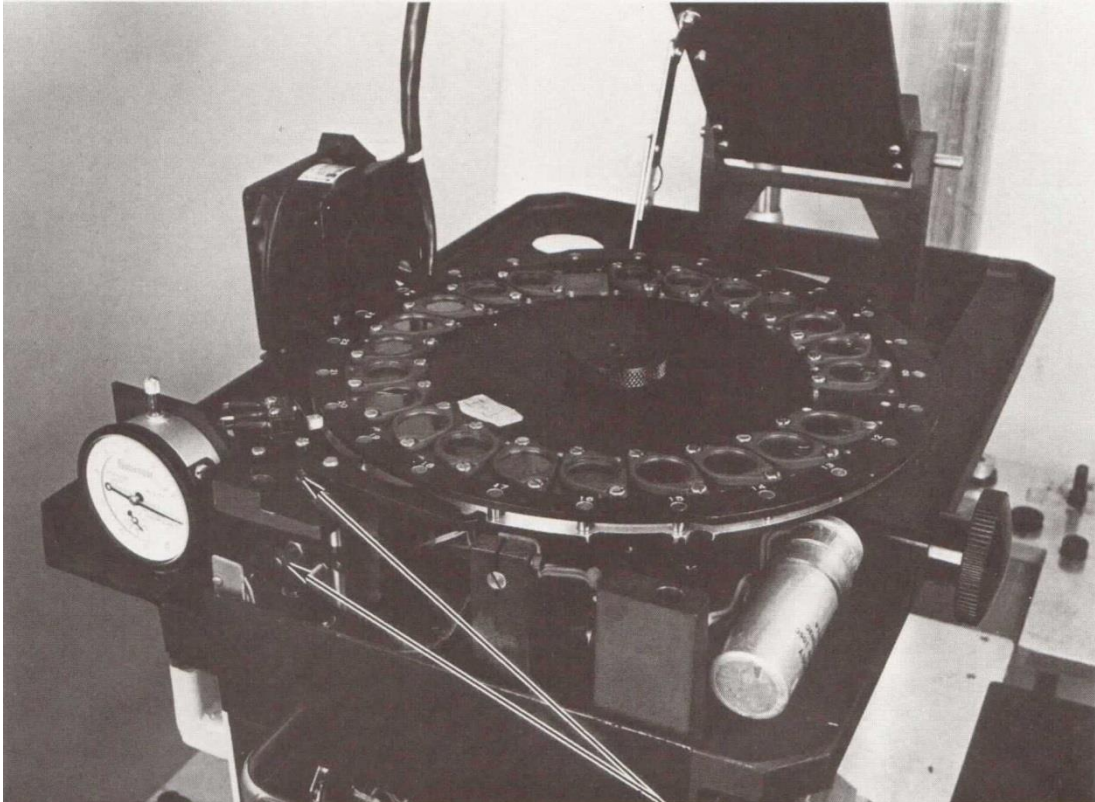
Standard Gerber has important drawbacks over the current Gerber file format and not a single advantage. It is not suited for automatic processing. We strongly recommend to always - always - use Extended Gerber (RS-274X) and to never use Standard Gerber. We can see no reason why anyone in his right mind would use Standard Gerber rather than Extended Gerber.



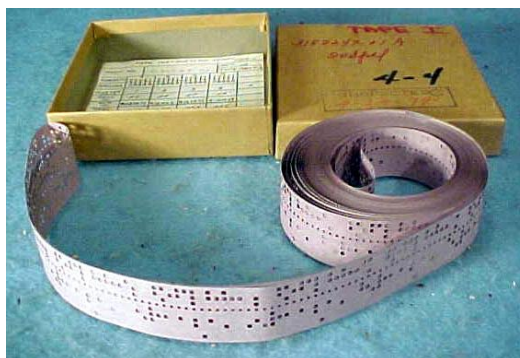
**Warning:** The responsibility of errors or misunderstandings about the wheel file when processing a Standard Gerber file rests solely with the party that decided to use Standard Gerber, with its informal and non-standardized wheel file, rather than Extended Gerber, which is unequivocally and formally standardized.

### 7.5.2 Origin and purpose of Standard Gerber

In the 1960s and 1970s, images were produced on lithographic film by a vector photoplotter, a precision optical Numerical Control machine. Images were produced by beaming light from the plotter's light source onto the film through an aperture on a wheel like that shown in the photograph below. This wheel was rotated to select the appropriate aperture, or it could be substituted by another aperture wheel if additional aperture sizes were needed.



The data for the exposure process was contained in a Standard Gerber file, which was typically recorded onto magnetic or paper tape (see pictures), which was in turn mounted onto the vector photoplottter by the operator.



The operator consulted the accompanying notes, typed the coordinate format on a machine console, mounted the appropriate aperture wheel, changed apertures if necessary, and started the plotter. The Standard Gerber file then drove the plotter through the required movements, controlled the aperture wheel and exposure light, and produced the desired image.

Standard Gerber was so well suited to this task that it became the industry standard.

That was decades ago. Vector photoplotters have not been used since, so Standard Gerber has lost its *raison d'être*. While it deserves a place of honor in the Computer History Museum, Standard Gerber has no place at all in the 21st century's electronics industry.

Of course, everyone is free. If you still send your data by paper tape or half-inch magtape, by all means, use Standard Gerber. Otherwise, use Extended Gerber.

### 7.5.3 Standard Gerber is a NC format, not an image format

From the above, it is clear that Standard Gerber is an NC (Numerical Control) machine format, and not an image description format. It contains neither the coordinate format definition, so the meaning of coordinate data is undefined, nor aperture definitions, so the meaning of flashes and interpolations is undefined.

Thus if an image is to be defined using Standard Gerber, additional information is essential. This typically comes in the form of a so called “wheel file” consisting of notes in an informal text format, plus drawings that define the more complex apertures. The problem is that there is no standard for this extra information, creating enormous potential for error and misunderstandings. This puts the onus squarely on operators' shoulders to ensure that all of the information is assembled and checked on a workstation – manually and with the help of software tools – in order to be sure that all the necessary image data is present.

As if this were not enough, an additional issue is that Standard Gerber renders the informal description of complex apertures, SMD apertures and areas so difficult that designers give up, and opt instead to paint them. This in turn creates such chaos that there is a very real risk of losing valuable data in both CAD and CAM operations. Thus the CAM engineer must be extremely careful to recover, and piece together, the pads in the design.

All of which renders Standard Gerber totally unsuitable for current CAD to CAM data transfer. This format, from the days of paper tape, punched cards, teletypes and electrical typewriters, offers not one single advantage over Extended Gerber.

So Standard Gerber, despite its name, is not an image definition standard, as it must be supported by a whole lot of extra non-standardized information in order to define an image. That's why Ucamco has defined the new Extended Gerber format. This, unlike its predecessor, is a standard, as it standardizes the additional data needed, puts it in the file header, and adds some sorely needed extensions.

### 7.5.4 A fallacy

The following is sometimes said: “The only difference between Standard Gerber and Extended Gerber is that in Extended Gerber the wheel file is embedded in the file. As software was developed to extract data automatically from the wheel files, this is no big deal.”

We beg to differ:

- It is not the only difference.
- This difference is a big deal.

Firstly, the other big difference is that Extended Gerber is a richer format that has all the constructs necessary for describing a PCB image efficiently. It has regions, positive/negative levels and powerful aperture macros. Planes and anti-pads can be described without painting. Pads are properly described as flashes, ensuring that no data is lost.

Secondly, this difference is a really big deal. While it is true that a lot of effort was spent on automating the task of inputting the accompanying notes, only a fraction of all data sets can in fact be read in automatically because they are in a free format. While this freedom was perfectly adequate for the vector photoplotter operator of old, it flies in the face of standardization and automation, which consequently becomes a less reliable and higher maintenance process. And what happens if the notes arrive in another language – imagine, for example, automating the input of a wheel file in Japanese. Or of its supporting drawings, for which again, there are no format definitions. It becomes clear pretty quickly that it is not possible to fully and reliably automate the transfer of such informal data, so the operator must carefully check all results for

errors. This is particularly important if we consider that a lack of standards can also mean lack of clarity about the intentions of the designer, and where responsibility lies in case of errors.

Compare this with the clarity of the formal, standardized aperture definitions in Extended Gerber: reading them in is straightforward, with no need to pore over the results for errors. And as there is a standard, it is clear what was intended, and who is responsible in case of a mistake. So yes, this difference is a big deal. It is the difference between using a published standard format and each individual using his own unspecified format. It is the difference between painstaking, minute manual work and inspection, and reliable, automatic data transfer.