Ucamco

Former Barco ETS

The Gerber Format Specification

A format developed by Ucamco

November 2014

Revision J4

Contents

Figures

Tables

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. 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. Attributes attached to the graphics objects transfer the meta-information needed by manufacturing. A well-constructed Gerber file precisely defines the PCB image and the functions of the objects. This results in a safe, reliable and productive transfer of PCB data from design to manufacturing.

Ucamco continuously clarifies this document based on input from the field and adapts it to current needs. We urge developers of Gerber software to monitor these revisions and adapt their software when needed.

The current Gerber file format is RS-274X or Extended Gerber version 2. Standard Gerber or RS-274-D is technically obsolete and superseded by RS-274X. Do not use Standard Gerber any longer.

Unfortunately, some applications generate mediocre Gerber file. The mail culprit is the use of painting (aka stroking) to create pads and copper pours. While not technically invalid – the correct image is generated – painted files require more manual work and increase the risk of errors in manufacturing. Painting is a relic of the days of vector photoplotters, devices as obsolete as the electrical typewriter. Do not output painted data. We urge all users to output proper Gerber files, without painting. The cooperation of all will make manufacturing more reliable, faster and cheaper for all. To quote a PCB fabricator: "If we would only receive proper Gerber files, it would be a perfect world."

Although many other data transfer formats have appeared they have not displaced Gerber. The reason is simple. Most of the problems in PCB data transfer are due not to limitations in the Gerber file format but to poor practices. The new formats are more complex and less transparent to the user. Poor practices in more complex formats make matters worse, not better. The new formats are often promoted by pointing to the few Gerber files with syntactic or semantic errors. This is of course a fallacy: the solution to bugs is to fix them and not to implement a new format. In fact, new implementations inevitably have more bugs. The remedy is worse than the disease. The industry has not adopted new formats. Gerber remains the standard.

The emergence of Gerber as a standard for PCB data exchange is the result of efforts by many individuals who developed outstanding software for Gerber files. Without their dedication there would be no standard format in the electronics manufacturing industry. Ucamco thanks these dedicated individuals.

Karel Tavernier

Managing Director,
Ucamco

1 **Introduction**

1.1 Scope and Target Audience

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

The Gerber format is the de-facto standard in the printed circuit board (PCB) industry but it also used in other industries. Standard meta-information is targeted at the PCB industry. A basic knowledge of PCB CAD/CAM is helpful in understanding this specification.

1.2 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

1.3 Questions & Feedback

Ucamco strives to make this specification easy to read and unequivocal. We are grateful for any suggestion for improvement. If you find a part of this specification not clear or it still leaves a question about the format, please ask. Your question will be answered and it will be taken into account to improve this document.

The Gerber format includes a set of standard attributes to transfer meta-information in the PCB industry. We are open to your suggestions for other new generally useful attributes.

We can be reached at gerber@ucamco.com.

See www.ucamco.com for more information about Gerber or Ucamco

1.4 Conformance

A Gerber file writer must write files according to this 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 the construct is invalid. An invalid Gerber file is meaningless and does not represent an image. 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.

A Gerber file reader must render a valid Gerber file according to this specification. Readers *must* give a warning when encountering an unknown command further ignore the command and continue processing to prepare for future extensions of the format. There is no other mandatory behavior on reading an invalid Gerber file. It is helpful but *not* mandatory to report any other errors or risky constructs - this would impose 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 reverse engineer the intended image by 'reading between the lines'; however, as an invalid Gerber file is meaningless, it cannot be stated one image is a correct interpretation of the file and another incorrect.

The responsibilities are obvious and plain. Writers must write valid and robust 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. These are the responsibilities but, where reasonable and safe, follow Postel's rule: *Be conservative in what you send, be liberal in what you accept.*

Current Gerber file writers must not use deprecated constructs. Current Gerber file readers may not support deprecated constructs or may support some or all of them as they can 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.

1.5 Copyright and Intellectual Property

1.6 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. (See www.ucamco.com) 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 technically obsolete and is superseded by RS-274X. Standard Gerber deserves a place of honor in the Museum for the History of Computing but it does not deserve a place in modern workflows.

1.7 Record of Revisions

1.7.1 Revision J4

The .AperFunction values "Slot", "CutOut" and "Cavity" were added. The text on standard attributes was made more explicit. An example of a poorly constructed plane was added.

1.7.2 Revision J3

The .FileFunction values for copper and drill layers were extended to contain more information about the complete job.

1.7.3 Revision J2

Associating aperture attributes with regions was much simplified. A section about numerical accuracy was added. The somewhat awkward term '(mass) parameter' was replaced by the more intuitive 'command'.

1.7.4 Revision J1

This revision created version 2 of Gerber format by adding attributes to what was hitherto a pure image format. See chapter 5. An shorthand for Gerber version 2 is "X2", with "X1" being Gerber without attributes. Gerber version 2 is backward compatible as attributes do not affect image generation.

Gerber version 2 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.

1.7.5 **Revision I4**

The commands 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. The somewhat awkward term '(mass) parameter' was replaced by the more intuitive 'parameter code'

1.7.6 **Revision I3**

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

1.7.7 **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.

1.7.8 **Revision I1**

General. The entire specification was extensively reviewed for clarity. The document was re-organize, the quality of the text and the drawings has been improved and many new drawings were added.

Deprecated elements. Elements of the format 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 2.6 and 2.5. *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 2.6. *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 4.12.3. 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 3.5.3. *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.

1.7.8.1 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.

1.8 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.

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, parameter codes* and/or *coordinate data.* Some commands control the *graphics state* (see 2.6). Other commands – the operation codes, see 2.5 - generate a stream of graphics object which combined produce the final image. The graphics state determines how the operation codes create the graphics objects.

A Gerber file can be processed in a single pass. This imposes constraints on the sequence of the commands.

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

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. (The name *track* can be used for either a draw or an arc.)

❑ 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 (also called copper pours).

2.3 Apertures

An aperture is a basic 2D geometric shape or figure. An example is a circle with a diameter of 2mm. Apertures are used for flashing or stroking – see 2.4 & 4.2.

An aperture has a *flash point*. When an aperture is flashed its flash point is positioned at the coordinates of a flash command (D03 – see 4.2). The flash point of a standard aperture is its geometric center. The flash point of a macro aperture is the origin of the coordinates used in the AM command.

The AD (Aperture Define) command creates and defines an aperture. The AD command contains the aperture template and parameters defining its shape and size. It also assigns the D-code or aperture number to identify it.

There are two kinds of apertures templates: *standard apertures* and *macro apertures*:
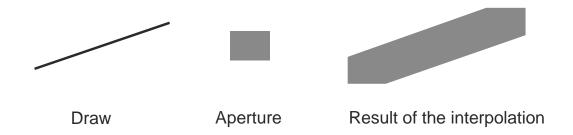
- Standard apertures are pre-defined: the circle (C), rectangle (R), obround (O) and regular polygon (P). See 4.10.

- Macro apertures are created and by an AM (Aperture Macro) command. They are identified by their given name. Any shape can be created by combining and parameterizing a small number or primitives. See 4.12**.**

Macros are a powerful and elegant feature of the Gerber format. Apertures of any shape can be created. A file writer can easily define the apertures needed. A file reader can handle any such aperture by implementing a small number of primitives. This single flexible mechanism, based on a small number of simple primitives, replaces the need for a large - but always insufficient - number of pre-defined apertures. New apertures types can be created without extending the format and update implementations.

Standard apertures can be viewed as built-in macro apertures.
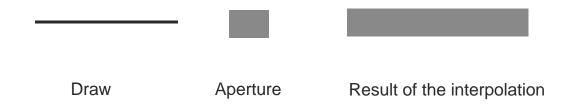
2.4 Stroking

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:

Draw   Aperture   Result of the interpolation

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:

Draw   Aperture   Result of the interpolation

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 macro apertures cannot be used for stroking, whatever their shape.

A zero size circular aperture can be used for stroking. They create graphics 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 graphics object is a draw or arc, not a flash.

Any valid aperture can be flashed.

2.5 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 used to be called a lights-off move.
- ❑ D01 creates a straight or circular line segment by interpolating from the current point to the coordinate pair. This used to be 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 2.7.

2.6 Graphics State

A Gerber file defines a graphics state after each command. The operation codes are controlled by the graphics state, see 2.5. A graphics state variable that affects an operation code must be defined before the operation code is issued.

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

All other graphics state variables are set explicitly by commands. 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 command	Fixed	Undefined
Unit	Inch or mm See MO command.	Fixed	Undefined
Current aperture	Standard or macro aperture. See AD and AM commands.	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 command	Changeable	1,1,-,-
Level polarity	Dark, Clear See LP command.	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.

2.7 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. An example is in 4.5.8.

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 command starts a new level and sets its polarity, see 4.14.

2.8 **Attributes**

Attributes add meta-information to a Gerber file. These are akin to labels providing information associated to image files, or features within them. Examples of meta-information conveyed by attributes are:

- *The function of the file.* Is the file the top solder mask, or the bottom copper layer, etc?
- *The part represented by the file.* Does it represent a single PCB, an array, a coupon?
- *The function of a pad.* Is the flash is an SMD pad, or a via pad, or a fiducial, etc.

The attribute syntax provides a flexible and standardized way to add meta-information to the files, independent of the specific semantics or application.

Example:

This command in adds a standard attribute indicating that the file is the top solder mask.

```
%TF.FileFunction,Soldermask,Top*
```

Attributes are not needed when just the image is needed. However, attributes are needed when PCB data is transferred from design to fabrication. The PCB fabricator needs more than just the image: for example he needs to know which pads are via pads to manufacture the solder mask. The attributes transfer this information in an unequivocal and standardized manner. They convey the design intent from CAD to CAM. This is sometimes rather grandly called "adding intelligence to the image". Without these attributes the fabricator has to reverse engineer the design intent of the features in the file, a time-consuming and error-prone process.
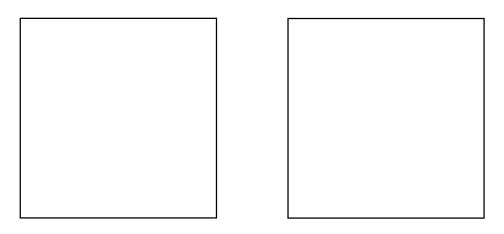
Attributes do not affect the image. A Gerber reader will generate the correct image if it ignores the attributes. If only the image is needed attributes can simply be ignored.

2.9 Example Files

These annotated samples illustrate the use of the elements of the Gerber file format. They will give you a feel for the Gerber file format if it is new to you and thus make the formal specification easier to read.

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
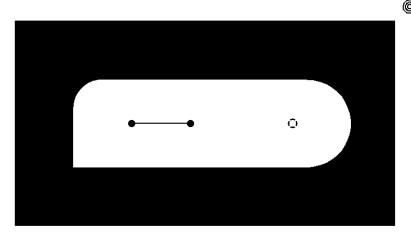
`G04 Ucamco ex. 1: Two square boxes*`	A comment
`%FSLAX25Y25*%`	Coordinate format specification: Leading zero's omitted Absolute coordinates 2 digits in the integer part 5 digits in the fractional part
`%MOMM*%`	Unit set to mm
`%TF.Part,Other*%`	The file is not a layer of a PCB part - it is just an example.
`%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)
`G01*`	Set linear interpolation
`X500000Y0D01*`	Draw to (5, 0) with D10
`Y500000D01*`	Draw to (5, 5) with D10
`X0D01*`	Draw to (0, 5) with D10
`Y0D01*`	Draw to (0, 0) with D10
`X600000D02*`	Move to (6, 0)
`X1100000D01*`	Draw to (11, 0) with D10
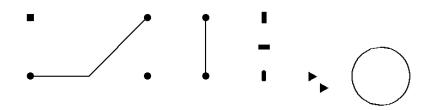
`Y500000D01*`	Draw to (11, 5) with D10
`X600000D01*`	Draw to (6, 5) with D10
`Y0D01*`	Draw to (6, 0) with D10
`M02*`	End of file

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

Example 2 illustrates the use of levels and various apertures.

`G04 Ucamco ex. 2: Shapes*`	A comment statement
`%FSLAX26Y26*%`	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 PCB production. It is used here to make the file easier to read by a human.
`%TF.Part,Other*%`	The file is not a layer of a PCB part - it is just an example.
`%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,.01,0.01,3,0.003, 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
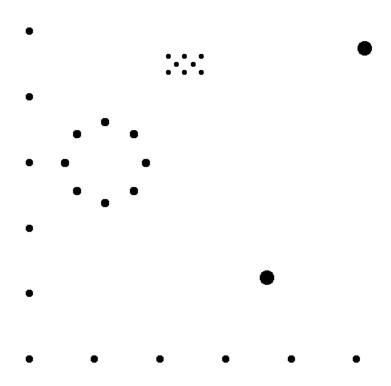
`%ADD17P,0.100X3*%`	Aperture definition: D17 is a polygon with 3 vertices and circumscribed circle with diameter 0.1 inch
`%ADD18TARGET125*%`	Aperture definition: D18 is the macro aperture called 'TARGET125' defined earlier
`%ADD19THERMAL80*%`	Aperture definition: D19 is the macro aperture called 'THERMAL80' defined earlier
`G04 Start image generation`	A comment
`D10*`	Select aperture with D-code 10
`X0Y250000D02*`	Move current point to (0, 0.25) inch
`G01*`	Set linear interpolation
`X0Y0D01*`	Linear interpolation (draw)
`X250000Y0D01*`	Linear interpolation (draw)
`X1000000Y1000000D02*`	Move current point
`X1500000D01*`	Linear interpolation (draw)
`X2000000Y1500000D01*`	Linear interpolation (draw)
`X2500000D02*`	Move current point. Since the X and Y coordinates are modal, Y is not repeated
`Y1000000D01*`	The X coordinate is not repeated and thus its previous value of 2.5 inch is used
`D11*`	Select aperture with D-code 11
`X1000000Y1000000D03*`	Flash D11 at (1.0, 1.0). Y is modal.
`X2000000D03*`	Flash D11 at (2.0, 1.0). Y is modal.
`X2500000D03*`	Flash D11 at (2.5, 1.0). Y is modal.
`Y1500000D03*`	Flash D11 at (2.5, 1.5). X is modal.
`X2000000D03*`	Flash D11 at (2.0, 1.5). Y is modal.
`D12*`	Select aperture with D-code 12
`X1000000Y1500000D03*`	Move to (1.0, 1.5) and flash
`D13*`	Select new aperture with D-code 13
`X3000000Y1500000D03*`	Move to (3.0, 1.5) and flash
`D14*`	Select new aperture with D-code 14
`Y1250000D03*`	Move to (3.0, 1.25) and flash
`D15*`	Select new aperture with D-code 15

`Y1000000D03*`	Move to (3.0, 1.0) and flash
`D10*`	Select new aperture with D-code 10
`X3750000Y1000000D02*`	Move current point. This sets the start point for the following arc interpolation
`G75*`	Set multi quadrant mode
`G03`	Set counterclockwise circular interpolation
`X3750000Y1000000I250000J0D01*`	Interpolate a complete circle
`D16*`	Select new aperture with D-code 16
`X3400000Y1000000D03*`	Flash D16
`D17*`	Select new aperture with D-code 17
`X3500000Y900000D03*`	Flash D17
`D10*`	Select new aperture with D-code 10
`G36*`	Start a region
`X500000Y2000000D02*`	Move current point to (0.5, 2.0)
`G01*`	Set linear interpolation
`Y3750000D01*`	Linear interpolation (draw)
`X3750000D01*`	Linear interpolation (draw)
`Y2000000D01*`	Linear interpolation (draw)
`X500000D01*`	Linear interpolation (draw)
`G37*`	Create the region by filling the contour
`D18*`	Select new aperture with D-code 18
`X0Y3875000D03*`	Flash D18
`X3875000Y3875000D03*`	Flash D18
`%LPC*%`	Level polarity is clear
`G36*`	Start a region
`X1000000Y2500000D02*`	Move current point to (1.0, 2.5)
`Y3000000D01*`	Linear interpolation (draw)
`G74*`	Set single quadrant mode
`G02*`	Set clockwise circular interpolation
`X1250000Y3250000I250000J0D01*`	Clockwise arc with radius 0.25
`G01*`	Set linear interpolation

`X3000000D01*`	Linear interpolation (draw)
`G75*`	Set multi quadrant mode
`G02*`	Set clockwise circular interpolation
`X3000000Y2500000I0J-375000D01*`	Clockwise arc with radius 0.375
`G01*`	Set linear interpolation
`X1000000D01*`	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
`X1500000Y2875000D02*`	Move current point
`X2000000D01*`	Linear interpolation (draw)
`D11*`	Select aperture with D-code 11
`X1500000Y2875000D03*`	Flash D11
`X2000000D03*`	Flash D11
`D19*`	Select aperture with D-code 19
`X2875000Y2875000D03*`	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

`%FSLAX26Y26*%`	Format specification: Leading zero's omitted Absolute coordinates Coordinate format is 2.6
`%MOIN*%`	Units are inches
`%TF.FileFunction,Plated,1,8,PTH*%`	This drill file describes plated-through holes
`%TF.Part,Single*%`	The file is part of a single PCB
`%LPD*%`	Dark level polarity
`%SRX1Y1I0J0*%`	Indicates that the following data is not stepped.
`%TA.DrillTolerance,0.01,0.005*%`	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.
`%TA.AperFunction,ComponentDrill%`	Indicates that the following apertures define component drill holes.
`%ADD10C,0.014000*%`	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 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
`G01*`	Set interpolation mode to linear interpolation
`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 describe perfectly. For more information, see 5.4.1.1.

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 graphics 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.

Block: A set of graphics objects that can be stepped and repeated in the image plane.

Circular interpolation: Creating an arc.

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

Contour: A closed curve defining a region.

Current aperture: The last aperture selected by a D-command. Flashes and strokes always are created with the current aperture.

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

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

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

Draw: Either a graphics 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 graphics 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 graphics 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.

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.

Parameter codes: Commands enclosed between '%', typically specifying the behavior of the operation codes. These used to be called (Mass) Parameters.

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 'Clear'.

Polygon fill: This is an old name for region fill. See section 4.5.

Region: A graphics 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°**.**

Macro aperture: An aperture template defined by a macro with an AM command.

Standard aperture: An aperture template standard or pre-defined in the format.

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 copies of a single image block are made to produce a multiple image.

Stroke: To create a draw or an arc object (interpolation).

Track: Either a draw or an arc. Typically used for a conductive track on a PCB.

3 **Syntax**

3.1 Formatting of 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.
- ❑ **<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)**

3.2 Character Set

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

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.

SP can only be used inside strings. It cannot be used inside or between commands and data blocks, etc.

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

3.3 Variable Types

3.3.1 Integers

Integers are a sequence of one or more digits optionally preceded by a '+' or a '-' sign. They must fit in a 32 bit signed integer.

3.3.2 Decimals

Decimals are a sequence of one or more digits with an optional decimal point optionally preceded by a '+' or a '-' sign.

3.3.3 Coordinate Number

Coordinate numbers are integers conforming to the rules set by the FS parameter. See 4.8.1. Coordinate numbers are used in coordinate data.

3.3.4 Names

Names are used to identify macros and attributes.

Names consist of upper or lower case letters, underscores ('_'), dots '.', dollar signs ('$') and digits. The first character cannot be a digit.

`Name =[a-zA-Z_$.]{[a-zA-Z_$.0-9]+}`

Names can be maximally 127 characters long.

Names are case-sensitive: `Name ≠ name`

Names beginning with a dot '.' are reserved for *standard names* defined in the specification. It is not allowed to have user defined names beginning with a dot!

3.3.5 **Strings**

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

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

Strings can be maximally 65,535 characters long. (65,535 fits in an unsigned int 16.)

Strings are case-sensitive: `String ≠ string`

3.4 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 function codes, coordinates or parameter codes.

Example:

```
X0Y0D02*
G01*
X50000Y0D01*
```

Data blocks are the low level syntactical elements of a Gerber file. The data blocks can be semantically interconnected and form a group representing 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.

3.5 Commands

3.5.1 **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 Parameter codes then it starts and ends with a '%' character.

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

The syntax of a command is as follows:

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

In the example below the command consists of a single data block a representing a G02 and D01 function code together with a coordinate pair and offset in X and Y..

Example:

```
G02X0Y100I-400J100D01*
```

The following example is an AM command built of three data blocks.

Example:

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

There are three types of commands:

❑ Function codes. See 3.5.2.

❑ Coordinate data. See 3.5.3.

❑ Parameter codes. See 3.5.4.

Function codes are identified by a code letter G, D or M followed by a code number, e.g. `G02`. Parameter codes are identified by two letters, e.g. `MO`. The codes and code letters originates from the original EIA RS-274-D specification. Parameter codes were an extension on RS-274-D. This difference is historic; please accept it as it is.

Example of commands:

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

3.5.2 **Function Codes**

Function codes are either
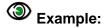
❑ Operation codes, operating on coordinate data, i.e. D01, D02 or D03.

❑ 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*
```

The function codes are described in detail in chapter 4.

3.5.3 **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.

The coordinate data block syntax is 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 a distance or offset in the X or Y direction
<Number>	Coordinate number - see section 3.3.3 - defining either a coordinate (X,Y) or an offset or distance(I,J). The number must have at least one digit.
D01|D02|D03	An operation code operation on the coordinate preceding it. Their meaning is explained in 2.5.

The FS and MO commands specify how to interpret the coordinate numbers. The coordinate numbers define points in the plane using a right-handed orthonormal 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*
```

3.5.4 **Parameter Codes**

Parameter codes define characteristics of the file.

Note: Originally Parameter codes were called (Mass) Parameters.

Parameter codes operating on the entire image must be placed in the header of the file. Other parameter codes are placed at the appropriate location.

Parameter codes consist of a two-character command code followed by command data. The command code identifies the command. The structure and meaning command data depends on the command code.

Parameter codes are enclosed into a pair of delimiter '%' characters. Usually a parameter code consists of a single data block ending with a '*'. The AM command however 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*%
```

There can be one or more parameter codes between each pair of '%' delimiters, up to a maximum of 4096 characters between these delimiters.

Example:

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

Line separators are permitted between parameter codes to improve readability, with the following syntax:

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

Example:

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

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

The syntax for an individual parameter code is:

%Code<required modifiers>[optional modifiers]*%

Syntax	Comments
Command code	2-character code (AD, AM, FS, etc…)
<Mandatory modifiers>	Must be entered to complete definition

<optional modifiers>	May be required depending on the required modifiers

We distinguish two classes of parameter codes:

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

4 **Graphics**

4.1 Graphics Overview

Processing the stream of commands creates a stream of graphics objects.

There are two types of graphics commands:

- ❑ Function codes
- ❑ Parameter codes

The tables below give an overview of the graphics function codes and parameter codes. They are explained in detail later.

Command	Name	Description	Comments
FS	Format Specification	Sets the 'Coordinate format' graphics state variable. See 4.8.	These commands can only be used once, in the header of the file.
MO	Mode (inch or mm)	Sets the 'Unit' graphics state variable. See 4.9.	
AD	Aperture Definition	Assigns a D code number to an aperture definition. See 4.10.	These commands can be used multiple times. It is recommended to put them in header of the file
AM	Aperture Macro	Defines macro apertures which can be referenced from the AD command. See 4.12.	
SR	Step and Repeat	Sets the 'Step and Repeat' graphics state variable. See 4.13.	These commands can be used multiple times over the whole file.
LP	Level Polarity	Starts a new level and sets the 'Level polarity' graphics state variable. See 4.14.	

Graphics parameter codes

Code	Description	Comments
D01	Interpolate operation code. See 4.2.	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. See 4.2.	D02 does not create a graphics object but move the current point to the coordinate.
D03	Flash operation code. See 4.2.	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
Dnn (nn≥10)	Set the current aperture.	Sets the current aperture to aperture nn. (Aperture numbers are set an AD command, see4.10.)
G01	Set the interpolation mode to linear. See 4.3.	A modifier of the interpolation operation code D01.
G02	Set the interpolation mode to 'Clockwise circular interpolation'. See 4.4.	
G03	Set the interpolation mode to 'Counterclockwise circular interpolation'. See 4.4.	
G04	Ignore data block. See 4.6.	Used for comments.
G36	Set region mode on. See 4.5.	Used to create regions.
G37	Set region mode off. See 4.5.	
G74	Set quadrant mode to 'Single quadrant'. See 4.4.	A modifier of the circular interpolation mode..
G75	Set quadrant mode to 'Multi quadrant. See 4.4.'	
M02	Indicates the end of the file. See 4.7.	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 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*.

❑ D02 moves the current point to the coordinates. No graphics object is generated. This operation is called to *move*.

❑ 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 2.7.

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. This construction is not recommended. It is a useless variation and will be deprecated in the near future.

The syntax for G01, G02, G02, 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 syntax for the D02 coordinate data block – a 'move' - is the following:

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

Syntax	Comments
X<Number>	Coordinate data defining the X coordinate of the new current point. If missing then the previous X coordinate is used. <Number> is a coordinate number – see section 3.3.3.
Y<Number>	Coordinate data defining the Y coordinate of the new current point. If missing then the previous Y coordinate is used. <Number> is a coordinate number – see section 3.3.3.
D02	Move operation code.

👁 **Example:**

```
X200Y1000D02*
```

It is allowed to specify G01/G02/G03 together with a D02 (move). This is not recommended.

The syntax for the D03 coordinate data block – a 'flash' - is:

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

Syntax	Comments
X<Number>	Coordinate data defining the X coordinate of the flash point. If missing then the previous X coordinate is used. <Number> is a coordinate number – see section 3.3.3.
Y<Number>	Coordinate data defining the Y coordinate of the flash point. If missing then the previous Y coordinate is used. <Number> is a coordinate number – see section 3.3.3
D03	Flash operation code

⊙ Example:

```
X1000Y1000D03*
```

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

⊙ Example:

```
G36*
X200Y1000D02*
G01*
X1200D01*
Y200D01*
X200D01*
Y600D01*
X500D01*
G75*
G03*
X500Y600I300J0D01*
G74*
G01*
X200D01*
Y1000D01*
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 graphics 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<Number>][Y<Number>][D(01|02)]*

Syntax	Comments
G(01|1)	G01 or G01 – Sets interpolation mode to 'Linear interpolation'
X<Number>	Coordinate data defining the X coordinate of the draw end point.
	If missing then the X coordinate of the current point is used.
	<Number> is a coordinate number – see section 3.3.3.
Y<Number>	Coordinate data defining the Y coordinate of the draw end point.
	If missing then the Y coordinate of the current point is used.
	<Number> is a coordinate number – see section 3.3.3.
D(01|02)	Interpolate or 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)	In single quadrant mode the arc is not allowed to extend over more than 90°. The following relation must hold: $0° ≤ A ≤ 90°$, where A is the arc angle 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.
Multi quadrant (G75)	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: $0° < A ≤ 360°$, where A is the arc angle If the start point of the arc is equal to the end point, the arc is a full circle of 360°.

Quadrant modes

The codes G74 and G75 allow switching between single- and multi-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 without a preceding G74 or G75 code is invalid.

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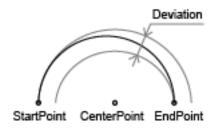
For an arc to be circular the center must be positioned at exactly the same distance - radius - from the start point and the end point. The two radii must be equal. The definition of an arc is then obvious.

However, as Gerber file has a finite resolution, the center point generally cannot be position 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 monotonic 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 6. Note that this curve is only mathematically circular if the deviation is zero.

6. 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 7.

7. 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. See 4.8.1.

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<Number>][Y<Number>]

[I<Number>][J<Number>][D(01|02)]*

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

X<Number>	Defines the X coordinate of the arc end point.
	If missing then the previous X coordinate is used.
	<Number> is a coordinate number – see section 3.3.3.
Y<Number>	Defines the Y coordinate of the arc end point.
	If missing then the previous Y coordinate is used.
	<Number> is a coordinate number – see section 3.3.3.
I<Number>	The distance between the arc start point and the center parallel to the X axis. Number is ≥ 0.
	If missing then a 0 distance is used.
	<Number> is a coordinate number – see section 3.3.3.
J<Number>	The distance between the arc start point and the center parallel to the Y axis. Number is ≥ 0.
	If missing then a 0 distance is used.
	<Number> is a coordinate number – see section 3.3.3.
D(01|02)	Interpolate or 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:**
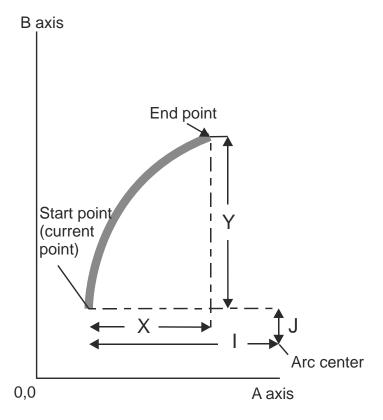
```
G74*
G03*
X700Y1000I400J0D01*
```

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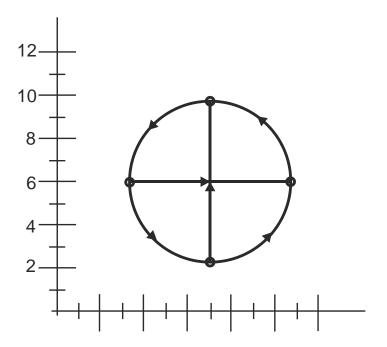
The coordinates of an arc endpoint and the center distances are interpreted according to the coordinate format specified by the FS command and the unit specified by the MO command. The following image illustrates how arcs are interpolated.

B axis

End point

Start point
(current
point)

Y

X

I

J

Arc center

0,0   A axis
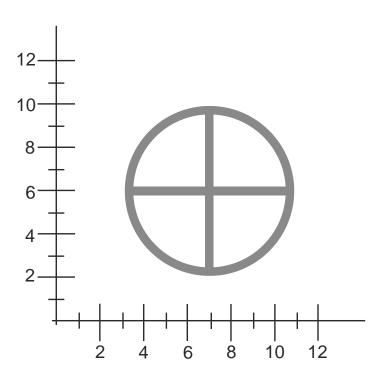
8.   Single quadrant mode

4.4.3.2.1 Example

Syntax	Comments
`G74*`	Single quadrant mode
`D10*`	Use aperture D10
`X1100Y600D02*`	Start from (11, 6)
`G03*`	Set counterclockwise interpolation
`X700Y1000I400J0D01*`	Quarter arc (radius 4) to (7, 10)
`X300Y600I0J400D01*`	Quarter arc (radius 4) to (3, 6)
`X700Y200I400J0D01*`	Quarter arc (radius 4) to (7, 2)
`X1100Y600I0J400D01*`	Quarter arc (radius 4) to (11, 6)
`X300D02*`	Start from (3 ,6)
`G01*`	Set linear interpolation
`X1100D01*`	Draw to (11, 6)
`X700Y200D02*`	Start from (7, 2)
`Y1000D01*`	Draw to (7, 10)
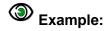
9.   Single quadrant mode example: arcs and draws

10.   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<Number>][Y<Number>]

[I<Number>][J<Number>][D(01|02)]*

Syntax	Comments
G(02|2|03|3)	Sets the interpolation mode: G02 or G02 – Clockwise circular interpolation G03 or G03 – Counterclockwise circular interpolation
X<Number>	Defines the X coordinate of the arc end point. If missing then the previous X coordinate is used. <Number> is a coordinate number – see section 3.3.3.
Y<Number>	Defines the Y coordinate of the arc end point. If missing then the previous Y coordinate is used. <Number> is a coordinate number – see section 3.3.3.
I<Number>	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. <Number> is a coordinate number – see section 3.3.3.
J<Number>	Defines the offset or signed distance between the arc start point and the center measured parallel to the Y axis. If missing then a 0 offset is used. <Number> is a coordinate number – see section 3.3.3.
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:
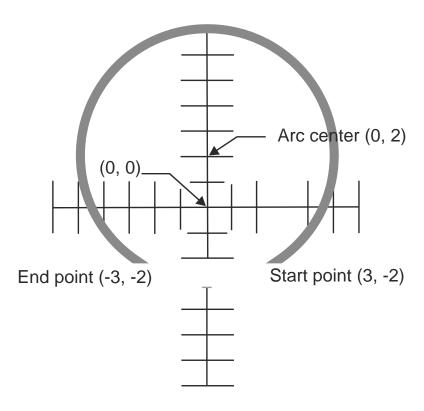
```
G75*
G03*
X-300Y-200I-300J400D01*
```

4.4.5 **Arc Example**

Syntax	Comments
X300Y-200D02*	Move to (3, -2)
G75*	Set multi quadrant mode
G03*	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)
X-300Y-200I-300J400D01*	

11.   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. See 4.8.1.

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
G01*	
X0Y600D02*	Start from (0, 6)
G74*	Single quadrant mode
G02*	
X0Y600I500J0D01*	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
G01*	
X0Y600D02*	Start from (0, 6)
G75*	Multi quadrant mode
G02*	
X0Y600I500J0D01*	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.

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 turns region mode *on* and G37 turns 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 always finishes a contour even the current point does not change, e.g. a `D02*`.) A D02 is only allowed if the preceding contour is closed. The next D01 encountered starts a new contour. Thus an unlimited number of contours can be created between a single G36/G37 pair.

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 when all contours are properly 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 fully coincident *draws*, used to create holes in a region with cut-ins; see 4.5.9. A pair of draws are said to be fully coincident if and only if the draws coincide completely, with the second draw starting where the first one ends.
3. Zero-length draws and arcs are allowed and have no effect. (Avoid them as they are useless and can only cause confusion.)

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 (draws not sharing both vertices), diagonal fully coincident draws, fully 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, full 360° arcs.

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

The region is associated with the current aperture. This has no graphical effect, but the region inherits the attributes from the current aperture.

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. Parameter codes are *not* allowed.

⚠️**Warning:** Use cut-ins only for simple configurations. Regions with many cut-ins are complex and error-prone; numerical rounding can create self-intersection which make the file invalid. See section 4.5.10 for an example on how *not* to use cut-ins.

⚠️**Warning:** For professional PCB production planes the holes (anti-pads, clearances) must be constructed by superimposing the flashing the anti-pads in dark polarity (LPC) and *not* be with cut-ins. See 4.5.8.

⚠️ 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.

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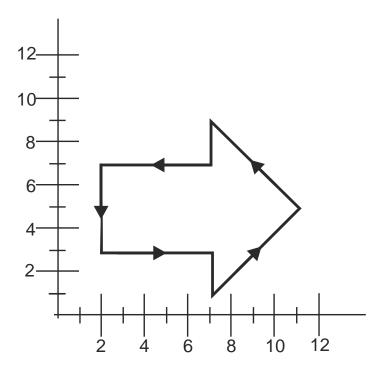
ℹ️ Note: In the 1960's and 1970s, the era of vector plotters, the only way to produce a region was by painting (aka filling or stroking) it with draws. This produces the correct image. However, the file size explodes. More importantly, painted data cannot be handled properly in PCB CAM and the painting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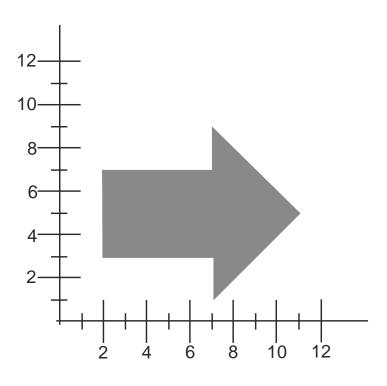
ℹ️ Note: In previous versions of this document "contour fill" was called "polygon fill".

4.5.2 **Example: a simple contour**

Syntax	Comments
`G36*`	Start a region
`X200Y300000D02*`	Move the current point to (2, 3)
`G01*`	Set linear interpolation
`X700000D01*`	Line segment to (7, 3)
`Y100000D01*`	Line segment to (7, 1)
`X1100000Y500000D01*`	Line segment to (11, 5)
`X700000Y900000D01*`	Line segment to (7, 9)
`Y700000D01*`	Line segment to (7, 7)
`X200000D01*`	Line segment to (2, 7)
`Y300000D01*`	Line segment to (2, 3)
`G37*`	Create the region by filling the contour

12.   Simple contour example: the segments

13.   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
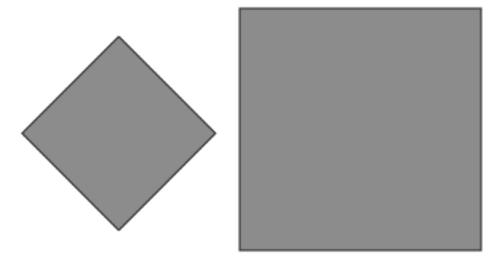
Example 1	Example 2	Example 3
... G01* D11* ... X300Y500D01* G36* X5000Y5000D02* X6000D01* Y6000D01* X5000D01* Y5000Y5000D01* G37* ...	... G01* D11* ... X300Y500D01* X5000Y5000D02* G36* X6000D01* Y6000D01* X5000D01* Y5000Y5000D01* G37* ...	... G01* D11* ... X300Y500D01* X5000Y5000D01* G36* X6000D01* Y6000D01* X5000D01* Y5000Y5000D01* G37* ...
This sequence creates a square contour after the stroked draw X300Y500D01*	Swap D02 and G0336. 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*%
G01*
%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:

14.   Use of D02 to start an new non-overlapping contour
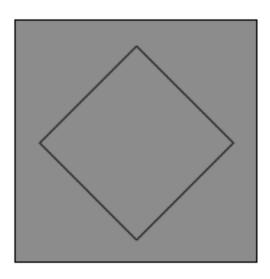
Two different contours were created. Each contour is filled individually. The filled area is the union of the filled areas.

4.5.5 **Example fle: Overlapping contours**

```
G04 Overlapping contours*
%FSLAX23Y23*%
%MOMM*%
%ADD10C,1.00000*%
G01*
%LPD*%
G36*
X0Y5000D02*
Y10000D01*
X10000D01*
Y0D01*
X0D01*
Y5000D01*
X1000D02*
X5000Y1000D01*
X9000Y5000D01*
X5000Y9000D01*
X1000Y5000D01*
G37*
M02*
```

This creates the following image:
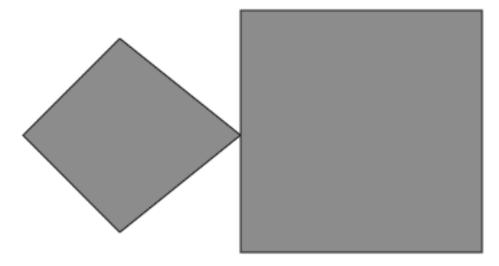
15.   Use of D02 to start an new overlapping contour

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*%
G01*
%LPD*%
G36*
X0Y5000D02*
Y10000D01*
X10000D01*
Y0D01*
X0D01*
Y5000D01*
D02*
X-5000Y1000D01*
X-9000Y5000D01*
X-5000Y9000D01*
X0Y5000D01*
G37*
M02*
```

This creates the following image:
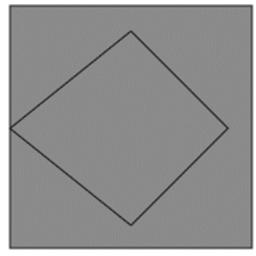
16.   Use of D02 to start an new non-overlapping contour

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*%
G01*
%LPD*%
G36*
X0Y5000D02*
Y10000D01*
X10000D01*
Y0D01*
X0D01*
Y5000D01*
D02*
X5000Y1000D01*
X9000Y5000D01*
X5000Y9000D01*
X0Y5000D01*
G37*
M02*
```

This creates the following image:

17.   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 Example: 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.

Example:

```
G04 This file illustrates how to use levels to create holes*
%FSLAX27Y27*%
%MOMM*%
G01*
G04 First level: big square - dark polarity*
%LPD*%
G36*
X2500000Y2500000D02*
X17500000D01*
Y17500000D01*
X2500000D01*
Y2500000D01*
G37*
G04 Second level: big circle - clear polarity*
%LPC*%
G36*
G75*
X5000000Y10000000D02*
G03*
X5000000Y10000000I5000000J0D01*
G37*
G04 Third level: small square - dark polarity*
%LPD*%
G36*
X7500000Y7500000D02*
X12500000D01*
Y12500000D01*
X7500000D01*
Y7500000D01*
G37*
G04 Fourth level: small circle - clear polarity*
%LPC*%
```

```
G36*
G75*
X11500000Y10000000D02*
G03*
X11500000Y10000000I2500000J0D01*
G37*
M02*
```
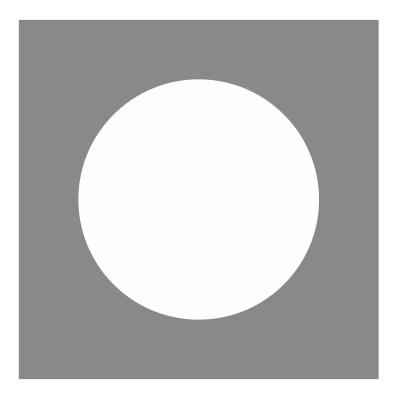
Below there are pictures which show the resulting image after adding each level.
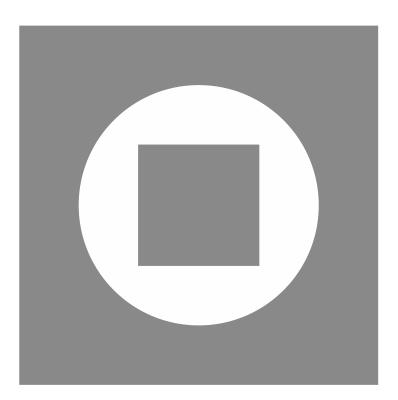
18.   Resulting image: first level only

```
G36*
G75*
```

19.   Resulting image: first and second levels
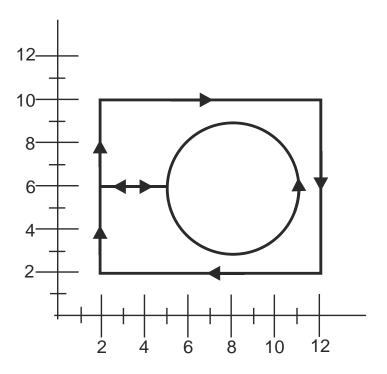
20.   Resulting image: first, second and third levels
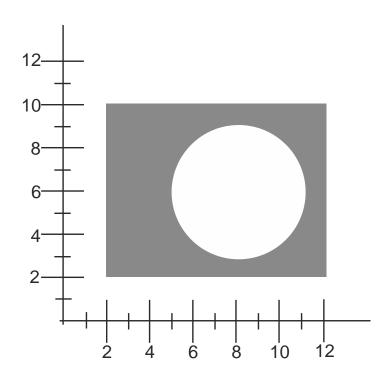
21.   Resulting image: all four levels

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)
G01*	
X1200D01*	Draw to (12,10)
Y200D01*	Draw to (12, 2)
X200D01*	Draw to (2, 2)
Y600D01*	Draw to (2, 6)
X500D01*	Draw to (5, 6),1st fully coincident draw
G03	
X500Y600I300J0D01*	Full counterclockwise circle with radius 300
G01*	Draw to (2, 6), 2nd fully coincident draw
X200D01*	Draw to (2, 10)
Y1000D01*	Create the region by filling the contour
G37*	

22.   Simple cut-in: the segments

23.   Simple cut-in: the image
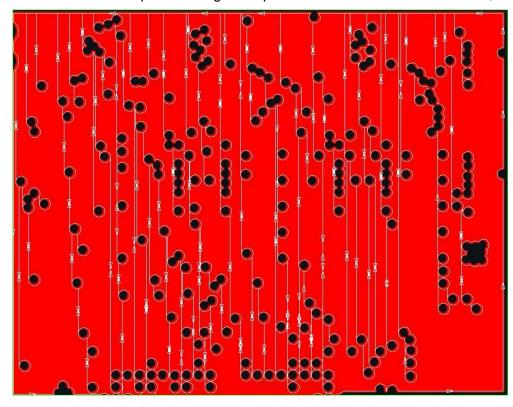
4.5.10   **Example: Power and ground planes**

The proper way to construct power and ground planes is as follows. First create the copper pour with a region in dark polarity (LPD), and then erase the clearances by switching to clear polarity (LPC) and flash the anti-pads:

```
G04 We define the antipad used to create the clearances
%TF.AperFunction,AntiPad*%
%AD11C….*%
….
G04 We now define the copper pour as a region*
LPD*
G36*
X…Y…D02*
X…Y…D01*
…
G37*
G04 We now we flash clearances
%LPC*%
D11*
X…Y…D03*
```

This is simple and clear. In the CAD layout the location of the anti-pads is known and it is transferred directly to CAM. CAM needs to know the locations of the anti-pads.

Sometimes the clearances in power and ground planes are constructed with cut-ins, as below.

*24.   How **not** to create power and ground planes.*
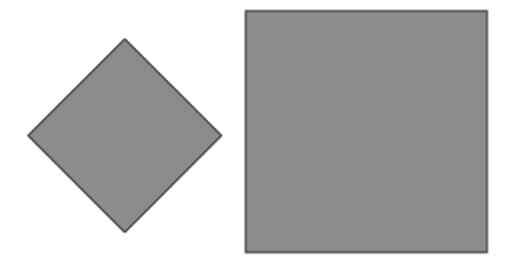
Don't use this complex construction unless you have a very good reason, and then be very careful, especially about rounding errors. A complex and hence error-prone algorithm is needed to create the clearances using cut-ins. Numerical rounding gives plenty of opportunities for errors and the inadvertent creation self-intersections, making the file invalid. Just creating an image of such a plane is one thing but CAM needs an equally complex algorithm to get rid of the cut-in lines and recover the locations of the anti-pads, again with plenty of opportunities for error.

4.5.11 Examples: Fully coincident draws

Example 1

```
G04 ex1: non overlapping*
%FSLAX23Y23*%
%MOMM*%
%ADD10C,1.00000*%
G01*
%LPD*%
G36*
X0Y5000D02*
Y10000D01*
X10000D01*
Y0D01*
X0D01*
Y5000D01*
X-1000D01*                    1st fully coincident draw
X-5000Y1000D01*
X-9000Y5000D01*
X-5000Y9000D01*
X-1000Y5000D01*
X0D01*                        2nd fully coincident draw
G37*
M02*
```

This creates the following image:
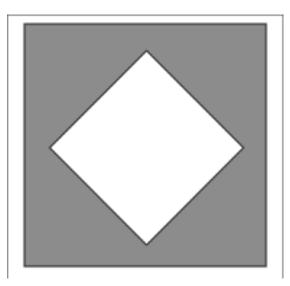
25.   Fully coincident edges in contours, example 1

Example 2

```
G04 ex2: overlapping*
%FSLAX23Y23*%
%MOMM*%
%SRX1Y1I0.000J0.000*%
%ADD10C,1.00000*%
G01*
%LPD*%
G36*
X0Y5000D02*
Y10000D01*
X10000D01*
Y0D01*
X0D01*
Y5000D01*
X1000D01*                    1st fully coincident draw
X5000Y1000D01*
X9000Y5000D01*
X5000Y9000D01*
X1000Y5000D01*
X0D01*                       2nd fully coincident draw
G37*
M02*
```

This creates the following image:

26.   Fully coincident edges in contours, example 2

4.5.12   **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 fully 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*
G01*
Y2720000D01*
X1310000D01*
Y2570000D01*
X1250000D01*
Y2600000D01*
X1290000D01*
Y2640000D01*
X1250000D01*
Y2670000D01*
X1290000D01*
Y2700000D01*
X1250000D01*
Y2670000D01*
Y2640000D01*
Y2600000D01*
Y2570000D01*
X1220000D01*
G37*
```
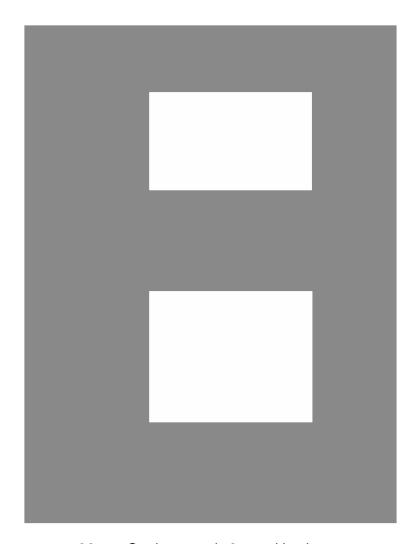
This results in the following contour:

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

This creates the following image:

28.   Cut-in example 2: resulting image

Example 3 attempts to creates 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*
G01*
Y2600000D01*
X1140000D01*
Y2640000D01*
X1110000D01*
Y2670000D01*
X1140000D01*
Y2700000D01*
X1110000D01*
Y2570000D01*
X1090000D01*
Y2720000D01*
X1170000D01*
Y2570000D01*
X1110000D01*
G37*
```

This results in the following contour:

G01X1170000D01*

G01X1110000*D01

G01Y2700000*D01

G01X1140000*D01

G01Y2670000*D01

G01X1110000*D01

G01Y2720000*D01

G01Y2570000*D01

G01Y2570000*D01

G01Y2640000*D01

G01X1140000*D01

G01Y2600000D01*

G01X1090000*D01

G01X1110000*D01

X1110000Y2570000D02*

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 content>*

The <Comment content> must follow the syntax for strings in section 3.3.5.

👁 **Example:**

```
G04 This is a comment*
G04 The characters SP, ',' and ';' are allowed here.*
```

4.7 End-of-file (M02)

The M02 function code indicates the end of the file.

The last data block in a Gerber file must be the M02 – this is mandatory. No data is allowed after an M02.

Gerber readers are encouraged to give an error on a missing M02 as this is an indication that the file has been truncated.

The syntax for M02 is as follows:

<End-of-file>: M02*

4.8 FS – Format Specification

The FS command specifies the format of the coordinate number. It must be used only once at the beginning of a file. It must be specified before the first use of coordinate data. It is recommended to put the FS command at the very first non-comment line.

The FS command specifies the following format characteristics:

❑ Number of integer and decimal places in a coordinate number
❑ Zero omission
❑ Absolute or incremental coordinate notation

4.8.1 Coordinate Format

The coordinate format specifies the number of integer and decimal places in a coordinate number. For example, the "23" format specifies 2 integer and 3 decimal places. A maximum of 6 integer and 6 decimal places can be specified (nnnnnnn.nnnnnn). The same format must be defined for X and Y. Signs are allowed. The '+' sign is optional.

ℹ **Note:** There are a small number of files that have 7 decimal digits. This is invalid but the meaning is quite clear. It may be advisable for readers to handle 7 or more digits. However, writers must not generate more than 6 digits; if this would be necessary for a high resolution application, it must be checked that the reader can handle such files.

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 command. See 4.9.

⚠️ **Warning:** For professional PCB production data 6 decimal places in inch and 5 or 6 decimal places in mm must be used. A lower number can lose vital precision. The option to use less 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 precision. 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 errors.

⚠️ **Warning:** Coordinate numbers are integers. Explicit decimal points are not allowed.

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.

If the coordinate data in the file does not omit zero's leading zero omission must be specified.

At least one character must be output in both modes as omitting a value indicates that the previous value should be used. Zero therefore should be encoded as "0" in both modes.

ⓘ **Note:** It is recommended to *use leading zero omission only*. Leading zero omission is easier to read and trailing zero omission does not offer any real benefits over it. Trailing zero omission will be deprecated in the near future.

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 strongly recommended to *use absolute notation only*. Incremental notation will be deprecated in a future revision of the specification. 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 command is:

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

Syntax	Comments
FS	FS for Format Specification
L	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 numbers. 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 a coordinate number (0 ≤ N ≤ 7) M - number of decimal positions in a coordinate number (0 ≤ M ≤ 7)

4.8.5 **Examples**

Syntax	Comments
%FSLAX25Y25*%	Coordinate numbers have 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 command sets the units used for coordinates and for parameters or modifiers indicating sizes or coordinates. The units can be either inches or millimeters. This command must be used once, at the beginning of the file.

Note: the FS command sets the format (i.e. number of integer and decimal positions) of the coordinate numbers.

4.9.1 **Data Block Format**

The syntax for the MO command is:

Copyright Ucamco NV.

<MO command>: 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 command creates an aperture. It starts with 'AD', followed by

- ❑ 'D' and D-code number (or aperture number)
- ❑ the aperture template name
- ❑ optional modifiers.

As a side effect AD sets the graphics state variable current aperture to the D-code number.

The AD command must precede the first use of the assigned aperture. It is recommended to put all AD commands in the file header.

The allowed range of D-code is from 10 up to 2.147.483.647 (max int 32). The D-codes 0 to 9 are reserved and *cannot* be used for apertures. Once a D-code number is assigned it cannot be re-assigned.

⚠️ **Warning:** The maximum D-code was 999 in older versions of the specification. Use low D-codes for compatibility with legacy readers by preference.

👁️ **Example:**

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

For readability it is recommended to enclose each AD command into a separate pair of '%' characters.

The data block for the AD command is as follows:

<AD command>: ADD<D-code number><Aperture name>[,<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 (≥10)
<Aperture name>[,<Modifiers set>]	The aperture name, optionally followed by modifiers

The **<Aperture name>** can either one of the standard aperture names (C, R ,O or P) or a macro aperture name previously defined by an AM command.

The required modifiers in <Modifiers set> depend on the <Aperture name>. Modifiers are separated by the 'X' character. All sizes are decimal numbers, units follow the MO command. The FS command has no effect on aperture sizes.

4.10.2 **Zero-size apertures**

Zero-size C standard (*circular)* apertures are allowed. No other standard or macro aperture with zero size is allowed, even if the normal shape would be a circle. Aperture size is defined as the size of the effective image represented by the aperture; consequently aperture parameter values resulting in an empty image are not allowed, apart for the zero-size C aperture

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 attached to 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.

4.10.3 **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 macro aperture described by aperture macro CIRC defined previously by an aperture macro (AM) command

4.11 Standard Apertures

4.11.1 Holes in standard apertures

Standard apertures may have a hole in them. When an aperture is flashed only the solid part affects the image, the hole does *not.* Objects under a hole remain visible through the hole. For image generation the area of the hole behaves exactly as the area outside the aperture. The hole is not part of the aperture.

⚠️ **Warning:** Make no mistake: holes do *not* clear the objects under them.

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

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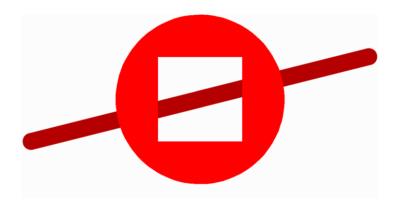
If only the **<X-axis hole size>** modifier is specified the hole is round, and the modifier specifies the diameter. If both X and Y is specified the hole is rectangular and the modifiers specify the X and Y size. If both parameters are omitted the aperture is solid. If present the sizes must be ≥ 0.

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

👁 **Example:**
```
%FSLAX26Y26*%
%MOIN*%
%ADD10C,10X5X5*%
%ADD11C,1*%
G01*
%LPD*%
D11*
X-10000000Y-2500000D02*
X10000000Y2500000D01*
D10*
X0Y0D03*
M02*
```


30.   Standard (circle) aperture with a hole above a track. The track is still visible through the hole.

4.11.2 **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, a decimal ≥0
<Hole>	Optional hole. If no hole is specified the aperture is solid. <Hole> is a decimal number ≥0.

⊚ **Examples:**

These commands define the apertures below

```
%ADD10C,0.5*%
%ADD10C,0.5X0.25*%
%ADD10C,0.5X0.29X0.29*%
```


31.   Circles with different holes

4.11.3   **Rectangle**

The syntax of the rectangle or square standard aperture:
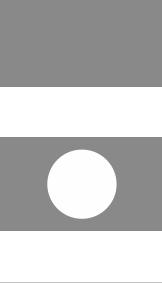
R,<X size>X<Y size>[X<Hole>]

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 Both numbers are decimals.
<Hole>	Optional hole. If no hole is specified the aperture is solid. <Hole> is a decimal number ≥0.

👁 **Examples:**

These commands define the apertures below

```
%ADD22R,0.044X0.025*%
%ADD22R,0.044X0.025X0.019*%
%ADD22R,0.044X0.025X0.024X0.013*%
```


32.   Rectangles with different holes

4.11.4 **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> font<Y size>	X and Y sizes of the obround sides. Both must be decimals > 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. <Hole> is a decimal number ≥0.
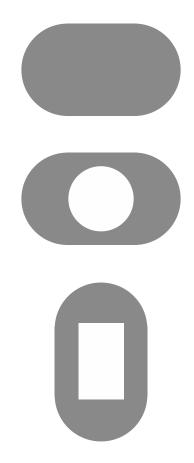
 Example:

These commands define the apertures below

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

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

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

33.   Obrounds with different holes

4.11.5　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 a decimal > 0.
<Number of vertices>	Number of polygon vertices, ranging from 3 to 12
<Degrees of rotation>	A decimal number specifying the 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. <Hole> is a decimal number ≥0.

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

👁 **Examples:**
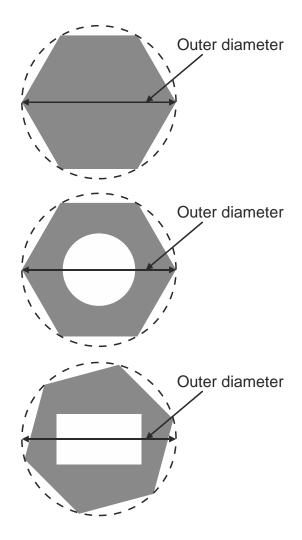
These commands define the apertures below

```
%ADD17P,.040X6*%
%ADD17P,.040X6X0.0X0.019*%
%ADD17P,.040X6X15.0X0.023 X0.013*%
```


34.   Polygons with different holes

4.12 AM - Aperture Macro

The AM command defines macro aperture templates. Apertures of any shape can be created. Multiple shapes called primitives can be combined in a single aperture. An aperture macro can contain variables whose actual values are defined by:

- ❑ Values provided by an AD command referencing the aperture macro
- ❑ Arithmetic expressions with other variables

A macro aperture defined by the AM command can be referenced from AD command in the same way as the standard apertures. A macro aperture must be defined before the first AD that refers to it. The AM command can be used multiple times.

4.12.1   Data Block Format

The syntax for the AM command is:

<AM command>:   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 3.3 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. (K is an integer >0.) An arithmetic expression may use arithmetic operators described later, constants and variables $X where the definition of $X precedes $K.
<Primitive>	A primitive is a basic shape to create the macro. It includes primitive code and modifiers (e.g. diameter for a circle). The modifiers depend on the primitive. All primitives are described in 4.12.3.
<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 command or defined within the AM by some previous <Variable definition>.
<Comment>	Comment does not affect the image.
<Text>	Single-line text string

4.12.2 **Modifiers**

The exposure modifier that can take two values:

- ❏ 0 means exposure is 'off'
- ❏ 1 means exposure is 'on'

Primitives with exposure 'on' create the solid part of the macro aperture. Primitives with exposure 'off' erase the solid part created before in the same macro definition. Exposure off is typically used to create a hole in the aperture – see also 4.11.1. The erasing action of exposure off is limited to the macro definition in which it occurs. When an aperture is flashed, the solid parts affect the underlying graphics objects. The removed parts do *not* affect the underlying graphics objects. Objects under removed parts remain visible.
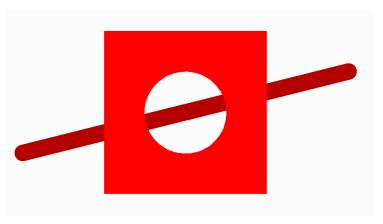
⚠ **Warning:** Make no mistake: primitives with exposure off do *not* clear graphics objects.

A rotation angle is expressed by a decimal number, in degrees. 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.

Coordinates and sizes are expressed by a decimal number in the unit set by the MO command.

```
%FSLAX26Y26*%
%MOIN*%
%AMSQUAREWITHHOLE*
21,1,10,10,0,0,0*
1,0,5,0,0*%
%ADD10MYSPECIALSHAPE*%
%ADD11C,1*%
G01*
%LPD*%
D11*
X-10000000Y-2500000D02*
X10000000Y2500000D01*
D10*
X0Y0D03*
M02*
```


35.   Macro with a hole above a track. The track is still visible through the hole.

4.12.3   **Primitives**

4.12.3.1 Comment, primitive code 0

The comment primitive has no image meaning. It is used to include human-readable comments into the AM command. 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.

```
%AMRECTROUNDCORNERS*
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*
```
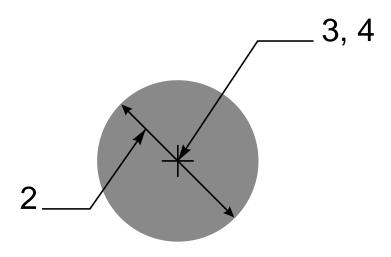
```
0 From now on, use width and height half-sizes. *
$9=$1/2*
$8=$2/2*
0 Add top and bottom rectangles. *
22,1,$1-$3-$3,$3,0-$9+$3-$4,$8-$3-$5,0*
22,1,$1-$3-$3,$3,0-$9+$3-$4,0-$8-$5,0*
0 Add circles at the corners. *
1,1,$3+$3,0-$4+$9-$3,0-$5+$8-$3*
1,1,$3+$3,0-$4-$9+$3,0-$5+$8-$3*
1,1,$3+$3,0-$4-$9+$3,0-$5-$8+$3*
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.
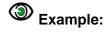
4.12.3.2 Circle, primitive code 1

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

Modifier number	Description
1	Exposure off/on (0/1)
2	Diameter, a decimal ≥ 0.
3	A decimal defining the X coordinate of center position.
4	A decimal defining the Y coordinate of center position.

36.   Circle primitive

Example:

```
%AMCIRCLE*
```

```
1,1,1.5,0,0*%
```
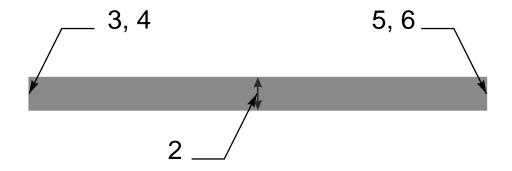
4.12.3.3   Vector Line, primitive code 2 or 20.

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, a decimal ≥ 0.
3	A decimal defining the X coordinate of start point.
4	A decimal defining the Y coordinate of start point.
5	A decimal defining the X coordinate of end point.
6	A decimal defining the Y coordinate of end point.
7	A decimal defining the rotation angle around the *origin* (rotation is *not* around the center of the object)

37.   Line (vector) primitive
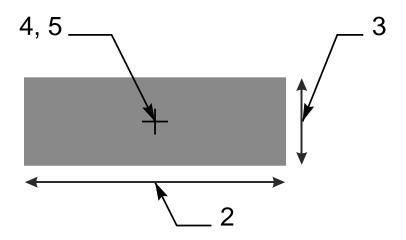
👁 **Example:**

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

Copyright Ucamco NV.

4.12.3.4 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, a decimal ≥ 0.
3	Rectangle height, a decimal ≥ 0.
4	A decimal defining the X coordinate of center point.
5	A decimal defining the Y coordinate of center point.
6	A decimal defining the rotation angle around the *origin* (rotation is *not* around the center of the object)

38.   Line (center) primitive
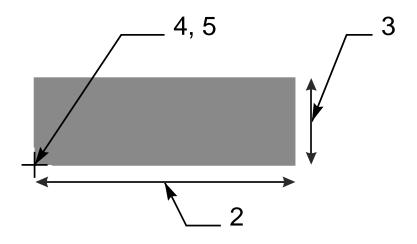
👁 **Example:**

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

4.12.3.5 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, a decimal ≥ 0.
3	Rectangle height, a decimal ≥ 0.
4	A decimal defining the X coordinate of lower left point.
5	A decimal defining the Y coordinate of lower left point.
6	A decimal defining the rotation angle around the *origin* (rotation is *not* around the center of the object)

39.   Line (lower left) primitive

⊙ **Example:**

```
%AMLINE2*22,1,6.8,1.2,0,0,0*%
```

Copyright Ucamco NV.

4.12.3.6 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).
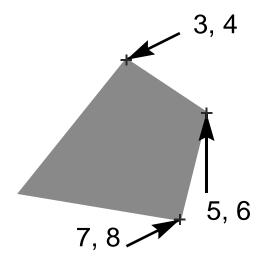
Self-intersecting outlines are *not allowed. S*egments *cannot* cross, overlap or touch except:

1. Consecutive segments connecting in their endpoints, needed to construct the outline.
2. Horizontal or vertical fully coincident segments, used to create holes in a region with cut-ins; see 4.5.9. A pair of segments are said to be fully 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≥1)
3, 4	Decimals specifying the X and Y coordinates of the start point.
5, 6	Decimals specifying the X and Y coordinates of subsequent point number 1.
...	Decimals specifying the X and Y coordinates of further subsequent points.
3+2n, 4+2n	Decimals specifying the X and Y coordinates of subsequent point number n. Must be equal to coordinates of start point.
5+2n	A decimal specifying the rotation angle around the *origin* (rotation is *not* around the center of the object)

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*%
```
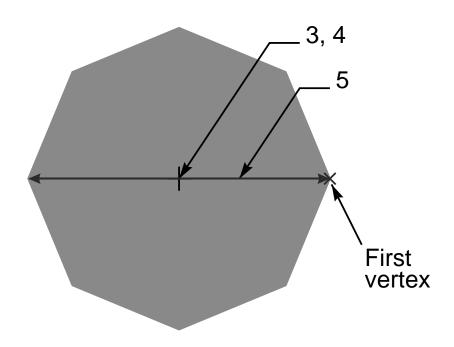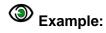
4.12.3.8 Polygon, primitive code 5

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 ≤ n ≤ 12
3	A decimal specifying the X coordinate of center point, conform the decimal syntax defined in section 3.3.2 .
4	A decimal specifying the Y coordinate of center point, conform the decimal syntax defined in section 3.3.2 .
5	A decimal specifying the diameter of the circumscribed circle, ≥ 0
6	A decimal specifying the rotation angle around the *origin.* 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

41.   Polygon primitive

👁 **Example:**
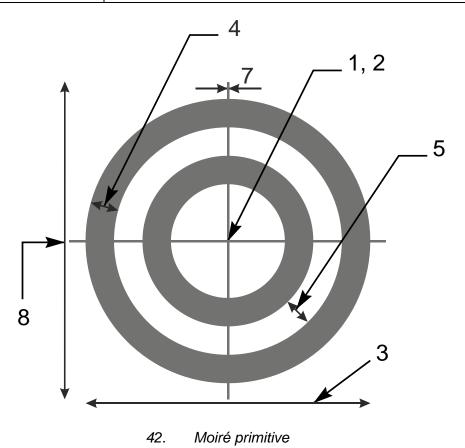
```
%AMPOLYGON*
5,1,8,0,0,8,0*%
```

4.12.3.9 Moiré, primitive code 6

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

Modifier number	Description
1	A decimal defining the X coordinate of center point.
2	A decimal defining the Y coordinate of center point.
3	A decimal defining the outer diameter of outer concentric ring
4	A decimal defining the ring thickness
5	A decimal defining the gap between rings
6	Maximum number of rings
7	A decimal defining the cross hair thickness
8	A decimal defining the cross hair length
9	A decimal defining the rotation angle around the *origin.* Rotation is only allowed if the center point is on the origin.

42.   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:
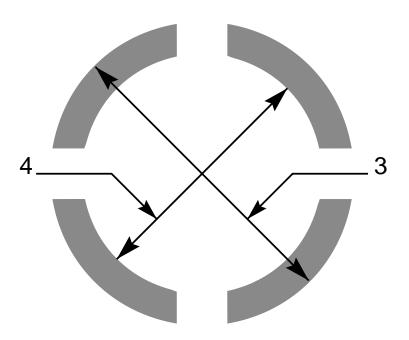
```
%AMMOIRE*
6,0,0,5,0.5,0.5,2,0.1,6,0*%
```

4.12.3.10 Thermal, primitive code 7

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

Modifier number	Description
1	A decimal defining the X coordinate of center point
2	A decimal defining the Y coordinate of center point
3	Outer diameter, must be a decimal and > inner diameter
4	Inner diameter, must be a decimal and ≥ 0
5	Gap thickness, must be a decimal < (outer diameter)/$\sqrt{2}$
6	A decimal defining the 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.

43.   Thermal primitive

Note: If the (gap thickness)*$\sqrt{2}$ ≥ (inner diameter) the inner circle disappears. This is not invalid.

4.12.4　**AM Command Syntax**

An AM command contains the following data blocks:

- ❑ The AM declaration with the macro name
- ❑ Primitives with their comma-separated modifiers
- ❑ Macro variables, defined by an arithmetic expression

Each data block must end with the '*' character.

An aperture macro definition contains the macro name used to identify a macro in an AD command.

An aperture macro definition also contains one or more aperture primitives described in 4.12.3. Each primitive, except the comment, is followed by modifiers setting its position, size, rotation etc. Primitive modifiers can use macro variables. The values for such variables is either provided by an AD command or calculated with arithmetic expression using other variables.

A modifier can be either:

- ❑ A decimal number, such as 0, 2, or 9.05
- ❑ A macro variable
- ❑ An arithmetic expression including numbers and variables

A macro name must comply with the syntax rules in 3.3.4. A macro variable name must be a '$' character followed by an integer >0, for example $12.

Each AM definition must be enclosed into a separate pair of '%' characters. Unlike other commands AM definitions cannot be grouped. Line separators can be added to enhance readability. These line separators do not affect the definition of the macro.

4.12.4.1 Variable values from an AD Command

An AM command can use variables whose actual values are provided by an AD command. Such variables are identified by '$n' where n indicates the serial number of the variable value in the list provided by an AD command. 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 command 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.

4.12.4.2 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 ')'

Note: Older versions of the specification stated that is no unary minus operator. Negative values had to be expressed by subtracting from zero, e.g. '0-$1'. When writing macro's it may be advisable to use this old syntax for compatibility with legacy applications. The current specification includes the unary minus operator, and current readers must support it.

Example:
```
%AMRECT*
21,1,$1,$2-$3-$3,0-$4,0-$5,0*%
```

Corresponding AD command could look like:
```
%ADD146RECT,0.0807087X0.1023622X0.0118110X0.5000000X0.3000000*%
```

4.12.4.3 Definition of a new variable

The AM command allows defining new macro 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
```

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

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

- ❑ All variables used in AM command are initialized to 0
- ❑ If an AD command 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 command; 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 command 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 command contains only 2 modifiers then the value of $3 will remain 0.

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

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

the variable values are calculated as follows: the AD command 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 command 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 command. 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 commands 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 command. 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=$1x0.75*
$5=($2+100)x1.75*
1,0,$4,$5,$3*%


%AMTEST*
$4=$1x0.75*
$5=100+$3*
1,1,$1,$2,$3*
1,0,$4,$2,$5*
$6=$4x0.5*
1,0,$6,$2,$5*%


%AMRECTROUNDCORNERS*
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*%
```

4.12.5 **Examples**

4.12.5.1 *Fixed Modifier Values*

The following AM command 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

An example of an AD command using this aperture macro:

```
%ADD33DONUTFIX*%
```

4.12.5.2 *Variable Modifier Values*

The following AM command 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 command $2 – X coordinate of the center is provided by AD command $3 – Y coordinate of the center is provided by AD command

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

The AD command 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.

4.12.5.3 Definition of a New Variable

The following AM command 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 command
	$2 – X coordinate of the center is provided by AD command
	$3 – Y coordinate of the center is provided by AD command
$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 command (same as in first circle)
	$3 – Y coordinate of the center is provided by AD command (same as in first circle)

The AD command using this aperture macro can look like the following:
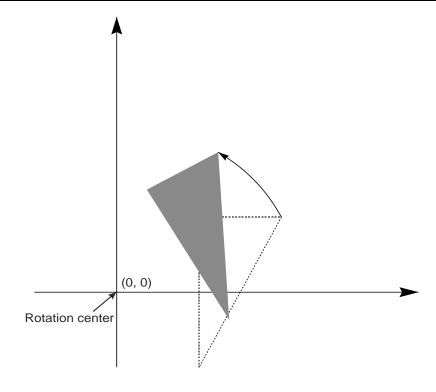
```
%ADD35DONUTCAL,0.020X0X0*%
```

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

4.12.5.4 Rotation Modifier

The following AM command 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

44.   Rotated triangle

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

```
%ADD33AMTRIANGLE_30*%
```
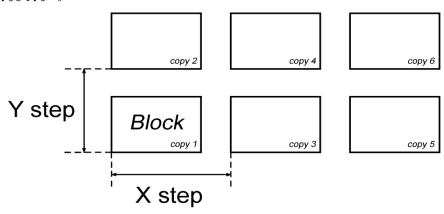
4.13 SR – Step and Repeat

The SR command 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 set; it is cleared by an SR command without repeats with both repeats equal to 1.

In step & repeat mode the graphics objects generated by the command stream of date are collected in a set called a *block* instead of being add the object stream directly. When another SR command is encountered the block is closed. The block is then stepped and repeated (copied) in the image plane according to the SR command starting the block. Each copy of the block contains identical graphics object.

⊚ **Example:**

```
%SRX3Y2I5.0J4.0*%
```


45.   Step and Repeat

The SR command can be used multiple times in a file. The end of the file also closes the block, but it is recommended to explicitly close it with an SR command.

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. If the repeat number is 1 it is recommended to set its step value to 0.

Blocks are copied first in the Y direction and then in the X direction.

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

Note that it is the graphics objects that are repeated, it is not Gerber source is copied and processed several times. The graphics objects in each copy are always identical, even if the graphics state is modified during the processing the source.

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.

⚠ **Warning:** It is recommended not to overlap block repeats containing clear and dark polarity. The effect is not so easy to handle correctly and is probably not always correctly implemented by Gerber readers. Expect errors in readers. (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 this is safe to use.)

4.13.1 **Data Block Format**

The syntax for the SR command is:

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

Syntax	Comments
SR	SR for Step and Repeat
X<Repeats>	Repeats defines the number of times the block is repeated along the X axis. It is an integer ≥ 1.
Y<Repeats>	Repeats defines the number of times the block is repeated along the Y axis. It is an integer ≥ 1.
I<Step>	Steps defines the step distance along the X axis. It is a decimal number ≥ 0, expressed in the unit specified by the MO command.
J<Step>	Steps defines the step distance along the Y axis. . It is a decimal number ≥ 0, expressed in the unit specified by the MO command.

4.13.2 **Examples**

Syntax	Comments
%SRX2Y3I2.0J3.0*%	Repeat the block 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 block 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 block 1 time along the X and Y axes, i.e. block is not repeated.
%SR	Repeats the accumulated blocks and clears $&R mode

4.14   LP – Level Polarity

The LP command starts a new level and sets its polarity to either dark or clear. The level polarity applies to all data following the LP command until superseded by another LP command. This command can be used multiple times in a file. See also 2.7.

An example can be found in 4.5.8.

4.14.1   Data Block Format

The syntax for the LP command is:

<LP command>: LP(C|D)*

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

4.14.2   Examples

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

4.15   Numerical accuracy in image processing and visualisation

The coordinates of all points and all geometric parameters (e.g. a diameter) have an exact numerical value. Graphics objects are therefore in principle defined with infinite precision, with the exception of arcs, which are intrinsically slightly fuzzy, see 4.4.2.) A Gerber file specifies an image with infinite precision.

However Gerber file writers cannot assume that readers will process their files with infinite precision as this is simply impossible. Nemo potest ad impossibile obligari. This raises the question to what a Gerber file reader is held, and what a Gerber writer cán assume.

4.15.1   Visualization

Gerber files are often used to visualize an image on a screen, a photo plotter, a direct imager. Visualization is unavoidably constrained by the limitations of the output device. However, visualization must comply with the following rules:

- Each individual graphics object must be rendered within the stated accuracy of the output device.

- No spurious holes may appear - solid objects must be visualized solid.

- No spurious objects may appear.

- Zero-size objects are not visualized.

- Graphics object can be rendered individually, without looking at other objects; in other words, each graphics object is handled individually, regardless of context.

Note that it is intentionally not specified if rendering must be "fat" or "thin" - fat meaning that features below the device accuracy are blown up to be visible, thin meaning that they disappear.

These rules have a number of noteworthy effects:

- When Gerber objects touch or only marginally overlap a gap may open between the visualized objects.

- When what is intended to be a single object is constructed by combining a number of elementary graphics objects, by painting or stroking, and these elementary objects do no sufficiently overlap, the resulting image may not be solid, it may have internal holes or may even break up in pieces. To avoid these effects the best and most robust approach is not to paint at all: the Gerber format has powerful primitives allowing to create almost any shape with a single graphics object.

- When Gerber objects are separated by a very small gap, the gap may disappear in the visualized image.

- Objects smaller or thinner than the devices resolution can totally disappear.

Construct files robustly.

4.15.2 **Image processing**

Gerber files are also used to transfer PCB design data from CAD to CAM. In CAM the images are subject to complex image processing algorithms: e.g. sophisticated etch compensation, design rule checks and so on. These algorithms perform long sequences of numerical calculations and rounding errors are unavoidably accumulated. This means that all object sizes and positions can move. We call these changes a perturbation. The specification imposes on the reader that the perturbation must be within [-0.5µm, +0.5 µm]. The writer can assume that the perturbation is within this limit.

The perturbation has some noteworthy consequences:

- Contours that are not self-intersecting by a margin of about 1µm can become self-intersecting under a valid perturbation and are therefore invalid. See section 4.5. Avoid such marginal contours like the plague.

- Objects that touch or overlap marginally can become separated under perturbation. This is important for electrical connection. An electrical connection that is realized by touching objects can get separated by a valid perturbation. Such marginal construction can be validly interpreted as either isolating or connecting. Make proper and robust electrical connections, with an overlap of the order of magnitude of at least the minimum conductor width.

If an application requires higher accuracy it must be checked that the applications downstream can handle this. It cannot be blindly assumed.

Construct files robustly.

5 **Attributes**

5.1 Attributes Overview

Attributes add meta-information to a Gerber file. These are akin to labels providing information attached to image files, or features within them. Examples of meta-information conveyed by attributes are:

- *The function of the file.* Is the file the top solder mask, or the bottom copper layer, etc?
- *The part represented by the file.* Does it represent a single PCB, an array, a coupon?
- *The function of a pad.* Is the flash is an SMD pad, or a via pad, or a fiducial, etc.

The attribute syntax provides a flexible and standardized way to add meta-information to the files, independent of the specific semantics or application.

Attributes do not affect the image. A Gerber reader will generate the correct image even if it ignores the attributes. If only the image is needed attributes can simply be ignored.

Each attribute consists of an *attribute name* and an optional *attribute value.* Attribute names must follow the naming syntax in section 3.3.4. Attribute values must follow the string syntax in section 3.3.5. Each attribute is defined by a single command containing a single data block. Put each attribute definition on a separate line for readability.

Example:

> In the following example the command TF defines an attribute with name ".FileFunction" and value "Soldermask,Top". The file represents the top solder mask.
>
> ```
> %TF.FileFunction,Soldermask,Top*%
> ```

There are two types of attributes by association domain:

- ❑ *File attributes* associating metadata with the file as a whole.
- ❑ *Aperture attributes* associating metadata with an aperture. All *graphics objects* inherit the attributes from the current aperture when they are created. (The current aperture is the last aperture selected by a Dnn, with nn≥10).

There are two types of attributes by generality:

- ❑ *Standard attributes.* Standard attribute names, values and semantics are defined in this specification. As they are standardized they can be used to exchange meta-information across all applications.
- ❑ *User attributes.* User attributes can be chosen freely by users to extend the format with meta-information for proprietary workflows. Users must agree on the names, values and semantics. Use these attributes only for unequivocally defined machine-readable information – do not use it for pure comment, the G04 is there for that purpose.

In accordance with the general rule laid out in 3.3.4 standard attribute names must begin with a dot '.' while user attribute names cannot begin with a dot.

⚠ **Warning:** Do not invent your own standard attribute names (names starting with a dot) or your own standard attribute values. This would defeat the purpose of standardization Files with such attributes or values are invalid.

Note that standard attribute values typically contain a value "Other" to cater for requirements not yet foreseen in the specification. The intention is to add new values as the need arises to reduce the use of "Other" over time.

It may of course be that there is a need for standard meta-information for which there is no attribute name or attribute value. Users are encouraged to contact Ucamco at gerber@ucamco.com to request extending the standard attributes where needed. All requests will be investigated. Authors will be properly acknowledged when their suggestions are included in the standard.

Attributes are not needed when the image only needs to be rendered. However, attributes are needed when transferring PCB data from design to fabrication. The PCB fabricator needs to process the images in CAM to prepare it for production, and not just render the image. For example, the fabricator needs to know whether an object is a via pad or a component pad to handle the solder mask properly. The standard attributes transfer this information in an unequivocal and standardized manner. Standard attributes are designed for the PCB CAD to CAM workflow. They convey the design intent from CAD to CAM. This is sometimes rather grandly called "adding intelligence to the image". Without these attributes the fabricator has to reverse engineer the design intent of the features in the file, a time-consuming and error-prone process.

Note that the use of the standard attributes is not "all or nothing". It is possible to use just one attribute, use all of them or use none at all. That said it is strongly recommended to use standard attributes as comprehensively as possible. Attributes provide vital information in a standard way – information that must otherwise be gathered from various documents, unwritten rules, conversations or reverse engineering, with all the risks of error and delay that this entails. Developers of Gerber file output software that cannot provide *all* the attributes or are unsure of their use are encouraged to provide all the attributes they are comfortable with. Partial information is better than no information. For professional PCB production the bare minimum is to set the .FileFunction 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.

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

> **<TFcommand>: TF<AttributeName>[,<AttributeValue>]**

The attribute name must follow the naming syntax in section 3.3.4. The attribute value must follow the string syntax in section 3.3.5.

5.3 Aperture Attributes

5.3.1 Aperture Attributes Overview

Each aperture attribute is attached to an aperture inherited by graphics objects as explained below. The term *aperture attribute* is shorthand for *graphics object attribute defined by aperture.*

The *current attribute dictionary* contains all current aperture attributes. When an AD command defines an aperture, all aperture attributes in the current dictionary are attached to that aperture. The current aperture dictionary is defined after each command in the file according to the following rules:

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

When a graphics object is created it inherits the attributes of the current aperture.

Therefore it is important that objects with different attributes, even if they are the same shape and size, are 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.

A region inherits the attributes of the current aperture, as any other graphics object. Note that the current aperture has no effect on the image of the regions, only on its attributes. Strictly speaking, an aperture created to flash, draw or arcs can be used to carry a region's attributes; we recommend creating dedicated dummy apertures to carry the region's attributes. As the shape of current aperture has no graphical effect on a region we can choose any shape; we recommend to use a zero-size circle standard aperture for dummy apertures.

Associating attributes to graphics objects via their apertures is elegant, compact and efficient

5.3.2 .Aperture Attributes Commands

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

Command	Name	Description
TA	Add aperture attribute	Adds an aperture attribute in the current dictionary.
TD	Delete attribute	Deletes an aperture attribute from the current dictionary.

Aperture attributes
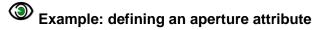
5.3.2.1 Add aperture attribute (TA)

The TA command adds an aperture attribute into the current dictionary. The syntax is the same as for the TF command:

<TA command>: TA<AttributeName>{,<AttributeValue>}

The attribute name must follow the naming syntax in section 3.3.4. 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 a TA command with the same name, but a new value.

The attribute value follows the syntax of strings defined in section 3.3.5.

⊚ **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 TD command deletes an attribute from the current dictionary. Note that the attribute remains attached to apertures and objects to which it was attached before it was deleted.

<TD command>: TD(<AttributeName>)

<AttributeName>: The name of the attribute to delete. If omitted, the whole dictionary is cleared.

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

5.4 **Standard Attributes**

5.4.1 **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.

Standard file attributes

5.4.1.1 .FileFunction

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

Of all the attributes it 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 layer structure. The syntax depends on the file type:

Type	Corresponding position substring
Copper layer	**L1, L2, L3**…to indicate the layer position followed by **Top**, **Inr** or **Bot**. L1 is always the top copper layer. E.g. L2,Inr.
Extra layer, e.g. solder mask	**Top or Bot** – defines the attachment of the layer.
Drill/rout layer	E.g. **1,4** – where 1 is the start and 4 is the end copper layer. The pair 1,4 defines the span of the drill/rout 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.

👁 **Example file function:**

```
%TF.FileFunction,Copper,L1,Top*%
```

The file functions are designed to support all file types in current use. If a type is missing please contact us at gerber@ucamco.com.

The existence of all these file functions does not mean that all types listed should always be included in PCB data sets. Include only the files that are required: no more, no less.

This specification 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 large round hole can be routed.

Each drill span (from-to layers) must be put in a separate Gerber file. Although the PTH and NPTH share the span they must also be split in two separate files. (With NC files the PTH and NPTH drill holes are sometimes lumped together in the same file. With NC files this is bad practice as it is hard to know which is which. With Gerber it is simply not allowed as it is then impossible to specify which holes are plated and which not.)

ℹ **Note:** Some people think drill information cannot be represented in Gerber but must be expressed in an NC format. This is not true – in CAD to CAM workflows Gerber files convey drill information perfectly – drill information 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 fabricator 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 and all the information exactly as needed by the driller. 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 problems. Most importantly, NC formats cannot handle attributes.

.FileFunction value	Usage
`Copper,L<p>,(Top|Inr|Bot)[,<label>]`	A conductor or copper layer. L<p> – where p is an integer - the position of the layer in the stack. The mandatory (Top|Inr|Bot) mark the top, inner or bottom layer. L1 must always be the top layer. The redundancy helps in handling partial data. The label Plane, Signal or Mixed is optional; as these are not exactly defined the file creator adds the label he considers reasonable.
`Soldermask,(Top|Bot)[,<i>]`	The image represents the solder mask *openings.* The optional index is used when there is more than one solder mask on the same side; the index counts from the PCB surface outwards.
`Legend,(Top|Bot)[,<i>]`	A legend is printed on top of the solder mask to show which component goes where. A.k.a. 'silk' or 'silkscreen'.
`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)`	A file containing only the SMD, BGA, component and test pads. Rarely used.
`Scoring,(Top|Bot)`	

`Plated,i,j,<type>[,<label>]`	Plated drill/rout from layer i to layer j. The mandatory type can take the values *PTH, Blind, Buried.*
	The optional label can take the values *Drill, Route, Mixed*; these labels are not exactly defined; the file creator uses the value he reasonably finds appropriate. .
`NonPlated,i,j,<type>[,<label>]`	Non-plated drill/rout from layer i to layer j. The mandatory type can take the values N*PTH, Blind, Buried.*
	The optional label is explained in the row above.
`Profile,(P|NP)`	Profile (outline). P indicates a plated profile (this is exceptional). NP indicates a non-plated profile (this is routine)
`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>`	The value 'Other' is to be used if none of the values above fits. By putting the value 'Other' rather than crudely omitting the attribute it is made explicit that the value is none of the above – an omitted attribute can be one of the above. Certainly do not abuse existing values by horseshoeing a file with a vaguely similar function into that value that does not fit perfectly – keep the identification clean by using 'Other'.
	The mandatory label informally describes 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,Top*%
%TF.FileFunction,Copper,L2,Inr,Plane*%
%TF.FileFunction,Copper,L3,Inr,Plane*%
%TF.FileFunction,Copper,L4,Bot*%
%TF.FileFunction,Soldermask,Bot*%
%TF.FileFunction,NonPlated,1,4,NPTH,Drill*%
%TF.FileFunction,NonPlatd,1,4,NPTH,Route*%
%TF.FileFunction,Plated,1,4,PTH*%
%TF.FileFunction,Profile,NP*%
%TF.FileFunction,Drillmap*%
%TF.FileFunction,Drawing,Stackup*%
```

5.4.1.2 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*%
```

5.4.1.3 .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*
```

5.4.2 **Standard Aperture Attributes**

5.4.2.1 .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.

Sometimes pads are painted, or stroked, rather than flashed. This is bad practice (see 6.2). It is strongly recommended to flash pads, but if you paint pads then at least give the apertures used for painting the function that the painted pad has. For example, if you use aperture 21 to paint SMD pads, then the function of aperture 21 is SMDPad. Similar considerations apply for painted regions. Regions should not be painted but should be defined by their contour, but if you paint regions then set the function of the aperture used for painting to the function of the region.

The possible values for this attribute are defined in the following table.

.AperFunction value(s)	Usage	Applicable File Function
ComponentDrill	A hole for component pins	Plated, NonPlated
ViaDrill,(Filled|NotFilled)	A via hole. Not used for components. The string (Filled|NotFilled) is mandatory. Filled for filled holes, NotFilled for open 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.	
Slot	Used to define PCB slots.	
CutOut	Used to define the PCB cut-outs. Just as the profile, it can be present in all PCB layers.	
Cavity	Used to define Cavities in a PCB.	
ComponentPad	A through-hole component pad. (Not to be used for SMD or BGA.)	Copper
SMDPad,(CuDef|SMDef)	An SMD pad. The label (CuDef|SMDef) is *mandatory*.	

	CuDef stands for *copper defined*; it is by far the most common SMD pad; the copper pad is completely free of solder mask; the area to be covered by solder paste is defined by the copper pad. SMDef stand for *solder mask defined;* the solder mask overlaps the copper pad; the area to be covered by solder paste is defined by the solder mask opening and not by the copper pad. (CuDef is sometimes rather awkwardly called *non solder mask defined.)*
`BGAPad,(CuDef|SMDef)`	A BGA pad. The label (CuDef|SMDef) is *mandatory*. CuDef stands for *copper defined*, SMDef for *solder mask defined*; see SMDPad for clarification.
`HeatsinkPad`	Heat sink pad (typically for SMDs)
`TestPad`	A test pad.
`ConnectorPad`	An edge connector pad. Through-hole or SMD connectors are classified as resp. through-hole or SMD components.
`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. The label (Global|Local) is *mandatory*. Local refers to a component fiducial, Global refers to a fiducial on the entire image or PCB.
`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. Note that setting the AntiPad attribute itself has no effect on the image, and therefore does not turn the pad into LPC as a side effect– this must be done explicitly by an %LPC*% command.	
`OtherPad,<string>`	A pad not specified above. The mandatory string informally describes the type.	
`Conductor,(NotC|ImpC),` `<string>))`	Copper whose function is to connect pads and/or to provide shielding. The label (NotC|ImpC) is mandatory. 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`	The copper border around a production panel.	
`OtherCopper,<string>`	Indicates another function. The mandatory string informally describes the type.	
`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. A Profile cannot have holes. Use the Cut-out attribute for this.	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
`Other,<string>`	The value 'Other' is to be used if none of the values above fits. By putting the value 'Other' rather than crudely omitting the attribute it is made explicit that the value is none of the above – an omitted attribute can be one of the above. Certainly do not abuse existing values by horseshoeing a file with a vaguely similar function into that value that does not fit perfectly – keep the identification clean by using 'Other'. The mandatory label informally describes the aperture function.	

.AperFunction aperture attribute values

5.4.2.2 Drill Tolerancs

.DrillTolerance 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>
```

5.5 Examples

Example 1, simple aperture attribute:

```
G01*
%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:

```
G01*
%TA.AperFunction,SMDPad*%     G04 Adds attribute .AperFunction in the
                             current dictionary with value "SMDPad" to
                             identify SMD pads*
%ADD11…*%                    G04 Ape. 11 gets the .AperFunction,SMDPad
                             attribute*
%TA.AperFunction,Conductor*% G04 Changes the value of .AperFunction
                             attribute to define conductors*
%ADD20…*%                    G04 Ap. 20 gets the .AperFunction,Conductor
                             att.*
%TACustAttr,val*%            G04 Adds attribute "CustAttr" in the current
                             dictionary and sets its value to "val".*
%ADD21…*%                    G04 Ap. 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 attached .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*                        G04 Use aperture 20 for graphical objects &
                            attach it to regions.

X2000Y1500D01*              G04 Draw a conductor*

G36*                        G04 Start a conductive region. (Still
                            attached to aperture 20)*

….

G37*

D21*                        G04 Select aperture 21*

G36*                        G04 Start a conductive region with CustAttr
                            = val.*

….

D22*                        G04 Select aperture 22*

X2000Y2000D03*              G04 A flash with CustAttr = val, but
                            undefined aperture function.*

D23*                        G04 Select aperture 23. *

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. **A G74 or G75 is mandatory if arcs are used.** *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 4.12.3.4.*
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. **The D01/D02/D03 operation code should always be included with the coordinate data.** *See 7.1.*
Objects unexpectedly appear or disappear under holes in standard apertures.	Some CAD systems incorrectly assume the hole in an aperture *clears* (erases) the underlying objects. This is wrong; the hole has no effect on the underlying image. *See **Error! Reference source not found.***
Objects unexpectedly appear or disappear under holes in macro apertures.	Some CAD systems incorrectly assume that exposure off in a macro aperture *clears* (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 4.12.1.*
Openings in areas disappear, typically with clearances in planes	Overlapping segments have been used to construct cut-ins. This is an error. Fully 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 4.11.5.*
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.
The MI command is used to mirror a macro definition but the result is not as expected.	With the MI command mirroring is not applied to aperture definitions. Do not use this confusing and deprecated command. Apply the transformation directly in the aperture definitions and object coordinates. *See 7.3.5.*

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 such as missing clearances.	Use 6 decimal places in imperial and 5 decimal places in metric for PCB manufacturing. 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.	**Use G74** 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*	**Always position arc center points precisely**
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.	**Always use flashed pads.** Define pads, including SMD pads, with the AD and AM commands

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.	**Always use contours (G36/G37) to define areas**
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. See section 4.5.10.	**Construct planes and anti-pads using an LPD layer for the plane and an LPC layer for the holes (anti-pads)**
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.	**Always use Extended Gerber.**

Common poor/good practices

7 Deprecated 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 till any other D code is encountered. In sequences of D01 data blocks this allows omitting an explicit D01 after the first coordinate data block

Example:

```
D10*
X700Y1000D01*
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. The operation mode after a D02, D03 or an aperture selection (Dnn with nn≥10) is undefined. Therefore a file containing coordinates without operation code after a D03 or an aperture selection (Dnn with nn≥10) is invalid.

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

7.2 Open Contours

Previous versions of the specification allowed leaving contours open in a region definition. All open contours are closed by connecting the last point to the first with a straight draw when turning region mode off. Closing the contour does *not* move the current point; the current remains at the last coordinate in the file. Moves (D02) with zero length in open contours are *not* allowed. Contours that become self-intersecting after the automatic closure are not allowed.

In all but the simplest situations open contours can be ambiguous. Therefore open contours are deprecated.

⚠️ **Warning:** Avoid open contours like the plague. The risk of using them lies solely with the writer of the file.

7.3 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 command. 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 command. 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 parameter codes. They are explained later in this chapter.

Command	Function	Description	Comments
AS	Axis Select	Sets the 'Axes correspondence' graphics state variable	These commands 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 parameter codes

The order of execution of these commands 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 parameter codes.

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

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 command	Yes	AXBY
Image mirroring	See MI command	Yes	A0B0
Image offset	See OF command	Yes	A0B0
Image polarity	POS, NEG See IP command	Fixed	Positive
Image rotation	0°, 90°, 180°, 270° See IR command	Yes	0°
Scale factor	See SF command	Yes	A1B1

Deprecated graphics state variables

7.3.1 **AS – Axis Select**

The AS command 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 command affects the entire image. It can only be used once, at the beginning of the file.

7.3.1.1 Data Block Format

The syntax for the AS command is:

<AS command>: 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

7.3.1.2 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

7.3.2 **IN - Image Name**

The IN command is deprecated. Use attributes to provide meta information about the file, see chapter 4.15.

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 3.3.5. This command can only be used once, at the beginning of the file.

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

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.

7.3.2.1 *Data Block Format*

The syntax for the IN command is:

<IN command>: IN<Name>*

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

7.3.2.2 *Examples*

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

7.3.3 **IP – Image Polarity**

The IP command is deprecated.

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

7.3.3.1 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.)

7.3.3.2 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.

In negative image polarity the first graphics object generated must have dark polarity, and therefore clears the dark background.

7.3.3.3 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 command is now deprecated.

7.3.3.4 Data Block Format

The syntax for the IP command is:

<IP command>: IP(POS|NEG)*

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

7.3.3.5 Examples

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

7.3.4 **IR – Image Rotation**

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

7.3.4.1 Data Block Format

The syntax for the **IR command** is:

<IR command>: 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

7.3.4.2 Examples

Syntax	Comments

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

7.3.5 **LN – Level Name**

The LN command 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 3.3.5. This command can be used multiple times in a file.

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

7.3.5.1 *Data Block Format*

The syntax for the LN command is:

<LN command>: LN<Name>*

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

7.3.5.2 *Examples*

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

7.3.6 **MI – Mirror Image**

The MI command 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 macro apertures!**

This command 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 command. Avoid it like the plague. The exception for macro apertures is confusing and leads to mistakes. If an image must be mirrored, write out the mirrored coordinates and apertures.

7.3.6.1 Data Block Format

The syntax for the MI command is:

<MI command>: 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

7.3.6.2 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

7.3.7 **OF - Offset**

The OF command 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 command 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 command.

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

7.3.7.1 Data Block Format

The syntax for the OF command is:

<OF command>: 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.

7.3.7.2 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

7.3.8 **SF – Scale Factor**

The SF command 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 numbers are multiplied by the specified factor value for the corresponding axis. Note that apertures are *not* scaled.

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

7.3.8.1 Data Block Format

The syntax for the SF command is:

<SF command>: 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 ≤ n ≤ 999.99999

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

7.3.8.2 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

7.4 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
- does not support any parameter codes

Consequently, Standard Gerber does not allow defining the coordinate format and aperture shapes. It is incomplete as an image description format. It lacks the imaging primitives needed to unequivocally transfer information from PCB CAD to CAM

7.4.1 Standard Gerber must no longer be used

Standard Gerber is technically obsolete, deprecated and superseded by RS-274X.

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. 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 was not designed for automatic processing. There is no reason why anyone in his right mind would use Standard Gerber rather than Extended Gerber.

Always use Extended Gerber (RS-274X) and to never use Standard 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.