Ucamco
Former Barco ETS

The Gerber File Format Specification

A format developed by Ucamco

Revision 2016.04

Table of 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 fabrication 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 fabrication. A well-constructed Gerber file precisely defines the PCB image and the function of the objects, resulting in a reliable and productive transfer of PCB fabrication data from design to fabrication.

Ucamco continuously clarifies this document based on input from the field and adapts it to current needs. Ucamco thanks the individuals that help us with comments, criticism and suggestions. We urge Gerber software developers to monitor these revisions.

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 It is revoked and superseded by RS-274X.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

Unfortunately, some applications stubbornly continue to use painting (aka stroking) to create pads and copper pours. While not technically invalid painted files require more manual work and increase the risk of errors in fabrication. Painting is a relic of the days of vector photoplotters, devices as obsolete as the electrical typewriter. The rationale for painting disappeared decades ago, its disadvantages remain. We urge all users and developers to help to banish painting from our industry.

Although other data formats have appeared they have not displaced Gerber. The reason is simple. The problems in PCB fabrication data transfer are not limitations in the Gerber file format but poor practices and poor implementations. The main culprits are painting and the use of Standard Gerber. To quote a PCB fabricator: “If we would only receive proper Gerber files, it would be a perfect world.” The new formats are more complex and less transparent to the user. Poor practices in unfamiliar and more complex formats are more damaging than in a well-known format. The new formats are sometimes promoted by pointing to Gerber files with syntactic or semantic errors. This is of course a fallacy: the solution to bugs is to fix them and not to leap to 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 fabrication data 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 fabrication industry. Ucamco thanks these dedicated individuals.

Karel Tavernier

Managing Director,
Ucamco

1 Introduction

1.1 About this document

1.1.1 Scope and Target Audience

This document specifies the Gerber file format, a vector format for representing a 2D binary (represented by two colors) 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.

1.1.2 Questions and 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.

1.1.3 References

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

https://en.wikipedia.org/wiki/MD5

https://en.wikipedia.org/wiki/Unicode

http://en.wikipedia.org/wiki/ISO_8601

1.2 Copyright and Intellectual Property

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

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

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

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

1.3 History of the Gerber File Format

1.3.1 **Standard Gerber**

The Gerber file format derives its name from the former Gerber Systems Corp. A leading supplier of vector photoplotters from the 1960s onwards, Gerber based its plotter input on a subset of the EIA RS-274-D NC format, and in1980, it published a well-written specification titled “Gerber Format: a subset of EIA RS-274-D; plot data format reference book”. The format was so well suited for its task that it was widely adopted and became the de-facto standard format for vector plotters, known as Standard Gerber.

1.3.2 **Extended Gerber**

Vector photoplotters are NC machines, and Standard Gerber, which is dedicated to vector photoplotters, is an NC format. As of the 1980s, vector photoplotters started losing ground to raster plotters. Based on bitmap technology, these newer devices demanded rather more than a simple NC format, so Gerber extended the original NC format with so called “Mass Parameters”, converting it to a fully-fledged image file formats. This resulted in a family of effective image description formats designed specifically to drive Gerber's PCB devices and raster plotters. In 1998 Gerber Systems Corp. was taken over by Barco and incorporated into its PCB division – Barco ETS, now Ucamco. At this point, Barco drew all the variants in Gerber's family of formats into a single standard image format.

On September 21, 1998 Barco-Gerber published the Gerber RS-274X Format User's Guide. The format became known as Extended Gerber or GerberX. This is a full image description format, which means that an Extended Gerber file contains the complete description of a PCB layer, providing everything needed for an operator to generate a PCB image, and enabling any aperture shape to be defined. Unlike Standard Gerber, it does not need the support of additional external files, and it specifies planes and pads clearly and simply without the need for painting or vector-fill. The Extended Gerber format quickly superseded Standard Gerber as the de facto standard for PCB image data, and is sometimes called "the backbone of the electronics industry". A sequence of revisions clarifying the specification was published over the years, culminating in revision H of January 2012.

1.3.3 **The Great Reform**

During the course of 2012, Ucamco reviewed the entire format in depth. A large number of historic format elements that were rarely, if ever, used were deprecated. Light was shone into the most remote corners of the format. Where there were conflicting interpretations of any part of the format in the market, that part was either deprecated or its interpretation was clarified. As part of this important work, over 10.000 files from all over the world were gathered into a representative library to help establish current practice. The result was a powerfully clear and simple format, without needless embellishments, focused on the current needs of the PCB industry.

The specification document itself was re-organized, the quality of the text and the drawings improved and many new drawings were added. Deprecated elements were relegated to a separate chapter.

This resulted in The Gerber Format Specification, revision I1 published in December 2012. Revisions I2, I3 and finally I4 from November 2013 further improved the document. The backbone of the electronics industry was now supported by a first-rate up-to-date specification document and was ready for the next step.

This version of the Gerber Format was developed by Karel Tavernier and Rik Breemeersch. They were assisted by an advisory group including Ludek Brukner, Artem Kostyukovich, Jiri Martinek, Adam Newington, Denis Morin, Karel Langhout and Dirk Leroy. Grateful thanks are extended to all those who helped the development of the revision by posting their questions, remarks and suggestions on the Ucamco website. Particular thanks to Paul Wells-Edwards whose insightful comments contributed substantially to the revision.

1.3.4 **The Second Extension**

Until this point, Gerber was purely an image description format. Recognizing that a PCB image must be supported with meta-information that describes, say, the function of an image file in the layer structure, Ucamco realized that it could convey that information clearly and unequivocally using attributes. Accordingly, and in June 2013, the company publicly proposed to extend the Gerber format using attributes, and invited feedback on its proposal from the Gerber user community.

The outcome of this was revision J1, completed in February 2014, during which Gerber got its attributes. It was a major step forward for the format, at least on a par with the changes made when Standard Gerber became Extended Gerber. Sometimes called the second extension, the latest version of the Gerber format is known as Gerber version 2, or X2 (as opposed to X1, which is Gerber without attributes). Gerber version 2 is fully backward compatible as attributes do not affect the image at all. Subsequent revisions, J2 to J4, clarified the specification and added new standard attributes.

Gerber version 2 was developed by Karel Tavernier, Ludek Brukner and Thomas Weyn. They were assisted by an advisory group including Roland Polliger, Luc Samyn, Wim De Greve, Dirk Leroy and Rik Breemeersch.

1.3.5 **Standard Gerber Revoked**

In September 2014 Ucamco published an open letter declaring Standard Gerber obsolete and revoking its specification. 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.

1.3.6 **Further Clarification**

Early in 2015, the entire specification was reviewed once again by Karel Tavernier, Thomas Weyn and Artem Kostyukovich whose focus centered on making the specification easier to read and understand, while taking great care to ensure consistent and precise terminology. Some further elements were identified as superfluous, so they were deprecated, further simplifying the format, and a number of overview tables and charts were added. Not least, special attention was given to the 'Overview' chapter, with the aim of turning it into a tutorial that can be understood by non-experts. The result of this work is revision 2015.06, published in June 2015.

1.4 Record of Revisions

1.4.1 **Revision 2016.04**

Added PressFit label to component drill and pad attributes; see ComponentPad and ComponentDrill. Revoked default on current point.

Text improvements that do not change the format: Removed superfluous concept of level and replaced ‘Level Polarity’ by ‘Load Polarity. Various others.

1.4.2 **Revision 2016.01**

Added drill and pad functions for castellated holes. Added optional types break-out and tooling on MechanicalDrill.

Deprecated closing an SR with the M02.

Text improvements that do not change the format: Clarified .AperFunction attribute values. Clarified when to use of standard or user attributes. Clarified how aperture attributes can be set on regions.

1.4.3 **Revision 2015.10**

Added items to section Errors and Bad Practices.

Added file function attribute .FilePolarity.

Refined drawing .FileFunction attributes Replaced Mechanical by FabricationDrawing and Assembly by AssemblyDrawing. Added definitions to the drawing types. Added mandatory (Top|Bot) to .AssemblyDrawing, as suggested by Malcolm Lear. Added ArrayDrawing.

1.4.4 **Revision 2015.07**

The superfluous and rarely, if ever, used macro primitives 2 and 22 were revoked. The .AperFunction aperture attribute was simplified:

- Filled / NotFilled option is removed for the ViaDrill function
- ImpC / NotC option is removed from the Conductor function

1.4.5 **Revision 2015.06**

The entire document was revised for clarity. The readability of the text was improved. The terminology was made consistent. The glossary was expanded. A number of additional images were added, including the Gerber file processing diagrams, command types diagram, aperture macro rotation illustration. Some of existing images were recreated to improve the quality. Several new tables were added to explain the relation between D code commands and graphics state parameters. The glossary was updated. The sections were rearranged. Several new sections (2.2, 2.6, 2.11, 4.3, 7.2, 7.4) and subsections (4.4, 4.5, 4.6, 5) were added.

The usage of G codes in a data block together with D codes was deprecated. The rectangular hole in standard apertures was deprecated. Usage of less than 4 decimal positions and trailing zero omission in the FS command was deprecated

The number after D/G/M letter in function code commands was allowed to contain more leading zeros. The mistakenly omitted rotation parameter of the circle macro primitive was restored. Unicode escape sequences in strings are now possible.

New file attributes were specified: *.GenerationSoftware* (5.4.1.4), *.CreationDate* (5.4.1.5) and *.ProjectId* (5.4.1.6).

As of now the revision numbering follows the yyyy.mm versioning scheme.

1.4.6 **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.

1.4.7 **Revision J3**

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

1.4.8 **Revision J2**

Associating aperture attributes with regions was much simplified. A section about numerical accuracy was added.

1.4.9 **Revision J1**

This revision created version 2 of Gerber format by adding attributes to what was hitherto a pure image format. See chapter 5. A 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.

1.4.10 **Revision I4**

The commands LN, IN and IP were deprecated. The possibility of re-assigning D codes was revoked.

The regions overview section 4.6.1 was expanded and examples were added different places in 4.6 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 term ‘(mass) parameter’ was replaced by ‘extended code’.

1.4.11 **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.

1.4.12 **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.

1.4.13 **Revision I1**

General. The entire specification was extensively reviewed for clarity. The document was re-organized, 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 focused. *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 2.8 and 2.7. *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 2.8. *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 4.13.3. 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 4.6. *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. Coordinate data without D01/D02/D03 in the same data block create some confusion. It therefore has been deprecated. See 7.2. *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 concept called a layer. These were often confused with PCB layers and have been renamed as levels. This is purely narrative and does not alter the Gerber files.

1.5 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.

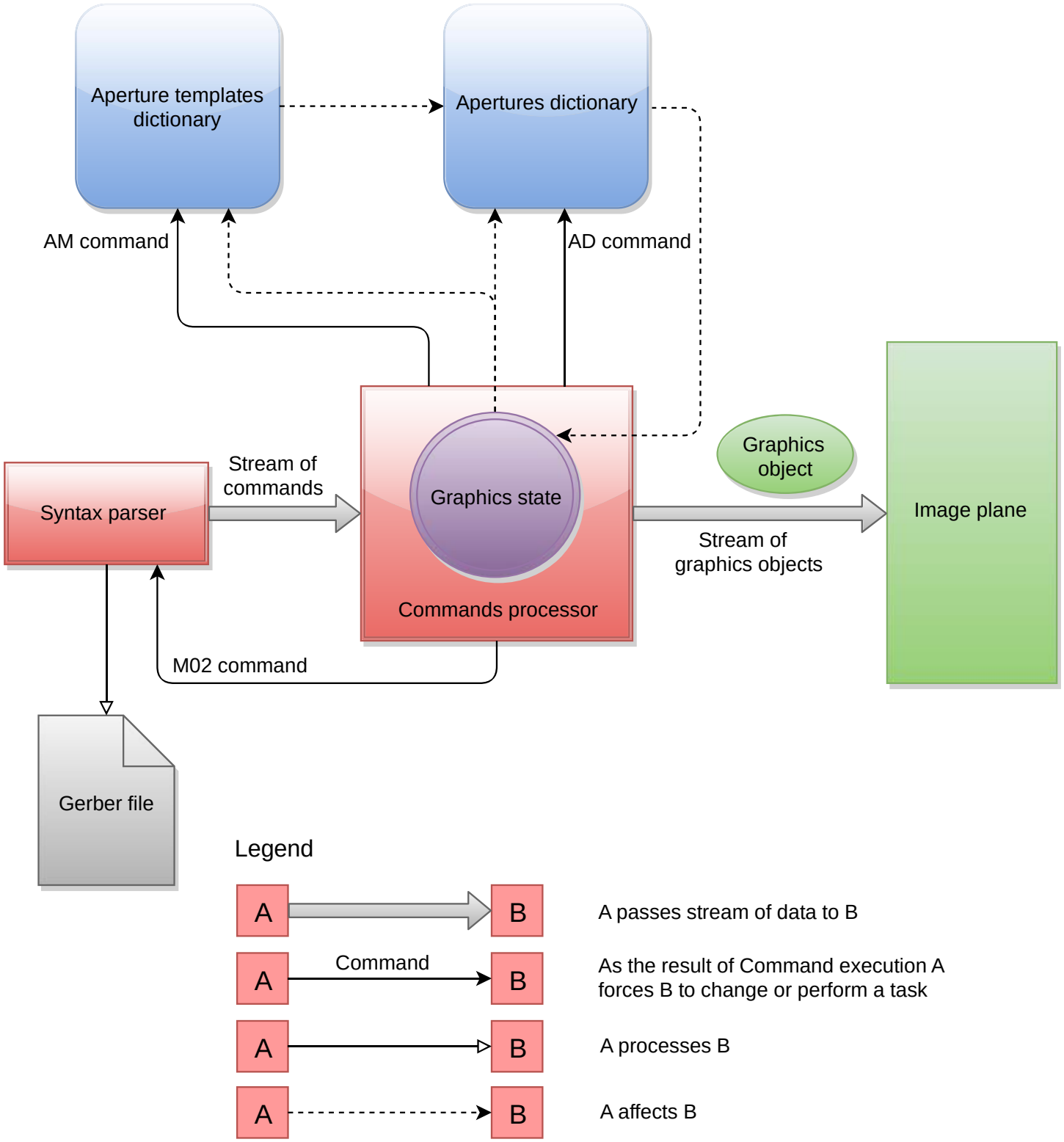
A single Gerber file specifies a single image. A Gerber file is complete: it does not need external files or parameters to be interpreted. One Gerber file represents one image. One image needs only one file.

A Gerber file is a stream of *commands.* A command is either a *function code* (see 3.5.2) or an *extended code* (see 3.5.3). Some commands control the *graphics state* (see 2.8), other commands generate a stream of graphics objects (see 2.3) which combined produce the final image. The graphics state determines how the operations create the graphics objects.

A Gerber file can be processed in a single pass. Names, parameters and objects must be defined *before* they are used.

2.2 Processing a Gerber File

The image below illustrates how a Gerber file is processed.



Having a question or remark about the spec? Please contact us at gerber@ucamco.com

1. Gerber file processing diagram

A Gerber file is the input for the syntax parser. The parser reads the file and produces the stream of commands for the commands processor. The commands processor is responsible for handling the stream of commands and as the result it generates the stream of graphics objects. All the created objects are superimposed on the image plane in order of their creation.

An important part of the commands processor is the graphics state. This is the internal object manipulated by the processor itself. The processor executes the commands which explicitly or implicitly change the graphics state (see 2.8). The graphics state holds the set of parameters which influence the operation codes (see 2.7) and thus define the resulting graphics objects.

The AM command (see 4.13) results in creating a macro aperture template (see 2.4). The template is generated by the commands processor and added to the aperture templates dictionary. The dictionary is responsible for holding all the templates available for an aperture instantiation. This includes standard (built-in) aperture templates (see 4.12) which are automatically added to the dictionary before file processing is started, and also macro aperture templates specified by AM commands in the file being processed. The templates are then used by AD command for instantiating apertures – this is how the aperture templates dictionary affects the apertures dictionary.

When the commands processor executes an AD command (see 4.11) it creates an aperture that is added into the apertures dictionary. The aperture is created using an aperture template from the templates dictionary.

The graphics state has the ‘current aperture’ parameter that is manipulated by Dnn command (see 4.3). When the processor executes a Dnn command a referenced aperture from apertures dictionary is set as the current aperture.

The graphics state also affects the generation of aperture templates and apertures: the templates and apertures depend on ‘coordinate format’ and ‘unit’ graphics state parameters (see 2.8).

The graphics object stream is without state. Objects are superimposed as they are, in their order of appearance.

After processing the M02 command (see 4.8) the processor interrupts the syntax parser and stops the graphics objects generation.

The image from above illustrates the processing of a Gerber file without attributes. For the complete schema see 2.11.

2.3 Graphics Objects

A Gerber file creates an ordered stream of graphics objects. A graphics object is a visual object of a certain type that is represented by an image in the plane. It has a shape, a size, a position and a polarity (dark or clear). The stream of the graphics objects forms the final image.

There are four types of graphics objects:

- A draw is a straight line segment, stroked with the current aperture. It has a thickness. The line endings depend on the current aperture: line endings are round for circle apertures and square or triangle for square apertures (see 2.5).
- An arc is circular segment, stroked with the current aperture. An arc has a thickness. Line endings are always round as only stroking with a circle is allowed (see 2.5).
- A flash is a replication of an aperture at a given location. An aperture is a basic geometric shape defined earlier in the file. An aperture is typically flashed many times. Any valid aperture can be flashed (see 4.2.4).
- A region is an area defined by its contour (see 4.6.1). A contour is constructed with a sequence of linear and circular segments (see 2.6).

Note: A *track* is a generic name for a graphics object that can either be a draw or an arc.

In PCB copper layers, draws and arcs are used to create conductive tracks, flashes to create pads and regions to create copper areas (also called copper pours).

2.4 Apertures

An aperture is a 2D geometric shape or figure, for example a circle with a diameter of 2mm. Apertures are used for flashing or stroking (see 2.5 and 4.2).

The AD (Aperture Define) command creates an aperture based on an aperture template by providing values for template parameters defining shape and size. It also assigns the D code or aperture number to identify the aperture for later use in the command stream.

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 4.12).
- Macro apertures are defined by means of AM (Aperture Macro) command. Any shape and parametrization can be created. They are identified by their given name. (See 4.13).

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

Standard apertures can be considered as built-in macro apertures with the center as origin.

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 aperture types can be created without extending the format and updating implementations.

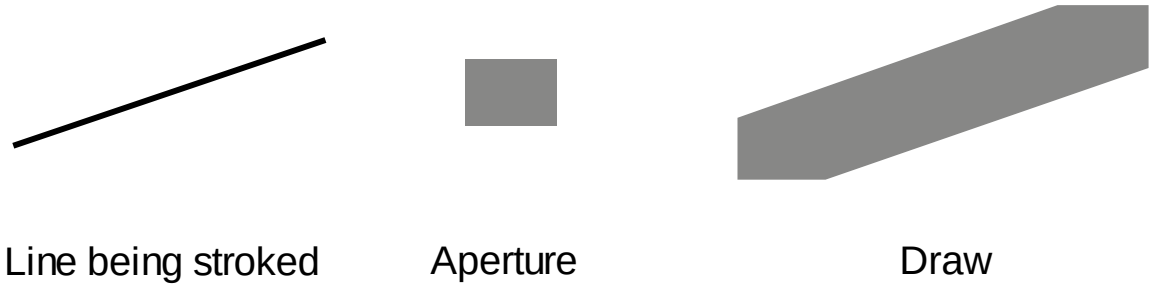
2.5 Draw and Arc Objects

A *draw object* is created by a command with D01 code in linear interpolation mode. The command results in 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.

If the rectangle aperture is aligned with the line being stroked the result is a draw with line endings which have right angles:

2. Creating a draw: the aperture is aligned with line

If the rectangle is not aligned the result is as in the illustration below. The rectangle is *not* automatically rotated to align with the line being stroked.



3. Creating a draw: the aperture is not aligned with line

The solid circle and the solid rectangle *standard* apertures are the only apertures allowed for creating *draw* objects. Neither other standard apertures nor any macro apertures can be used to create a draw, even if their effective shape is circle or a rectangle.

An *arc* object is created by a command with D01 code in circular interpolation mode. In this case the command results in 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* object cannot be created using a rectangle or any other aperture.

A circle aperture with diameter zero can be used for creating a draw or an arc. It creates 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 a replica of the aperture. This is also the limiting image of a draw/arc when the draw/arc length approaches zero. Thus the image is what is expected if a small draw/arc is accidentally rounded to a zero-length draw/arc. Although the image is coincidentally identical to a *flash* of the same aperture the resulting graphics object is not a flash but a draw/arc.

Note: Do not use zero-length draws to represent pads as pads must be represented by flashes.

2.6 Contours

A contour is an important concept in the Gerber format description. Contours are used to create regions (see 4.6) and outline primitives in macro apertures (see 4.13.4.5).

A contour is a sequence of connected linear or circular 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.

In Gerber format *self-intersecting contours are not allowed*. Segments *cannot* cross, overlap or touch except:

- Consecutive segments connecting in their endpoints, needed to construct the contour
- Horizontal or vertical fully coincident linear segments, used to create holes in a region with cut-ins; see 4.6.11. A pair of linear 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.
- Zero-length linear and circular segments 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 linear segments (linear segments not sharing both vertices), diagonal fully coincident linear segments, fully coincident circular segments, partially coinciding circular segments, circular segments tangent to another segment of any form, vertices on a segment but not on its endpoints, vertices with more than two segments, full 360° circular segments.

If a contour violates any of the restrictions above it is considered invalid and thus the whole file is invalid.

The segments are not graphics objects in themselves; segments are part of the contour which is then used either to create a region graphics object, or to specify an outline aperture. The segments have no thickness.

 **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 circular segments, which are intrinsically fuzzy. Construct contours defensively. Observe sufficient clearances between the circular segments. 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 4.9.

 **Warning:** A circular segment can be validly interpreted by any curve within a range, see **Error! Reference source not found.**. If any of these curves results in a self-intersecting c ontour the file is invalid and the result is unpredictable.

2.7 Operation Codes

D01, D02 and D03 are the *operation codes*. An *operation* contains the coordinate data followed by a single operation code: each operation code is associated with a single coordinate pair and vice versa. Operations create the graphics objects and/or change the current point by operating on the coordinate data.

Example:

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

The operation codes have the following effect.

- D02 moves the current point (see 2.8) to the coordinate pair. No graphics object is created.
- D01 creates a straight or circular line segment by interpolating from the current point to the coordinate pair. When region mode (see 2.8) is off these segments are converted to draw or arc objects by stroking them with the current aperture (see 2.5). When region mode is on these segments form a contour defining a region (see 4.6).
- D03 creates a flash object by replicating (flashing) the current aperture. The origin of the current aperture is positioned at the specified coordinate pair.

Only D01 and D03 operation codes result in a graphics object creation. The effect of the operation codes depends on the graphics state (see 2.8).

2.8 Graphics State

A Gerber file defines a graphics state after each command. The graphics state is a set of parameters that determine the effect of the upcoming operation codes (see 2.7). A graphics state parameter that affects an operation code must be defined before the operation code is issued.

The most important graphics state parameter is the *current point*. The current point is a point in the image plane that is set by any operation (D01, D02, D03: *after* performing an operation the current point is set to the coordinates in that operation command.

All other graphics state parameters are set explicitly by corresponding commands. Their values remain constant until explicitly changed.

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

Graphics state parameter	Value range	Fixed or changeable	Initial value
Coordinate format	See FS command in 4.9	Fixed	Undefined
Unit	Inch or mm See MO command in 4.10	Fixed	Undefined
Current point	Point in plane	Changeable	Undefined
Interpolation mode	Linear, clockwise circular, counterclockwise circular See G01/G02/G03 commands in 4.4 and 4.5	Changeable	Undefined
Quadrant mode	Single-, multi-quadrant See G74/G75 commands in 4.5	Changeable	Undefined
Current aperture	Standard or macro aperture See AD in 4.11 and AM in 4.13	Changeable	Undefined
Polarity mode	Dark, clear See LP command in 4.14	Changeable	Dark
Region mode	On/Off See G36, G37 commands in 4.6	Changeable	Off

Graphics state parameters

The graphics state determines the effect of an operation. If a parameter is undefined when it is required to perform an operation the Gerber file is *invalid*. If a graphics state parameter is not needed then it can remain undefined. For example, if the interpolation mode has been set by G02 or G03 code command (circular interpolation) the quadrant mode is required to perform a D01 code operation and thus must be defined; if the interpolation mode has been set by G01

code command (linear interpolation) then the quadrant mode is not needed and may remain undefined.

The relevance of the graphics state parameters for the operations is represented in the table below.

Graphics state parameter	Operation codes		
	D01	D02	D03
Coordinate format	Yes	Yes	Yes
Unit	Yes	Yes	Yes
Current aperture	Yes if Region mode off No if Region mode on	No	Yes
Quadrant mode	Yes if interpolation mode is clockwise or counterclockwise circular interpolation No if interpolation mode is linear	No	No
Interpolation mode	Yes	No	No
Current point	Yes (interpolation starting point)	No	No
Polarity	Yes	No	Yes
Region mode	Yes	Yes. In region mode D02 has the side effect of closing the current contour	Not used by the operation if region mode is off Not allowed if region mode is on

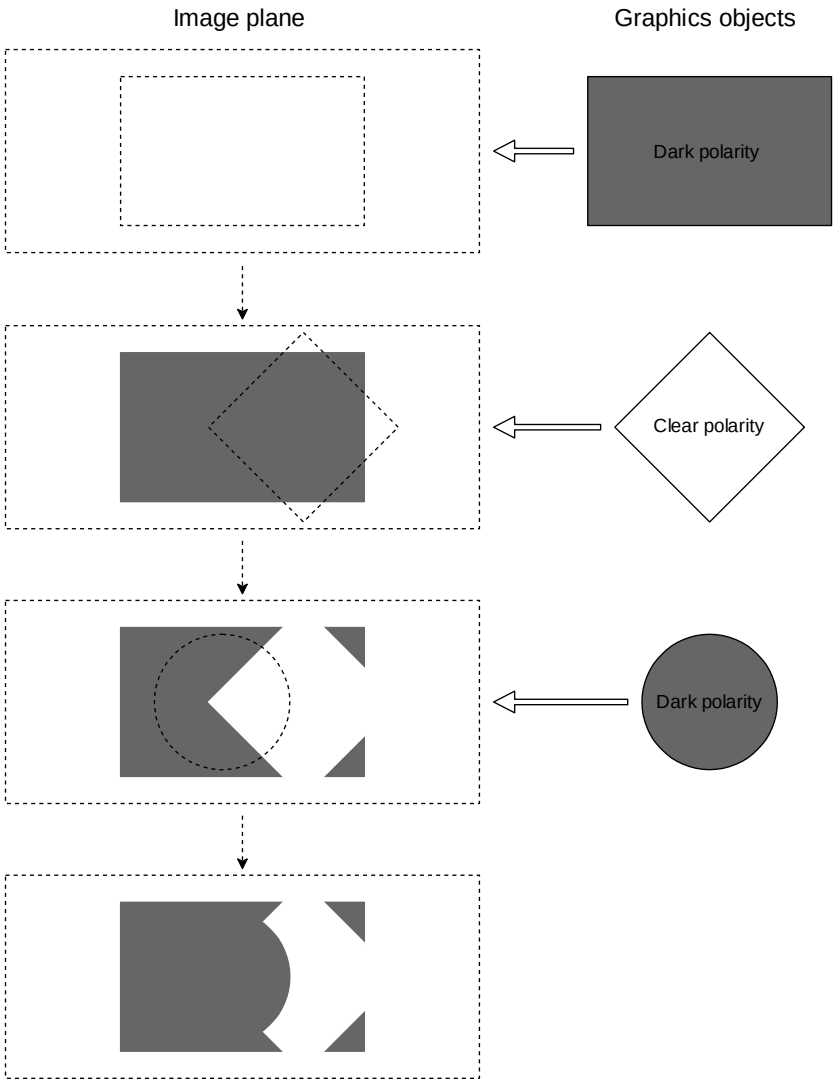
Relevance of graphics state parameters for operation codes

If a table cell contains 'Yes' it means the graphics state parameter is relevant for the corresponding operation. Thus the graphics state parameter must be defined before the operation code is used in the file. If the parameter does not have an automatically assigned initial value it must be explicitly set by the corresponding command.

The current aperture is not needed for image generation in region mode. However it can be used to attach attributes to a region. If the current aperture is undefined, no attributes are attached to the region. The region takes the attribute of the current aperture.

2.9 Dark and Clear Polarity

The final image of the Gerber file is created by superimposing the objects in the order of their creation. Objects have a polarity, either clear or dark. Objects can overlap. A dark polarity object darkens its image in the plane. A clear polarity object clears its image in *all objects beneath it (generated before)*. Subsequent dark objects may again darken the cleared area. See illustration below. Another example is in 4.6.10.



4. Superimposing objects with dark and clear polarities

An object is totally dark or totally clear. It cannot be partially dark and partially clear.

The *order* of superimposed objects with different polarities affects the final image.

The LP command sets the polarity mode, a graphics state parameter (see 4.14). Objects that are created when the polarity mode is dark are dark; when the mode is clear the objects are clear.

2.10 Attributes

Attributes add meta-information to a Gerber file. These are akin to labels providing additional information about the file or features within them. Examples of such meta-information conveyed by attributes are:

- The function of the file: the file is the top solder mask, or the bottom copper layer etc.
- The part represented by the file: the file represents a single PCB, an array, or a coupon
- The function of a pad: the flash is an SMD pad, or a via pad, or a fiducial, etc.

The command below defines an attribute indicating that the file represents the top solder mask.

Example:

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

Attributes do not affect the image. If only the image is needed the Gerber reader can safely ignore the attributes.

Attributes can be associated with either the file as a whole or with individual apertures.

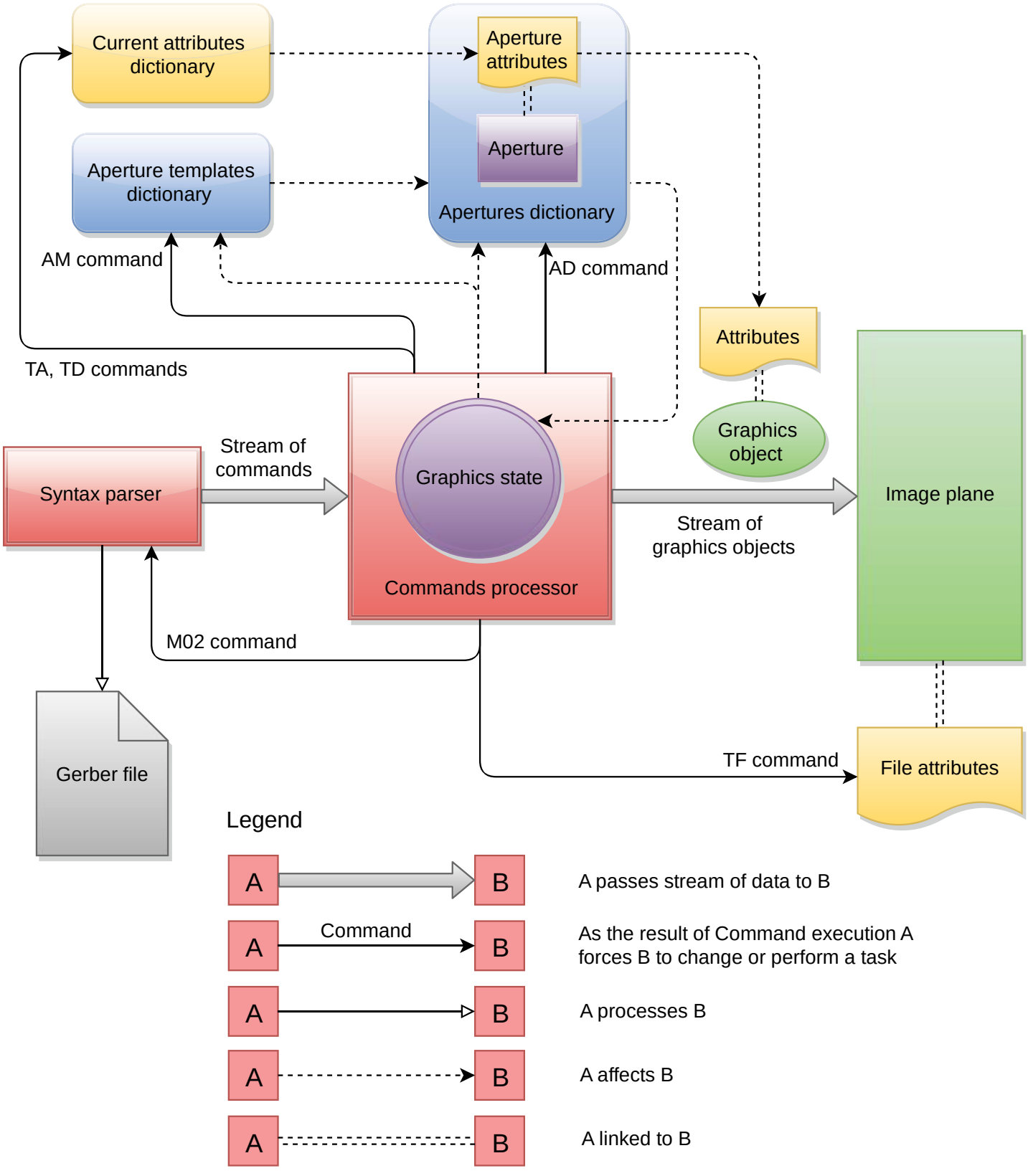
The attribute syntax provides a flexible and standardized way to add meta-information to the images, independent of the specific semantics or application.

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, which is a time-consuming and error-prone process.

The attributes are described in detail in the chapter 5.

2.11 Processing Gerber File with Attributes

The image below illustrates processing Gerber file with file and aperture attributes.



5. Gerber file with attributes processing schema

The schema is the extended version of the schema from the section 2.2.

The current attributes dictionary is responsible for holding all the currently defined aperture attributes. The dictionary is updated by the commands processor as the result of execution TA and TD commands (see 5.3.2 and 5.3.3).

When an aperture is added to the apertures dictionary it gets the attached set of aperture attributes which come from the current attributes dictionary. When the aperture is set as a current aperture and used for creating a graphics object, the aperture attributes are attached to the resulting graphics object.

When the commands processor executes TF command (see 5.2.1) the corresponding file attribute is attached to the resulting image.

2.12 Conformance

If the interpretation of a construct is not specified or not obvious the construct is invalid. A file violating any requirement of the specification or containing any invalid part is wholly invalid. An invalid Gerber file is meaningless and does *not* represent an image.

A conforming Gerber file writer must write files according to this specification. A current conforming Gerber file writer cannot use deprecated constructs. A 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. A current reader may support some or all deprecated format elements as they can be present in legacy files. To prepare for future extensions of the format, a Gerber file reader *must* give a warning when encountering an unknown command or macro primitive; it must then continue processing ignoring the unknown construct. Otherwise there is *no* mandatory behavior on reading an invalid Gerber file. It is *not* mandatory to report any other errors – this would impose an unreasonable burden on readers and may result in useless messages in some applications. It 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 interpretation of the file is valid and another invalid. A reader must also give a warning when it processes a file exceeding its implementation limits.

The responsibilities are obvious and plain. Writers must write valid and numerically 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. Keep in mind Postel's rule: "*Be conservative in what you send, be liberal in what you accept.*"

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. See 7.8.

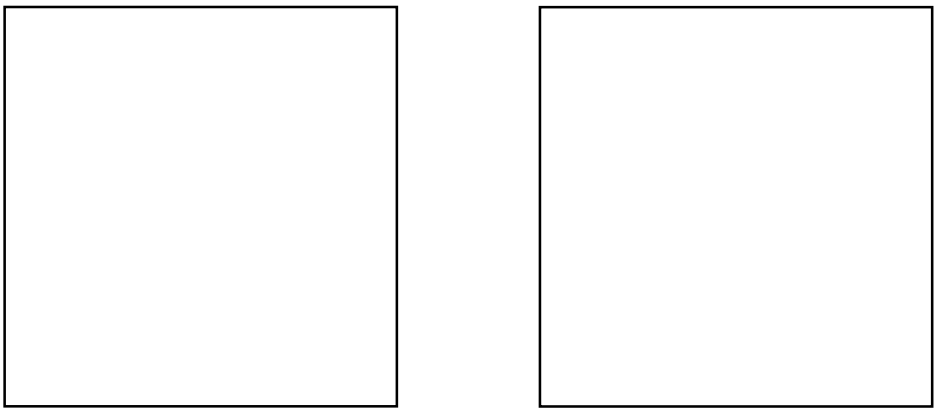
This document is the sole specification of the Gerber format. Gerber viewers, however useful, are not the reference and do not overrule this document.

2.13 Example Files

These annotated sample files 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 will make the formal specification easier to read.

2.13.1 Example: Two Square Boxes

This example represents a single polarity image with two square boxes.



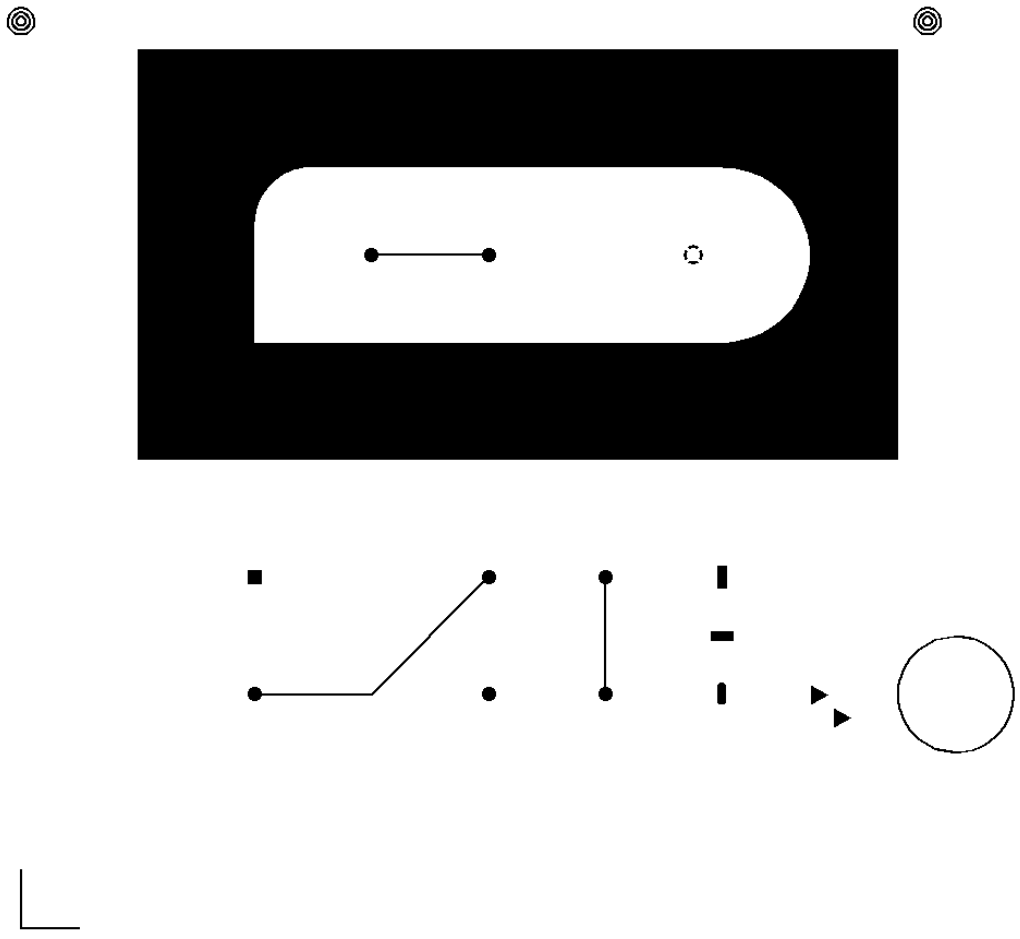
6. Example: two square boxes

Syntax	Comments
`G04 Ucamco ex. 1: Two square boxes*`	A comment
`%FSLAX25Y25*%`	Coordinate format specification: Leading zero's omitted Absolute coordinates Coordinates format is 2.5: 2 digits in the integer part 5 digits in the fractional part
`%MOMM*%`	Unit set to mm
`%TF.Part,Other*%`	Attribute: the file is not a layer of a PCB part - it is just an example
`%LPD*%`	Set the polarity to dark
`%ADD10C,0.010*%`	Define aperture with D-code 10 as a 0.01 mm circle
`D10*`	Set aperture with D-code 10 as current aperture
`X0Y0D02*`	Set current point to (0, 0)
`G01*`	Set linear interpolation mode
`X500000Y0D01*`	Create draw graphics object using the current aperture D10: start point is the current point (0,0), end point is (5, 0)
`Y500000D01*`	Create draw using the current aperture: (5, 0) to (5, 5)
`X0D01*`	Create draw using the current aperture: (5, 5) to (0, 5)

Syntax	Comments
`Y0D01*`	Create draw using the current aperture: (0, 5) to (0, 0)
`X600000D02*`	Set current point to (6, 0)
`X1100000D01*`	Create draw using the current aperture: (6, 0) to (11, 0)
`Y500000D01*`	Create draw using the current aperture: (11, 0) to (11, 5)
`X600000D01*`	Create draw using the current aperture: (11, 5) to (6, 5)
`Y0D01*`	Create draw using the current aperture: (6, 5) to (6, 0)
`M02*`	End of file

2.13.2 **Example: Use of Polarities and Various Apertures**

This example illustrates the use of polarities and various apertures.



7. Example: various shapes

Syntax	Comments
`G04 Ucamco ex. 2: Shapes*`	A comment statement

Syntax	Comments
%FSLAX26Y26*%	Format specification: Leading zero's omitted Absolute coordinates Coordinates format is 2.6: 2 digits in the integer part 6 digits in the fractional part
%MOIN*%	Units are inches
%TF.Part,Other*%	Attribute: the file is not a layer of a PCB part - it is just an example
%LPD*%	Set the polarity to dark This command confirms the default and makes the intention unequivocal
G04 Define Apertures*	Comment
%AMTARGET125*	Define the aperture macro 'TARGET125'
6,0,0,0.125,.01,0.01,3,0.003,0.150,0*%	Use moiré primitive in the macro
%AMTHERMAL80*	Define the aperture macro 'THERMAL80'
7,0,0,0.080,0.055,0.0125,45*%	Use thermal primitive in the macro
%ADD10C,0.01*%	Define the aperture: D10 is a circle with diameter 0.01 inch
%ADD11C,0.06*%	Define the aperture: D11 is a circle with diameter 0.06 inch
%ADD12R,0.06X0.06*%	Define the aperture: D12 is a rectangle with size 0.06 x 0.06 inch
%ADD13R,0.04X0.100*%	Define the aperture: D13 is a rectangle with size 0.04 x 0.1 inch
%ADD14R,0.100X0.04*%	Define the aperture: D14 is a rectangle with size 0.1 x 0.04 inch
%ADD15O,0.04X0.100*%	Define the aperture: D15 is an obround with size 0.04 x 0.1 inch
%ADD16P,0.100X3*%	Define the aperture: D16 is a polygon with 3 vertices and circumscribed circle with diameter 0.1 inch
%ADD18TARGET125*%	Define the aperture: D18 is the instance of the macro aperture called 'TARGET125' defined earlier
%ADD19THERMAL80*%	Define the aperture: D19 is the instance of the macro aperture called 'THERMAL80' defined earlier
G04 Start image generation*	A comment
D10*	Set the current aperture: use aperture with D-code 10
X0Y250000D02*	Set the current point to (0, 0.25) inch
G01*	Set linear interpolation mode

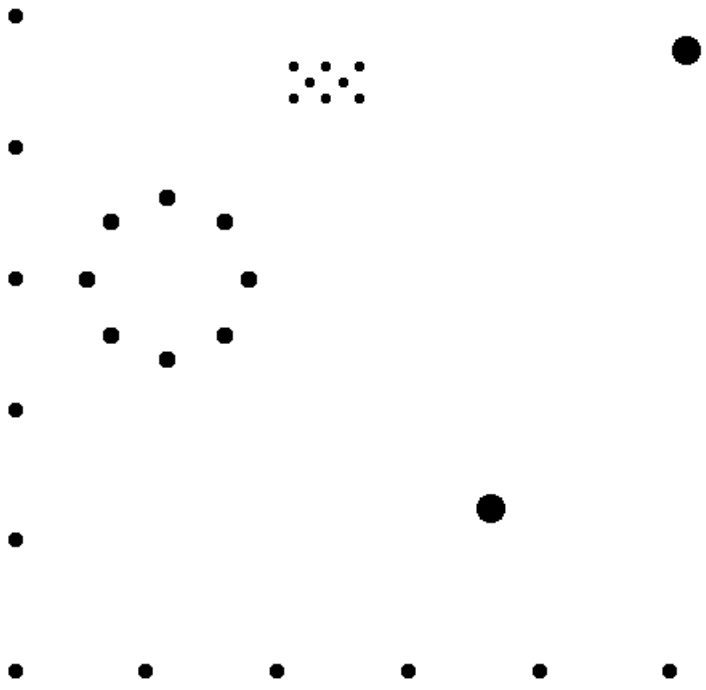
Syntax	Comments
X0Y0D01*	Create draw using the current aperture
X250000Y0D01*	Create draw using the current aperture
X1000000Y1000000D02*	Set the current point
X1500000D01*	Create draw using the current aperture
X2000000Y1500000D01*	Create draw using the current aperture
X2500000D02*	Set the current point. Since the X and Y coordinates are modal, Y is not repeated
Y1000000D01*	Create draw using the current aperture The X coordinate is not repeated and thus its previous value of 2.5 inch is used
D11*	Set the current aperture: use aperture with D-code 11
X1000000Y1000000D03*	Create flash using the current aperture D11 at (1.0, 1.0). Y is modal.
X2000000D03*	Create flash using the current aperture D11 at (2.0, 1.0). Y is modal.
X2500000D03*	Create flash using the current aperture D11 at (2.5, 1.0). Y is modal.
Y1500000D03*	Create flash using the current aperture D11 at (2.5, 1.5). X is modal.
X2000000D03*	Create flash using the current aperture D11 at (2.0, 1.5). Y is modal.
D12*	Set the current aperture: use aperture with D-code 12
X1000000Y1500000D03*	Create flash using the current aperture at (1.0, 1.5)
D13*	Set the current aperture: use aperture with D-code 13
X3000000Y1500000D03*	Create flash using the current aperture at (3.0, 1.5)
D14*	Set the current aperture: use aperture with D-code 14
Y1250000D03*	Create flash using the current aperture at (3.0, 1.25)
D15*	Set the current aperture: use aperture with D-code 15
Y1000000D03*	Create flash using the current aperture at (3.0, 1.0)
D10*	Set the current aperture: use aperture with D-code 10
X3750000Y1000000D02*	Set the current point. This sets the start point for the following arc object
G75*	Set multi quadrant mode
G03	Set counterclockwise circular interpolation mode
X3750000Y1000000I250000J0D01*	Create arc using the current aperture D10. This creates a complete circle

Syntax	Comments
D16*	Set the current aperture: use aperture with D-code 16
X3400000Y1000000D03*	Create flash using the current aperture D16
X3500000Y900000D03*	Create flash using the current aperture D16 again
D10*	Set the current aperture: use aperture with D-code 10
G36*	Enable region mode
X500000Y2000000D02*	Set the current point to (0.5, 2.0)
G01*	Set linear interpolation mode
Y3750000D01*	Create linear segment of the contour
X3750000D01*	Create linear segment of the contour
Y2000000D01*	Create linear segment of the contour
X500000D01*	Create linear segment of the contour
G37*	Disable region mode This creates the region by filling the created contour
D18*	Set the current aperture: use aperture with D-code 18
X0Y3875000D03*	Create flash using the current aperture D18
X3875000Y3875000D03*	Create flash using the current aperture D18
%LPC*%	Set the polarity to clear
G36*	Enable region mode
X1000000Y2500000D02*	Set the current point to (1.0, 2.5)
Y3000000D01*	Create linear segment
G74*	Set single quadrant mode
G02*	Set clockwise circular interpolation mode
X1250000Y3250000I250000J0D01*	Create clockwise circular segment with radius 0.25
G01*	Set linear interpolation mode
X3000000D01*	Create linear segment
G75*	Set multi quadrant mode
G02*	Set clockwise circular interpolation mode
X3000000Y2500000I0J-375000D01*	Create clockwise circular segment with radius 0.375
G01*	Set linear interpolation mode
X1000000D01*	Create linear segment
G37*	Disable region mode This creates the region by filling the created contour

Syntax	Comments
%LPD*%	Set the polarity to dark
D10*	Set the current aperture: use aperture with D-code 10
X1500000Y2875000D02*	Set the current point
X2000000D01*	Create draw using the current aperture
D11*	Set the current aperture: use aperture with D-code 11
X1500000Y2875000D03*	Create flash using the current aperture D11
X2000000D03*	Create flash using the current aperture D11
D19*	Set the current aperture: use aperture with D-code 19
X2875000Y2875000D03*	Create flash using the current aperture D19
%TF.MD5,6ab9e892830469cdff7e3e346331d404*%	Attribute: the MD5 checksum of the file
M02*	End of file

2.13.3 **Example: A Drill File**

This example is a drill file.



8. Example: drill file

Syntax	Comments

Syntax	Comments
%FSLAX26Y26*%	Format specification: Leading zero's omitted Absolute coordinates Coordinate format is 2.6: 2 digits in the integer part 6 digits in the fractional part
%MOIN*%	Units are inches
%TF.FileFunction,Plated,1,8,PTH*%	Attribute: this drill file describes plated-through holes
%TF.Part,Single*%	Attribute: the file is part of a single PCB
%LPD*%	Set the polarity to dark
%TA.DrillTolerance,0.01,0.005*%	Set the drill tolerance attribute 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%	Attribute indicates that the following apertures define component drill holes.
%ADD10C,0.014000*%	Define the aperture: 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*%	Attribute indicates that the following apertures are special drill holes
%ADD11C,0.024000*%	Define the aperture: 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 attribute for the following apertures to 15 mil in both directions
%TA.AperFunction,MechanicalDrill*%	Change the tool function attribute in the dictionary to mechanical
%ADD12C,0.043000*%	Define the aperture: 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*%	Define the aperture: another tool with the same attributes but a smaller diameter
%TD.AperFunction*%	Remove the .AperFunction aperture attribute from the current attributes dictionary
%TD.DrillTolerance*%	Remove the .DrillTolerance aperture attribute from the current attributes dictionary
G01*	Set linear interpolation mode
D10*	Set the current aperture: use drill tool 10
X242000Y275000D03* Y325000D03*	Create several flash graphics objects using the current aperture D10: drill plated component drill holes with

Syntax	Comments
X217000Y300000D03*	diameter 14 mil at indicated coordinates
X192000Y325000D03*	
X292000Y275000D03*	
X192000D03*	
X292000Y325000D03*	
X267000Y300000D03*	
D11*	Set the current aperture: use drill tool 11
X124000Y0D03*	Create several flash graphics objects using the current aperture D11: 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*	Set the current aperture: use drill tool 12
X792000Y350000D03*	Create several flash graphics objects using the current aperture D12: drill plated mechanical drill holes with diameter 43 mil at indicated coordinates
X492000Y-350000D03*	
D13*	Set the current aperture: use drill tool 13
X767000Y-600000D03*	Create several flash graphics objects using the current aperture D13: 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%	Attribute: 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 is image information that Gerber files describe perfectly. For more information, see 5.4.1.1.

2.14 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.)

Aperture macro: The content of an Aperture Macro (AM) command. Provides a definition of a custom aperture that is created by combining a number of primitives.

Aperture template: A template that is used to instantiate an aperture. There are two types of templates: standard (built-in) apertures and macro apertures. Templates are added to aperture templates dictionary and later they are used by AD command for creating apertures.

Aperture templates dictionary: The element of the Gerber file processing that is responsible for holding all the defined aperture templates. Initially the dictionary contains standard aperture templates only; additional macro aperture templates can be added to the dictionary by AM command.

Apertures dictionary: The element of the Gerber file processing that is responsible for holding all the defined apertures.

Arc: A graphics object created by D01 code command in a circular interpolation mode.

Attribute: Metadata that is associated with the file as a whole or with some of its graphics objects; it provides extra information without affecting the image.

Binary image: A two-dimensional (2D) image represented by two colors.

Block: A stream of graphics objects that can be added to the final objects stream.

Circular interpolation: Creating a circular segment (circular arc) that (depending on the region mode) is either converted to an arc graphics object or used as a circular contour segment.

Clear: Clear the shape of a graphics object on the image plane; this happens when a graphics object with clear polarity is added to the image.

Command: A higher-level element of a Gerber file that consists of one or more data blocks. Any command contains a command code optionally with additional data necessary for the command execution. Commands are responsible for manipulating graphics state, creating graphics objects, defining apertures, managing attributes and so on.

Command code: A code that identifies the command.

Contour: A closed a sequence of connected linear or circular segments. Contours are used to create regions and outline primitives in macro apertures.

Coordinate data: A data that specifies a position (X and Y coordinates) of a point in the image plane; when the data is used for circular interpolation it may also include the distance or offset in the X and Y direction from the point.

Coordinate format: The coordinate data specification defined by FS command. The effective coordinate format is stored as the value of the corresponding graphics state

parameter.

Current aperture: The graphics state parameter that specifies the last aperture selected by a Dnn command. Flashes, draws and arcs are always created using the current aperture.

Current attributes dictionary: The element of the Geber file processing that is responsible for holding all the currently defined aperture attributes.

Current point: The graphics state parameter that specifies the coordinates of a point in the plane used as a begin point of a circular or linear interpolation.

Darken: Darken the shape of a graphics object on the image plane; this happens when a graphics object with dark polarity added to the image.

Data block: The low level syntactical element of a Gerber file that is represented by a sequence of characters ending with '*' character. Data blocks are used to build commands.

Draw: A graphics object created by D01 code command in linear interpolation mode.

Extended code: A command code consisting of two letters, e.g. 'FS'.

Extended code commands: Commands defined by the extended codes and enclosed in a pair of '%' characters, typically manipulating graphics state or attributes.

File image: A binary image that is the visual representation of a Gerber file. It is created by superimposing the graphics objects in the plane.

Flash: A graphics object created by D03 code command. This command replicates (flashes) the current aperture; an arc or draw that is the result of a zero length stroking is not flash, although it is also a graphics object with the shape of the current aperture.

Function code: A command code consisting of a letter 'D', 'G' or 'M' followed by a code number.

Function code commands: Commands defined by function codes. The D01, D02 and D03 function code commands are called operations.

Gerber file: A file in the Gerber format.

Gerber format: The vector format defined by the current specification and used for representing a binary image.

Graphics object: A graphics object is a 2D image in the image plane. It has a shape, a size, a position and a polarity (dark or clear). It is of one of the following types: flash, draw, arc or region. The file image is created by superimposing graphics objects on the image plane.

Graphics state: The set of parameters that at each moment determine the effect of the upcoming operation codes. The graphics state defines the graphics objects which are being created.

Graphics state parameter: An element of the graphics state defining one specific parameter that influences on the graphics object creation, e.g. interpolation mode.

Header: The part of the file from the file beginning to the point where the first operation code is encountered**.**

Image plane: The 2D plane in which the image defined by the file is created.

Interpolation mode: The graphics state parameter defining the current interpolation mode. See linear and circular interpolation.

Linear interpolation: Creating a straight segment that (depending on the region mode) is either converted to a draw graphics object or used as a linear contour segment.

Macro aperture: An aperture template defined using AM command.

Multi quadrant mode: A mode defining how circular interpolation is performed. In this mode a circular arc is allowed to 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: A command containing one of the operation codes D01, D02 or D03 and a coordinate data. The operation code defines the type of the operation that is performed using the coordinate data. Operations may create graphics objects, create contours, and change the current point of the graphics state.

Operation codes: The function codes D01, D02 or D03.

Polarity: A graphics state parameter that can take the value dark or clear. It determines the polarity of the graphics objects generated. Dark means that the object exposes or marks the image plane in dark and clear means that the object clears or erases everything underneath it. See also ‘Darken’ and ‘Clear’.

Quadrant mode: The graphics state parameter defining the current quadrant mode. See multi quadrant mode and single quadrant mode.

Region: A graphics object with an arbitrary shape defined by its contour.

Region fill: Creating a region by darkening or clearing a generated contour.

Region mode: The graphics state parameter defining if the region mode is currently enabled or disabled. The region mode allows creating a region by defining 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 a circular 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**°.**

Standard aperture: A built-in aperture template that has pre-defined format.

Standard attribute: Pre-defined attribute conveying meta-information required for PCB data transfer from design to fabrication.

Step and repeat: A method by which replications of an accumulated block are made to produce multiple copies of a set of graphics objects. The current step and repeat settings are defined by ‘Step & Repeat’ graphics state parameter.

Stroke: To create a draw or an arc graphics object using the current aperture.

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

Unit: The measurement unit ‘mm’ or ‘inch’ used to interpret the coordinate data. The effective unit is stored as the value of the corresponding graphics state parameter.

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

3 Syntax

3.1 Conventions for Syntax Rules

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

3.2 File Extension, MIME Type and UTI

The Gerber Format has a standard file name extension, a registered mime type and a UTI definition.

Standard file extension: .gbr or .GBR

Mime type: application/vnd.gerber

(see http://www.iana.org/assignments/media-types/application/vnd.gerber)

Mac OS X UTI:

```
<key>UTExportedTypeDeclarations</key>
<array>
  <dict>
    <key>UTTypeIdentifier</key>
    <string>com.ucamco.gerber.image</string>
    <key>UTTypeReferenceURL</key>
    <string>http://www.ucamco.com/gerber</string>
    <key>UTTypeDescription</key>
    <string>Gerber image</string>
    <key>UTTypeConformsTo</key>
    <array>
        <string>public.plain-text</string>
        <string>public.image</string>
    </array>
    <key>UTTypeTagSpecification</key>
    <dict>
        <key>public.filename-extension</key>
        <array>
            <string>gbr</string>
        </array>
        <key>public.mime-type</key>
        <string>application/vnd.gerber</string>
    </dict>
  </dict>
</array>
```

3.3 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.

Space characters can only be used inside strings (see 3.6.6). They cannot be used inside or between commands and data blocks, etc.

Gerber files are case-sensitive. Command codes must be in upper case.

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 '*****'. A data block may contain function code, coordinates data, extended code, aperture primitive description, variable definition and so on.

Example:

```
X0Y0D02*
G01*
X50000Y0D01*
%AMDONUTCAL*
1,1,$1,$2,$3*
$4=$1x0.75*
1,0,$4,$2,$3*%
```

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.

Note: The pair of '%' characters in the above example does not belong to any data block. This pair is the special syntax of the extended code commands (see 3.5.3).

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.

A command consists of one or more data blocks. Most commands consist of a single data block. Any command contains a command code optionally with additional data necessary for the command execution.

Commands are responsible for manipulating graphics state, creating graphics objects, defining apertures, managing attributes and so on.

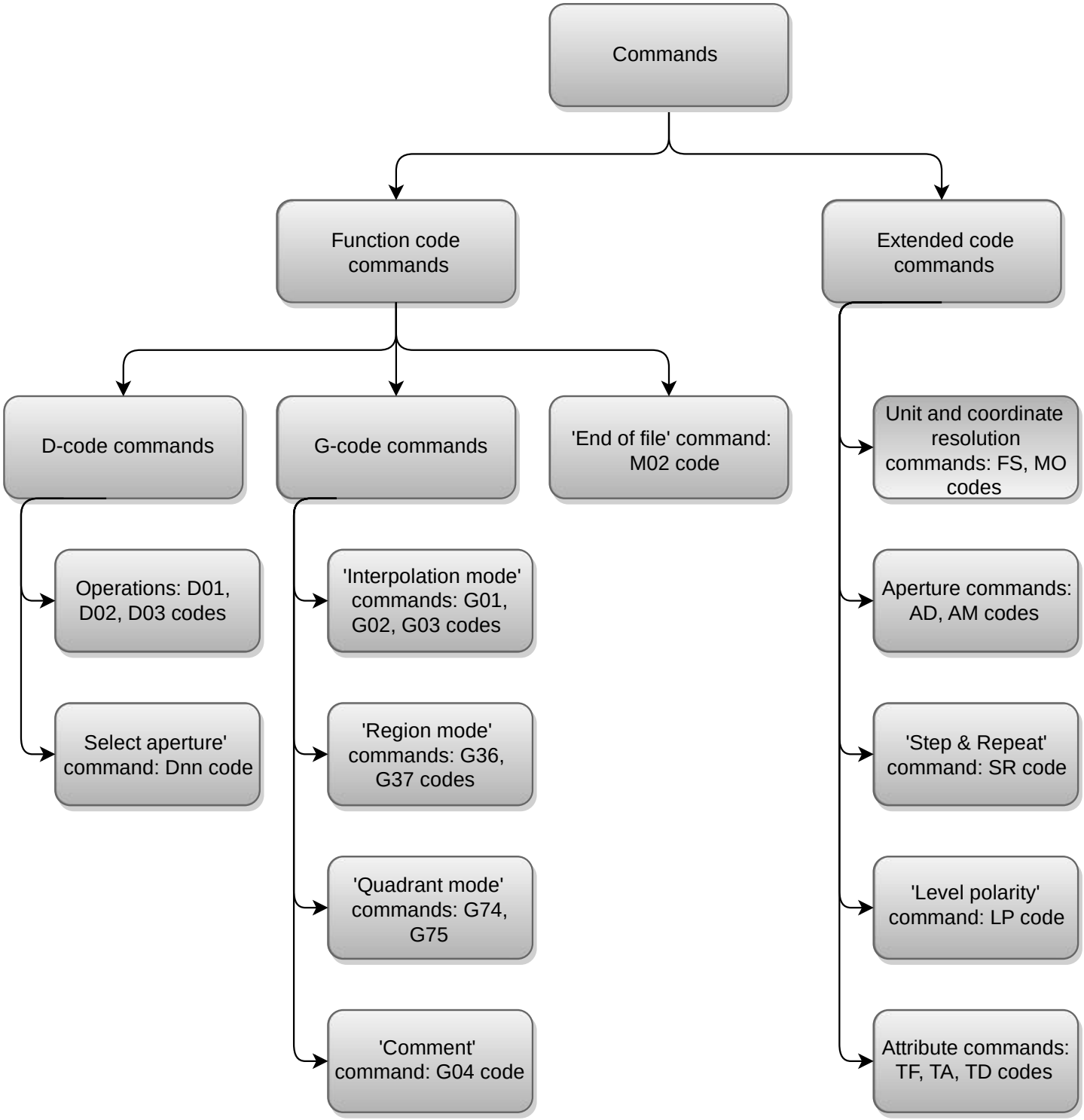
A Gerber file consists of a stream of commands. There is no limitation on the number of commands.

Command syntax:

<Command> = <Data Block>{<Data Block>}

For historic reasons there are two types of commands: function code and extended code commands. The difference between them is that each extended code command must be included into a separate pair of '%' characters.

The commands can be divided into groups as represented on the image below.



9. Gerber file commands

The function code commands are described in 3.5.2.

The extended code commands are described in 3.5.3.

The example below shows the stream of Gerber file commands of different types.

Example:

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

3.5.2 Function Code Commands

Function code commands are identified by a code letter G, D or M followed by a code number, e.g. `G02`.

A code number is a positive integer number without preceding '+'. The available code numbers are described in this specification. A code number can be padded with leading zeros, but the resulting number record must not contain more than 10 digits.

Example:

```
X100Y125D1*
X100Y125D01*
X100Y125D0001*
G002*
G0000074*
```

The conventional representation of a code number contains exactly two digits, so if the number is less than 10, it is padded with one leading zero. This representation is used everywhere in the specification.

Example:

```
X100Y125D01*
X100Y125D02*
G01*
G74*
```

Function code commands are either

- Operations: commands identified by D01, D02, D03 function codes
- Other commands, that set a graphics state parameter

The codes D01, D02, D03 have a special function and are called operation codes. They are used together with coordinate data to form commands called *operations*.

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

Example:

```
X0Y100I-400J100D01*
```

Each operation must end with a one and only one operation code. The operation code defines how the preceding coordinate data is used.

In the next example there are two operations. The first operation sets the current point to (300, 200). The second operation creates a graphics object (arc or draw, depending on the interpolation mode) from the current point to the end point (1100, 200).

Example:

```
X300Y200D02*
X1100Y200D01*
```

Operations are described in detail in chapter 4.2. Other function code commands are described in chapters from 4.3 to 4.8.

3.5.3 **Extended Code Commands**

Extended code commands are responsible for setting graphics state parameters, defining macro aperture templates and instantiating apertures, manipulating attributes.

Extended codes commands affecting the entire image must be placed in the header of the file. Other extended codes are placed at the appropriate location.

An extended code command consists of a two-character command code followed by command data. The command code identifies the command. The structure and meaning of the command data depends on the command code.

An extended code command is enclosed into a separate pair of delimiter '%' characters. Usually a command 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.

Example:

```
%FSLAX24Y24*%
%AMDONUTFIX*1,1,0.100,0,0*1,0,0.080,0,0*%
```

There can be only one extended code command between each pair of '%' delimiters. It is allowed to put line separators between data blocks of a single command.

The following example is an AM function code command built of three data blocks. The data blocks of the AM command are separated by a newline character for better readability.

Example:

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

Tip: For readability it is recommended to have one command per line; in case of AM command it is recommended to have one data block per line.

The syntax for an individual extended code command is:

<Command> = <Command code><Command data>*{<Additional command data>*}

Syntax	Comments
Command code	2-character code (AD, AM, FS, etc…)
Command data	The data necessary for the command. Normally it includes: required modifiers: must be entered to complete definition optional modifiers: may be necessary depending on the required modifiers
Additional command data	Additional command data in the extra data blocks (used for AM command only)

We distinguish two *classes* of extended codes:

- Graphics commands affect the image generation. They define how the function codes and coordinates are processed. The graphics commands are described in the section 4.
- Attribute commands do not affect the image generation but attach attributes to either the image as a whole or to the individual graphics objects. The attribute commands are described in the section 5.

3.6 Data Types

3.6.1 **Integers**

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

3.6.2 **Decimals**

Decimals are a sequence of one or more digits with an optional decimal point optionally preceded by a '+' or a '-' sign. They must fit in an IEEE double.

3.6.3 **Coordinate Number**

Coordinate numbers are integers conforming to the rules set by the FS command. See 4.9.1. Coordinate numbers are used to express coordinates.

3.6.4 **Hexadecimal**

An hexadecimal is a number expressed using a positional numeral system with a base of 16. It is represented using sixteen distinct characters: 0–9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen. Thus a hexadecimal number is a sequence of characters that matches the regular expression:

```
[a-fA-F0-9]+
```

The letters in a hexadecimal number can be upper case or lower case characters. It means the sequences 'A9' and 'a9' represent the same number.

3.6.5 **Names**

Names consist of upper or lower case letters, underscores ('_'), dots ('.'), a dollar sign ('$') 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. User defined names *cannot* begin with a dot.

The scope of a name starts at its definition and runs till the end of the file.

Note: The variable names within macro's follow their own rules.

3.6.6 **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`

Any character with a Unicode code lower than 65,536 can be included in a string by specifying the Unicode character code in hexadecimal in the Unicode escape sequence:

```
\uXXXX
```

The four characters XXXX are a hexadecimal number (see 3.6.4) indicating the code of the Unicode character represented by the escape sequence. For example, \u00a9 represents the copyright symbol.

Unicode escape sequence must be six characters long. It means there must be exactly four characters following \u. If the character code contains less hexadecimal digits, it must be padded with leading zeros.

A hexadecimal number syntax allows upper case and lower case letters so both '\u00A9' and '\u00a9' are allowed and represent the same character.

The Unicode escape sequence syntax conforms to the regular expression:

```
\\u[a-fA-F0-9]{4}
```

A literal backslash character '\' inside a string shall be represented using the backslash character code as \u005c, otherwise, if '\' character and 5 next characters conform to the regular expression `\\u[a-fA-F0-9]{4}`, the whole sequence will be interpreted as the Unicode escape sequence.

For the string length the Unicode escape sequence is counted as one character.

Note: The Unicode escape sequences can be used only inside strings.

4 Graphics

4.1 Graphics Overview

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

The table gives an overview of all the commands. They are explained in detail further in this chapter.

Command	Command description	Comments
D01	Interpolate operation. See 4.2.	If region mode is off D01 creates a draw or arc object using the current aperture. When region mode is on D01 creates a linear or circular contour segment. The current aperture is not used. After the D01 command the current point is moved to the coordinate.
D02	Move operation. See 4.2.	D02 does not create a graphics object but moves the current point to the coordinate.
D03	Flash operation. See 4.2.	With region mode is off D03 flashes the current aperture. D03 is not allowed when region mode is on. After the D03 command the current point is moved to the coordinate.
Dnn (nn≥10)	Sets the current aperture.	Sets the current aperture to aperture nn. (Aperture numbers are set by AD command, see 4.11)
G01	Sets the interpolation mode to linear. See 4.4.	Used to alter the effect of the interpolate operation (D01).
G02	Sets the interpolation mode to ‘Clockwise circular interpolation’. See 4.5.	
G03	Sets the interpolation mode to ‘Counterclockwise circular interpolation’. See 4.5.	
G04	Ignore data block. See 4.7.	Used for comments.
G36	Sets region mode on. See 4.6.	Used to create regions.
G37	Sets region mode off. See 4.6.	
G74	Sets quadrant mode to ’Single quadrant’. See 4.5.	A modifier of the circular interpolation mode.
G75	Sets quadrant mode to ’Multi quadrant. See 4.5.	
M02	Indicates the end of the file. See 4.8.	Every Gerber file must end in a M02. No data is allowed after M02.
FS	Sets the ‘Coordinate format’ graphics state parameter. See 4.9.	These commands are mandatory and must be used only once, in the header of a file.
MO	Sets the ‘Unit’ graphics state parameter. See 4.10.	
AD	Assigns a D code number to an aperture definition. See 4.11.	These commands can be used multiple times. It is recommended to put them in header of a file.
AM	Defines macro aperture templates. See 4.13.	
LP	Sets the ‘Polarity’ graphics state parameter. See 4.14.	These commands can be used multiple times over the whole file.
SR	Open or closes a ‘Step and Repeat’ statement. See 4.14.	

Gerber file commands

4.2 Operations (D01/D02/D03)

D01, D02 and D03 are the *operation codes*. Together with coordinate data the operation codes define commands called *operations*. An operation operates on its coordinate data.

Syntactically an operation contains the coordinate data followed by its operation code. An operation must contain a single (1) operation code: each operation code is associated with a single coordinate pair and vice versa.

The operations have the following effect.

- Operation with D01 code is called *interpolate* operation. It creates a straight line segment or a circular segment by interpolating from the current point to the operation coordinates. The segment is then converted to a graphics object depending on the value of region mode graphics state parameter.
- Operation with D02 code is called *move* operation. It moves the current point to the operation coordinates. No graphics object is generated.
- Operation with D03 code is called *flash* operation. It creates a flash object by replicating the current aperture at the operation coordinates.

Note: The code representation 01, 02, 03 (with one leading zero) is conventional; it is allowed to use a different number of leading zeros: 1, 001, 0002, etc. See 3.5.2 for more details.

The operations are controlled by the graphics state (see 2.8).

The D03 operation directly creates a flash object by replicating (flashing) the current aperture. When the aperture is flashed its origin is positioned at the coordinates of the operation. The origin of a standard aperture is its geometric center. The origin of a macro aperture is the origin of the coordinates (the origin of the macro definition) used in the AM (Aperture Macro) command.

Sequences of D01 and D02 operations create segments that are turned into a graphics objects by one of two following methods:

- Stroking. The segments are stroked with the current aperture, see 2.5.
- Region building. The segments form contour that defines a region, see 4.6.

The region mode graphics state parameter determines which object generating method is used. If the region mode is *off* the stroking is used to convert a segment into draw or arc graphics object. If the region mode is *on* the region building is used: a segment becomes the linear or circular contour segment then.

There is another graphics state parameter called interpolation mode that affects operations. It defines the form of the interpolated segment: linear interpolation mode results in a draw or linear contour segment; circular interpolation mode results in an arc or circular contour segment. This is described in detail in the section 4.4.

The circular interpolation mode can be clockwise and counterclockwise. Also in circular interpolation mode the quadrant mode parameter becomes relevant. It defines the arc angle. See 4.5 for more details.

The table below summarizes the results of the operations depending on the graphics state parameter values.

Operation code	Graphics state parameters values					
	Region mode on (G36)			Region mode off (G37)		
	Linear interpolation mode (G01)	Circular interpolation mode (G02, G03)		Linear interpolation mode (G01)	Circular interpolation mode (G02, G03)	
		Single quadrant mode (G74)	Multi quadrant mode (G75)		Single quadrant mode (G74)	Multi quadrant mode (G75)
D01	Linear contour segment	Circular contour segment (0°≤α≤90°)	Circular contour segment (0°<α≤360°)	Draw	Arc (0°≤α≤90°)	Arc (0°<α≤360°)
D02	Closes current contour and moves current point			Moves current point		
D03	Not allowed			Flash		

Effect of operation codes depending on graphics state parameters

The table describes only the parameters which have direct influence on the types of objects created by the operation codes. The effect of the other parameters is described elsewhere.

4.2.1 Coordinate Data Syntax

Coordinate data is the part of an operation. The syntax of the data is the following:

<Coordinate data> = [X<Number>][Y<Number>][I<Number>][J<Number>]

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 This data allowed only in D01 operations in circular interpolation mode (see 4.2.2)
<Number>	Coordinate number - see section 3.6.3 - defining either a coordinate (X, Y) or an offset or distance (I, J). The number must have at least one digit

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.

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. It means when the current point value is changed the offsets are ignored and only the values for X and Y are used.

Examples:

`X200Y200D02*` point (+200, +200) operated upon by D02

`Y-300D03*`	point (+200, -300) 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

In an operation without explicit X and Y the coordinates of the current point are used. In the example below D03 results in a flash at the current point.

Example

```
D03*
```

4.2.2 **D01 Command**

The D01 code (interpolate) operation syntax is as follows:

<D01 operation> = [X<Number>][Y<Number>][I<Number>][J<Number>]D01*

Syntax	Comments
X<Number>	Coordinate data defining the X coordinate of the interpolation end point. For more details see 4.4.2 and 4.5.6 The X coordinate is then used to set the new current point If missing then the previous X coordinate is used <Number> is a coordinate number – see section 3.6.3
Y<Number>	Coordinate data defining the Y coordinate of the interpolation end point. For more details see 4.4.2 and 4.5.6 The Y coordinate is then used to set the new current point If missing then the previous Y coordinate is used <Number> is a coordinate number – see section 3.6.3
I<Number>	Coordinate data defining the distance or offset in the X direction This coordinate data is only allowed in circular interpolation mode <Number> is a coordinate number – see section 3.6.3
J<Number>	Coordinate data defining the distance or offset in the Y direction This coordinate data is only allowed in circular interpolation mode <Number> is a coordinate number – see section 3.6.3
D01	Interpolate operation code

The use of the coordinate data in D01 code operation depends on the interpolation mode. For the additional details see 4.4.2 and 4.5.6.

Example:

```
X200Y200D01*
X200Y200I50J50D01*
```

4.2.3 **D02 Command**

The syntax for the D02 code (move) operation is the following:

<D02 operation> = [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 3.6.3
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 3.6.3
D02	Move operation code

The effect of D02 operation depends on the region mode (see 2.8). If the region mode is *off* the operation sets the new value for the current point graphics state parameter. If the region mode is *on* the operation closes the current contour and then sets the new value for the current point graphics state parameter (see 4.6).

Example:

```
X200Y1000D02*
```

4.2.4 **D03 Command**

The syntax for the D03 code (flash) operation is the following:

<D03 operation> = [X<Number>][Y<Number>]D03*

Syntax	Comments
X<Number>	Coordinate data defining the X coordinate of the aperture origin. If missing then the previous X coordinate is used. <Number> is a coordinate number – see section 3.6.3.
Y<Number>	Coordinate data defining the Y coordinate of the aperture origin. If missing then the previous Y coordinate is used. <Number> is a coordinate number – see section 3.6.3
D03	Flash operation code

Warning: D03 operation is not allowed when the region mode is *on*.

Example:

```
X1000Y1000D03*
```

4.2.5 **Example**

The example shows a stream of commands in a Gerber file. Some of the commands are operation codes, others are G code commands (G01, G03, G36, G37, G74, and G75). The G code commands set the graphics state parameters that are relevant for the operations: interpolation mode (G01 – see 4.4, G03 – see 4.5), region mode (G36, G37 – see 4.6), quadrant mode (G74, G75 – see 4.5).

Example:

```
G36*
X200Y1000D02*
G01*
X1200D01*
Y200D01*
X200D01*
Y600D01*
X500D01*
G75*
G03*
X500Y600I300J0D01*
G01*
X200D01*
Y1000D01*
G37*
```

4.3 Current Aperture (Dnn)

The command with code Dnn sets current aperture graphics state parameter.

The syntax is:

<Dnn command> = D<D-code number>*

Syntax	Comments
D	Command code
<D-code number>	The D-code number (≥10) of an aperture from the apertures dictionary The aperture must be previously added in the apertures dictionary by AD command

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.

The D01 and D03 commands use the current aperture to create track and flash graphics objects. The current aperture must be explicitly defined before it is used – see 2.8.

Example:

```
D10*
```

4.4 Linear Interpolation Mode (G01)

When linear interpolation mode is enabled a D01 code operation generates a straight line from the current point to the point with X, Y coordinates specified by the operation. The current point is then set to the X, Y coordinates.

To enable linear interpolation mode the G01 command is used.

4.4.1 G01 Command

The syntax for the command to enable linear interpolation mode:

<G01 command> = G01*

Syntax	Comments
G01	Sets interpolation mode graphics state parameter to 'linear interpolation'

Example:

```
G01*
```

4.4.2 D01 Command

The section 4.2.2 defines the general syntax of the D01 code operation. However in the linear interpolation mode I and J coordinate parts are not allowed, so the D01 command syntax is:

<D01 operation> = [X<Number>][Y<Number>]D01*

Syntax	Comments
X<Number>	Coordinate data defining the X coordinate of the straight segment (if the region mode is off – draw graphics object; if the region mode is on – linear contour segment) end point The X coordinate is then used to set the new current point If missing then the previous X coordinate is used <Number> is a coordinate number – see section 3.6.3
Y<Number>	Coordinate data defining the Y coordinate of the straight segment (if the region mode is off – draw graphics object; if the region mode is on – linear contour segment) end point The Y coordinate is then used to set the new current point If missing then the previous Y coordinate is used <Number> is a coordinate number – see section 3.6.3
D01	Interpolate operation code

Example:

```
G01*
X200Y200D01*
```

4.5 Circular Interpolation (G02/G03) and (G74/G75)

4.5.1 Circular Arc Overview

A circular arc is a circular segment created by a D01 (interpolate) operation with the graphics state set to circular interpolation. When region mode is off a track is added to the graphics object stream. When region mode is on a contour segment is added to the current region.

D01 code operation in circular interpolation mode generates a circular arc from the current point to the point with X, Y coordinates specified by the operation; 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 operation.

There are two orientations:

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

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

There are two quadrant modes:

- Single quadrant mode, set by G74 command
- Multi quadrant mode, set by G75 command

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 separate operation is required for each quadrant. A minimum of four operations 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

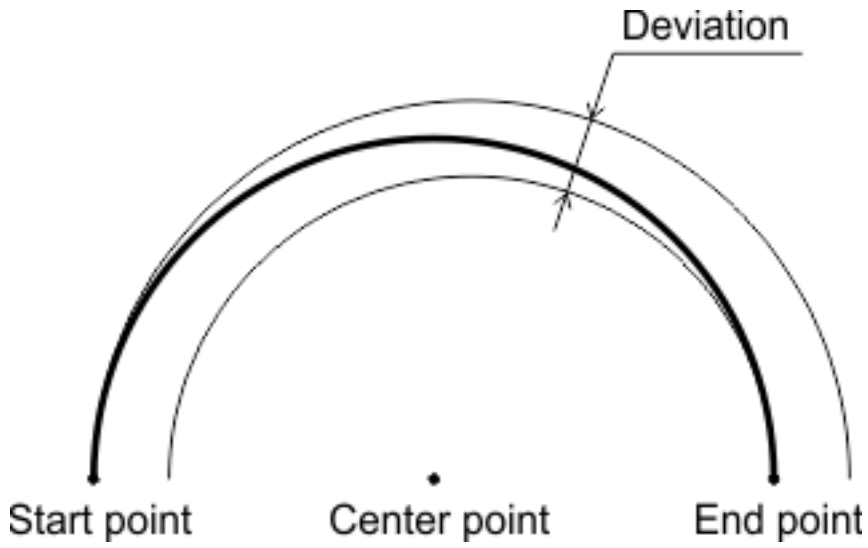
The commands with codes G74 and G75 allow switching between single- and multi-quadrant modes. G75 command activate multi quadrant mode. Every operation following it will be interpreted as multi quadrant, until cancelled by a G74 command. G74 command turns on single quadrant mode.

Warning: A Gerber file that attempts to interpolate circular arcs without a preceding G74 or G75 code is invalid.

For a strictly circular arc the distance of from the center to the start point must be exactly equal to the distance to the end point. This distance is the radius and the interpretation of the arc is then obvious.

However, as a Gerber file has a finite resolution, the center point generally cannot be positioned such that the distances – radii - are indeed exactly equal. Furthermore the software generating the Gerber file unavoidably adds rounding errors of its own. The two radii are unavoidably different for almost all real-life arcs. We will call the difference the *arc deviation*. An exact circle of course has only one radius, the same everywhere. This raises the question which curve is represented by a “circular arc” with a non-zero deviation.

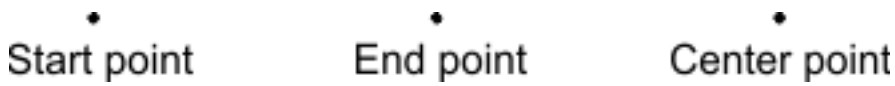
The arc defined *as a continuous and monotonic curve starting at the start point and ending at the end point, approximating the ring with the given center point and radii equal to the start radius and end radius.* See figure 10.



10. Arc with a non-zero deviation

The arc definition has 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 however not allowed to place the center point close to the straight line through begin and end point except when it is strictly in between these points. When the center is on or outside the segment between start and end point the construct is nonsensical. See figure 11.



11. Nonsensical center point

Note that self-intersecting contours are not allowed, see 2.6. If any of the valid arc interpretations turns the contour in a self-intersecting one, the file is invalid, with unpredictable results.

The root cause of most problems with arcs is the use a low resolution. One sometimes attempts to force arcs of size of the order of e.g. 1/10 of a mil in a file with resolution of 1/10. This is asking for problems. Use higher resolution. See 4.9.1.

4.5.2 **G02 Command**

The syntax for the command to enable clockwise circular interpolation mode:

<G02 command> = G02*

Syntax	Comments
G02	Sets interpolation mode graphics state parameter to 'clockwise circular interpolation'

Example:

```
G02*
```

4.5.3 **G03 Command**

The syntax for the command to enable counterclockwise circular interpolation mode:

<G03 command> = G03*

Syntax	Comments
G03	Sets interpolation mode graphics state parameter to 'counterclockwise circular interpolation'

Example:

```
G03*
```

4.5.4 **G74 Command**

The syntax for the command to enable single quadrant mode:

<G74 command> = G74*

Syntax	Comments
G74	Sets quadrant mode graphics state parameter to 'single quadrant'

Example:

```
G74*
```

4.5.5 **G75 Command**

The syntax for the command to enable multi quadrant mode:

<G75 command> = G75*

Syntax	Comments
G75	Sets quadrant mode graphics state parameter to ‘multi quadrant’

Example:

```
G75*
```

4.5.6 D01 Command

The section 4.2.2 defines the general syntax of the D01 code operation. This section explains the meaning of the coordinate data parts for the case when a circular interpolation mode is enabled:

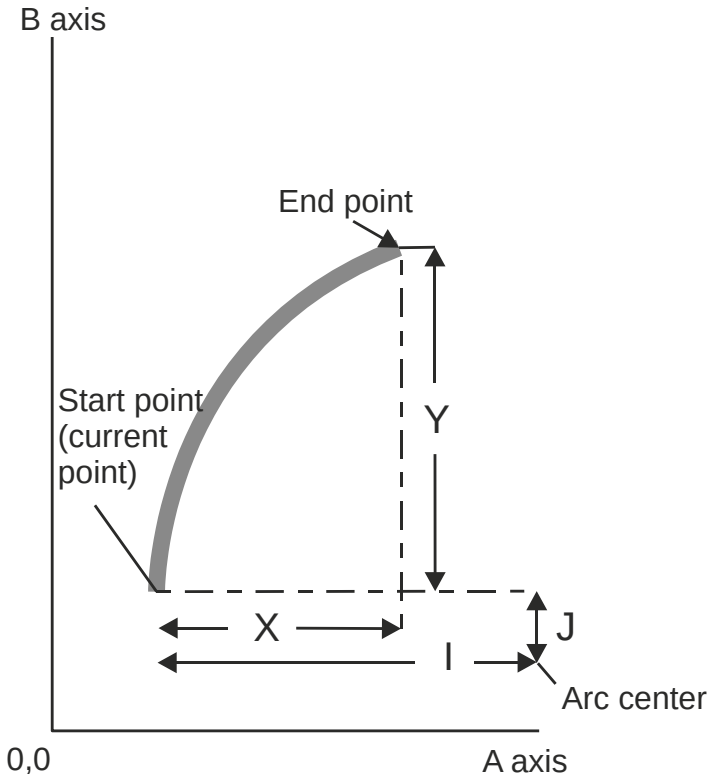
<D01 operation> = [X<Number>][Y<Number>][I<Number>][J<Number>]D01*

Syntax	Comments
X<Number>	Defines the X coordinate of the circular arc (if the region mode is off – arc graphics object; if the region mode is on – circular contour segment) end point. If missing then the previous X coordinate is used. <Number> is a coordinate number – see section 3.6.3.
Y<Number>	Defines the Y coordinate of the circular arc (if the region mode is off – arc graphics object; if the region mode is on – circular contour segment) end point. If missing then the previous Y coordinate is used. <Number> is a coordinate number – see section 3.6.3.
I<Number>	**In single quadrant mode**: the distance between the circular arc start point and the center measured parallel to the X axis. Number is ≥ 0. **In multi quadrant mode**: the offset or signed distance between the circular arc start point and the center measured parallel to the X axis. If missing then a 0 distance is used. <Number> is a coordinate number – see section 3.6.3.
J<Number>	**In single quadrant mode**: the distance between the circular arc start point and the center measured parallel to the Y axis. Number is ≥ 0. **In multi quadrant mode**: the offset or signed distance between the circular arc start point and the center measured parallel to the Y axis. If missing then a 0 distance is used. <Number> is a coordinate number – see section 3.6.3.

Syntax	Comments
D01	Interpolate operation code

The coordinates of a circular 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 circular arc are interpolated.



12. Circular interpolation example

Note: In single quadrant mode, 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°.

Example:

```
G74*
G03*
X700Y1000I400J0D01*
```

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

Example:

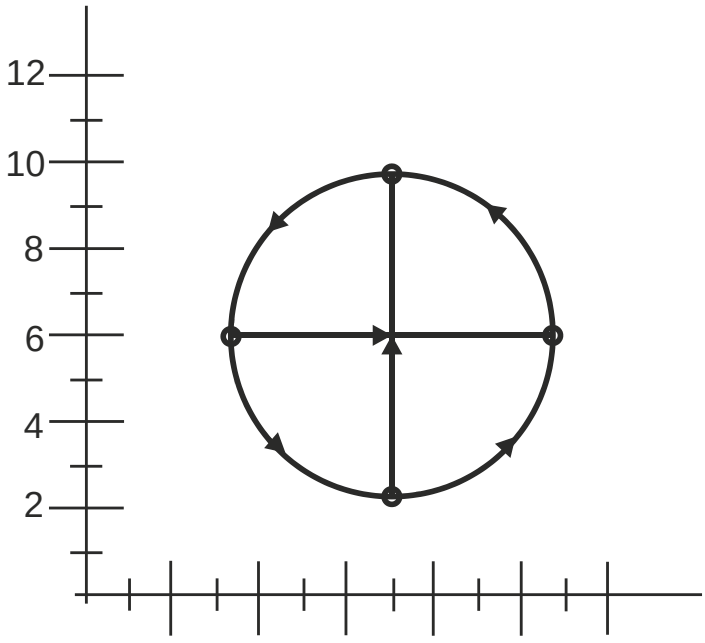
```
G75*
G03*
```

```
X-300Y-200I-300J400D01*
```

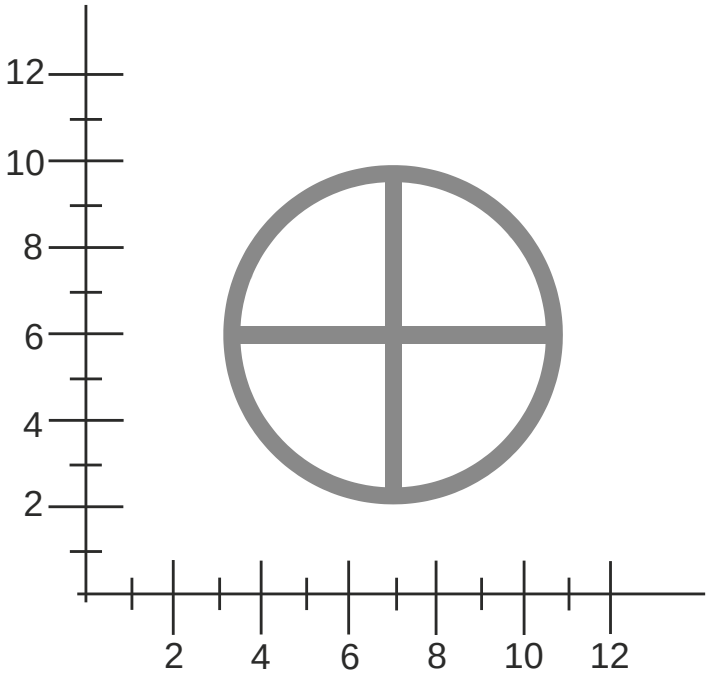
Warning: If the center is not precisely positioned, there may be none or more than one candidate fits. In that case the arc is invalid. The creator of the file accepts any interpretation.

4.5.7 Example: Single Quadrant Mode

Syntax	Comments
G74*	Set single quadrant mode
D10*	Set the current aperture to D10 aperture
X1100Y600D02*	Set the current point to (11, 6)
G03*	Set counterclockwise interpolation mode
X700Y1000I400J0D01*	Create quarter arc object (radius 4) to (7, 10)
X300Y600I0J400D01*	Create quarter arc object (radius 4) to (3, 6)
X700Y200I400J0D01*	Create quarter arc object (radius 4) to (7, 2)
X1100Y600I0J400D01*	Create quarter arc object (radius 4) to (11, 6)
X300D02*	Set the current point to (3 ,6)
G01*	Set linear interpolation mode
X1100D01*	Create draw object to (11, 6)
X700Y200D02*	Set the current point to (7, 2)
Y1000D01*	Create draw object to (7, 10)



13. Single quadrant mode example: arcs and draws

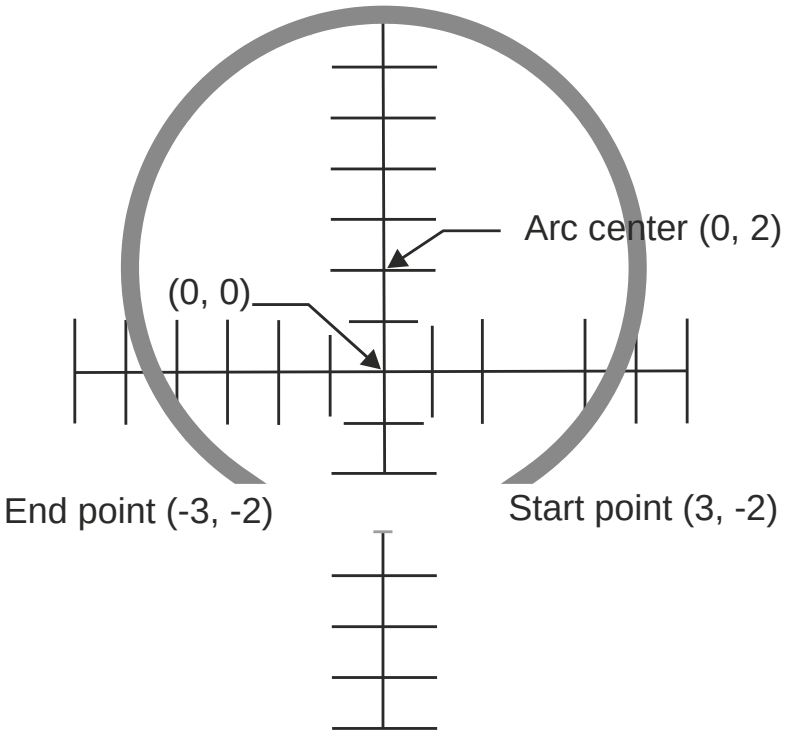


14. Single quadrant mode example: resulting image

4.5.8 **Example: Multi Quadrant Mode**

Syntax	Comments

`X300Y-200D02*`
`G75*`
`G03*`
`X-300Y-200I-300J400D01*`
	Set the current point to (3, -2)
	Set multi quadrant mode
	Set counterclockwise interpolation mode
	Create arc object counterclockwise to (-3,-2). The 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)



15. Multi quadrant mode example: resulting image

4.5.9 **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 4.5.1. 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 4.9.1.

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.

4.5.10 **Using G74 or G75 May 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* G01* X0Y600D02* G74* G02* X0Y600I500J0D01*	Set the current aperture to D10 aperture Set linear interpolation mode Set the current point to (0, 6) Set single quadrant mode Set clockwise circular interpolation mode Create arc object to (0, 6) with radius 5

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

Syntax	Comments
D10* G01* X0Y600D02* G75* G02* X0Y600I500J0D01*	Set the current aperture to D10 aperture Set linear interpolation mode Set the current point to (0, 6) Multi quadrant mode Set clockwise circular interpolation mode Create arc object to (0, 6) with center (5,6)

The image is a full circle.

Warning: It is mandatory to always specify quadrant mode (G74 or G75) if circular interpolation mode (G02 or G03) is used.

4.6 Region Mode (G36/G37)

4.6.1 **Region Overview**

A region is a graphics object defined by its contour(s) - see 2.6.

The G36 command turns region mode *on* and G37 turns it *off*. With region mode *on* the D01 and D02 commands 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 command is encountered the contour is considered finished. (Note that a D02 without effect on the current point, e.g. a `D02*`, still has the effect to finish the current contour.) A D02 is only allowed if the preceding contour is closed. The next D01 command starts a new contour. Thus an unlimited number of contours can be created between a single G36/G37 commands pair.

When G37 command is encountered region mode is turned off and a set of region graphics objects is created by filling all the newly created contours. Each contour is filled individually. The overall filled area is the union of the filled areas of each individual contour. The number of region objects created by a single G36/G37 pair is intentionally *not* specified. It may depend on the geometry of the contours - for example, two overlapping contours may be merged in a single region object.

A G37 is only allowed when all contours are properly closed. A G37 automatically finishes the last contour in the absence of a closing D02.

Using contours with horizontal or vertical fully coincident linear segments (see 2.6) it is possible to create holes in a region with cut-ins (see 4.6.11).

D01 and D02 are the *only* D code commands allowed in region mode; in other words D03 and Dnn (nn≥10) are *not* allowed. Extended codes are *not* allowed. The M02 (end-of-file) command is *not* allowed. However, G code commands *are* allowed – they are needed to control the interpolation modes.

A contour and its segments are not in themselves graphics objects –they define the regions which are the graphics objects. The attributes from the current aperture, if defined, are attached to the region objects. This is the only mechanism to attach aperture attributes are to regions.

 **Warning:** As the current aperture has no graphical effect in region mode it is easy to overlook that it still transfers its attributes. When using attributes be careful to set the current aperture correctly before issuing a G36.

 **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 4.6.12 and 4.6.14 for examples 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 clear polarity (LPC) and *not* by using cut-ins. See 4.6.10.

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

4.6.2 **G36 Command**

The syntax for the command to turn region mode on:

<G36 command> = G36*

Syntax	Comments
G36	Sets region mode graphics state parameter to ‘on’

Example:

```
G36*
```

4.6.3 **G37 Command**

The syntax for the command to disable region mode off:

<G37 command> = G37*

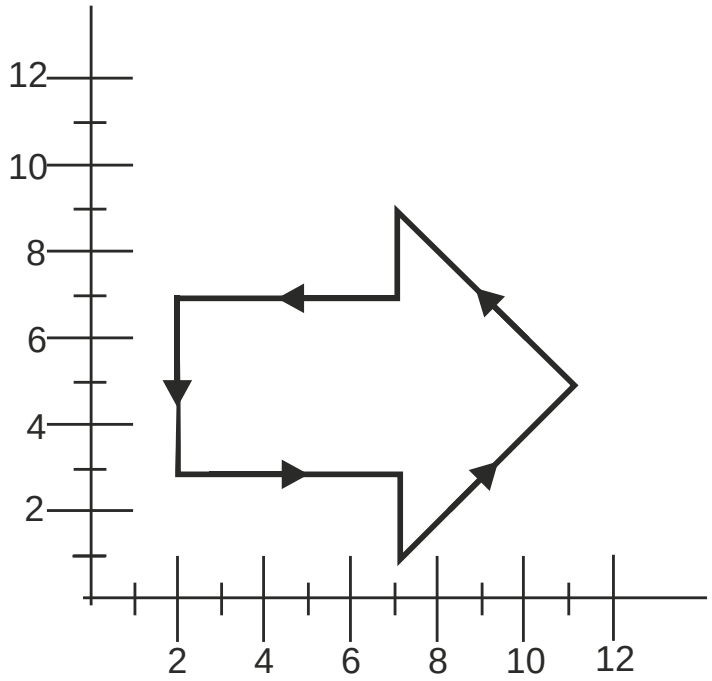
Syntax	Comments
G37	Sets region mode graphics state parameter to ‘off’ This creates the set of region graphics object by filling the contours created since the previous G36 command.

Example:

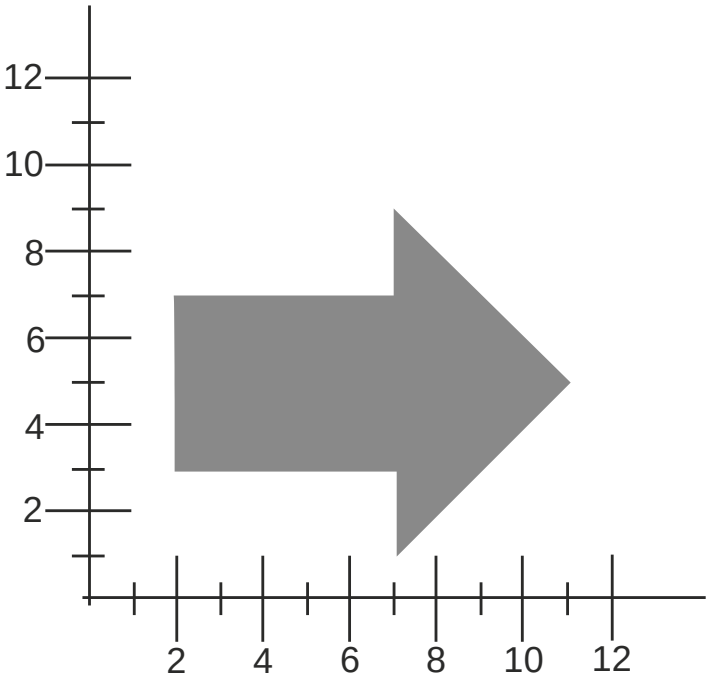
```
G37*
```

4.6.4 **Example: A Simple Contour**

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



16. Simple contour example: the segments



17. Simple contour example: resulting image

4.6.5 Example: How to Start a Single Contour

The first D01 starts the contour at the current point, independent of how the current point is set.

Below there are three examples of similar images; differences with the previous column are highlighted and explained in the last table row.

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 linear segment created by the operation: X300Y500D01*	Swap D02 and G36 commands. Exactly the same image.	Replace D02 by D01 command. The same contour is created. But the difference is that the additional draw object is added to the image by this operation: X5000Y5000D01*

4.6.6 Example: Use D02 to Start a Second Contour

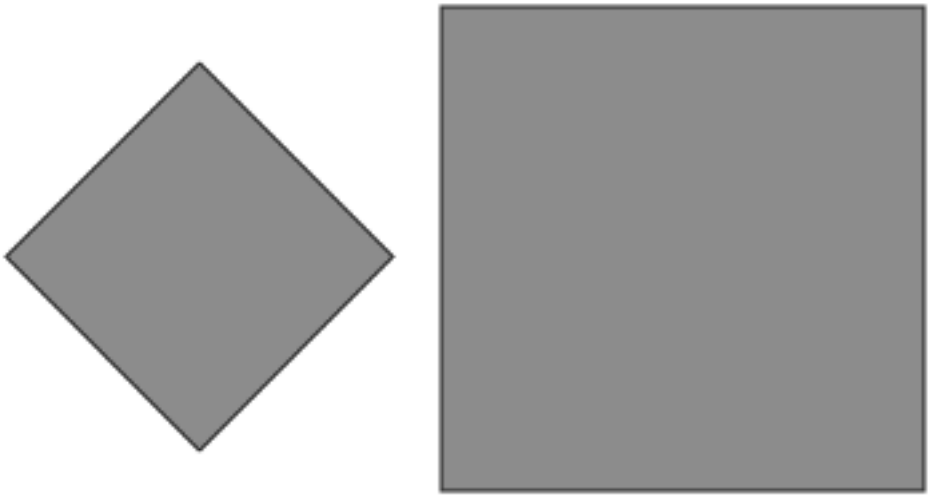
D02 command can be used to start the new contour. All the created contours are converted to regions when the command G37 is encountered. The example below creates two non-overlapping contours which are then converted into two regions.

Example:

```
G04 Non-overlapping contours*
%FSLAX24Y24*%
%MOMM*%
%ADD10C,1.00000*%
G01*
%LPD*%
G36*
X0Y50000D02*
Y100000D01*
X100000D01*
Y0D01*
X0D01*
Y50000D01*
X-10000D02*
```

```
X-50000Y10000D01*
X-90000Y50000D01*
X-50000Y90000D01*
X-10000Y50000D01*
G37*
M02*
```

This creates the following image:



18. 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.

4.6.7 **Example: Overlapping Contours**

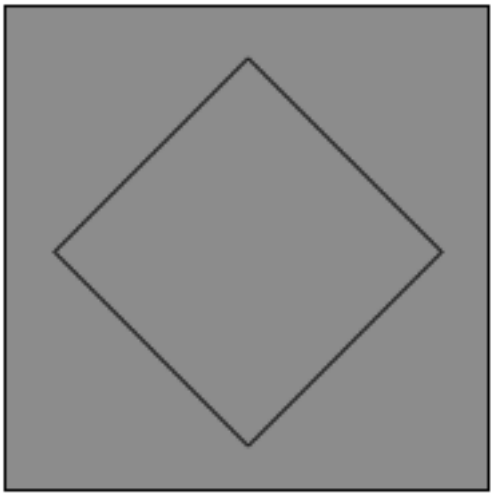
The example below creates two overlapping contours which are then converted into one region.

Example:

```
G04 Overlapping contours*
%FSLAX24Y24*%
%MOMM*%
%ADD10C,1.00000*%
G01*
%LPD*%
G36*
X0Y50000D02*
Y100000D01*
X100000D01*
Y0D01*
X0D01*
Y50000D01*
X10000D02*
X50000Y10000D01*
```

```
X90000Y50000D01*
X50000Y90000D01*
X10000Y50000D01*
G37*
M02*
```

This creates the following image:



19. 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. So the created region object is the same as would be defined by the first contour only.

4.6.8 **Example: Non-overlapping and Touching**

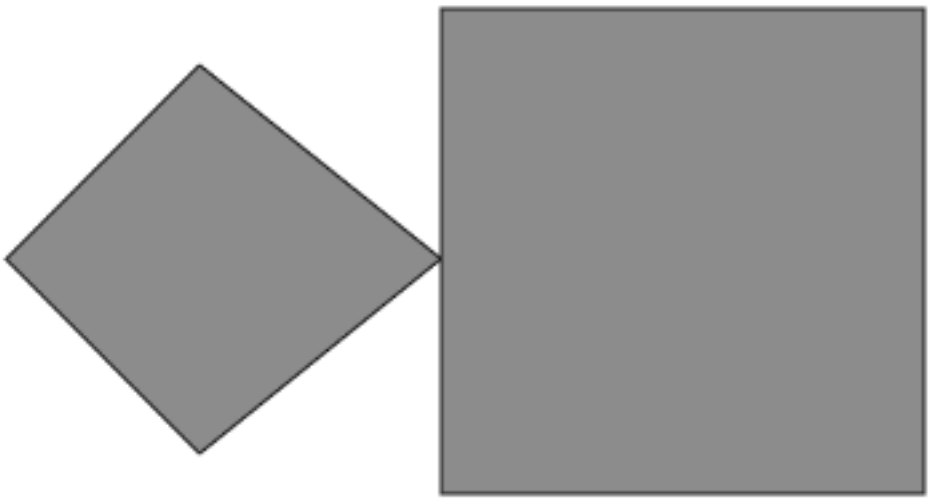
The example below creates two non-overlapping touching contours which are then converted into one region.

Example:

```
G04 Non-overlapping and touching*
%FSLAX24Y24*%
%MOMM*%
%ADD10C,1.00000*%
G01*
%LPD*%
G36*
X0Y50000D02*
Y100000D01*
X100000D01*
Y0D01*
X0D01*
Y50000D01*
```

```
D02*
X-50000Y10000D01*
X-90000Y50000D01*
X-50000Y90000D01*
X0Y50000D01*
G37*
M02*
```

This creates the following image:



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

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

4.6.9 **Example: Overlapping and Touching**

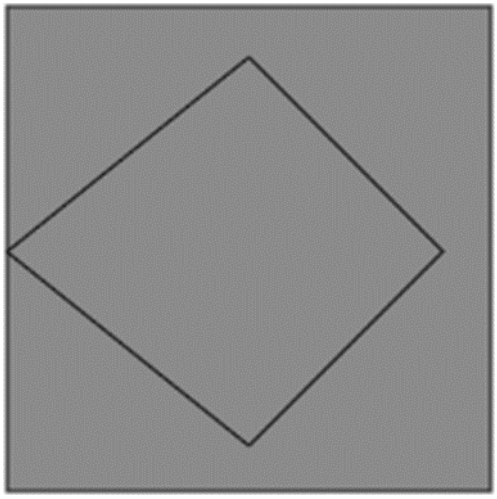
The example below creates two overlapping touching contours which are then converted into one region.

Example:

```
G04 Overlapping and touching*
%FSLAX24Y24*%
%MOMM*%
%ADD10C,1.00000*%
G01*
%LPD*%
G36*
X0Y50000D02*
Y100000D01*
X100000D01*
Y0D01*
X0D01*
Y50000D01*
D02*
```

```
X50000Y10000D01*
X90000Y50000D01*
X50000Y90000D01*
X0Y50000D01*
G37*
M02*
```

This creates the following image:



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

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

4.6.10 **Example: Using Polarity to Create Holes**

The recommended way to create holes in regions is by alternating dark and clear polarity, as illustrated in the following example. Initially the polarity mode is dark. A big square region is generated. The polarity mode is set to clear and a circular disk is added to the object stream; the disk is cleared from the image and creates a round hole in the big square. Then the polarity is set to dark again and a small square is added to the stream, darkened the image inside the hole. The polarity is set to clear again and a small disk added, clearing parts of the big and the small squares.

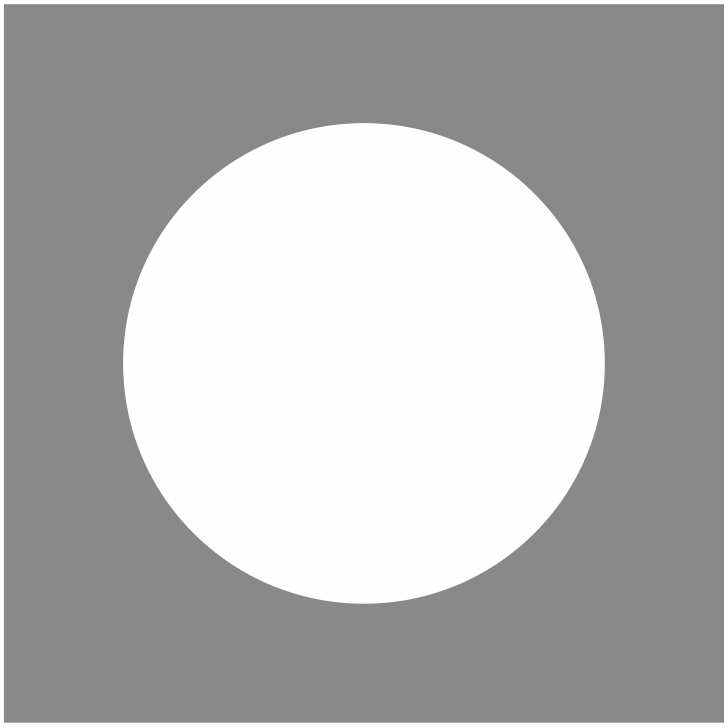
Example:

```
G04 This file illustrates how to use polarity to create holes*
%FSLAX25Y25*%
%MOMM*%
G01*
G04 First object: big square - dark polarity*
%LPD*%
G36*
X2500000Y2500000D02*
X17500000D01*
Y17500000D01*
X2500000D01*
```

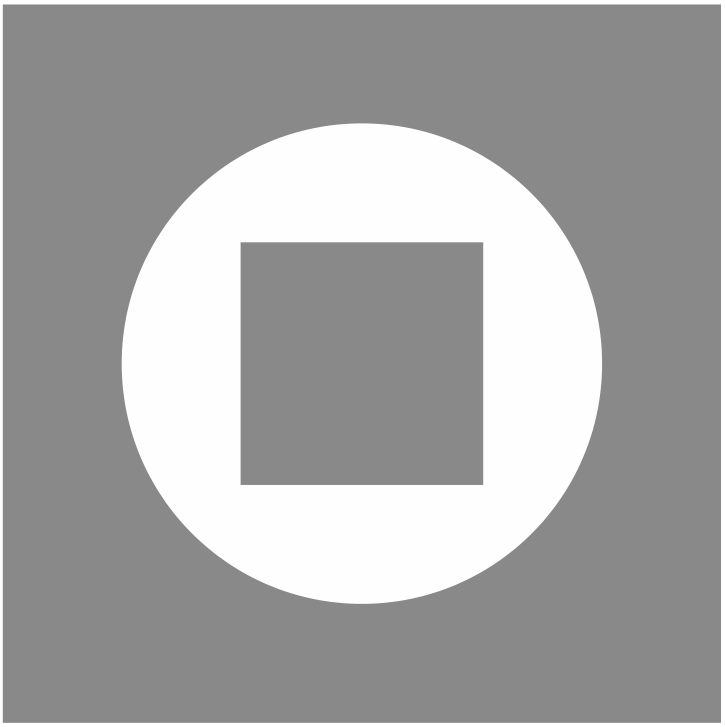
```
Y2500000D01*
G37*
G04 Second object: big circle - clear polarity*
%LPC*%
G36*
G75*
X5000000Y10000000D02*
G03*
X5000000Y10000000I5000000J0D01*
G37*
G04 Third object: small square - dark polarity*
%LPD*%
G36*
X7500000Y7500000D02*
X12500000D01*
Y12500000D01*
X7500000D01*
Y7500000D01*
G37*
G04 Fourth object: small circle - clear polarity*
%LPC*%
G36*
G75*
X11500000Y10000000D02*
G03*
X11500000Y10000000I2500000J0D01*
G37*
M02*
```

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

22. Resulting image: first object only



23. Resulting image: first and second objects



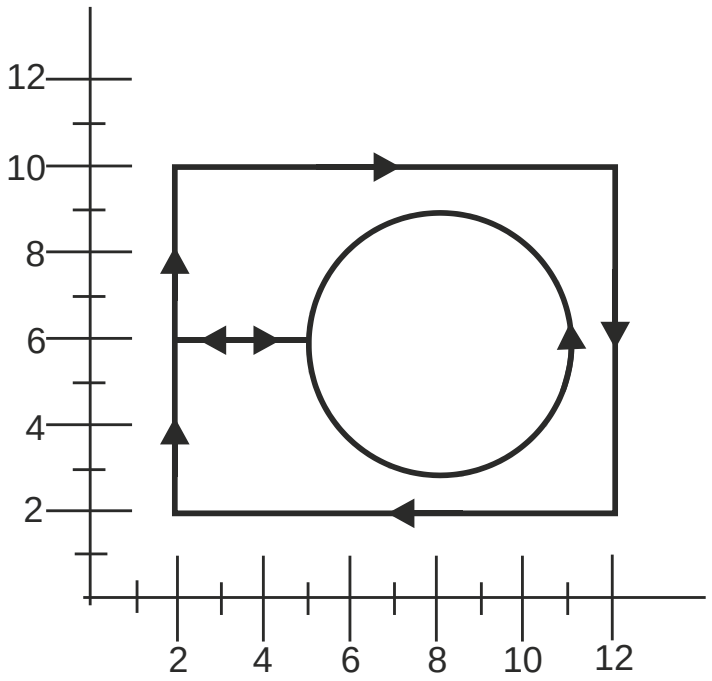
24. Resulting image: first, second and third objects

25. Resulting image: all four objects

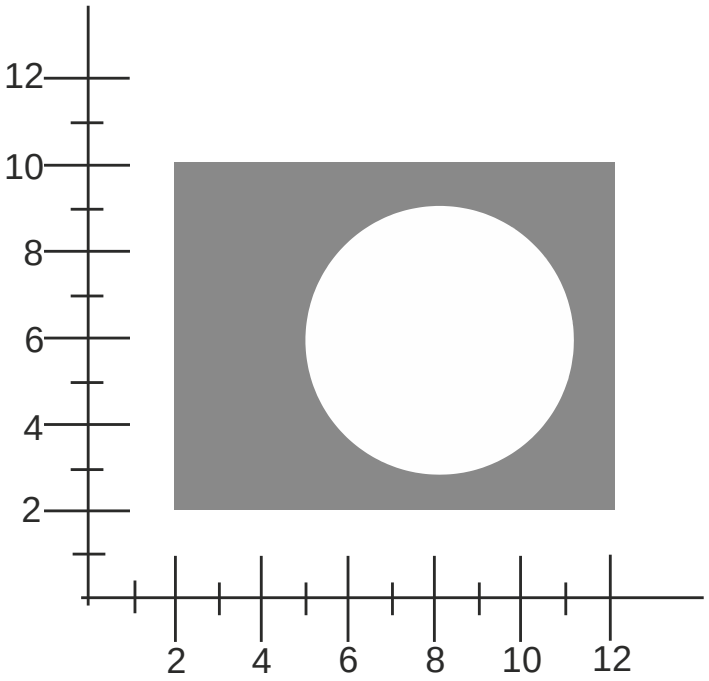
4.6.11 **Example: A Simple Cut-in**

The example below illustrates how a simple cut-in can be used to create a hole in a region. The coinciding contour segments must follow the requirements defined in 2.6.

Syntax	Comments
%FSLAX24Y24*% … G75* G36* X20000Y100000D02* G01* X120000D01* Y20000D01* X20000D01* Y60000D01* X50000D01* G03* X50000Y60000I30000J0D01* G01* X20000D01* Y100000D01* G37*	The 4 decimal positions is rather low precision. It is only used to improve readability of this example. Always use a precision of 5 decimal positions or more in production files. Set multi quadrant mode Enable region mode Set the current point to (2,10) Set linear interpolation mode Create linear contour segment to (12,10) Create linear contour segment to (12, 2) Create linear contour segment to (2, 2) Create linear contour segment to (2, 6) Create linear contour segment to (5, 6),1st fully coincident segment Set counterclockwise circular interpolation mode Create counterclockwise circle with radius 3 Set linear interpolation mode Create linear contour segment to (2, 6), 2nd fully coincident segment Create linear contour segment to (2, 10) Create the region by filling the contour



26. Simple cut-in: the segments



27. Simple cut-in: the image

4.6.12 **Example: Power and Ground Planes**

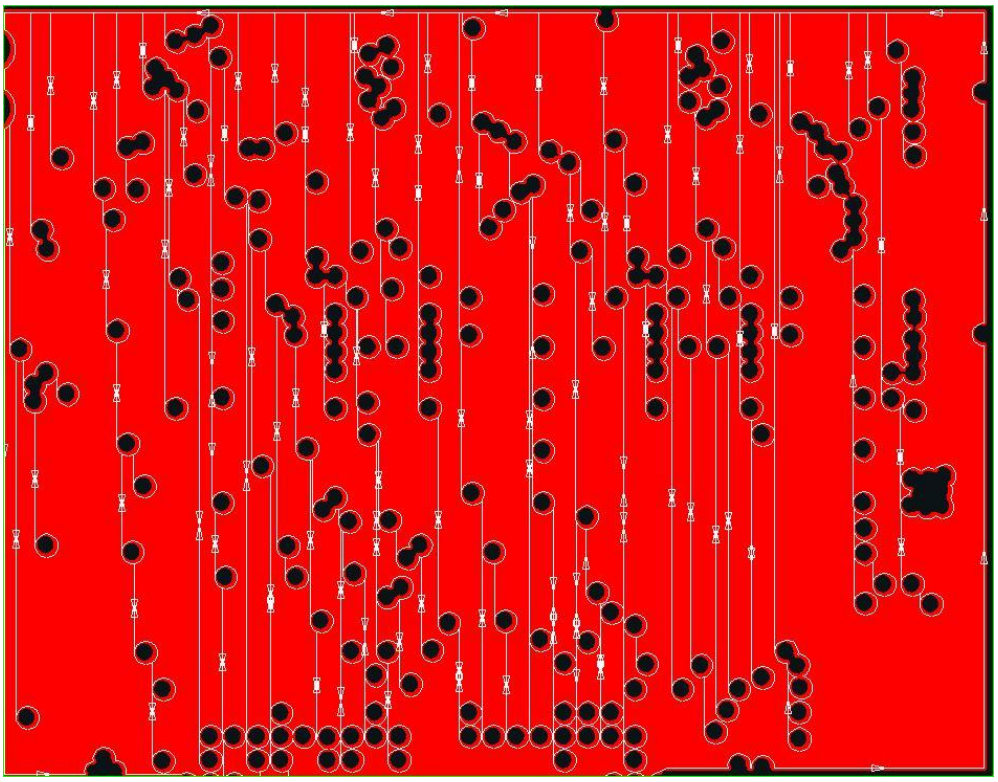
The proper way to construct power and ground planes is as follows. First create the copper pour with a region in dark polarity (LPD), and then erase the clearances by switching to clear polarity (LPC) and flash the anti-pads:

Example:

```
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 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.



*28. 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 an error.

4.6.13 **Example: Fully Coincident Segments**

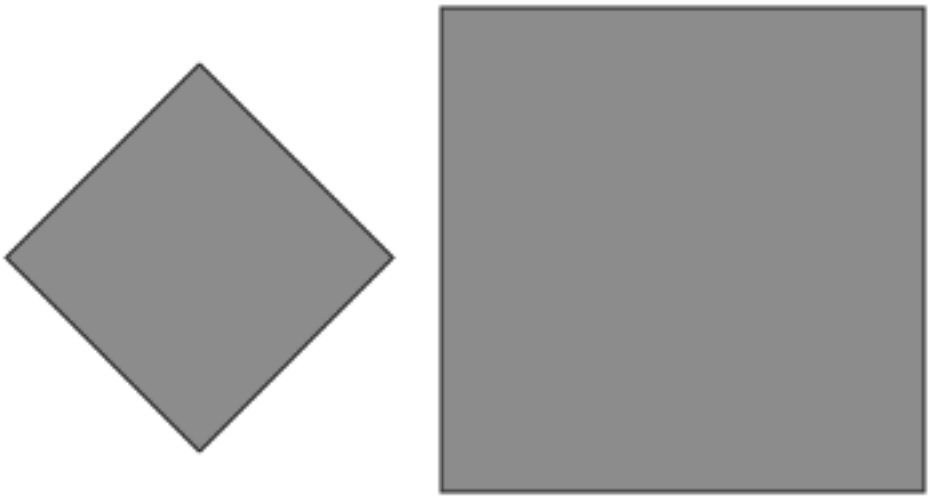
The first example below illustrates how one contour may result in two regions. This happens because there are two fully coincident linear segments which give the gap between filled areas.

Example:

```
G04 ex1: non overlapping*
%FSLAX24Y24*%
%MOMM*%
%ADD10C,1.00000*%
G01*
%LPD*%
G36*
X0Y50000D02*
Y100000D01*
```

```
X100000D01*
Y0D01*
X0D01*
Y50000D01*
G04 first fully coincident linear segment*
X-10000D01*
X-50000Y10000D01*
X-90000Y50000D01*
X-50000Y90000D01*
X-10000Y50000D01*
G04 second fully coincident linear segment*
X0D01*
G37*
M02*
```

This creates the following image:



29. Fully coincident segments in contours: two regions

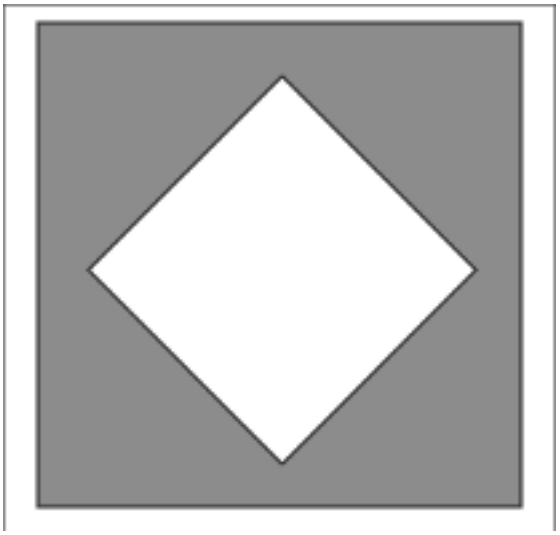
The second example illustrates how one contour allows creating region with hole.

Example:

```
G04 ex2: overlapping*
%FSLAX24Y24*%
%MOMM*%
%ADD10C,1.00000*%
G01*
%LPD*%
G36*
X0Y50000D02*
Y100000D01*
X100000D01*
Y0D01*
```

```
X0D01*
Y50000D01*
G04 first fully coincident linear segment*
X10000D01*
X50000Y10000D01*
X90000Y50000D01*
X50000Y90000D01*
X10000Y50000D01*
G04 second fully coincident linear segment*
X0D01*
G37*
M02*
```

This creates the following image:



30. Fully coincident segments in contours: region with hole

4.6.14 **Example: Valid and Invalid Cut-ins**

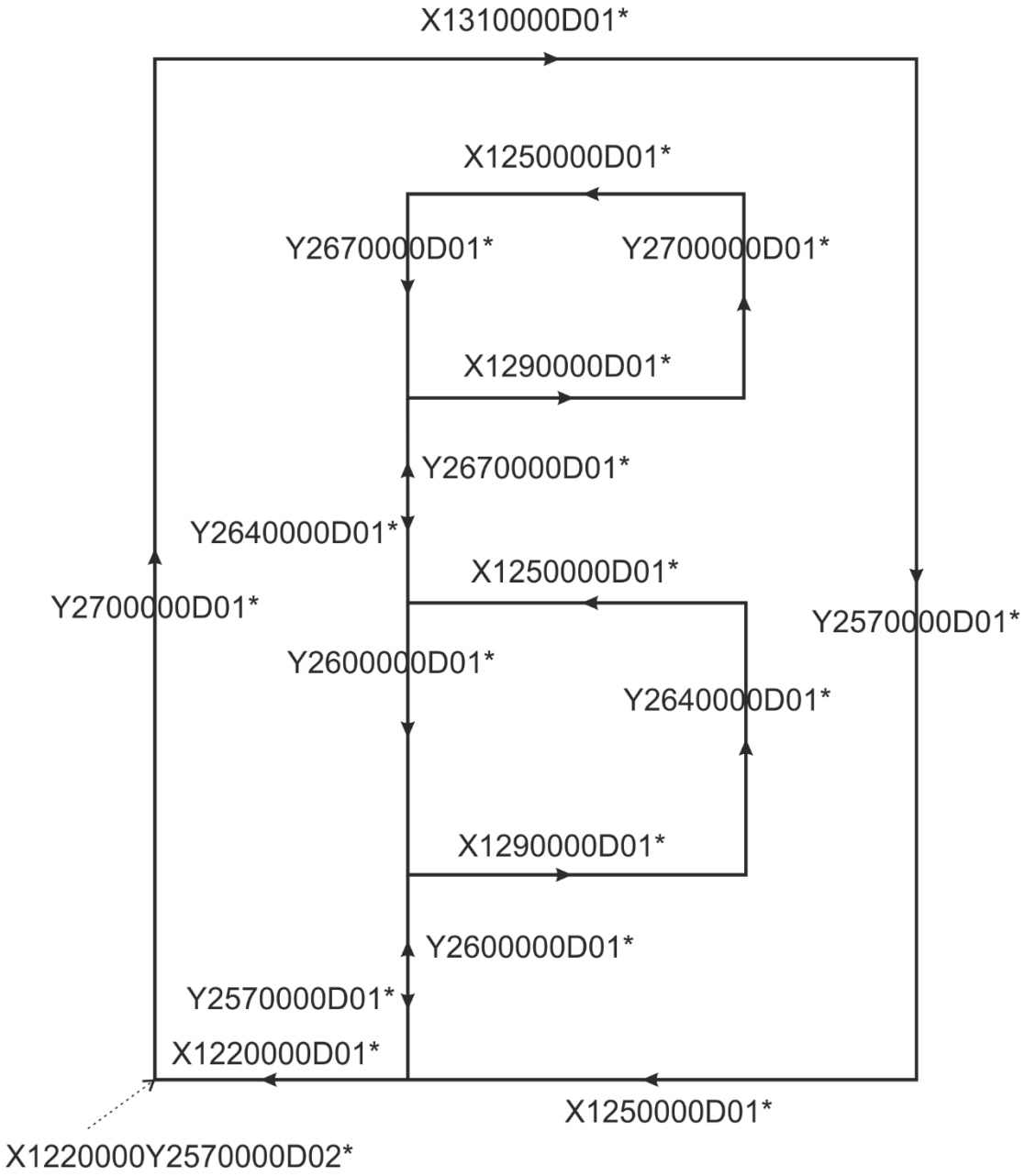
Contours with cut-ins are susceptible to rounding problems: when the vertices move due to the rounding the contour may become self-intersecting. This may lead to unpredictable results. The first example below is a cut-in with valid fully coincident segments, where linear segments which are on top of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

Example:

```
G36*
X1220000Y2570000D02*
G01*
Y2720000D01*
X1310000D01*
Y2570000D01*
X1250000D01*
Y2600000D01*
```

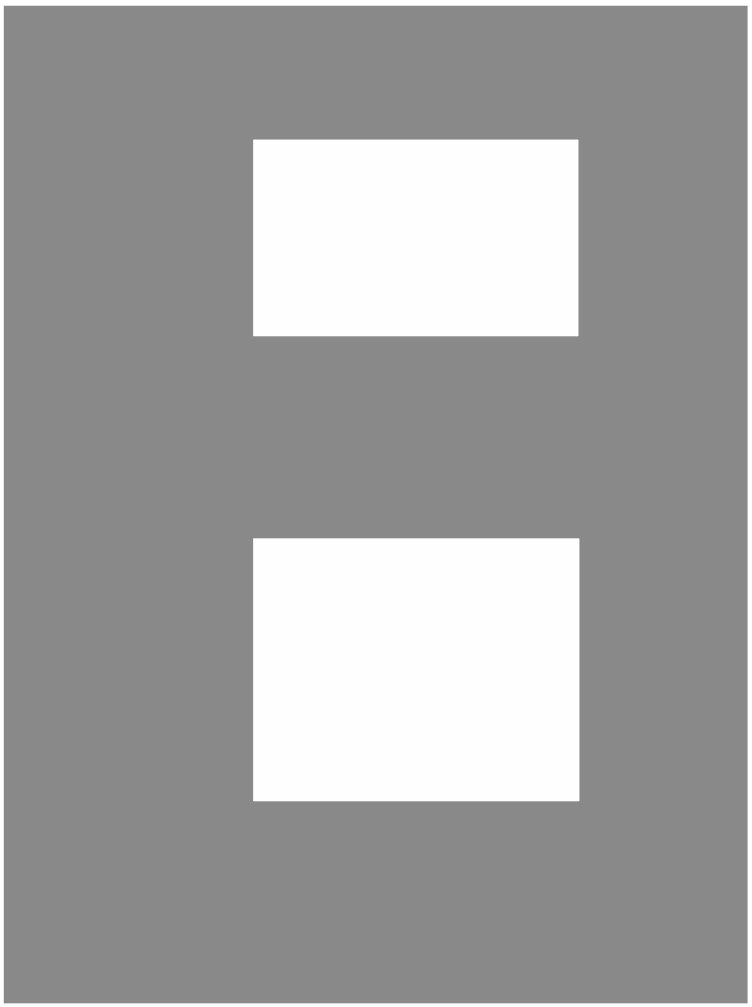
```
X1290000D01*
Y2640000D01*
X1250000D01*
Y2670000D01*
X1290000D01*
Y2700000D01*
X1250000D01*
Y2670000D01*
Y2640000D01*
Y2600000D01*
Y2570000D01*
X1220000D01*
G37*
```

This results in the following contour:



31. Valid cut-in: fully coincident segments

This creates the following image:



32. Valid cut-in: resulting image

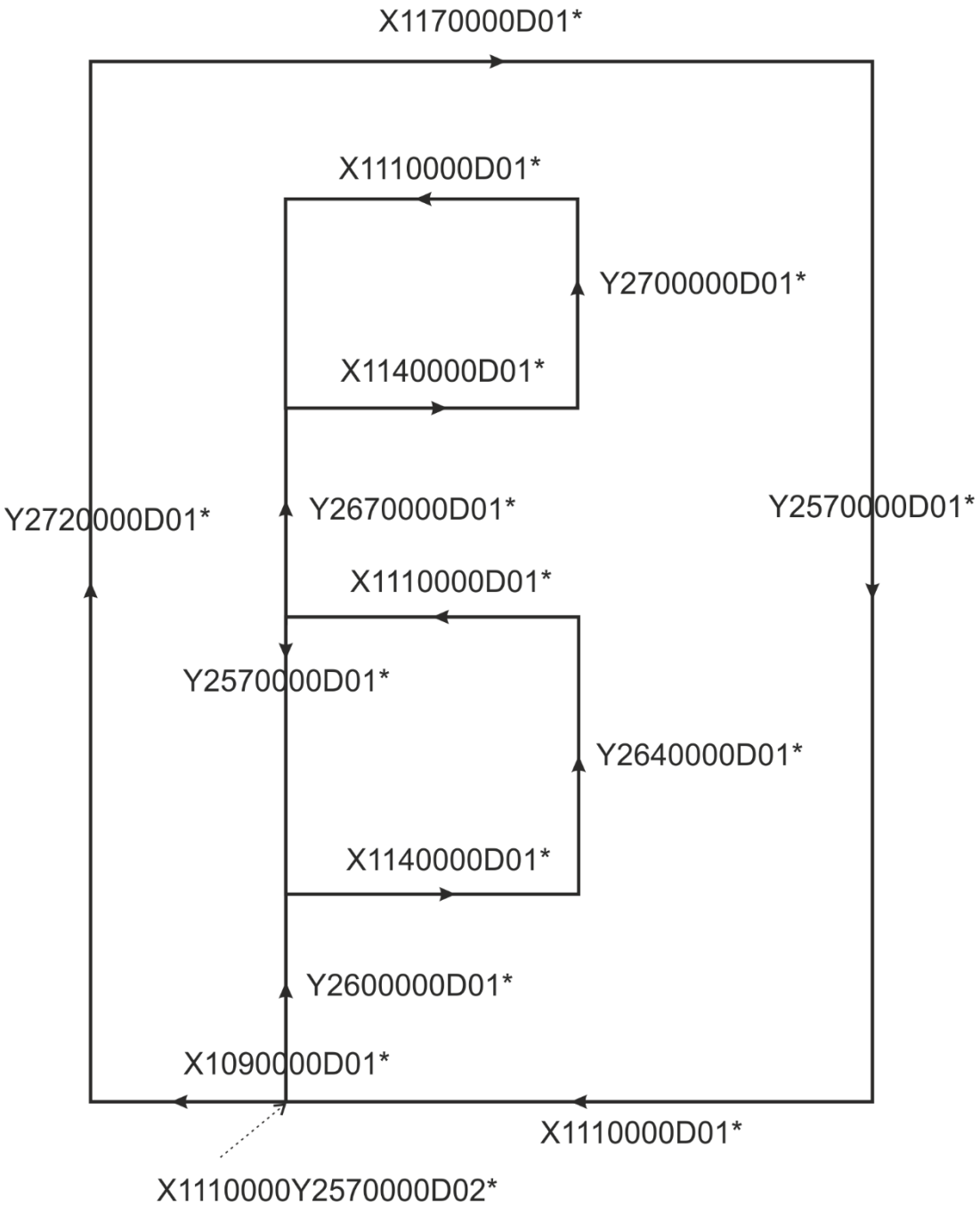
The next example attempts to create the same image as the first example from above, but it is *invalid* due to the use of invalid partially coinciding segments (see the description of a valid contour in 2.6). The number of linear segments has been reduced by eliminating vertices between collinear segments, creating invalid overlapping segments. This construction is *invalid*. It is prohibited is because it is not robust and hard to handle: when the vertices move slightly due to rounding, the segments that were on top of one another may become intersecting, with unpredictable results.

Example:

```
G36*
X1110000Y2570000D02*
G01*
Y2600000D01*
X1140000D01*
Y2640000D01*
X1110000D01*
Y2670000D01*
X1140000D01*
Y2700000D01*
X1110000D01*
```

```
Y2570000D01*
X1090000D01*
Y2720000D01*
X1170000D01*
Y2570000D01*
X1110000D01*
G37*
```

This results in the following contour:



33. Invalid cut-in: overlapping segments

4.7 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.

<G04 command> = G04<Comment content>*

The <Comment content> must follow the syntax for strings in section 3.6.6.

Example:

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

Comments cannot start with " #@!". This is reserved for future use as standardized comments.

4.8 End-of-file (M02)

The M02 command 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:

<M02 command> = M02*

 **Example:**

```
M02*
```

 **Warning:** M02 command is not allowed if the region mode is enabled. If a G36 command was used to enable region mode then a G37 command must be used before M02 to explicitly disable region mode.

4.9 Coordinate Format (FS)

Coordinate values are expressed as absolute coordinates.

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

A coordinate number in a Gerber file is represented by a sequence of digits without any separator between integer and decimal parts of the number. The integer and decimal parts are specified by their lengths in a coordinate number. The FS command defines the lengths of the integer and decimal parts for all coordinate numbers in the file.

4.9.1 Coordinate Format

The coordinate format specifies the number of integer and decimal places in a coordinate number. For example, the “24” format specifies 2 integer and 4 decimal places. The number of decimal places must be 4, 5 or 6. The number of integer places must be not more than 6. Thus the longest representable coordinate number is ‘nnnnnn.nnnnnn’. The same format must be defined for X and Y. Signs in coordinates are allowed; the ‘+’ sign is optional.

The unit in which the coordinates are expressed is set by the MO command (see 4.10).

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.

Coordinate numbers are integers. Explicit decimal points are not allowed.

A coordinate number must have at least one character. Zero therefore must be encoded as “0”.

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

 **Warning:** Using less than 4 decimal places is deprecated. 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.

If the FS command defines N places for integer part and M for decimal part it means the maximum allowed length of a coordinate number is N+M. To interpret the coordinate string, it is first padded with zero’s in front until its length is equal to N+M. And then first N digits are interpreted as the integer part, and remaining M digits are interpreted as the decimal part.

For example, with the “24” coordinate format, “015” is padded to “000015” and therefore represents 0.0015.

4.9.2 FS Command

The syntax for the FS command is:

<FS command> = FSLAX<Format>Y<Format>*

Syntax	Comments
FS	FS for Format Specification
LA	The predefined characters necessary for backwards compatibility (see 7.4 for more details)
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 ≤ 6) M - number of decimal positions in a coordinate number (4 ≤ M ≤ 6)

4.9.3 **Examples**

Syntax	Comments
`%FSLAX25Y25*%`	Coordinates has 2 integer and 5 decimal positions for both axes.

4.10 Unit (MO)

The MO (Mode) 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 is mandatory and must be used only once at the beginning of a file, before the first use of coordinate data. Normally MO command follows immediately after FS command (see 4.9).

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

The syntax for the MO command is:

<MO command> = MO(IN|MM)*

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

Examples:

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

4.11 Aperture Definition (AD)

4.11.1 **AD Command**

The AD command creates an aperture and puts it into apertures dictionary. It starts with 'AD' letters, followed by

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

The D-code identifies the aperture. The Dnn command uses the D-code to select it as the current aperture (see 4.3).

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 – thus apertures are uniquely identified by their D-code.

The syntax 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' is the command code 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> is either a standard aperture template name (C, R ,O or P – see 4.12) or a name of a macro aperture template previously defined by an AM command (see 4.13). AD command uses the name to find the referenced aperture template in the aperture templates dictionary (see 2.2).

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.

Note: Since this command is extended code command it shall be enclosed in a pair of '%' characters (see 3.5.3).

Example:

```
%ADD10C,.025*%
%ADD10C,0.5X0.25*%
```

4.11.2 Zero-size Apertures

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

Graphics objects created with a zero-size circular aperture are valid objects. They do not affect the image. Attributes can be attached to zero-objects. Zero-objects can be used to provide meta-information such as reference points or an outline

Warning: Only add zero-objects to provide meta-information. Certainly do not abuse a zero-object to indicate the *absence* of an object, e.g. by flashing a zero-size aperture to indicate the absence of a flash. Needless zero-objects are *not* allowed as they direct the reader to look for meta-information that is not there. If there is nothing there, put nothing.

4.11.3 Examples

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

4.12 Standard Aperture Templates

4.12.1.1 Circle

The syntax of the circle standard aperture template:

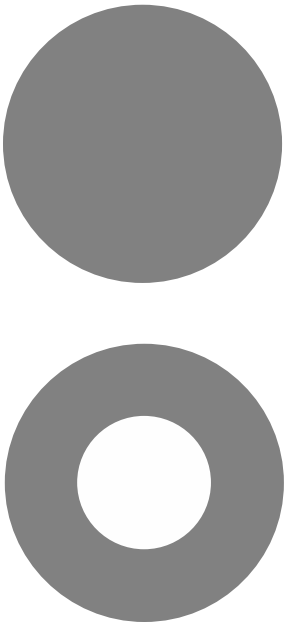
C,<Diameter>[X<Hole diameter>]

Syntax	Comments
C	Indicates that this is a circle aperture
<Diameter>	Circle diameter, a decimal ≥0
<Hole diameter>	Diameter of a round hole. If missing the aperture is solid See section 4.12.1.5 for more details.

Examples:

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

These commands define the following apertures:



34. Circles

4.12.1.2 Rectangle

The syntax of the rectangle or square standard aperture template:

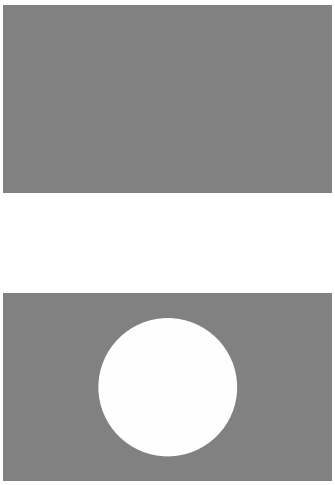
R,<X size>X<Y size>[X<Hole diameter>]

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 diameter>	Diameter of a round hole. If missing the aperture is solid See section 4.12.1.5 for more details.

Examples:

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

These commands define the following apertures:



35. Rectangles

4.12.1.3 Obround

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

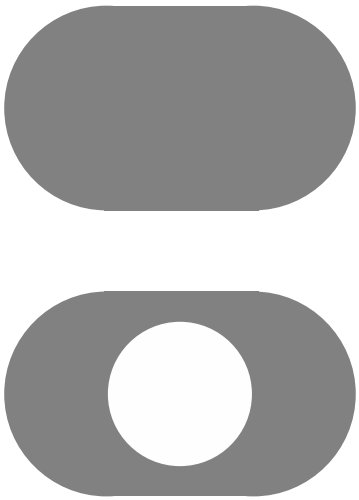
O,<X size>X<Y size>[X<Hole diameter>]

Syntax	Comments
O	Indicates that this is an obround aperture
<X size> <Y size>	X and Y sizes of enclosing box. Both must be decimals - See 3.6.2 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 diameter>	Diameter of a round hole. If missing the aperture is solid See section 4.12.1.5 for more details.

Example:

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

These commands define the following apertures:



36. Obrounds

4.12.1.4 Regular Polygon

The syntax of the polygon standard aperture template:

P,<Outer diameter>X<Number of vertices>[X<Rotation>[X<Hole diameter>]]

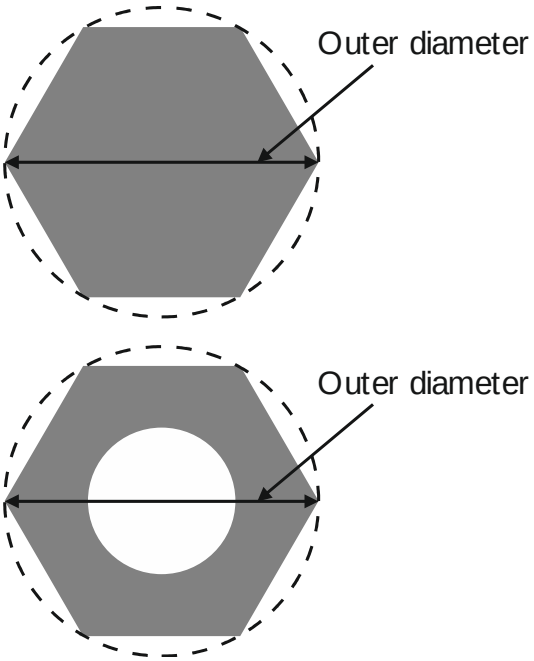
Syntax	Comments
P	Indicates that this is a polygon aperture
<Outer diameter>	Diameter of the circumscribed circle, i.e. the circle through the polygon vertices. Must be a decimal > 0
<Number of vertices>	Number of polygon vertices, ranging from 3 to 12
<Rotation>	A decimal number specifying the rotation in degrees 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 diameter>	Diameter of a round hole. If missing the aperture is solid See section 4.12.1.5 for more details. The hole modifier can be specified only after a rotation angle; set an angle of zero if the aperture is not rotated.

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

Examples:

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

These commands define the following apertures:



37. Polygons

4.12.1.5 Round Hole in Standard Apertures

Standard apertures may have a round 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.

For all standard apertures the round hole is defined by specifying its diameter as the last modifier: <Hole diameter>

If <Hole diameter> is omitted the aperture is solid. If present the diameter must be ≥ 0.

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

Example:

```
%FSLAX26Y26*%
%MOIN*%
%ADD10C,10X5*%
%ADD11C,1*%
G01*
%LPD*%
D11*
X-10000000Y-2500000D02*
X10000000Y2500000D01*
D10*
X0Y0D03*
M02*
```

38. Standard (circle) aperture with a hole above a draw

Note that the draw is visible through the hole.

4.13 Macro Aperture (AM)

The AM command creates a macro aperture template and adds it to the aperture template dictionary (see 2.2). A template is a parametrized shape. The AD command instantiates a template into an aperture by suppling values to the template parameters.

Templates of any shape or parametrization can be created. Multiple simple shapes called primitives can be combined in a single template. An aperture macro can contain variables whose actual values are defined by:

- Values provided by an AD command referencing the template
- Arithmetic expressions with other variables

The template is created by positioning primitives in a coordinate space. The origin of that coordinate space will be the origin of all apertures created with the state.

A template must be defined before the first AD that refers to it. The AM command can be used multiple times in a file.

4.13.1 **AM Command**

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 3.6.5 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 identifying the primitive and primitive-specific modifiers (e.g. center of a circle). All primitives are described in 4.13.4. The primitives are positioned in a coordinates system whose origin is the origin of the resulting apertures.
<Primitive code>	A code specifying the primitive (e.g. polygon).

Syntax	Comments
<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

Note: Each AM command must be enclosed in a pair of '%' characters (see 3.5.3).

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

4.13.2 **Exposure Modifier**

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 earlier in the same macro definition. Exposure off is typically used to create a hole in the aperture – see also 4.12.1.5. The erasing action of exposure off is limited to the macro definition in which it occurs.

Warning: When the macro aperture is flashed, the erased area does *not* clear the underlying graphics objects. Objects under removed parts remain visible.

Example:

```
%FSLAX26Y26*%
%MOIN*%
%AMSQUAREWITHHOLE*
21,1,10,10,0,0,0*
1,0,5,0,0*%
%ADD10SQUAREWITHHOLE*%
%ADD11C,1*%
G01*
%LPD*%
D11*
X-10000000Y-2500000D02*
X10000000Y2500000D01*
D10*
X0Y0D03*
M02*
```

39. Macro aperture with a hole above a draw

Note that the draw is still visible through the hole.

4.13.3 **Rotation Modifier**

All primitives can be rotated.

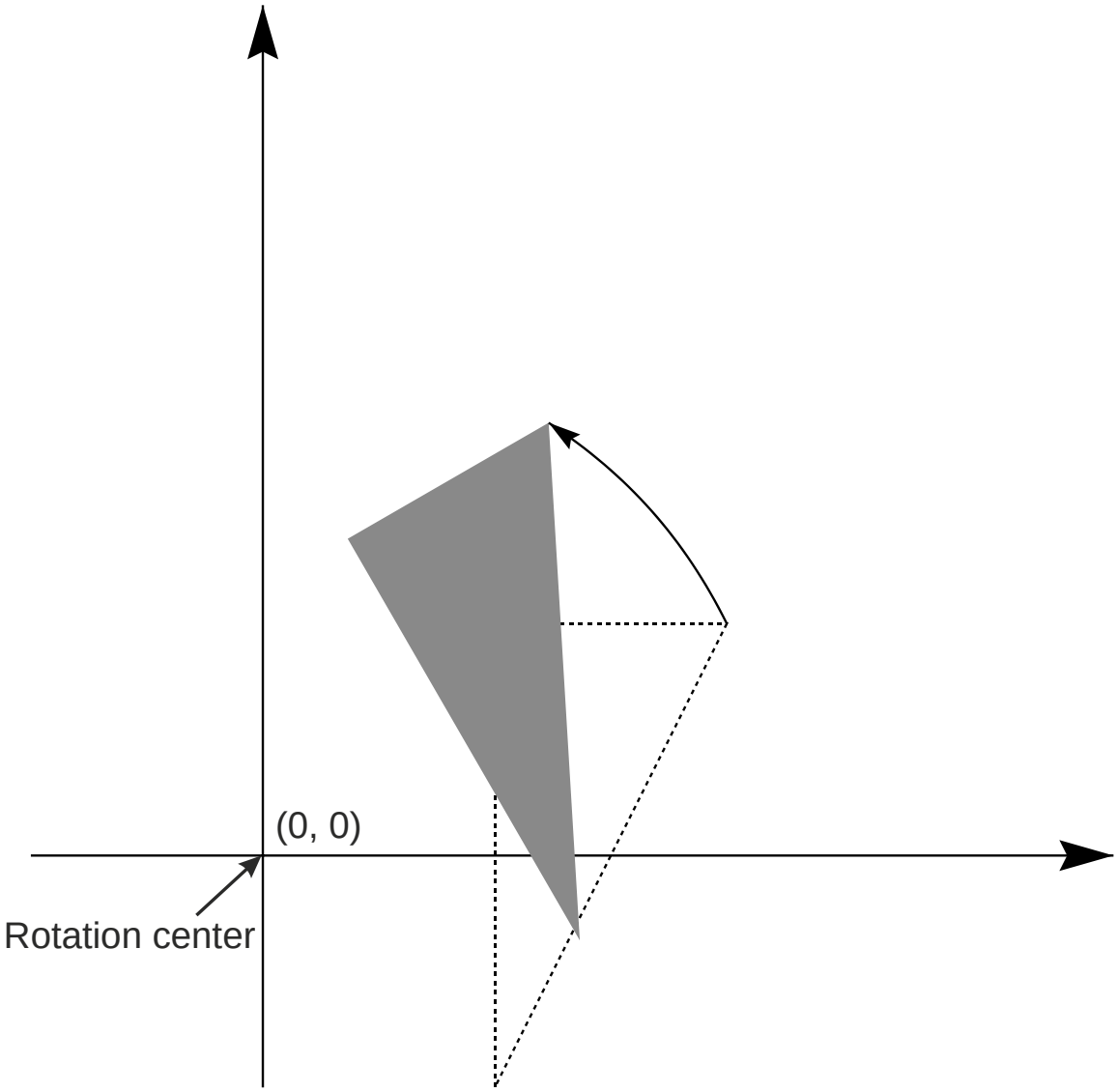
The rotation is always around the origin of the macro definition, i.e. its point (0, 0). Note that the origin may be different from the geometric center of the primitive – then the rotation is *not* around the geometric center.

A rotation angle is expressed by a decimal number, in degrees. A positive value means counterclockwise rotation, a negative value means clockwise rotation. The rotation angle of any primitive is defined by the rotation modifier that is the last in the list of the primitive modifiers.

The following AM command defines an aperture macro named ‘TRIANGLE_30’. The macro is a triangle rotated 30 degrees around the origin of the macro definition:

```
%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) Note that the last point is the same as the start point
30	Rotation angle is 30 degrees counterclockwise



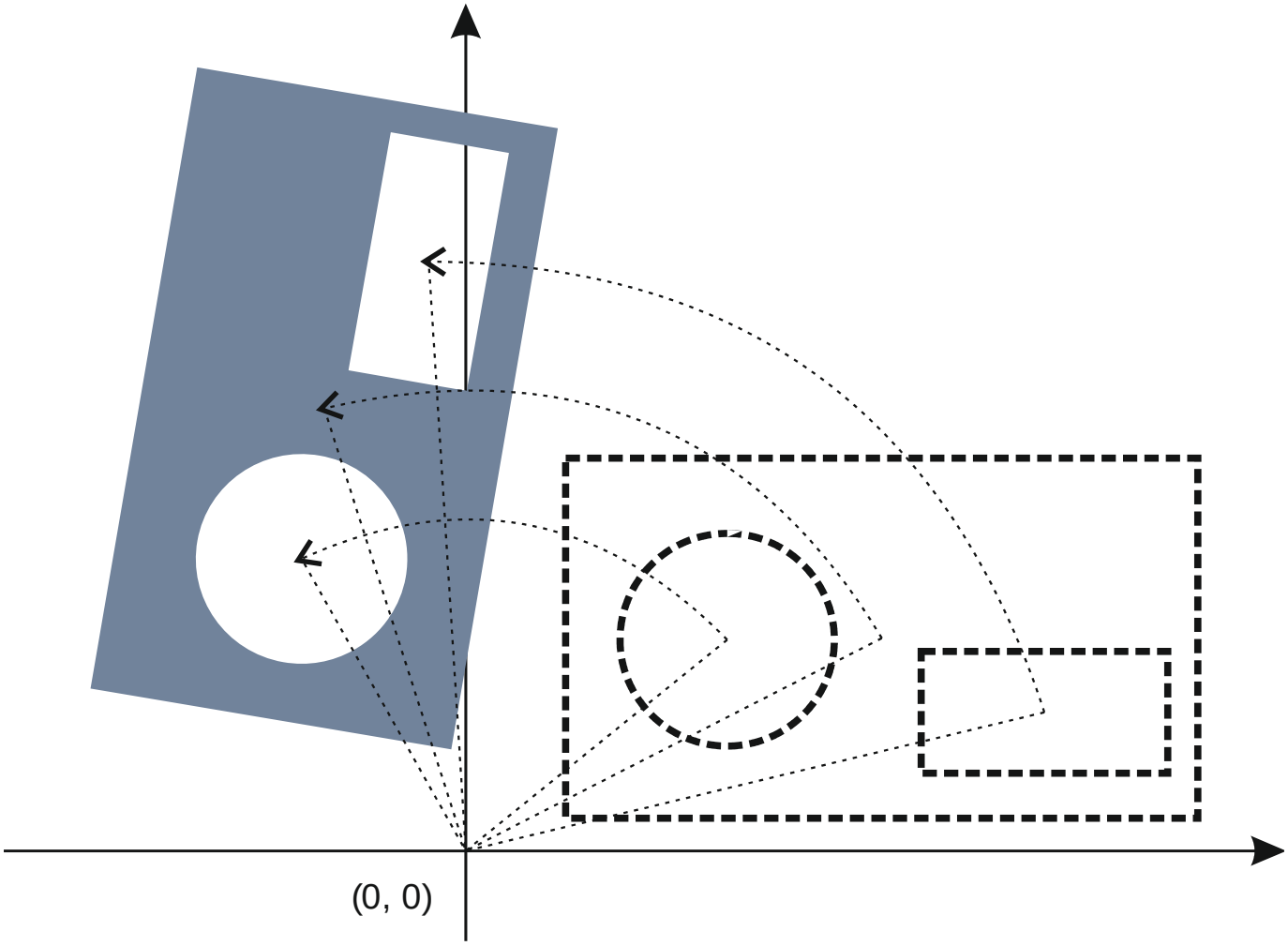
40. Rotated triangle

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

```
%ADD33AMTRIANGLE_30*%
```

To rotate a macro composed of several primitives it is then sufficient to rotate all primitives.

The picture below illustrates how the set of primitives are rotated together using the same rotation angle for each primitive in order to rotate the whole macro.



41. Rotation of an aperture macro composed of several primitives

4.13.4 **Primitives**

4.13.4.1 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.

Example:

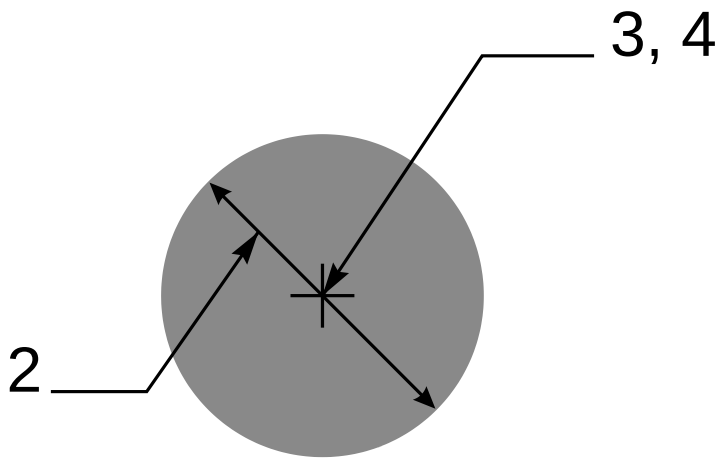
```
%AMRECTROUNDCORNERS*
0 Rectangle with rounded corners. *
0 Offsets $4 and $5 are interpreted as the *
0 offset of the flash origin from the pad center. *
0 First create horizontal rectangle. *
21,1,$1,$2-$3-$3,-$4,-$5,0*
0 From now on, use width and height half-sizes. *
$9=$1/2*
$8=$2/2*
0 Add top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

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

4.13.4.2 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	X coordinate of center position, a decimal
4	Y coordinate of center position, a decimal
5	Rotation angle. The rotation angle is specified by a decimal, in degrees. The primitive is rotated around the origin of the macro definition, i.e. the (0, 0) point of macro coordinates. The rotation modifier is optional. The default is no rotation. (Note that this modifier is optional only with the circle primitive. The reasons are historic. We recommend not using the default but always setting the angle explicitly.)



42. Circle primitive

Below there is the example of the AM command that uses the circle primitive.

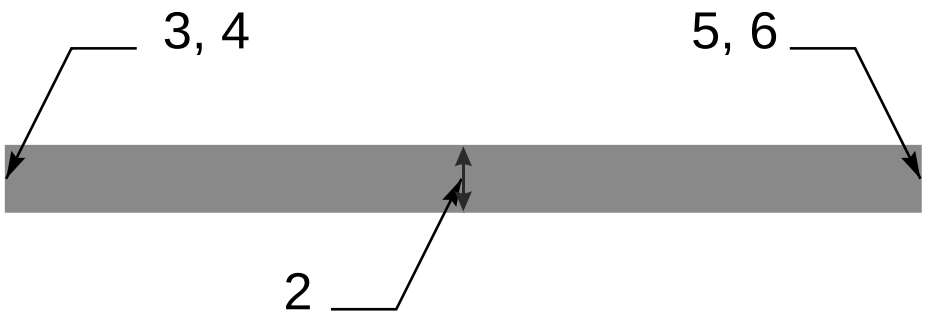
Example:

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

4.13.4.3 Vector Line, Primitive Code 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	X coordinate of start point, a decimal
4	Y coordinate of start point, a decimal
5	X coordinate of end point, a decimal
6	Y coordinate of end point, a decimal
7	Rotation angle of the vector line primitive The rotation angle is specified by a decimal, in degrees. The primitive is rotated around the origin of the macro definition, i.e. the (0, 0) point of macro coordinates



43. Vector line primitive

Below there is the example of the AM command that uses the vector line primitive.

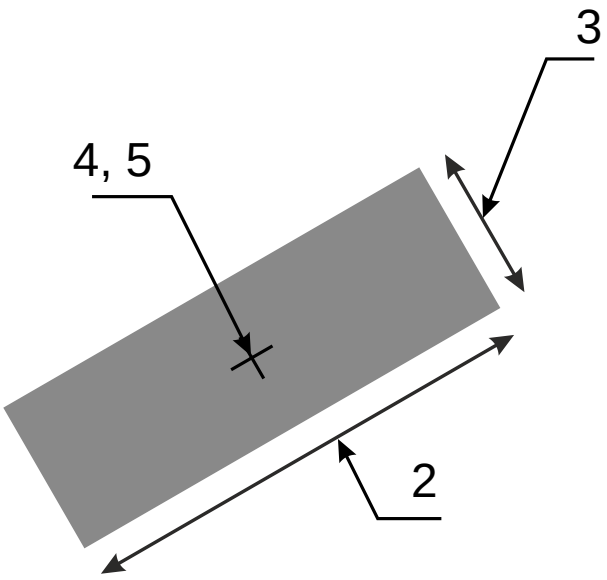
Example:

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

4.13.4.4 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	X coordinate of center point, a decimal
5	Y coordinate of center point, a decimal
6	Rotation angle The rotation angle is specified by a decimal, in degrees. The primitive is rotated around the origin of the macro definition, i.e. (0, 0) point of macro coordinates.



44. Center line primitive

Below there is the example of the AM command that uses the center line primitive.

Example:

```
%AMRECTANGLE*
21,1,6.8,1.2,3.4,0.6,30*%
```

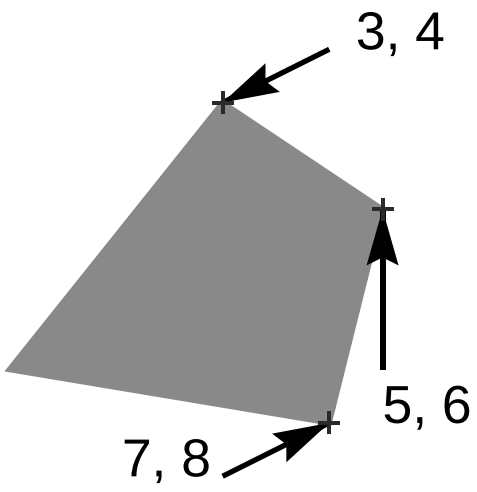
4.13.4.5 Outline, Primitive Code 4

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).

The outline of the primitive is actually the contour (see 2.6) that consists of linear segments only, so it must conform to all the requirements described for contours.

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	X and Y coordinates of the start point, decimals
5, 6	X and Y coordinates of subsequent point number 1, decimals
...	X and Y coordinates of further subsequent points, decimals
3+2n, 4+2n	X and Y coordinates of subsequent point number n, decimals Must be equal to coordinates of the start point
5+2n	Rotation angle of the outline primitive The rotation angle is specified by a decimal, in degrees. The primitive is rotated around the origin of the macro definition, i.e. the (0, 0) point of macro coordinates.



45. Outline primitive

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

Note: The maximum number of subsequent points n is 5000.

Below there is the example of the AM command that uses the outline primitive.

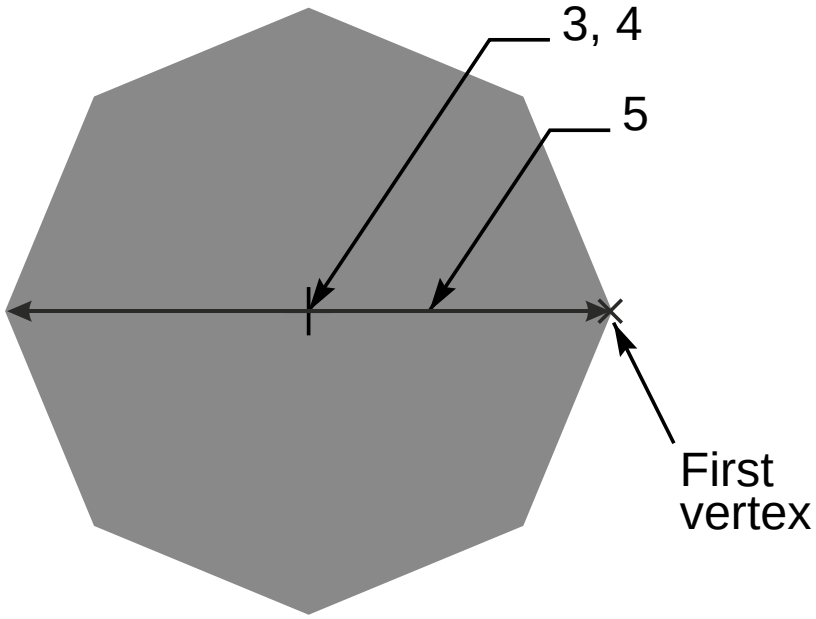
Example:

```
%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*%
```

4.13.4.6 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	X coordinate of center point, a decimal
4	Y coordinate of center point, a decimal
5	Diameter of the circumscribed circle, a decimal ≥ 0
6	Rotation angle of the polygon primitive The rotation angle is specified by a decimal, in degrees. The primitive is rotated around the origin of the macro definition, i.e. the (0, 0) point of macro coordinates. The first vertex is on the positive X-axis through the center point when the rotation angle is zero. Note: Rotation is only allowed if the primitive center point coincides with the origin of the macro definition.



46. Polygon primitive

Below there is the example of the AM command using the polygon primitive.

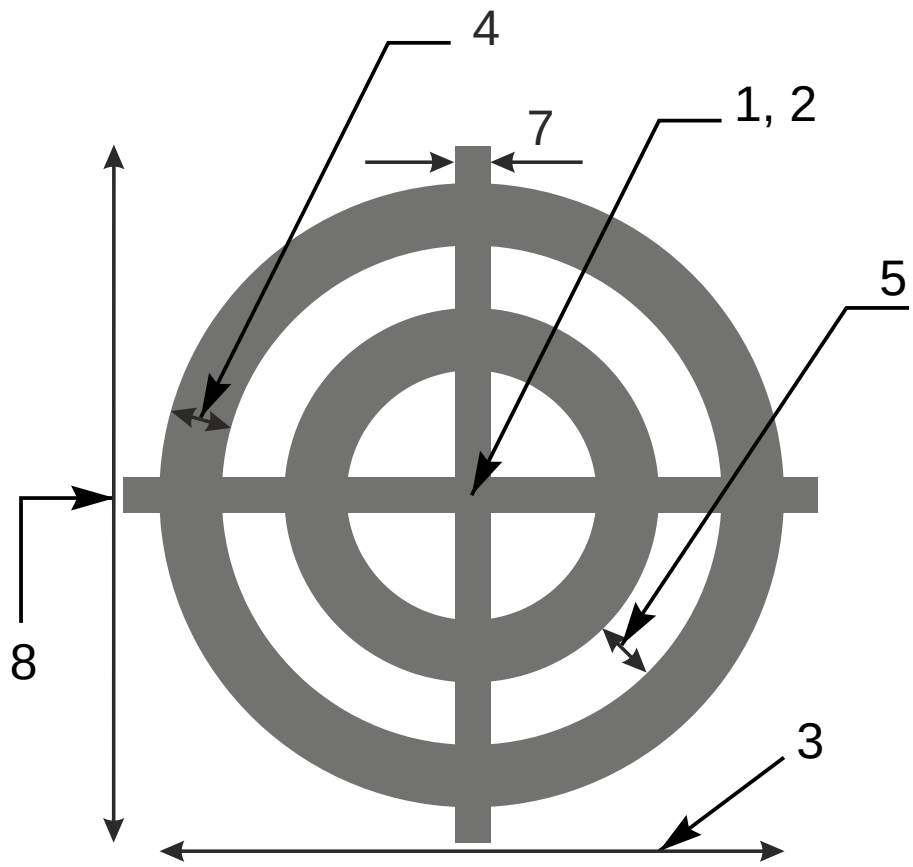
Example:

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

4.13.4.7 Moiré, Primitive Code 6

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

Modifier number	Description
1	X coordinate of center point, a decimal
2	Y coordinate of center point, a decimal
3	Outer diameter of outer concentric ring, a decimal ≥ 0
4	Ring thickness, a decimal ≥ 0
5	Gap between rings, a decimal ≥ 0
6	Maximum number of rings, an integer ≥ 0
7	Cross hair thickness, a decimal ≥ 0
8	Cross hair length, a decimal ≥ 0
9	Rotation angle of the moiré primitive The rotation angle is specified by a decimal, in degrees. The primitive is rotated around the origin of the macro definition, i.e. the (0, 0) point of macro coordinates Note: Rotation is only allowed if the primitive center point coincides with the origin of the macro definition



47. 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.

Below there is the example of the AM command that uses the moiré primitive.

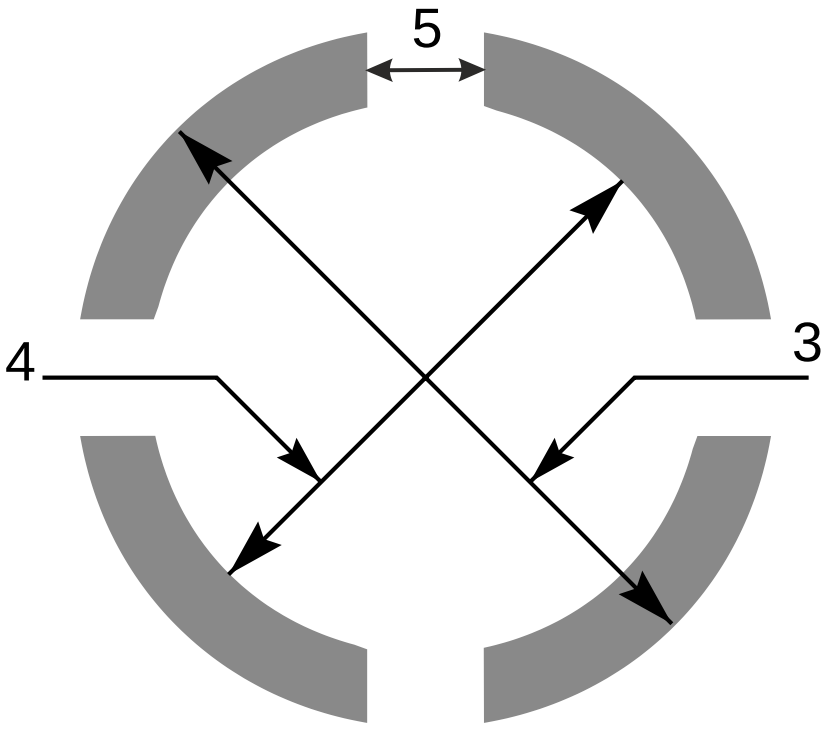
Example:

```
%AMMOIRE*
6,0,0,5,0.5,0.5,2,0.1,6,0*%
```

4.13.4.8 Thermal, Primitive Code 7

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

Modifier number	Description
1	X coordinate of center point, a decimal
2	Y coordinate of center point, a decimal
3	Outer diameter, a decimal > inner diameter
4	Inner diameter, a decimal ≥ 0
5	Gap thickness, a decimal < (outer diameter)/$\sqrt{2}$
6	Rotation angle of the thermal primitive The rotation angle is specified by a decimal, in degrees. The primitive is rotated around the origin of the macro definition, i.e. (0, 0) point of macro coordinates. The gaps are on the X and Y axes through the center when the rotation angle is zero Note: Rotation is only allowed if the primitive center point coincides with the origin of the macro definition.



48. Thermal primitive

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

4.13.5 **Syntax Details**

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 (see 3.4).

An aperture macro definition contains the macro name used to identify a template created by the macro. An AD command uses the macro name that is the name of the corresponding template in aperture templates dictionary.

An aperture macro definition also contains one or more aperture primitives described in 0. 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 3.6.5. A macro variable name must be a '$' character followed by an integer >0, for example $12.

Each AM command must be enclosed into a separate pair of '%' characters. Line separators between data blocks of a single command can be added to enhance readability. These line separators do not affect the definition of the macro.

4.13.5.1 Variable Values from an AD Command

An AM command can use variables whose actual values are provided by an AD command that instantiates the template. 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 the values of corresponding modifiers in the DONUTVAR macro.

4.13.5.2 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 ')'

Example:

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

Corresponding AD command could look like:

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

4.13.5.3 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 1.25 the following arithmetic expression can be used: $4=$1x1.25

Example:

```
%AMDONUTCAL*
1,1,$1,$2,$3*
$4=$1x1.25*
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,-$4,-$5,0*
$9=$1/2*
$8=$2/2*
```

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

4.13.6 **Examples**

4.13.6.1 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*%
```

4.13.6.2 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
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.

4.13.6.3 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.

4.14 Load Polarity (LP)

The LP command sets the polarity mode to either dark or clear. The polarity mode is a graphics state parameter and applies to all data until superseded by another LP command. This command can be used multiple times in a file. See also 2.9.

An example can be found in 4.6.10.

The syntax for the LP command is:

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

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

Examples:

Syntax	Comments
%LPD*%	Set the polarity mode graphics state parameter to dark
%LPC*%	Set the polarity mode graphics state parameter to clear

4.15 Step and Repeat (SR)

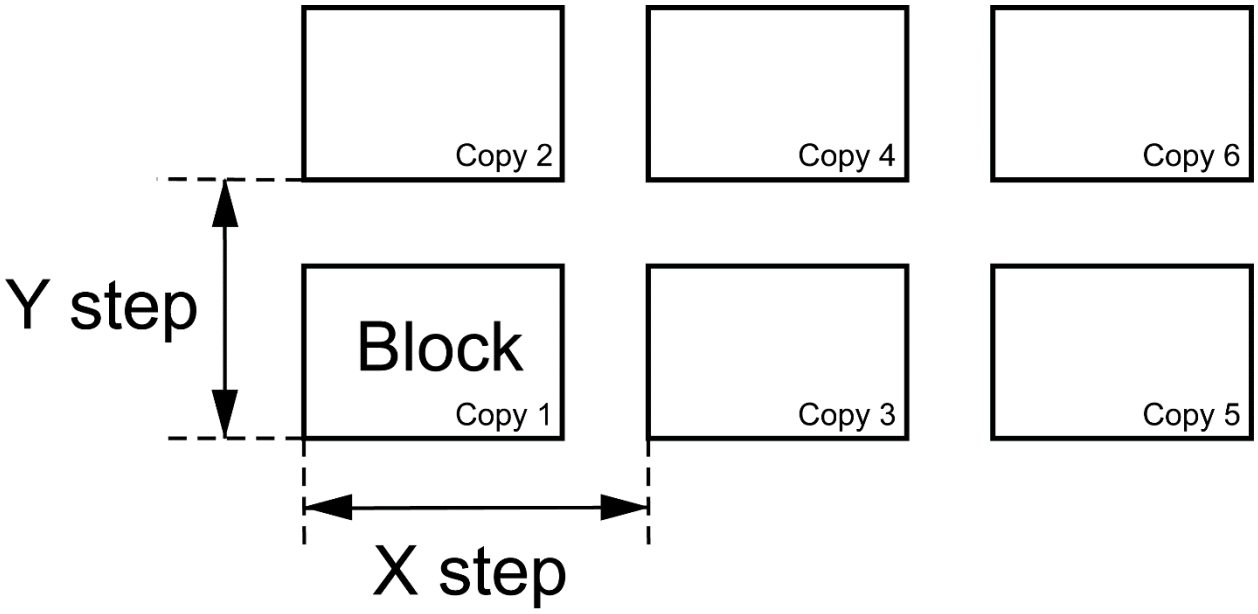
The purpose of the SR command is to replicate a set of graphics objects on the image plane without duplicating the commands creating the graphics objects.

The SR command with a number of repeats greater than 1 in either X or Y starts a *step & repeat statement*s. All subsequent commands are part of the step & repeat statement until it is terminated by an empty SR command `%SR*%`.

In a step & repeat statement all the graphics objects generated by the command stream are collected in a *block* instead of being added the object stream directly. When another SR command is encountered the block is step-repeated (replicated) in the image plane according to the parameters in the opening SR command. Each copy of the block contains identical graphics objects.

Example:

```
%SRX3Y2I5.0J4.0*%
G04 Block accumulation started. All the graphics*
G04 objects created below added to the block*
…
G04 Block accumulation is about to finish*
%SR*%
G04 The block is finished and replicated*
```



49. Blocks replication with SR command

The SR command can be used multiple times in a file. Each time it is encountered a previously accumulated block (if any) is closed and replicated; if the commands defines more than one repeat in any direction the new block is initiated.

The number of repeats and the step distances 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 repeats number for an axis is 1 it is recommended to set the step value to 0 for this axis.

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 – see 4.14).

Note that a block contains the graphics objects, not the Gerber source code. It is the graphics objects that are copied. The graphics objects in each copy are always identical, even if the graphics state is modified during the processing of the source. The reference point of a block is its origin. The origin of a block is (0, 0) point of the image global coordinate space.

A clear object in a block clears *all* objects beneath it, including objects outside the block. 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 prudent to avoid overlapping blocks containing clear and dark polarity objects. The image depends on the *order* in which objects are handled. This is not always correctly implemented in Gerber readers. (When all objects have identical polarity the behavior is straightforward and it is safe to use overlapping blocks.)

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>	Defines the number of times the block is repeated along the X axis <Repeats> is an integer ≥ 1
Y<Repeats>	Defines the number of times the block is repeated along the Y axis <Repeats> is an integer ≥ 1
I<Step>	Defines the step distance along the X axis <Step> is a decimal number ≥ 0, expressed in the unit of the MO command
J<Step>	Defines the step distance along the Y axis <Step> is a decimal number ≥ 0, expressed in the unit of the MO command

 **Note:** In addition to what is written in the comments column all the commands in the table below also close and repeat the previously accumulated block, if any.

Examples:

Syntax	Comments
%SRX2Y3I2.0J3.0*%	Opens a step & repeat statement and starts block accumulation When block accumulation is finished the block will be repeated 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*%	Opens step & repeat statement and starts block accumulation When block accumulation is finished the block will be repeated 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
%SR*%	Closes the step & repeat statement and repeats the previously accumulated block

4.17 Numerical Accuracy in Image Processing and Visualization

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 **Error! Reference source not found.**). A Gerber file specifies an image with infinite precision.

However Gerber file writers cannot assume that file 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 *can* assume.

4.17.1 Visualization

Gerber files are often used to *visualize* an image on a screen, a photoplotter, a direct imager. Visualization is unavoidably constrained by the limitations of the output device. Nonetheless, 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 taking into account neighboring objects. In other words, each graphics object is handled individually, regardless of context.

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 consequences:

- Gerber objects separated by a very small gap may touch in the visualized image.
- Gerber objects that touch or marginally overlap may be separated by a gap in the visualized image.
- Gerber objects smaller or thinner than the device resolution may totally disappear in the visualized image.
- When what is intended to be a single object is broken down in a number of elementary graphics objects, e.g. by stroking, and these elementary objects do not sufficiently overlap, the resulting image may *not* be solid - it may have internal holes or even break up in pieces. To avoid these effects the best and most robust approach is not to break up the single object at all: the Gerber format has powerful primitives to create almost any shape with a single graphics object or possible a succession of dark and clear objects.

Construct files robustly.

4.17.2 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 unavoidably accumulate. This means that all object positions can move and their sizes can vary. We call these changes a perturbation. The specification

imposes the restriction 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. Such contours are therefore invalid. See section 2.6. 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 higher accuracy is required it must be checked that the applications downstream can handle this. Higher accuracy 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 about the file or features within them. Examples of such 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> = <AttributeName>[,<AttributeValue>]*

Attribute names must follow the naming syntax in section 3.6.5. Attribute value has the following structure:

<AttributeValue> = <Field>{,<Field>}

This means an attribute value consists of one or more fields separated by comma. The fields composing the attribute value must follow the string syntax in section 3.6.6 with the additional restriction that a field must not contain commas. Consequently the whole attribute value also follows the string syntax.

Each attribute is defined by an extended code command consisting of a single data block with the TA, TF or TD command code. (Attribute commands start with a T as the A is taken by aperture commands.)

In the following example the command TF defines an attribute with name ".FileFunction" and value composed of two fields: "Soldermask,Top".

Example:

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

There are two types of attributes separated by the domain they attach to:

- File attributes attaching metadata to the file as a whole.
- Aperture attributes attaching metadata to 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 command, with nn≥10, see 4.3).

There are two types of attributes by scope:

- *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, use the G04 for comments.

In accordance with the general rule laid out in 3.6.5 standard attribute names *must* begin with a dot '.' while user attribute names *cannot* begin with a dot.

Example of a user attributes:

```
%TFMyAttribute,Yes*%
%TFZap*%
%TFZonk*%
```

The syntactical difference with standard attributes is the absence of a dot. The dot is not the separator between the command and the attribute name. The dot, if present, is part of the attribute name and indicates that it is a standard attribute whose syntax and semantics are defined in section 5.4.

5.2 File Attributes

File attributes provide meta-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.

When a file attribute is defined it is attached directly to the image as shown in 2.11.

5.2.1 **TF Command**

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

<TF command> = TF<AttributeName>[,<AttributeValue>]*

<AttributeValue> = <Field>{,<Field>}

The attribute name must follow the naming syntax in section 3.6.5. The fields composing the attribute value must follow the string syntax in section 3.6.6 with the additional restriction: a field must not contain commas.

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 attributes 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 or updated 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.

Attributes on regions. Although apertures are not needed to create a region, regions can carry aperture attributes. 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, any aperture created to flash or stroke 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 you can choose any shape; we recommend using a zero-size circle standard aperture for dummy apertures.

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

The handling of aperture attributes is illustrated in the diagram in 2.11.

5.3.2 TA Command

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

<TA command> = TA<AttributeName>[,<AttributeValue>]*

<AttributeValue> = <Field>{,<Field>}

The attribute name must follow the naming syntax in section 3.6.5. 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 fields composing the attribute value must follow the string syntax in section 3.6.6 with the additional restriction: a field must not contain commas.

The example below defines several attributes.

Example:

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

The next example shows how to overrule the value of an aperture attribute.

Example:

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

5.3.3 **TD Command**

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

<TD command> = TD[<AttributeName>]*

The <AttributeName> is 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

Attributes are not needed when the image only needs to be rendered. However, attributes are needed when transferring PCB data from design to fabrication. A PCB fabricator needs to process the image 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.

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.

 **Warning: Do not invent your own standard attribute names** (names starting with a dot), your own standard attribute values or your own semantics on them. This would defeat the purpose of standardization. Files with such attributes are anyhow invalid. Standard attributes can only be defined in this specification. If you need attributes not covered in this specification there is no problem. The user attributes cater to this need. Feel free to invent any user attribute you wish.

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.

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 explained in detail subsequently. All standard attributes names and values are case-sensitive.

Name	Usage
`.Part`	Identifies the part the file represents, e.g. a single PCB
`.FileFunction`	Identifies the file's function in the PCB
`.FilePolarity`	Defines whether the files represents the presence or absence of material in the PCB layer
`.GenerationSoftware`	Identifies the software creating the file.
`.CreationDate`	Formally defines the creation time of the file
`.ProjectId`	Defines project and revisions
`.MD5`	Sets the MD5 file signature or checksum

Standard file attributes

5.4.1.1 .Part

The value of the .Part file attribute identifies which part is described.

.Part value	Usage
`Single`	Single PCB
`Array`	A.k.a. customer panel, assembly panel, shipping panel, biscuit
`FabricationPanel`	A.k.a. working panel, production panel
`Coupon`	A test coupon
`Other,<mandatory field>`	None of the above. The mandatory field informally indicates the part

.Part file attribute values

Example:

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

The attribute – if present - must be defined in the header.

5.4.1.2 .FileFunction

The .FileFunction file attribute identifies the function of the file in the PCB. Of all the attributes it is the most important.

The attribute value consists of a number of fields separated by ‘,’:

- Type. Such as copper, solder mask etc. See list in the table 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

Position values

Example:

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

The attribute – if present - must be defined in the header.

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 fabrication processes are identical from the point of view of their image descriptions: the image simply represents where material is removed.

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.)

Note: It may come as a surprise that in CAD to CAM workflows drill information can and indeed is better represented in Gerber than in an NC as they are used to represent it in an NC format. 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)[,<type>]`	A conductor or copper layer. L<p> specifies the position in the stack. (p is an integer). The mandatory mark (*Top*|*Inr*|*Bot*) specifies it as the top, an inner or the bottom layer; this redundant information helps in handling partial data. Note that the top copper layer is L1; its specification is "`Copper,L1,Top[,label]`"; *L0 does not exist!* The type is optional. If present it must take one of the following: *Plane, Signal, Mixed* or *Hatched*.
`Soldermask,(Top|Bot)[,<index>]`	The image represents the solder mask *openings.* The index is not present if there is only one solder mask on a side. If there are more than one solder masks the numerical index numbers the masks on a side from the PCB surface outwards, starting with 1 for the mask closest to the surface.
`Legend,(Top|Bot)[,<index>]`	A legend is printed on top of the solder mask to show which component goes where. A.k.a. 'silk' or 'silkscreen'. See the Soldermask row for an explanation of the index.
`Goldmask,(Top|Bot[,<index>]`	See the Soldermask row for an explanation of the index.
`Silvermask,(Top|Bot)[,<index>]`	See the Soldermask row for an explanation of the index.
`Tinmask,(Top|Bot)[,<index>]`	See the Soldermask row for an explanation of the index.
`Carbonmask,(Top|Bot) [,<index>]`	See the Soldermask row for an explanation of the index.
`Peelablesoldermask,(Top|Bot)[,<index>]`	See the Soldermask row for an explanation of the index.
`Glue,(Top|Bot)[,<index>]`	See the Soldermask row for an explanation of the index.
`Viatenting,(Top|Bot)`	Indicates via's that must be tented
`Viafill`	Indicates via's that must be filled
`Heatsink,(Top|Bot)`	

.FileFunction value	Usage
`Paste,(Top|Bot)`	
`Keep-out,(Top|Bot)`	
`Pads,(Top|Bot)`	A file containing only the pads (SMD, BGA, component, …).
`Scoring,(Top|Bot)`	
`Plated,i,j,(PTH|Blind|Buried)[,<label>]`	Plated drill/rout data, from layer i to layer j. The (PTH|Blind|Buried) is mandatory. The label is optional. If present it must take one of the following values: *Drill, Route* or *Mixed*.
`NonPlated,i,j,(NPTH|Blind|Buried)[,<label>]`	Non-plated drill/rout data, from layer i to layer j. The (NPTH|Blind|Buried) is mandatory. The optional label is explained in the row above.
`Profile,(P|NP)`	The profile or outline. P indicates a plated profile (this is exceptional). NP indicates a non-plated profile (this is routine). The (P|NP) is mandatory.
`Drillmap`	A drawing with the locations of the drilled holes. It often also contains the hole sizes, tolerances and plated/non-plated info.
`FabricationDrawing`	A drawing with additional information for the fabrication of the bare PCB: the location of holes and slots, the board outline, sizes and tolerances, layer stack, material, finish choice, etc.
`ArrayDrawing`	A drawing of the array, aka biscuit, assembly panel, shipment panel, customer panel.
`AssemblyDrawing,(Top|Bot)`	A drawing with the locations and reference designators of the components. It is mainly used in PCB assembly.
`Drawing,<mandatory field>`	Any other drawing. The mandatory field informally describes its topic.

.FileFunction value	Usage
`Other,<mandatory field>`	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 field informally describes the file function.

.FileFunction attribute values

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!

Example:

```
%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*%
```

5.4.1.3 .FilePolarity

The value of the .FilePolarity specifies whether the image represents the *presence or absence* of material. This attribute can only be used when the file represents a pattern in a material layer, e.g. copper, solder mask, legend. Together with.FileFunction it defines the role of that image in the layer structure of the PCB. Note that the .FilePolarity attribute does *not* change the image - no attribute does. It changes the interpretation of the image.

For example, in a copper layer in positive polarity a round flash generates a copper *pad*. In a copper layer in negative polarity it generates a *clearance*. Negative copper is sometimes used for copper plane layers. Note that there is no longer a reason to output copper layers in negative. It is recommended to output copper layers in positive, but if you output them in negative, please make this clear by setting this attribute.

Solder mask images usually represent solder mask *openings* and are then *negative*. This may be counter-intuitive.

.FilePolarity value	Usage
`Positive`	The image represents the *presence* of material (recommended)
`Negative`	The image represents the *absence* of material

.FilePolarity attribute values

Example:

```
%TF.FileFunction,Copper,L2,Inr,Plane*%
%TF.FilePolarity,Positive*%
```

The attribute – if present - must be defined in the header.

5.4.1.4 .GenerationSoftware

The .GenerationSoftware file attribute identifies the software that generated the Gerber file.

The attribute – if present - must be defined in the header.The attribute value has the following syntax:

```
<vendor>,<application name>[,<application version>]
```

Example:

```
%TF.GenerationSoftware,Ucamco,Ucam*%
```

5.4.1.5 .CreationDate

The .CreationDate file attribute identifies the moment of creation of the Gerber file.

The attribute – if present - must be defined in the header. The attribute value must conform to the full version of the ISO 8601 date and time format, including the time and time zone. A formally defined creation date can be helpful in tracking revisions – see also 5.4.1.6

Example:

```
%TF.CreationDate,2015-02-23T15:59:51+01:00*%
```

5.4.1.6 .ProjectId

Usually a Gerber file is part of a PCB project with a sequence of revisions. It is important to be able to determine if different files belong to the same revision of a project, different revisions of the same project or completely different projects. This is the purpose of the .ProjectId file attribute. It uniquely identifies project and revision.

The attribute – if present - must be defined in the header. The attribute value has the following syntax:

```
<project id>,<project guid>,<revision id>
```

The field <project id> defines the project id in a custom format, <project guid> defines the project using a global unique ID and <revision id> specifies its revision. All parameters must conform to the string syntax, with the additional restriction that the ‘,’ character cannot be used. The <project guid> must be a random 32 digit hexadecimal number (see 3.6.4).

Examples:

```
%TF.ProjectId,My Incredible PCB,fd82b6a0966042718f6aad92285b3de3,2*%

%TF.ProjectId,project#8,c919f25ccf844f1fb2560e9c46f85f5d,/main/18*%
```

5.4.1.7 .MD5

The .MD5 file attribute sets a file signature (checksum, hash, digest) that uniquely identifies the file and provides a high degree of security against accidental modifications. The 128 bit signature is calculated with the MD5 algorithm and expressed as a 32 digit hexadecimal number (see 3.6.4).

The signature is calculated over the bits from the beginning of the file up to but excluding the .MD5 file attribute command. Note that this excludes the closing `M02*`. The complete .MD5 file attribute command, with both '%'and '*', is excluded. Any CR and LF are excluded from signature calculation. As CR and LF do not affect the interpretation of the file but may be altered when moving platforms excluding them makes the signature portable without sacrificing security.

The signature, if present, 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:

Consider the following Gerber file segment, without checksum:

```
…
D11*
X1500000Y2875000D03*
X2000000D03*
D19*
X2875000Y2875000D03*
M02*
```

As the CR and LF characters are skipped the checksum is taken over the following data:

```
…D11*X1500000Y2875000D03*X2000000D03*D19*X2875000Y2875000D03*
```

With the checksum the file then becomes:

```
…
D11*
X1500000Y2875000D03*
X2000000D03*
D19*
X2875000Y2875000D03*
%TF.MD5,6ab9e892830469cdff7e3e346331d404*%   <- Excluded from the MD5
M02*                                         <- Excluded from the MD5
```

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. Gerber file creators are encouraged to identify the function of all apertures. PCB CAM needs to know this information and if it is not defined by this attribute CAM has to reverse engineer it.

If this is not possible to define the function of all apertures, define it for those where it is possible - partial information is better than no information. The bare minimum is to identify the via pads – the fabricator really cannot do his job properly if he does not know where the vias are. Without attribute he must reverse engineer this information, and that is always a risk.

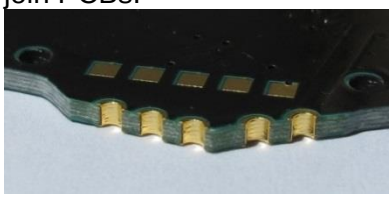
One function, one aperture. Create objects with different functions with separate apertures, even if they are of the same shape and size. Mixed function attributes are hard to handle in CAM. If you do use mixed attributes you cannot define their value - an aperture can only carry one function attribute. It is *invalid* set one of its values only on a mixed function in an attempt to convey partial information. For example, if an edge connector and SMD pads are created with the same aperture it is wrong to give it the SMDPad value – this defines the edge connector as SMD and could result in paste on your edge connector. The same considerations apply for drill tools.

Stroking. The function of the stroked objects is set by the function of apertures used for stroking to the object. For example, if you use aperture 21 to stroke SMD pads then the function of aperture 21 is SMDPad. If you use aperture 50 to paint a conductive region, the function of aperture 50 is conductor. Note that it is recommended not to use stroking, see 6.2.

Regions. Remarkably, regions can carry aperture attributes, see Attributes on regions. Use this to define the function of the regions.

The values for this attribute are defined in the tables below. Note that the applicable functions depend on the layer - for example, an SMD pad can only be defined on an outer copper layer.

Drill and rout files- .Filefunction Plated and/or NonPlated

AperFunction value	Usage
`ViaDrill`	A via hole. This is reserved for holes whose *sole* function is to connect different layers. Not to be used for component leads.
`BackDrill`	A hole to remove plating in a hole over a depth by drilling with a slightly larger diameter.
`ComponentDrill[,PressFit]`	A hole for through-hole component leads. The optional label PressFit indicates that the drill holes are intended for press fit component leads. Press fit leads are pressed in properly sized plated-through holes to realize the electrical contact. It is an alternative to soldering. It can only be applied on PTH holes. See ComponentPad.
`CastellatedDrill`	Plated holes cut- through by the board edge; used to join PCBs.  Image courtesy Eurocircuits.

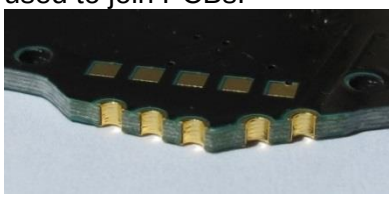
`MechanicalDrill[,<type>]`	A hole with mechanical function (registration, screw, etc.) The specifier <type> is optional. If present it can take one of the following values:
	• `BreakOut`: Non-plated holes forming a break-out tab used in break routing.
	 
	• `Tooling`: Tooling holes to attach the board or panel temporarily to test fixtures during assembly and test. Also called mounting holes.
	• `Other`
	Example:
	`.AperFunction,MechanicalDrill,Breakout`
	`.AperFunction,MechanicalDrill`

`Slot`	PCB slots.

`CutOut`	PCB cut-outs. Can be viewed as holes in the profile. Just as the profile, it can be present in all PCB layers.
`Cavity`	Cavities in a PCB.
`OtherDrill,<mandatory field>`	A hole, but none of the above. The mandatory field informally describes the type.

Copper layers - .FileFunction Copper

AperFunction value	**Usage**
`ComponentPad[,PressFit]`	A pad for through-hole component leads. Only applicable on outer layers. Non-electrical fixing elements are not leads - use WasherPad for these. The optional label PressFit indicates a press fit component pad. See ComponentDrill. Only trough-hole components, not for SMD and BGA.
`SMDPad,(CuDef|SMDef)`	An SMD pad. Excluded BGA pads which have their own type. The specifier (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.)* Only applicable for outer layers. Note: Edge connecters are *not* SMDs. Surely one does not want solder paste on an edge connector. Use ConnectorPad.
`BGAPad,(CuDef|SMDef)`	A BGA pad. Only applicable for outer layers. The specifier (CuDef|SMDef) is *mandatory*.CuDef stands for *copper defined*, SMDef for *solder mask defined*; see SMDPad.
`ConnectorPad`	An *edge connector* pad. Only applicable for outer layers. Through-hole or SMD connector pads are classified as resp. through-hole or SMD pads.
`HeatsinkPad`	Heat sink pad (typically for SMDs)

`ViaPad`	A via pad. It provides a ring to attach the plating in the barrel. This is reserved for pads that have no other function than making the connection between layers. If the pad also has another function, e.g. test pad or component pad then it takes the other function.
`TestPad`	A test pad. Only applicable for outer layers. Sometimes a test pad is drilled by a via drill and hence also used as a via pad. Such a pad *must* be specified as test pad. The fabricator must know it is a test pad when fabricating the solder mask. This is consistent with component pads: a component hole is often also used as a via; the pad must however be specified as a component pad as this is the more important function.
`CastellatedPad`	Pads on plated holes cut- through by the board edge; used to join PCBs.  Image courtesy Eurocircuits.
`FiducialPad,(Global|Local)`	A fiducial pad. The specifier (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 non-plated hole that is not used for component leads. Several applications, e.g. a pad that strengthens the PCB where fixed with a bolt – hence the name washer.
`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,<mandatory field>`	A pad not specified above. The mandatory field informally describes the type.
`Conductor`	Copper whose function is to connect pads and possibly to provide shielding, typically tracks and copper pours such as power and ground planes. Note that conductive copper pours can and should carry this attribute, whether made properly by a G36/G37 or by stroking,– see **Regions** and **Stroking**

NonConductor	Copper that does not serve as a conductor; typically text and graphics without electrical function. This value can only be applied to copper that is part of the PCB, not to drawing elements; see NonMaterial
CopperBalancing	Copper pattern added to balance copper coverage for the plating process.
Border	The copper border around a production panel.
OtherCopper,<mandatory field>	Indicates another function. The mandatory field informally describes the type.

All layers	
AperFunction value	**Usage**
`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.
`NonMaterial`	This is for objects that do not represent physical material, 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 function 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. This function cannot be used for copper and drill layers as this value is split of much more specific functions.
`Other,<mandatory field>`	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 an attribute with a vaguely similar function into that value that does not fit perfectly – keep the identification clean by using ‘Other’. The mandatory field informally describes the aperture function.

.AperFunction aperture attribute values

Functions on extra layers. The solder mask, paste and other extra layers openings *cannot* take the pad values. Pad values are reserved for outer copper layers. The solder mask openings and paste pads take their function from the underlying copper pads. The reason for this is that a single solder mask opening may have multiple underlying copper pads – e.g. an SMP pad with an embedded via pad - and hence multiple functions. Consequently, solder mask openings have the aperture function ‘Material’. (Admittedly this is somewhat a misnomer in this context as solder masks are usually negative, and the presence of image indicates therefore the absence of material; this has nothing to do with the pad functions but with the layer being negative.)

5.4.2.2 .DrillTolerance

.DrillTolerance defines the plus and minus tolerance of a drill hole. Both values are positive decimals expressed in the MO units. The attribute value has the following syntax:

```
<plus tolerance>,<minus tolerance>
```

Examples:

```
%TA.DrillTolerance,0.01,0.005*%
```

5.5 Examples

The example below shows the usage of a simple aperture attribute.

Example:

```
G01*
%ADD13R,200X200*%       G04 this aperture has no attribute*
D13*
%TAIAmATA*%             G04 add attribute IAmATA in current attributes
                        dictionary*
X0Y0D03*                G04 this flash object has no attribute*
%ADD11R,200X200*%       G04 this aperture now has attached attribute
                        IAmATA*
%TDIAmATA*%             G04 delete attribute IAmATA from current
                        attributes dictionary*
%ADD12C,5*%             G04 this aperture does not have attribute IAmATA*
D11*
X100Y0D03*              G04 this flash object has attribute IAmATA*
X150Y50D02*
D12*
X100Y150D01*            G04 this draw object has no attribute*
```

The next example illustrates an aperture attribute definition and changing value.

Example:

```
G01*
%TA.AperFunction,SMDPad*%      G04 Adds attribute .AperFunction in the
                               current dictionary with value SMDPad to
                               identify SMD pads*
%ADD11…*%                      G04 Aperture with D-code 11 gets the
                               .AperFunction,SMDPad attribute attached*
%TA.AperFunction,Conductor*%   G04 Changes the value of .AperFunction
                               attribute to define conductors*
%ADD20…*%                      G04 Aperture with D-code 20 gets the
                               .AperFunction,Conductor attribute attached*
%TACustAttr,val*%              G04 Adds attribute CustAttr in the current
                               attributes dictionary and sets its value to
                               val*
%ADD21…*%                      G04 Aperture with D-code 21 is a conductor
                               with attached attribute CustAttr = val*
%TD.AperFunction*%             G04 Deletes the .AperFunction attribute from
                               the current attributes dictionary*
%ADD22…*%                      G04 Aperture with D-code 22 has no attached
                               .AperFunction attribute, but has attribute
                               CustAttr = val*
%TDCustAttr *%                 G04 Deletes the CustAttr attribute from the
                               current attributes dictionary*
%ADD23…*%                      G04 Aperture with D-code 23 has no attached
                               aperture attributes*
…
D11*                           G04 Set current aperture to aperture 11*
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. The region
                               takes on the value of D20*
….
G37*
D21*                           G04 Set current aperture to aperture 21*
G36*                           G04 Start a conductive region with CustAttr
                               = val*
….
D22*                           G04 Set current aperture to aperture 22*
X2000Y2000D03*                 G04 A flash with CustAttr = val, but
                               undefined aperture function*
D23*                           G04 Set current aperture to aperture 23*
G36*                           G04 Start a region, without attributes*
….
G37*
```

6 Errors and Bad Practices

6.1 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 commands. This is invalid because quadrant mode is undefined by default. **A G74 or G75 command is mandatory if arcs are used.** *See 2.8 and 4.5.9.*
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* 4.13.4.4*.*
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. **Coordinate date must always be combined with an explicit D01/D02/D03 operation code.** *See 7.2.*
Objects unexpectedly appear or disappear under holes in standard apertures.	Some CAD systems incorrectly assume the hole in an aperture *clears* the underlying objects. This is wrong; the hole has no effect on the underlying image. *See 4.12.1.5.*
Objects unexpectedly appear or disappear under holes in macro apertures.	Some systems incorrectly assume that exposure off in a macro aperture *clears* 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 4.13.2.*
Clearances in planes disappear.	This is typically the result of sloppy cut-ins in contours. Only fully coincident segments are allowed for cut-ins. Note that cut-ins are not intended for complex planes; use LPC to make clearances in a plane. *See 4.6* and 4.14*.*

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* 4.12.1.4.
A single Gerber file contains more than one image, separated by M00, M01 or M02	This is invalid. A Gerber file can contain only one image. One file, one image. One image, one file.
The MI command is used to mirror a file with macro's but the result is not as expected.	With the MI command mirroring is not applied to aperture definitions. This exception is unfortunately ignored by a number of input and output implementations. Do not use this deprecated command. Apply the transformation directly in the aperture definitions and object coordinates. *See* 7.1.6.
%ICAS*%	Some files contain %ICAS*%. One wonders what this pseudo-command is supposed to achieve. Anyhow, it is invalid.
%FSD….*%	The only valid zero omission options in the %FS are L and T. D is invalid. See 7.4.1.
%FSLAN2X26Y26%	The N2 in the format statement is invalid. See 4.9 One wonders what it is supposed to do.
…X5555Y5555IJ001 ..	Missing zero after the "I". The number after I must have at least one digit, see 4.9.1.

Reported errors

6.2 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
PCB fabrication data sets without netlist.	PCB fabrication data is complex geometric data with an infinite number of variations. Differences in the interpretation of image data is very rare but does happen and then is costly. A netlist is a powerful check on the image data – it is akin to the redundancy checks used in all data transfer protocols. Omitting a netlist is omitting a basic security check.	**Always include a netlist (IPC-D-356A) in a PCB fabrication data set.**
Writing files with deprecated constructs.	Each construct was deprecated for a reason. Many carry the risk of a misinterpretation. Continuing to use deprecated constructs is bad corporate citizenship as it blocks the industry from taking the next steps.	**Generated files with current constructs only.** (Note: it is OK for readers to handle deprecated constructs to cater for legacy files.)
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 fabrication. See 4.9. Do not sacrifice precision to save a few bytes.
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 4.6.12.	**Construct planes and anti-pads using dark polarity for the plane and clear polarity for the holes (anti-pads).**
Invalid and sloppy cut-ins, e.g. resulting in self-intersections.	The file becomes invalid with undefined interpretation. See section 4.6.12.	**Avoid cut-ins for complex images.** If you insist to use them, be very careful, especially with rounding errors, to construct them correctly.
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 care with 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* **Error! Reference source not found.**	**Always position arc center points precisely.**
Stroked (painted) pads	Stroked 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.
Stroked (painted) regions	As above, stroked regions 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 regions.**
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. See 7.8.	**Always use Extended Gerber.**
Using a non-standard file extension	Using a non-standard file extension for Gerber files makes it impossible to determine the file type without reading the file. See 3.2.	**Please use ".gbr" or ".GBR" as file extension for all your Gerber files.**

Poor/good practices

7 Deprecated Elements

7.1 Deprecated Commands

The next table lists deprecated commands.

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 4.10. 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 4.9. 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. See 4.8. It is superfluous and deprecated.
M01	Optional stop	This historic code has no effect. It is superfluous and deprecated.
AS	Sets the 'Axes correspondence' graphics state parameter	These commands can only be used once, at the beginning of the file.
IN	Sets the name of the file image	
IP	Sets the 'Image polarity' graphics state parameter	
IR	Sets 'Image rotation' graphics state parameter	
MI	Sets 'Image mirroring' graphics state parameter	
OF	Sets 'Image offset' graphics state parameter	
SF	Sets 'Scale factor' graphics state parameter	
LN	Has no effect on the image. It is no more than a comment	Can be used many times in the file. It is deprecated. To add a comment G04 command can be used. See 4.7.

Current Gerber writers must not use the deprecated commands. Gerber readers may implement them to support legacy applications and files.

Note: The function code commands with codes G54, G70 and G71 are still found from time to time. The other deprecated function code commands are very rarely found.

The order of execution of the deprecated extended code commands is always MI, SF, OF, IR and AS, independent of their order of appearance in the file.

Note: There are a few legacy files with these deprecated extended codes but this is nearly always, if not always, to confirm the default value; in other words they have no effect. *It is a waste of time to fully implement them. It is probably sufficient to handle them as an unsupported command: give a warning, further ignore them and continue to process the file.*

The table below contains deprecated graphics state parameters.

Graphics state parameter	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	Yes	Positive
Image rotation	0°, 90°, 180°, 270° See IR command	Yes	0°
Scale factor	See SF command	Yes	A1B1

Deprecated graphics state parameters

7.1.1 **Axis Select (AS)**

The AS command is deprecated.

AS 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.

7.1.1.1 AS Command

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

7.1.1.2 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

7.1.2 **Image Name (IN)**

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

IN 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 3.6.6. 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.

7.1.2.1 IN Command

The syntax for the IN command is:

<IN command> = IN<Name>*

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

7.1.2.2 Examples

Syntax	Comments
`%INPANEL_1*%`	Image name is ‘PANEL_1’

7.1.3 Image Polarity (IP)

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.

7.1.3.1 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.)

7.1.3.2 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.

7.1.3.3 IP Command

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

7.1.3.4 Examples

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

7.1.4 Image Rotation (IR)

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.

7.1.4.1 IR Command

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

7.1.4.2 Examples

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

7.1.5 **Load Name (LN)**

This historic command has *no* effect on the image and can be ignored. It is deprecated.

LN associates a name to the subsequent part of the file. It was intended as a human-readable comment. Use the normal G04 command for human-readable comment.

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

7.1.5.1 LN Command

The syntax for the LN command is:

<LN command> = LN<Name>*

Syntax	Comments
LN	LN for Load Name
<Name>	The name must comply with the syntax for a string, see section 3.6.6.

7.1.5.2 Examples

Syntax	Comments
%LNVia_anti-pads*%	The name 'Via_anti-pads' is to the subsequent file section

7.1.6 **Mirror Image (MI)**

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: Quite a number of Gerber input and output implementations implement MI incorrectly: they overlook the restriction on macro apertures and assume they are mirrored too. These incorrect implementation make is risky to use MI. Be careful when reading a file with MI and macro's as you do not know how the file was intended. We strongly recommended *not* to use MI on output as you do know how the reader will interpret the file. If an image must be mirrored, write out the mirrored coordinates and apertures.

7.1.6.1 MI Command

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

7.1.6.2 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

7.1.7 **Offset (OF)**

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.

7.1.7.1 OF Command

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 ≤ n ≤ 99999.99999

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

7.1.7.2 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

7.1.8 **Scale Factor (SF)**

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.

7.1.8.1 SF Command

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.

7.1.8.2 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

7.2 Coordinate Data without Operation Code

Previous versions of the specification allowed coordinate data *without explicit operation code* after a D01. A D01 code sets the deprecated operation mode to interpolate. It remains in interpolate mode till any other D code is encountered. In sequences of D01 operations this allows omitting an explicit D01 code after the first operation.

 **Example:**

```
D10*
X700Y1000D01*
X1200Y1000*
X1200Y1300*
D11*
X1700Y2000D01*
X2200Y2000*
X2200Y2300*
```

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:** However, coordinate data without explicit operation code saves a few bytes but is not intuitive and lead to errors in the field. This risk far outweighs the meager benefit

7.3 Rectangular Hole in Standard Apertures

In addition to the round hole described in the section 4.12 the previous versions of this specification also allowed rectangular holes. Rectangular holes are now deprecated.

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

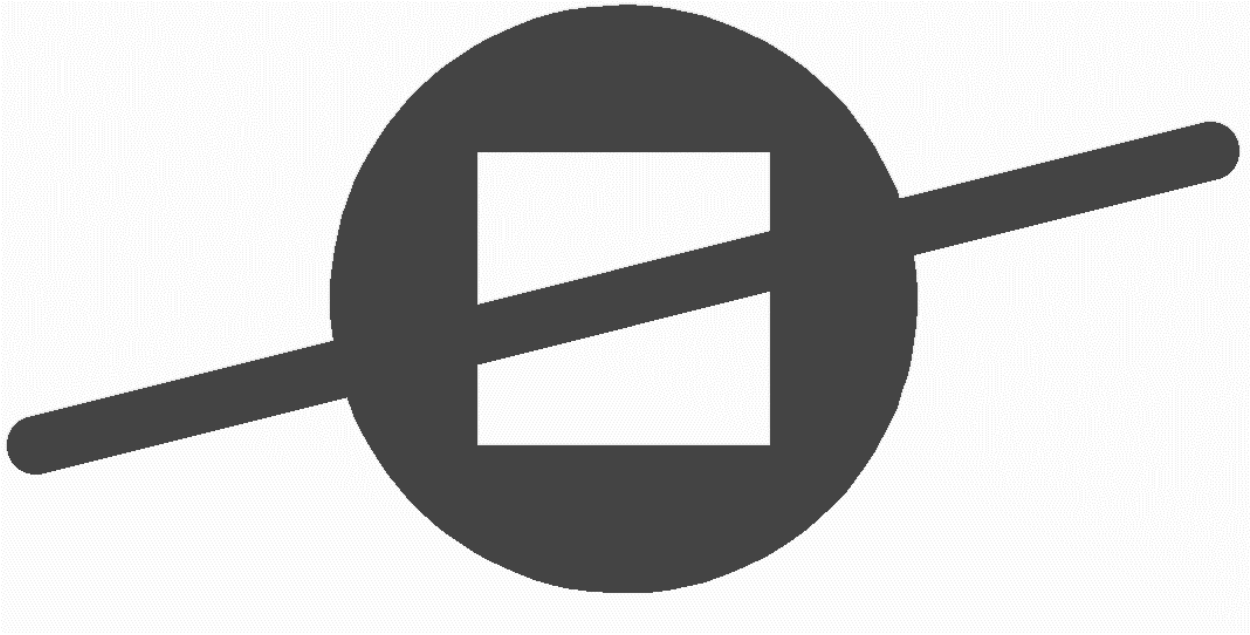
<Hole> = <X-axis hole size>X<Y-axis hole size>

The modifiers specify the X and Y sizes of the rectangle and 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*
```



50. Standard (circle) aperture with a rectangular hole above a draw

Note that the draw is visible through the hole.

7.4 Deprecated Options of the Format Specification

This section describes deprecated options of the FS command (see 4.9).

The FS command could also be used to specify the following format characteristics:

- Zero omission
- Absolute or incremental coordinate notation

Warning: Using less than 4 decimal places is deprecated.

7.4.1 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 the 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.

The leading zero omission is specified by 'L' after FS code in the command.

The trailing zero omission is specified by 'T' after FS code in the command.

Example:

```
%FSTAX25Y25*%
```

Warning: Trailing zero omission is deprecated.

7.4.2 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).

The absolute notation is specified by 'A' after FSL or FST in the command.

The incremental notation is specified by 'I' after FSL or FST in the command.

Example:

```
%FSLIX25Y25*%
%FSTIX36Y36*%
```

Warning: Currently the *only allowed notation is absolute notation*. The incremental notation is deprecated.

7.5 Using G01/G02/G03 in a data block with D01/D02

The function codes G01, G02, G03 could 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 a useless variation and now it is deprecated.

The syntax for G01, G02, G02 codes in operations with codes D01 and D02 was the following:

<Operation> = G(1|01|2|02|3|03)<Coordinate data>D(01|02)*

Example:

```
G01*
X100Y100D01*
G01X500Y500D01*
X300Y300D01*
G01X100Y100D01*
```

7.6 Closing SR with the M02

When an SR statement is not explicitly closed with an `%SR*%` command and the end-of-file (M02) command is encountered the M02 closes and replicates the block. However it t is recommended to close the statement explicitly with an `%SR*%` command.

7.7 Deprecated Terminology

Originally extended codes were called (Mass) Parameters, an awkward term.

In previous versions of this document:

- The “contour fill” was called “polygon fill”
- The “operation” was called “coordinate data block”
- The following synonyms for “darken” could be used: mark, expose, paint
- The following synonyms for “clear” could be used: unmark, rub, erase, scratch
- The term “paint” could also be used as the synonym of “stroke”
- Incremental position: position expressed as a distance in X and Y from the current point

7.8 Revoked 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.

Standard Gerber is revoked and superseded by Extended Gerber, which is the current Gerber format. Consequently, Standard Gerber no longer complies with the Gerber specification. Files in that format can no longer be correctly called Gerber files. Standard Gerber files are not only deprecated, they are no longer valid.

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

It differs from the current Gerber file format (RS-274X), in that it:

- does not support G36 and G37 codes
- does not support any extended codes

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

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 major drawbacks compared to the current Gerber file format and does not offer a single advantage. Standard Gerber is obsolete. There is not a single valid reason to use standard Gerber rather than Extended Gerber.

Always use Extended Gerber (RS-274X). 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 revoked Standard Gerber, with its informal and non-standardized wheel file, rather than Extended Gerber, which is unequivocally and formally standardized.