The Gerber Format Specification

A format developed by Ucamco

Revision 2017.05

Contents

Copyright Ucamco NV

2

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

Copyright Ucamco NV

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

3

Copyright Ucamco NV

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

5

Copyright Ucamco NV   6

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

Figures

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

Tables

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

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.

The current Gerber formats are RS-274X or Extended Gerber, either X1 or X2. Standard Gerber or RS-274-D is obsolete and revoked. It was superseded by RS-274X decades ago. Do not use Standard Gerber any longer.

Some applications stubbornly continue to use painting (aka stroking) to create pads and copper pours. While not formally 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. We urge all users and developers to banish painting from our industry.

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.

Although other data formats have appeared they have not displaced Gerber. The reason is simple. The problems in PCB fabrication data transfer are not so much limitations in the Gerber format but poor practices and poor implementations. 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 simple format. The new formats are sometimes promoted by pointing to poor Gerber implementations. 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 Scope and Target Audience

This document specifies the Gerber file format, an ASCII vector image file format representing a 2D bi-level (binary, having two colors) images. 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. A basic knowledge of PCB CAD/CAM is helpful for understanding this specification.

1.2 Further Reading

The Ucamco website contains articles about the use of the Gerber format as well as sample files. For more information about Gerber or Ucamco see www.ucamco.com.

1.3 Questions

Ucamco strives to make this specification easy to read and unequivocal. If you find a part of this specification unclear, please ask. Your question will be answered and it will be considered to improve this document. We are grateful for any suggestion for improvement.

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.

![Ucamco logo - Former Barco ETS]

1.4 Copyright and Intellectual Property

Copyright Ucamco NV   12

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

2 **Overview**

2.1 File Structure

The Gerber format is a 2D bi-level vector 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 one file only.

A Gerber file is uses printable 7-bit ASCII. It is printable and human readable.

A Gerber file is a stream of *commands.* The commands create a stream of graphics objects (see 2.2) that are put on the image plane to create the final image. Other commands add attributes to structures in the image- attributes are labels defining meta-information.

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

As an illustration here is a small Gerber file with one command per line. It creates a circle of 1.5 mm in the origin.

```
%FSLAX2.6Y2.6*%
%MOMM*%
%AD100C,1.5*%
D100*
X0Y0D03*
M02*
```

2.2 Apertures

An aperture is a 2D plane figure. Apertures are used to create pads or tracks.

The AD (Aperture Define) command creates an aperture based on an aperture template and parameter values and gives it a unique D code or aperture number for reference later in the command stream.

There are three 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 4.4).

❑ Macro apertures are created with the AM (Aperture Macro) command. Any shape and parametrization can be created. They are identified by their given name. (See 4.5).

❑ Block apertures

Standard apertures can be considered as built-in macro apertures. The example AD command below creates an aperture with D-code D123. It uses the standard aperture R with parameters 2.5 and 1.5 mm which creates a rectangle of 2.5 by 1.5 mm.

```
%ADD123R,2.5X1.5
```

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 the small number of macro primitives. This single flexible mechanism replaces the need for a large - but always insufficient - set of built-in apertures. New apertures can be created without extending the format.

The AB command creates a block aperture from a block and directly assigns a D-code to it (see 2.8).

An aperture has an *origin*. When an aperture is flashed, its origin is positioned at the coordinates in the D03 flash command (see 4.1). 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 the macro. The origin of a block aperture is the origin used when defining the block.

2.3 Graphics Objects

A Gerber file creates an ordered stream of graphics objects. A graphics object represents plane figure. It has a shape, a size, a position and a polarity (dark or clear). The stream of the graphics objects create the final image by superimposing the objects on the plane in the order of the stream, with dark polarity objects darkening the plane and clear ones erasing all dark areas under them.

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 2.4).

❑ 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 2.4).

❑ A flash is a replication of an aperture at a given location. An aperture is a basic 2D shape defined earlier in the file. An aperture is typically flashed many times. Any valid aperture can be flashed (see 4.8.4).

❑ A region is an area defined by its contour (see 4.12.1). A contour is a closed sequence of connected linear or circular segments.

In PCB copper layers, tracks are typically represented by draws and arcs, pads by flashes and copper pours by regions. Tracks is then a generic name for draws and arcs.

2.4 Operations

D01, D02 and D03 are the *operation codes*. An *operation* is a command containing 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 2.6) 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. Outside a region statement (see 2.6) these segments are converted to draw or arc objects by stroking them with the current aperture (see 2.4). Within a region statement these segments form a contour defining a region (see 4.12).

❑ D03 creates a flash object by flashing (replicating) 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 2.6).

Copyright Ucamco NV   15

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

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:

Line being stroked   Aperture   Draw

1. 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.

Line being stroked   Aperture   Draw
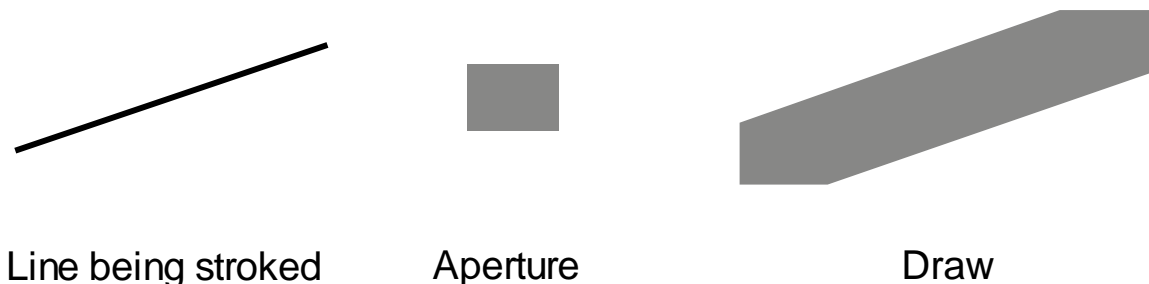
2. Creating a draw: the aperture is not aligned with line

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 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 2.4). 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 'Constant or variable' 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	Constant or variable during file processing	Initial value
Coordinate parameters			
Coordinate format	See the FS command in 4.1	Constant	Undefined
Unit	Inch or mm - See MO command in 4.2	Constant	Undefined
Generating parameters			
Current point	Point in plane	Variable	Undefined
Current aperture	D01 and D03 use this aperture. See 4.1	Variable	Undefined
Interpolation mode	Linear, clockwise circular, counterclockwise circular See G01, G02 and G03 in 4.9 and 4.10	Variable	Undefined
Quadrant mode	Single-, multi-quadrant See G74 and G75 in 4.10	Variable	Undefined
Object transformation parameters			
Object polarity	See the LP command in 4.11.2	Variable	Dark
Object mirroring	See the LM command in 4.11.3	Variable	No mirror
Object rotation	See the LR command in 4.11.4	Variable	No rotation (=0 degree)
Object scaling	See the LS command in 4.11.5	Variable	No scaling (=1.0)

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 outside a region statement, no inside	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

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.

2.7 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 4.12.5.7.
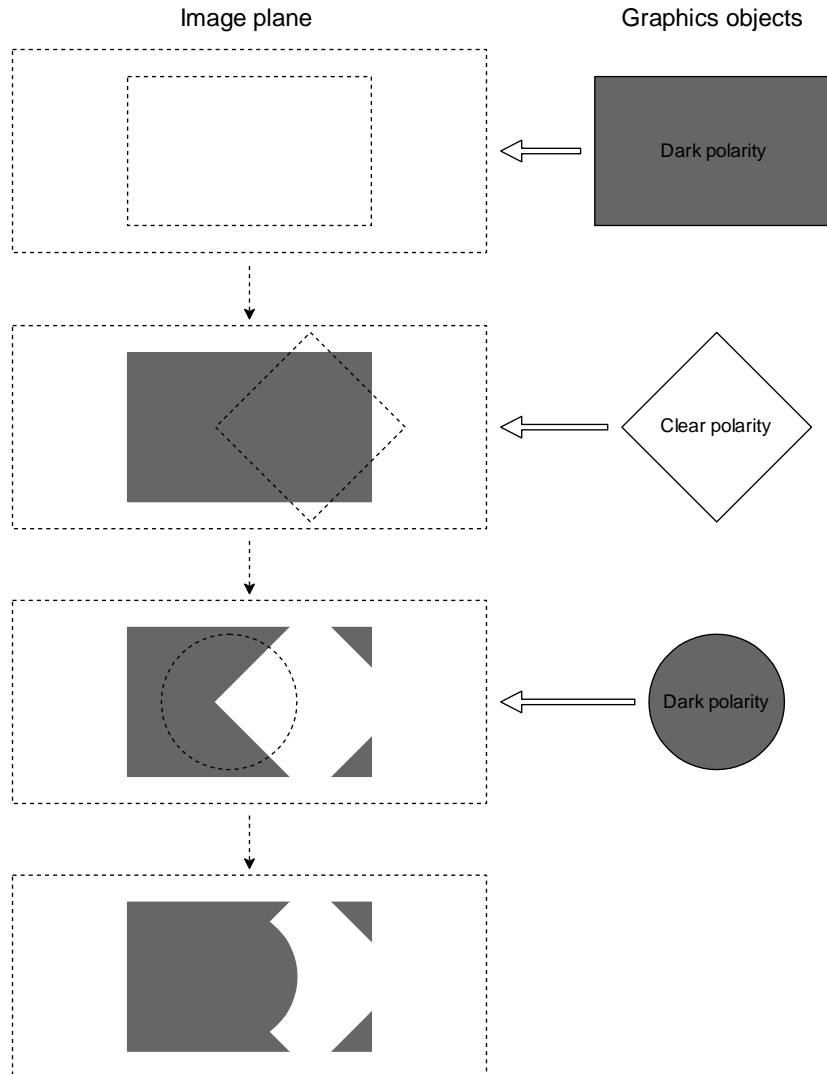
3. Superimposing objects with dark and clear polarities

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 4.11). Objects that are created when the polarity mode is dark are dark; when the mode is clear the objects are clear.

2.8 Blocks

A *block* is a substream of graphics objects that can be added one or more times to the final graphics objects stream. Blocks can be mirrored, rotated, scaled, shifted and its polarity can be toggled. By using blocks sub-images that occur multiple times must only be defined once, thus slashing file size, boosting processing speed and preserving the information that these sub-images are identical.

A block is *not* a macro of commands called repeatedly in the command stream. The command stream is processed sequentially, in one pass, without procedure or macro calls. Gerber is not a programming language.

Blocks can contain objects with different polarities (LPD and LPC). Blocks can overlap.

Once a block is added to the graphics objects stream its objects becomes part of the overall stream. The effect of these objects does not depend on whether they were part of a block or not. Only their order is important. A clear object in a block clears *all* objects beneath it, not only the objects *not* contained in the block.

There are two commands to create a block: SR and AB. They open and close block statements, a sequence of commands that define blocks. A block statement can contain other block statements.

⚠️ **Warning:** It is recommended to avoid overlapping blocks containing both clear and dark polarity objects. The order in which they are added to the object stream may affect the final image; that order is not always correctly implemented in Gerber readers. Overlapping blocks containing only dark objects are safe to use as the order does not affect the image.

2.9 Attributes

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

❏ The function of the file: is it the top solder mask, or the bottom copper layer etc.

❏ The function of a pad: is the pad an component pad, or a via pad, or a fiducial, etc.

👁 **Example:**

This command defines an attribute indicating the file represents the top solder mask.

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

Attributes do not affect the image. A Gerber reader will generate the correct image even if it ignores the attributes.

Attributes can be attached to apertures, objects or to the file as a whole.

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 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 what are the via pads to manufacture the solder mask. The attributes transfer the design intent from CAD to CAM in an unequivocal and standardized manner. This is sometimes rather grandly called "adding intelligence to the image". Without these attributes the fabricator must reverse engineer the design intent of the features in the file, which is a time-consuming and error-prone process.

Attributes are described in detail in the chapter 5.

2.10 Command Summary

Command	Description	Ref.
FS	Format Statement. Sets the coordinate format, e.g. the number of decimals.	4.1
MO	Mode. Sets the unit to inch or mm.	4.2
AD	Aperture Define. Defines an aperture and assigns a D code to it.	4.3
AM	Aperture Macro. Defines a macro aperture template.	4.4
AB	Aperture Block. Defines a block aperture and assigns a D-code to it.	4.6
Dnn (nn≥10)	Sets the current aperture to the aperture with D code nn.	4.7
D01	Interpolate operation. Outside a region statement D01 creates a draw or arc object using the current aperture. Inside it creates a linear or circular contour segment. After the D01 command the current point is moved to the coordinate.	4.8
D02	Move operation. D02 does not create a graphics object but moves the current point to the coordinate.	4.8
D03	Flash operation. Outside a region statement D03 flashes the current aperture. Inside a region statement D03 is not allowed. After the D03 command the current point is moved to the coordinate.	4.8
G01	Sets the interpolation mode to linear.	4.9
G02	Sets the interpolation mode to clockwise circular interpolation.	4.10
G03	Sets the interpolation mode to counterclockwise circular interpolation.	4.10
G74	Sets quadrant mode to single quadrant.	4.10
G75	Sets quadrant mode to multi quadrant.	4.10
LP	Load Polarity. Loads the polarity object transformation parameter.	4.11.2
LM	Load Mirror. Loads the mirror object transformation parameter.	4.11.3
LR	Load Rotation. Loads the rotation object transformation parameter.	4.11.4
LS	Load Scale. Loads the scale object transformation parameter.	4.11.5
G36	Start region statement. This creates a region by defining its contour.	4.12.
G37	Ends region statement.	4.12
SR	Step and Repeat. Open or closes a step and repeat statement.	4.13
G04	Comment.	4.15
TF	Set a file attribute.	5.2
TA	Add an aperture attribute to the dictionary or modify it.	5.3
TO	Add an object attribute to the dictionary or modify it.	5.4
TD	Delete one or all attributes in the dictionary.	5.5
M02	End of file.	4.16

Command Summary

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

2.11 Processing a Gerber File

The image below illustrates how a Gerber file is processed.

Aperture templates dictionary

Apertures dictionary

AM command

AD command

Stream of commands

Syntax parser

Graphics state

Graphics object

Image plane

Commands processor

Stream of graphics objects

M02 command

Gerber file

Legend

A	→	B	A passes stream of data to B
A	Command →	B	As the result of Command execution A forces B to change or perform a task
A	→	B	A processes B
A	---→	B	A affects B

4. 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 2.6). The graphics state holds the set of parameters which influence the operation codes (see 2.4) and thus define the resulting graphics objects.

The AM command (see 4.4) results in creating a macro aperture template (see 2.2). 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 4.4) which are automatically added to the dictionary before file processing is started, as well as 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 4.3) 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 4.7). 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 2.6).

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

After processing the M02 command (see 4.15) 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.

2.12 Glossary

Aperture: A 2D 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. Defines a custom aperture template by combining built-in primitives.

Aperture template: A template is used to create the specific apertures used in the file. The AD command defines the parameters to instantiate the template to a defined aperture. There are three types of templates: standard or built-in apertures, macro apertures and block apertures.

Aperture templates dictionary: The object that holds all the defined aperture templates during the processing of a Gerber file.

Apertures dictionary: The object that holds all the defined apertures during the processing of a Gerber file.

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

Attribute: Metadata that is attached to the file as a whole or to objects in it; it provides extra information without affecting the image.

Attributes dictionary: The object that holds all the current attributes during the processing of a Gerber file.

Bi-level image: A two-dimensional (2D) image represented by two colors, usually black and white.

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

Circular interpolation: Creating a circular segment (circular arc) that is either an arc graphics object or used as a circular contour segment.

Clear: Clearing or erasing part of the image in the image plane. When a graphics object with clear polarity is added to the stream it erases its shape from any image that was already there.

Command: Commands are the basic unit of a Gerber file. Commands create graphics objects, define apertures, manage attributes, modify the graphics state and so on. For historic reasons, there are two syntax styles for commands: function code commands and extended commands.

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 or outline primitives in macro apertures.

Coordinate data: A number whose interpretation is determined by the FS command. It is used to specify the X and Y coordinates of a point in the image plane or a a distance or offset in the X and Y direction.

Coordinate format: The specification of how to convert coordinate data to coordinates. It is file-dependent and is defined by an FS command.

Current aperture: The graphics state parameter that specifies the last aperture selected by a Dnn command. The current aperture is always used to create flashes, draws and arcs.

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

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. Many commands consist of a single data block.

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

Extended commands: Commands enclosed in a pair of '%' characters.

File image: The bi-level 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 commands: Commands consisting of a single data block containing a function code. A function code is a letter 'D', 'G' or 'M' followed by a code number.

Gerber file: A file in the Gerber format.

Gerber format: The vector image file format defined by the current specification and used for representing a bi-level image.

Graphics object: A graphics object is a 2D object with a shape, a size, a position in the plane 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. Attributes can optionally be attached to a graphics object.

Graphics state: The set of parameters that at each moment determine how the operation codes create graphics objects. For example, it determines whether a D01 operation code creates a draw or an arc.

Header: The part of the file from the file beginning to the point where the first operation code is encountered**.** The header typically holds the definitions of file attributes, aperture definitions, scale and unit.

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 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 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.

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 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 statement: The statement creates 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.

Standard attribute: A built-in attribute with a pre-defined semantics. See also user attribute.

Statement: A coherent sequence of commands delimited by an open and close command defining a higher-level structure (block or region).

Step and repeat: A method by which replications of an accumulated block are made to produce multiple copies of a set of graphics objects.

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.

Ucamco
Former Barco ETS

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.
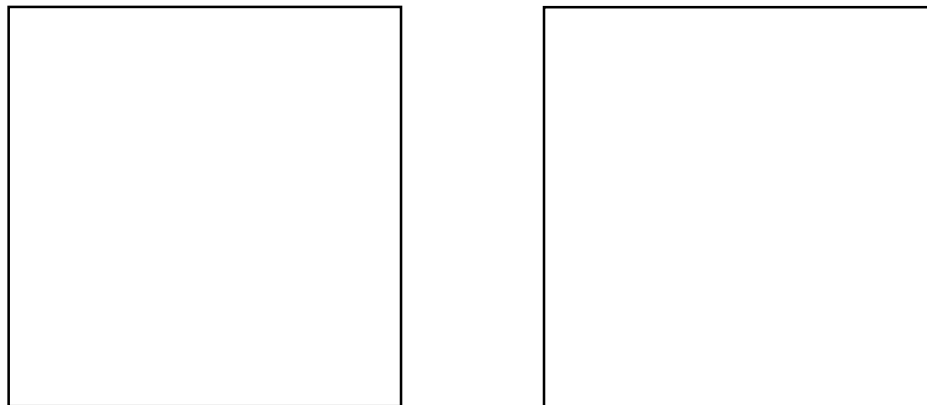
5. Example: two square boxes

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,example*%`	Attribute: the file does not describe 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)

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

Syntax	Comments
X0D01*	Create draw using the current aperture: (5, 5) to (0, 5)
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: Polarities and Apertures**

This example illustrates the use of polarities and various apertures.

6. Example: various shapes

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

Syntax	Comments
G01*	Set linear interpolation mode
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

Syntax	Comments
`X3750000Y1000000I250000J0D01*`	Create arc using the current aperture D10. This creates a complete circle
`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*`	Start a region statement
`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*`	Close the region statement
	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*`	Start the region statement
`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

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

Syntax	Comments
`G37*`	Close the region statement
	This creates the region by filling the created contour
`%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
`M02*`	End of file

2.13.3 **Example: A Drill File**

This example is a drill file.
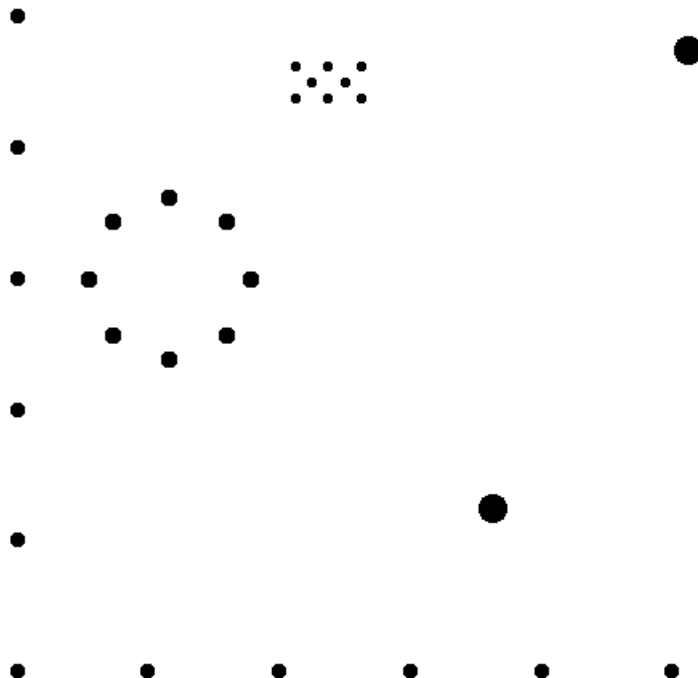
7. Example: drill file

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

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,0.002,0.001*%`	Set the drill tolerance attribute to 2 mil in plus and 1 mil in minus in the 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 plated hole for a component lead, with 14 mil end (or inner) diameter and 2 mil positive and 1 mil negative tolerance.
`%TA.AperFunction,Other,SpecialDrill*%`	Attribute indicates that the following apertures are special drill holes
`%ADD11C,0.024000*%`	Define the aperture: a plated hole for a special purpose, with 24 mil end (or inner) diameter and 2 mil positive and 1 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 plated mechanical drill hole with 43 mil end diameter, and a tolerance of 15 mil in both directions.
`%ADD13C,0.022000*%`	Define the aperture: another tool with the same attributes but a smaller end diameter
`%TD.AperFunction*%`	Remove the .AperFunction aperture attribute from the attributes dictionary
`%TD.DrillTolerance*%`	Remove the .DrillTolerance aperture attribute from the attributes dictionary
`G01*`	Set linear interpolation mode
`D10*`	Set the current aperture: use drill tool 10
`X242000Y275000D03*`	
`Y325000D03*`	

Syntax	Comments

`X217000Y300000D03*`
`X192000Y325000D03*`
`X292000Y275000D03*`
`X192000D03*`
`X292000Y325000D03*`
`X267000Y300000D03*`
	Create several flash graphics objects using the current aperture D10: drill plated component drill holes with diameter 14 mil at indicated coordinates

`D11*`	Set the current aperture: use drill tool 11

`X124000Y0D03*`
`X0Y-124000D03*`
`X-124000Y0D03*`
`X88000Y88000D03*`
`X-88000D03*`
`X0Y124000D03*`
`X88000Y-88000D03*`
`X-88000D03*`
	Create several flash graphics objects using the current aperture D11: drill plated special drill holes with diameter 24 mil at indicated coordinates

`D12*`	Set the current aperture: use drill tool 12
`X792000Y350000D03*` `X492000Y-350000D03*`	Create several flash graphics objects using the current aperture D12: drill plated mechanical drill holes with diameter 43 mil at indicated coordinates
`D13*`	Set the current aperture: use drill tool 13

`X767000Y-600000D03*`
`X567000D03*`
`X-233000Y200000D03*`
`Y400000D03*`
`Y0D03*`
`Y-200000D03*`
`Y-600000D03*`
`Y-400000D03*`
`X-33000Y-600000D03*`
`X167000D03*`
`X367000D03*`
	Create several flash graphics objects using the current aperture D13: drill plated mechanical drill holes with diameter 22 mil at indicated coordinates

`%TF.MD5,b5d8122723797ac635a1814c0` `4c6372b%`	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 5.6.2.

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

2.14 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 consider 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 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.

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. They are used to improve human readability.

Space characters can *only* be used inside strings (see 3.6.6).

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 end-of-block character asterisk '*'. A data block may contain a function code, coordinates data, an extended command, aperture primitive description, variable definition and so on.

Example:

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

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

Tip: One of the strengths of the Gerber format is its human readability. Use line breaks to enhance readability; put one command per line; avoid lines longer than a page width. Do not put a needless line separator within a data block, except after a comma separator in long data blocks.

Copyright Ucamco NV   38

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

3.5 Commands

3.5.1 Commands Overview

Commands are higher level semantic elements of a Gerber file. Commands define the graphics state, create graphics objects, defe apertures, manage attributes and so on. A Gerber file consists of a stream of commands. There is no limitation on the number of commands.

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

Command syntax:

<Command> = <Function code command>|<Extended command>

<Function code command> = <Data block>

<Extended command> = %<Data block>{<Data block>}%

All extended command except the AM consist of a single data block.

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

Copyright Ucamco NV
39

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

```
                            Commands
```

Function code commands

Extended code commands

D-code commands

G-code commands

'End of file' command: M02 code

Unit and coordinate resolution commands: FS, MO codes

Operations: D01, D02, D03 codes

'Interpolation mode' commands: G01, G02, G03 codes

Aperture commands: AD, AM codes

Select aperture' command: Dnn code

'Region mode' commands: G36, G37 codes

'Step & Repeat' command: SR code

'Quadrant mode' commands: G74, G75

'Level polarity' command: LP code

'Comment' command: G04 code

Attribute commands: TF, TA, TD codes

8. Gerber file commands

Copyright Ucamco NV   40

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

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*
```

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 4.1. Other function code commands are described in chapters from 4.7 to 4.15.

3.5.3 **Extended Commands**

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

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

An extended 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 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 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.

👁 **Example:**
```
%AMDONUTFIX*
1,1,0.100,0,0*
1,0,0.080,0,0*%
```

💡 **Tip:** For readability it is recommended to put one data block per line in the AM command.

The syntax for an individual extended command is:

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

Syntax	Comments

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

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 commandss:

❑ 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 Data**

Coordinate data are integers conforming to the rules set by the FS command. See 4.1.1. Coordinate data are used to express coordinates.

3.6.4 **Hexadecimal**

A hexadecimal value is a sequence of characters that matches the regular expression:

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

The letters in a hexadecimal value can be upper case or lower case characters; 'A9' and 'a9' represent the same value.

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.

3.6.7 **Fields**

The fields follow the string syntax in section 3.6.6 with the additional restriction that a field must not contain commas. Fields are intended to represent comma-separated items in strings. If a field must contain a comma it can be represented by the Unicode above.

4 Graphics

4.1 Format Statement (FS)

Coordinate values are expressed as absolute coordinates.

The FS (Format Specification) command specifies the format of the coordinate data. 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.

Coordinate data 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 coordinate data. The FS command defines the lengths of the integer and decimal parts for all coordinate data in the file.

4.1.1 Coordinate Format

The coordinate format specifies the number of integer and decimal places in coordinate data. 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 data 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 4.2).

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.

Coordinate data must have at least one character. Zero therefore must be encoded as "0".

Note: A few files 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 the maximum allowed length of coordinate data 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.

4.1.2 FS Command

The syntax for the FS command is:

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

Copyright Ucamco NV
46

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

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 Ycoordinate datas. The format of X and Y coordinates **must be the same**! <Format> must be expressed as a number NM where N - number of integer positions in coordinate data (0 ≤ N ≤ 6) M - number of decimal positions in coordinate data (4 ≤ M ≤ 6)

4.1.3 **Examples**

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

4.2 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 4.1).

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

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

Copyright Ucamco NV

48

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

4.3 Aperture Definition (AD)

4.3.1 AD Command

The AD command creates an aperture and puts it the 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 4.7).

The AD command must precede the first use of the D-code. 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 AD command uses the name to find the referenced aperture template in the aperture templates dictionary (see 0).

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

👁 **Example:**
```
%ADD10C,.025*%
%ADD10C,0.5X0.25*%
```

4.3.2 Zero-size Apertures

As a general rule, apertures with size zero are not valid, and so are objects created with them.

There is one exception. The C (circular) standard aperture with zero diameter is allowed, and so are objects created with it. Attributes can be attached to them. For the avoidance of doubt, it is the C aperture only where zero-size that can be valid, not another aperture whose shape fortuitously happens to be circular.

Zero-size objects do not affect the image. They can be used to provide meta-information to locations in the image plane.

Allowed does *not* mean recommended, quite in the contrary. If you are tempted to use a zero-size circle, consider whether it is useful, or if there is no proper way to convey the intended information. The goal is not to have zero-size apertures. (Of course, do not simply replace zero-size by a positive size when there is no object just to avoid zero size; this would falsify the image.)

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 just confusing as they direct the reader to look for meta-information that is not there. If there is nothing, put nothing.

4.3.3 **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

4.4 Standard Aperture Templates

4.4.1.1 Circle

The syntax of the circle standard aperture template:

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

Syntax	Comments
C	Indicates the circle aperture template.
<Diameter>	Diameter. A decimal ≥0.
<Hole diameter>	Diameter of a round hole. A decimal >0. If omitted the aperture is solid. See also section 4.4.1.5.

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

👁 **Examples:**
```
%ADD10C,0.5*%
%ADD10C,0.5X0.25*%
```

These commands define the following apertures:

9. Circles

Copyright Ucamco NV

51

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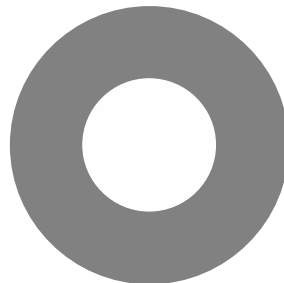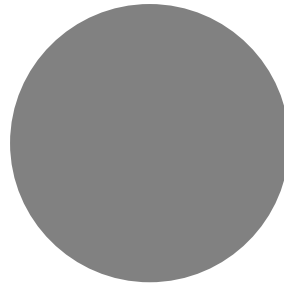
4.4.1.2 Rectangle

The syntax of the rectangle or square standard aperture template:

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

Syntax	Comments
R	Indicates the rectangle aperture template.
<X size> <Y size>	X and Y sizes of the rectangle. Decimals >0. If <X size> = <Y size> the effective shape is a square.
<Hole diameter>	Diameter of a round hole. A decimal >0. If omitted the aperture is solid. See also section 4.4.1.5.

Examples:

```
%ADD22R,0.044X0.025*%
%ADD23R,0.044X0.025X0.019*%
```

These commands define the following apertures:

10. Rectangles

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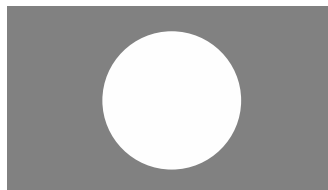
4.4.1.3 Obround

Obround (oval) is a shape consisting of two semicircles connected by parallel lines tangent to their endpoints. It can be viewed as a rectangle where the smallest side is rounded to a half-circle. The syntax of the obround standard aperture template:

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

Syntax	Comments
O	Indicates the obround aperture template.
<X size> <Y size>	X and Y sizes of enclosing box. Decimals >0. If <X size> = <Y size> the effective shape is a circle.
<Hole diameter>	Diameter of a round hole. A decimal >0. If omitted the aperture is solid. See also section 4.4.1.5.

👁 **Example:**
```
%ADD22O,0.046X0.026*%
%ADD22O,0.046X0.026X0.019*%
```

These commands define the following apertures:

11. Obrounds

Copyright Ucamco NV

53

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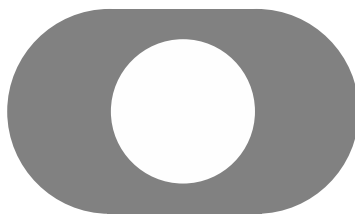
4.4.1.4 Polygon

Creates a *regular* polygon aperture. The syntax of the polygon standard aperture template:

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

Syntax	Comments
P	Indicates the polygon aperture template.
<Outer diameter>	Diameter of the circumscribed circle, i.e. the circle through the polygon vertices. A decimal > 0.
<Number of vertices>	Number of vertices n, 3 ≤ n ≤ 12. A decimal.
<Rotation angle>	The rotation angle, in degrees counterclockwise. A decimal.
	With rotation angle zero there is a vertex on the positive X-axis through the aperture center.
<Hole diameter>	Diameter of a round hole. A decimal >0. If omitted the aperture is solid.
	See also section 4.4.1.5.

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

These commands define the following apertures:

12. Polygons

Having a question or remark about the spec? Please contact us at gerber@ucamco.com
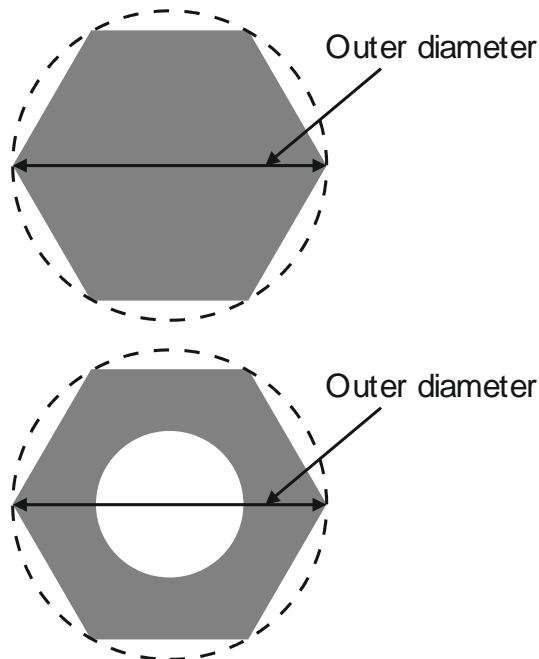
4.4.1.5 Round Hole in Standard Apertures

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 strictly 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*
```


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

Note that the draw is visible through the hole.

4.5 Macro Aperture (AM)

The AM command creates a macro aperture template and adds it to the aperture template dictionary (see 0). 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.

4.5.1 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 4.5.4. 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).

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

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.

4.5.2 **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 4.4.1.5. 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*
```


14. Macro aperture with a hole above a draw

Note that the draw is still visible through the hole.

Copyright Ucamco NV   58

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

4.5.3 **Rotation Modifier**

All primitives can be rotated.

The rotation is always around the origin of the macro definition, i.e. its point (0, 0). The rotation is *not* around the geometric center, unless of course this center happens to coincide with the origin.

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 which 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

Copyright Ucamco NV

59

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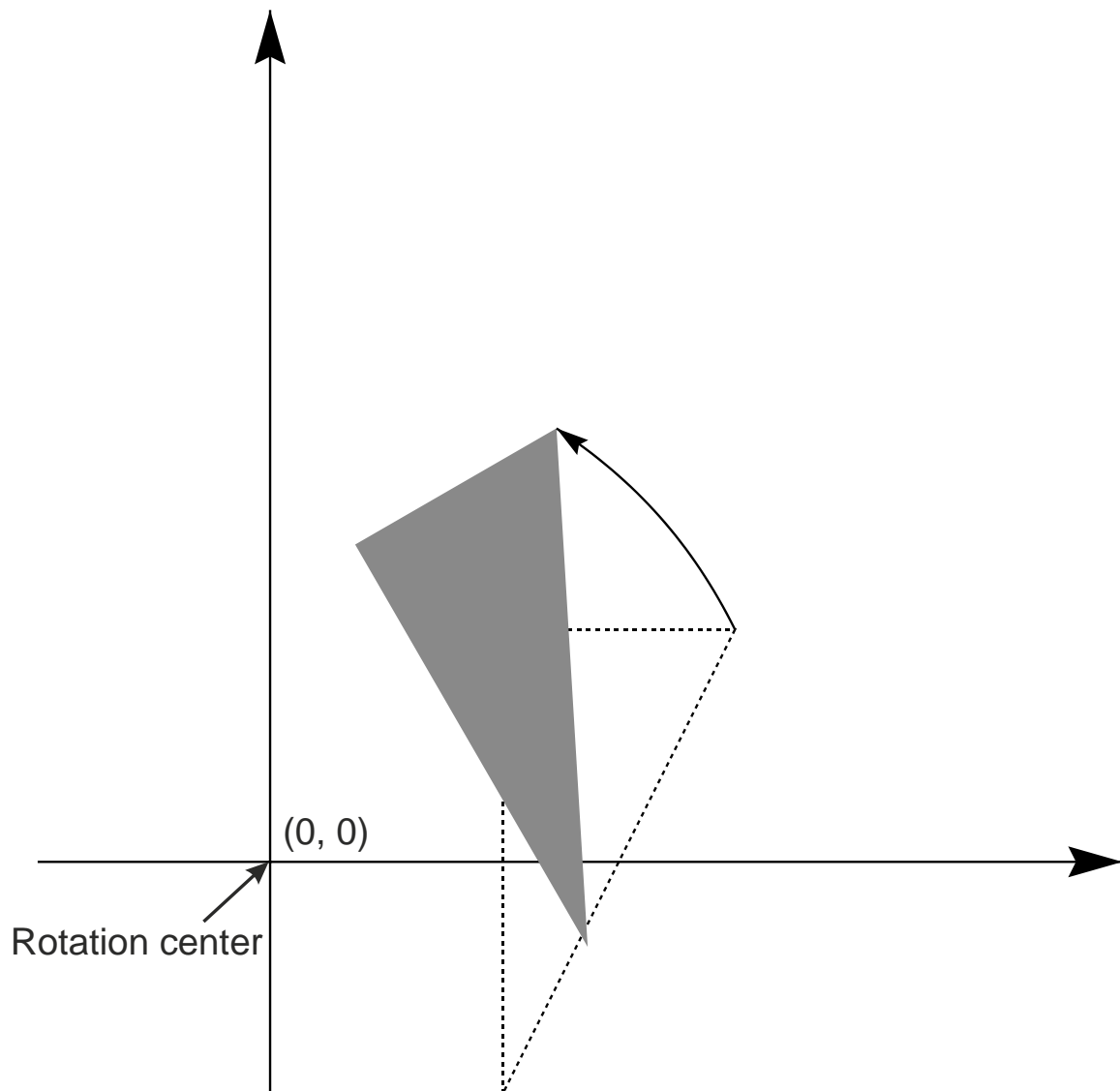
15. Rotated triangle

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

```
%ADD33AMTRIANGLE_30*%
```

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

The picture below illustrates how the whole macro is rotated by rotating all primitives with the same angle.
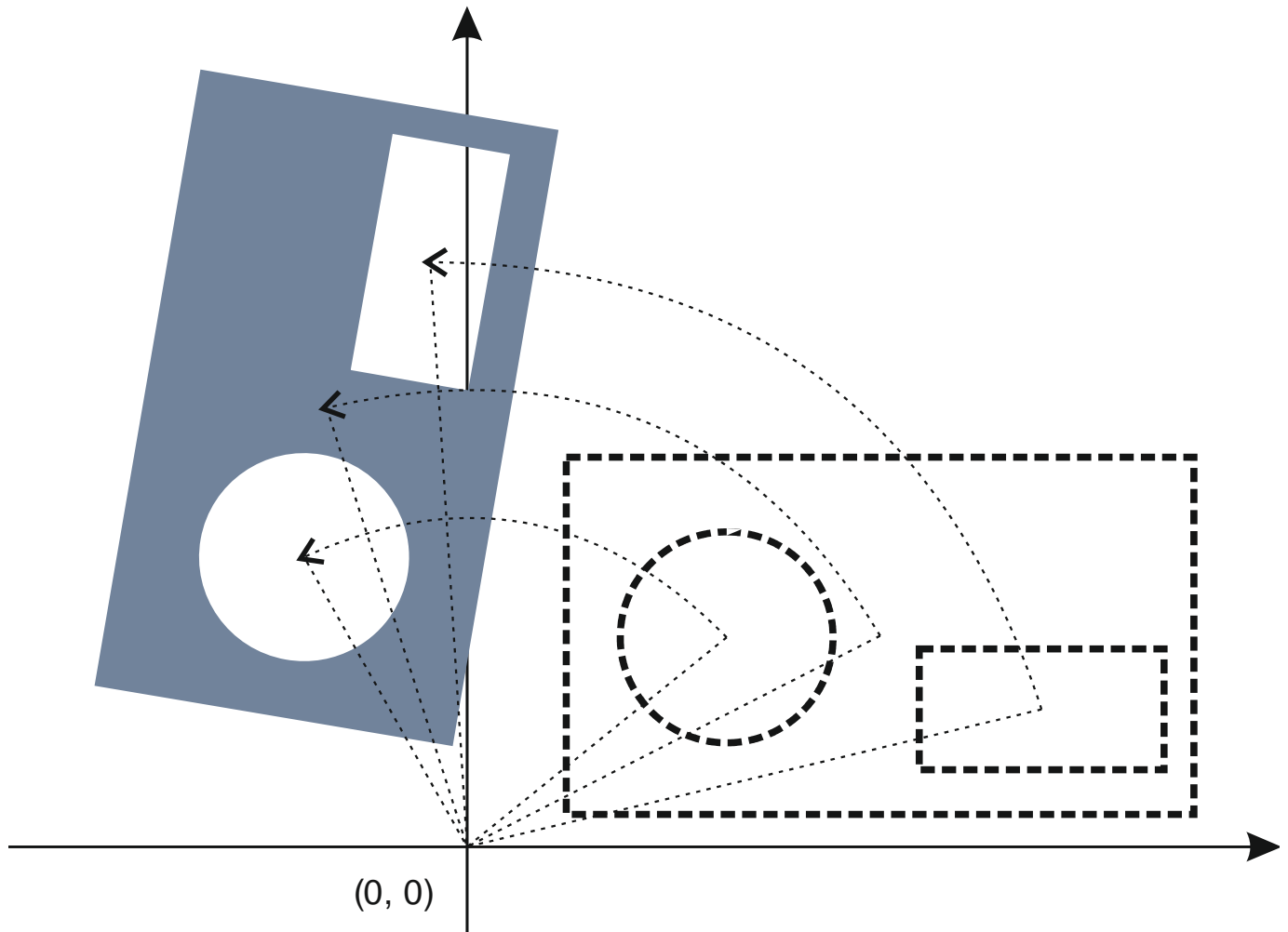
16. Rotation of an aperture macro composed of several primitives

4.5.4 **Primitives**

4.5.4.1 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 in section 3.1.

👁 **Example:**

```
%AMBox*
0 Rectangle with rounded corners, with rotation*
0 The origin of the aperture is it's center*
0 $1 X-size*
0 $2 Y-size*
0 $3 Rounding radius*
0 $4 Rotation angle, in degrees counterclockwise*
0 Add two overlapping rectangle primitives as box body*
21,1,$1,$2-$3-$3,0,0,$4*
21,1,$2-$3-$3,$2,0,0,$4*
0 Add four circle primitives for the rounded corners*
$5=$1/2*
$6=$2/2*
$7=2X$3*
1,1,$7,$5-$3,$6-$3,$4*
1,1,$7,-$5+$3,$6-$3,$4*
1,1,$7,-$5+$3,-$6+$3,$4*
1,1,$7,$5-$3,-$6+$3,$4*%
```

The lines starting with 0 are comments and do not affect the image.

Copyright Ucamco NV 62

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

4.5.4.2 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	Center X coordinate. A decimal.
4	Center Y coordinate. A decimal.
5	Rotation angle of the center, in degrees counterclockwise. A decimal.
	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.
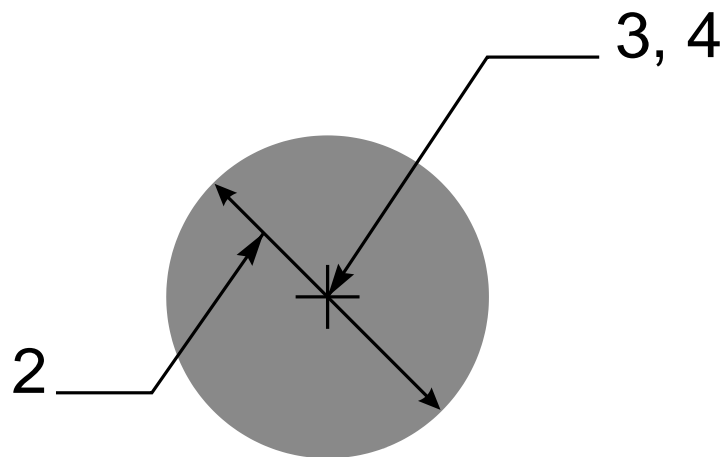
17. Circle primitive

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

👁 **Example:**
```
%AMCIRCLE*
1,1,1.5,0,0,0*%
```

Copyright Ucamco NV

63

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

4.5.4.3 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	Width of the line. A decimal ≥ 0.
3	Start point X coordinate. A decimal.
4	Start point Y coordinate. A decimal.
5	End point X coordinate. A decimal.
6	End point Y coordinate. A decimal.
7	Rotation angle, in degrees counterclockwise. A decimal. The primitive is rotated around the origin of the macro definition, i.e. the (0, 0) point of macro coordinates
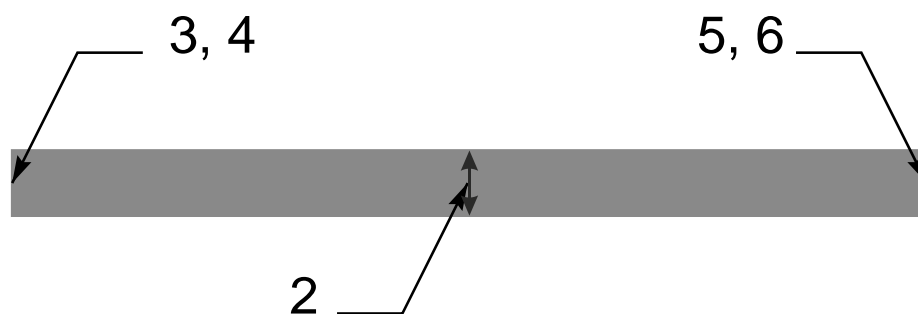
18. Vector line primitive

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*%
```

4.5.4.4 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	Width. A decimal ≥ 0.
3	Height. A decimal ≥ 0.
4	Center point X coordinate. A decimal.
5	Center point Y coordinate. A decimal.
6	Rotation angle, in degrees counterclockwise. A decimal. The primitive is rotated around the origin of the macro definition, i.e. (0, 0) point of macro coordinates.
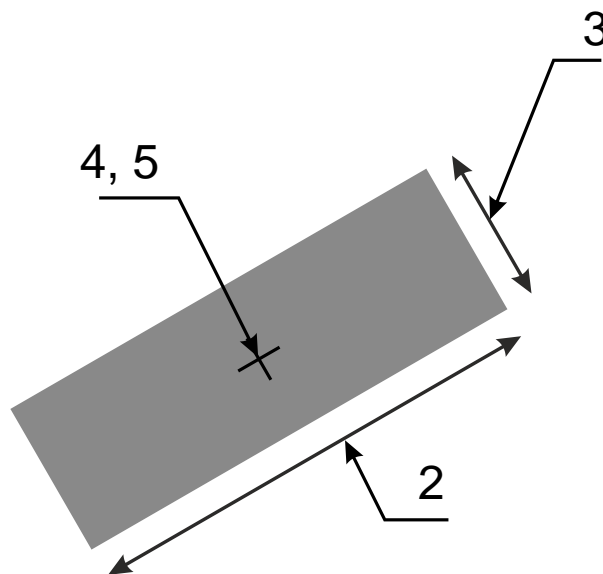
19. Center line primitive

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*%
```

Copyright Ucamco NV

65

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

4.5.4.5 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; consequently there must be at least one subsequent point (to close the outline). The outline of the primitive must be a contour according to 4.12.2, consisting of linear segments only.

Modifier number	Description
1	Exposure off/on (0/1)
2	The number of subsequent points n. An integer ≥1.
3, 4	Start point X and Y coordinates. Decimals.
5, 6	First subsequent point X and Y coordinates. Decimals.
...	Further subsequent point X and Y coordinates. Decimals.
3+2n, 4+2n	Last subsequent point X and Y coordinates. Decimals. *Must* be equal to the start point.
5+2n	Rotation angle, in degrees counterclockwise, a decimal. The primitive is rotated around the origin of the macro definition, i.e. the (0, 0) point of macro coordinates.

⚠️ **Warning:** Make no mistake: The number of subsequent points is the number of vertices of the outline or the number of coordinate pairs *minus one*.
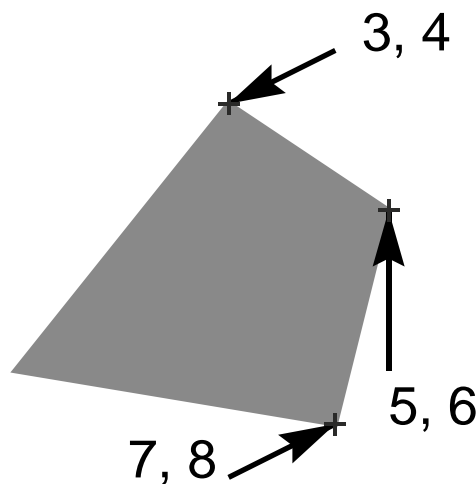
20. Outline primitive

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*%
```

4.5.4.6 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. A decimal. The first vertex is on the positive X-axis through the center point when the rotation angle is zero.
3	Center point X coordinate. A decimal.
4	Center point Y coordinate. A decimal.
5	Diameter of the circumscribed circle. A decimal ≥ 0.
6	Rotation angle, in degrees counterclockwise. A decimal. With rotation angle zero there is a vertex on the positive X-axis through the aperture center. 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.
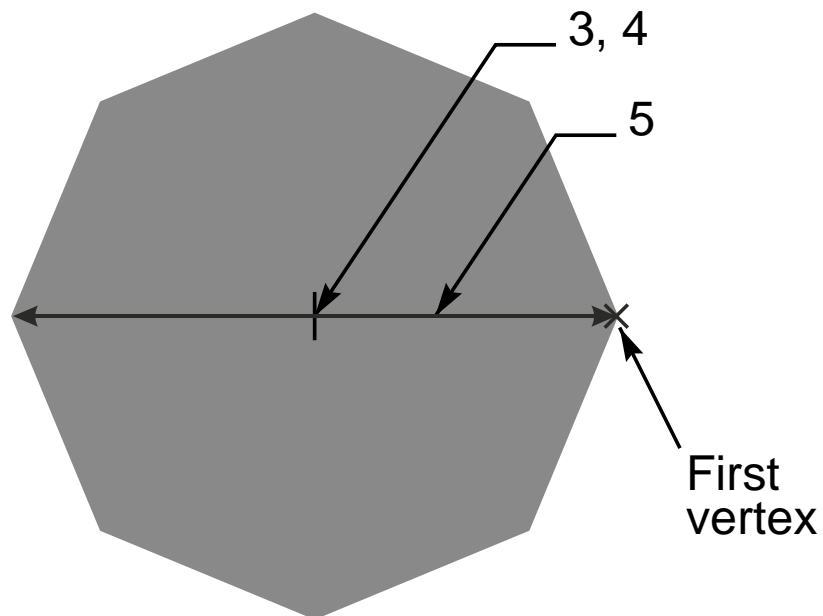
21. Polygon primitive

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

👁 **Example:**

Copyright Ucamco NV   68

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

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

4.5.4.7 Moiré, Primitive Code 6

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

Modifier number	Description
1	Center point X coordinate. A decimal.
2	Center point Y coordinate. 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, in degrees counterclockwise. A decimal. 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.
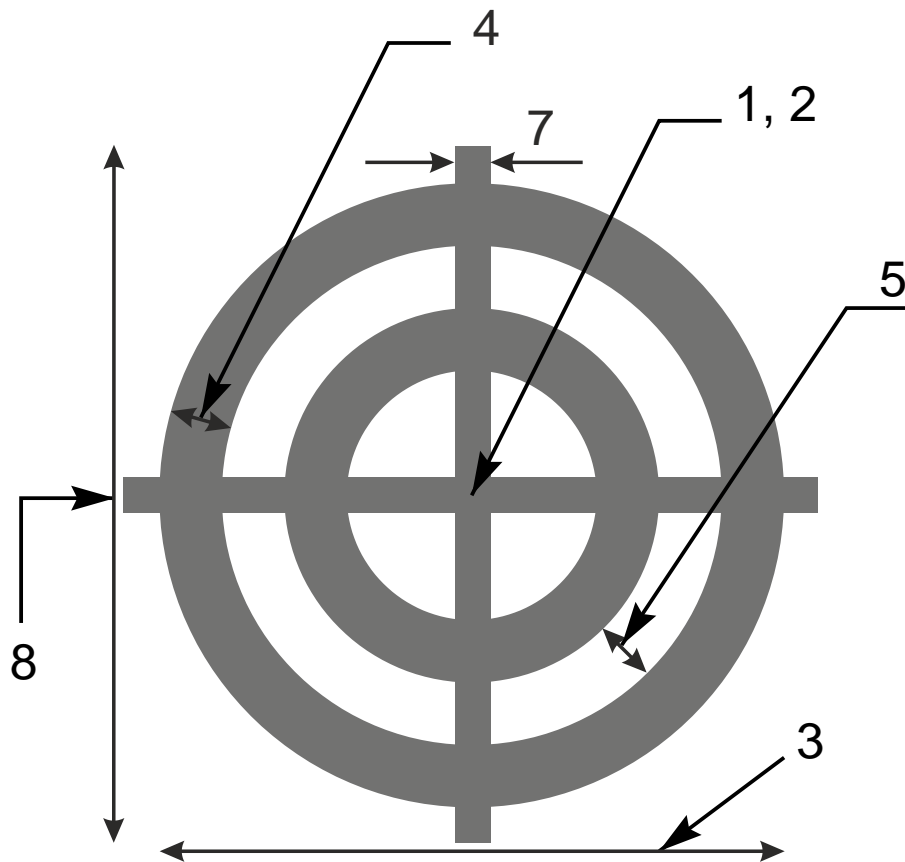
22. Moiré primitive

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*%
```

4.5.4.8 Thermal, Primitive Code 7

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

Modifier number	Description
1	Center point X coordinate. A decimal.
2	Center point Y coordinate. A decimal.
3	Outer diameter. A decimal > inner diameter
4	Inner diameter. A decimal ≥ 0
5	Gap thickness. A decimal < (outer diameter)/√2. The gaps are on the X and Y axes through the center without rotation. They rotate with the primitive.
6	Rotation angle, in degrees counterclockwise. A decimal. The primitive is rotated around the origin of the macro definition, i.e. (0, 0) point of macro coordinates. Note: Rotation is only allowed if the primitive center point coincides with the origin of the macro definition.
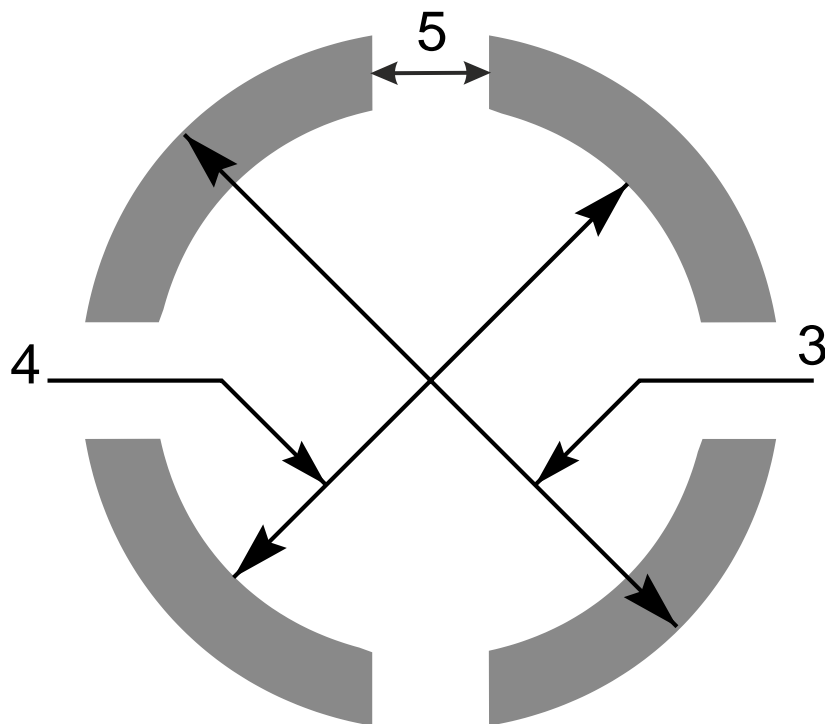
23. Thermal primitive

ⓘ **Note:** If the (gap thickness)*√2 ≥ (inner diameter) the inner circle disappears. This is not invalid.

Copyright Ucamco NV

72

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

4.5.5 **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 A macro variable name must be a '$' character followed by an integer >0, for example $12. (This is a subset of names allowed in 3.6.5.)

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 macro definition.

4.5.5.1 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.

4.5.5.2 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 (lower case, preferred)	Multiply
X (upper case, *not* preferred)	Multiply
/	Divide. The result is a decimal; it is not rounded or truncated to an integer.

Arithmetic operators

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

- ❑ '(' and ')'
- ❑ 'x', 'X' and '/'
- ❑ '+' and '-'

Example:

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

The corresponding AD command could look like:

```
%ADD146Rect,0.0807087X0.1023622X0.0118110X0.5000000X0.3000000*%
```

4.5.5.3 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*%
```

Copyright Ucamco NV   75

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

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.

👁 **Examples:**
```
%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*%
```

4.5.6 **Examples**

4.5.6.1 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*%
```

4.5.6.2 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.

4.5.6.3 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.

4.5.6.4 A useful macro

The following example creates a rectangle with rounded corners, useful as SMD pad.

It uses the following construction:
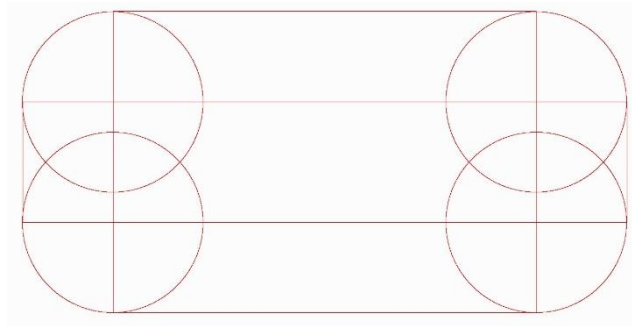
24. Construction of the Box macro

```
%AMBox*
0 Rectangle with rounded corners, with rotation*
0 The origin of the aperture is it's center*
0 $1 X-size*
0 $2 Y-size*
0 $3 Rounding radius*
0 $4 Rotation angle, in degrees counterclockwise*
0 Add two overlapping rectangle primitives as box body*
21,1,$1,$2-$3-$3,0,0,$4*
21,1,$2-$3-$3,$2,0,0,$4*
0 Add four circle primitives for the rounded corners*
$5=$1/2*
$6=$2/2*
$7=2X$3*
1,1,$7,$5-$3,$6-$3,$4*
1,1,$7,-$5+$3,$6-$3,$4*
1,1,$7,-$5+$3,-$6+$3,$4*
1,1,$7,$5-$3,-$6+$3,$4*%
```

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

4.6 Block Aperture (AB)

4.6.1 Overview of block apertures

The AB command creates *block apertures*. The command stream between the opening and closing AB command defines a block aperture which is then stored in the aperture dictionary. Thus the AB commands add an aperture to the dictionary directly, without needing a template and an AD command. The effect of block apertures is governed by the LM, LR, LS and LP commands as any other aperture. When a block aperture is flashed the transformed –mirrored, rotated and scaled – objects are added to the graphics object stream.

While a standard or macro aperture always adds a *single* graphics object to the stream, a block aperture can add *any number* of objects, with different polarities. A block aperture is not a single graphics object but a set of objects. Standard and macro apertures always have a single polarity while block apertures can contain both dark and clear parts.

As with any other aperture, the flash operation updates the current point but otherwise leaves the graphics state unmodified. (The graphics state is *not* set to the value it had after the block statement defining the block. A block is not a macro command but simply a set of graphics objects.)

If the polarity is clear (LPC) when the block aperture is flashed the polarity of all objects in the block is toggled (clear becomes dark, and dark becomes clear). These toggles proceeds through all nesting levels. If the polarity is dark (LPD) then the block aperture is inserted as is. In the following example the polarity of objects in the flash of block D12 will be toggled.

```
%ABD12*%
…
%AB*%
….
D12*
%LPC*%
X-2500000Y-1000000D03*
```

4.6.2 AB - Aperture Block command

The syntax for the AB command is:

<AB command> = AB[<block D-code>]*

Syntax	Comments
AB	AB for Aperture Block. Opens a block statement.
<block D-code>	The D-code under which the block is stored in the aperture dictionary.

Examples:

Syntax	Comments
%ABD12*%	Opens of the definition of aperture D12
%AB*%	Closes the current block statement.

The section between the opening and closing AB commands can contain nested AB commands. The resulting apertures are stored in the library and are available subsequently over the file, also outside the enclosing AB section.

AB statements can contain other AB or SR statements – see 4.14. The names defined by an AB statement is not restricted in scope but is global over the whole file.

The AB command itself does not affect the graphics state.

The AB command was introduced in revision 2016.12

4.6.3 **Usage of Block Apertures**

The main purpose of block apertures is to repeat a sub-image without copying the generating commands. Aperture blocks can be repeated at *any* location and *individually* mirrored, rotated and scaled. Blocks are used to create panels without duplicating the data. Clear (LPC) objects are used to mask out underlying copper balancing patterns in the panel. Block apertures are more powerful that the SR command and will replace it over time:the SR only allows repeats on a regular grid without mirror, rotate or scale, without nesting.

The second purpose of block apertures is to complement macro apertures. Blocks are simpler to create. However, macros can have parameters and blocks cannot. Macro outline primitives support linear segments only while blocks can contain contours with both linear and circular segments. A block aperture consisting of a single region creates a single object with one polarity– as with standard or macro apertures. Thus, single object apertures of any shape can easily be created. In PCB design to fabrication data transfer block apertures can define pads. Such block apertures ideally consist of a single object (region). However, multi-object single polarity blocks can have a use. Pads are sometimes painted or stroked; such jobs are very hard to handle in CAM as pad locations must be reverse-engineered. Defining a block aperture with the painting of a single pad and then flashing it at the pad locations is a big step forward as the pad locations are now clear. Such a usage may be an intermediate step towards flashing pads with proper single object apertures.

Do not use blocks – or macros - when a standard aperture is available. Standard apertures are built-in and therefore are processed faster.

4.6.4 **Example**

Example 2: a complete Gerber file with nested blocks

```
G04 Ucamco copyright*
%TF.GenerationSoftware,Ucamco,UcamX,2016.04-160425*%
%TF.CreationDate,2016-04-25T00:00;00+01:00*%
%FSLAX36Y36*
%MOMM*%*
G04 Define standard apertures*
%ADD10C,7.500000*%
%ADD11C,15*%
%ADD12R,20X10*%
%ADD13R,10X20*%
G04 Define block aperture D100, consisting of two draws and a round dot*
%ABD100*%
D10*
X65532000Y17605375D02*
Y65865375D01*
X-3556000D01*
D11*
X-3556000Y17605375D03*
%AB*%
G04 Define nested block D101, consisting of 2x2 flashes of D100
%ABD101%*
D100*
X-30000000Y10000000D03*
X-30000000Y10000070D03*
X-29999900Y10000000D03*
X-29999900Y10000070D03*
%AB*%
G04 Define nested block D102, consisting of 2x3 flashes of D101 and 1 flash of D12*
%ABD102%*
D101*
X-30000000Y10000000D03*
X-30000000Y10000160D03*
X-30000000Y10000320D03*
X-29999770Y10000000D03*
X-29999770Y10000160D03*
X-29999770Y10000320D03*
D12*
X19500000Y-10000000D03*
%AB*%
G04 Flash D13 twice outside of blocks*
D13*
X-30000000Y10000000D03*
X143000000Y-30000000D03*
G04 Flash nested block D102 3x2 times
D102*
X-30000000Y10000000D03*
X-30000000Y10000520D03*
```

```
X-29999500Y10000000D03*
X-29999500Y10000520D03*
X-29999000Y10000000D03*
X-29999000Y10000520D03*
M02*
```

Having a question or remark about the spec? Please contact us at gerber@ucamco.com
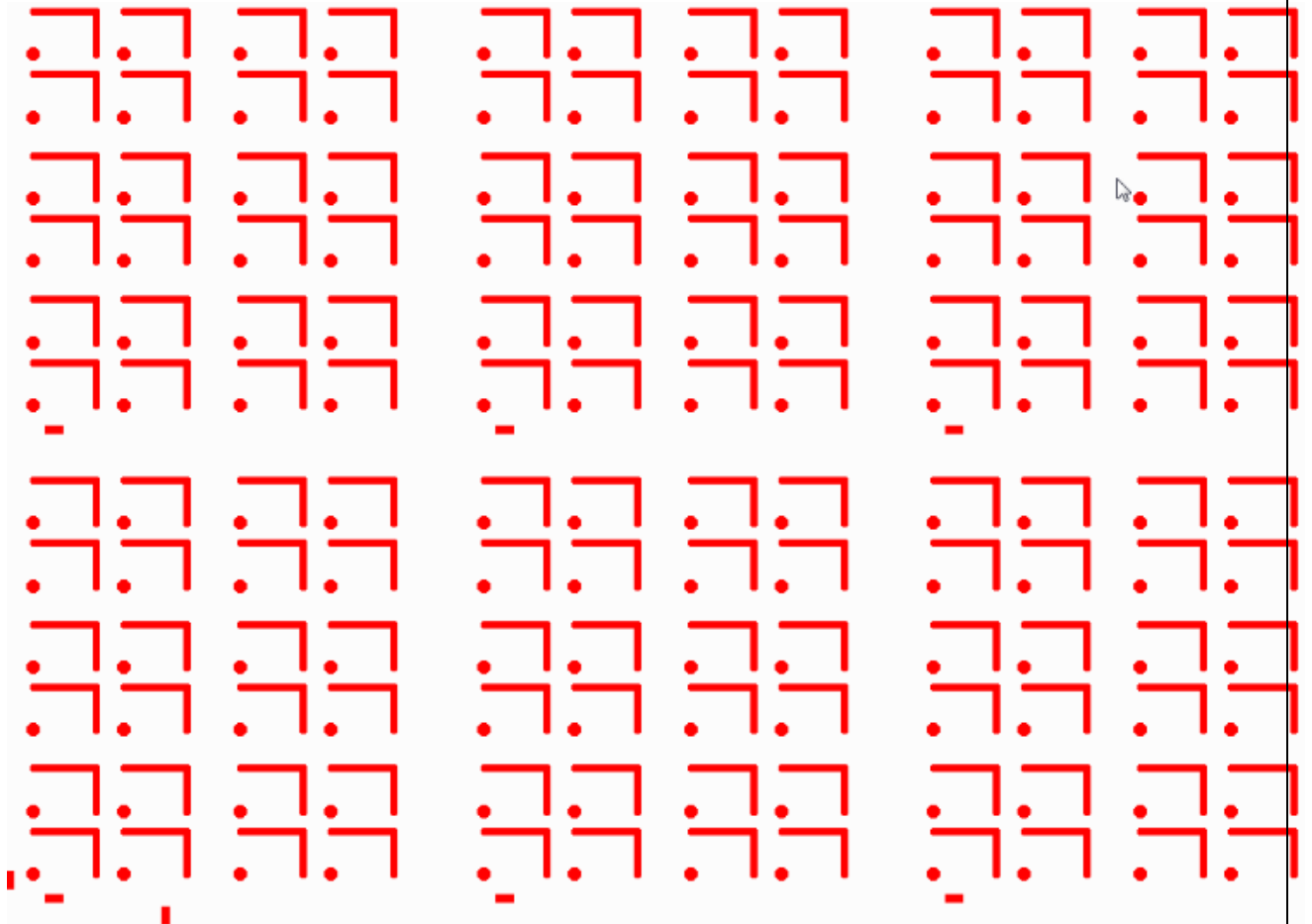
4.7 Current Aperture (Dnn)

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 2.6.

👁 **Example:**

```
D10*
```

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

4.8 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 outside a region statement or to a contour segment inside.

❑ 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 2.6).

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 2.4.

❑ Region building. The segments form contour that defines a region, see 4.12.

Outside a region statement stroking is used to convert a segment into draw or arc graphics object. Inside a region statement a segment becomes the linear or circular contour segment.

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 4.9.

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 4.10 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					
	Inside a region statement (G36)			Outside a region statement (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°)
	Moves current point			Moves current point		
D02	Closes current contour and moves current point			Moves current point only		
D03	Not allowed			Flash; moves current point		

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.

4.8.1 **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. They are mandatory in D01 operations in circular interpolation mode (see 4.8.2) and only allowed there.
<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 data. The coordinate data 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. Similar for Y.

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

👁 **Examples:**

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

`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*
```

4.8.2 **D01 Command**

The interpolation command defines a draw or arc graphics object, depending on the interpolation mode. See sections 4.9.2 and 4.10.5.

4.8.3 **D02 Command**

Performs a move operation, moving the current point to a new value. The syntax for the D02 code (move) operation is the following:

> **<D02 operation> = [X<Number>][Y<Number>]D02***

Syntax	Comments
X<Number>	<Number> is coordinate data – see section 3.6.3. It defines the X coordinate of the new current point. The default is the X coordinate of the old current point.
XY<Number>	<Number> is coordinate data – see section 3.6.3. It defines the Y coordinate of the new current point. The default is the Y coordinate of the old current point.
D02	Move operation code

The D02 command sets the new value for the current point. On top of that, inside a region statement it also closes the current contour. (see 4.12).

👁 **Example:**

```
X200Y1000D02*
```

4.8.4 **D03 Command**

Performs a flash operation, creating a flash graphics object. After the flash operation the current point is set to the origin of the flash

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

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

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

Syntax	Comments
X<Number>	<Number> is coordinate data – see section 3.6.3. It defines the X coordinate of the aperture origin.
	The default is the X coordinate of the old current point.
Y<Number>	<Number> is coordinate data – see section 3.6.3. It defines the Y coordinate of the aperture origin.
	The default is the Y coordinate of the old current point.
D03	Flash operation code

⚠ **Warning:** D03 operation is not allowed in a region statement.

👁 **Example:**
```
X1000Y1000D03*
```

4.8.5 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 4.9, G03 – see 4.10), region statement (G36, G37 – see 4.12), quadrant mode (G74, G75 – see 4.10).

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

4.9 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.

The G01 command sets linear operation.

4.9.1 G01 Command

The syntax for the G01 command is as follows:

<G01 command> = G01*

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

Example:

```
G01*
```

4.9.2 D01 Command

In G01 mode the interpolate command Dcreates a draw. After the D01 command the current point is moved to the end point of the draw. The D01 command syntax is:

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

Syntax	Comments
X<Number>	<Number> is coordinate data – see section 3.6.3. It defines the X coordinate of the straight segment. The default is the X coordinate of the old current point.
Y<Number>	<Number> is coordinate data – see section 3.6.3. It defines the Y coordinate of the straight segment. The default is the Y coordinate of the old current point.
D01	Interpolate operation code

Example:

```
G01*
X200Y200D01*
```

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

4.10.1 Circular Arc Overview

A circular arc is a circular segment created by a D01 (interpolate) operation with the graphics state set to circular interpolation. Outside a region statement a track is added to the graphics object stream, inside a contour segment is added to the current contour.

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 25.
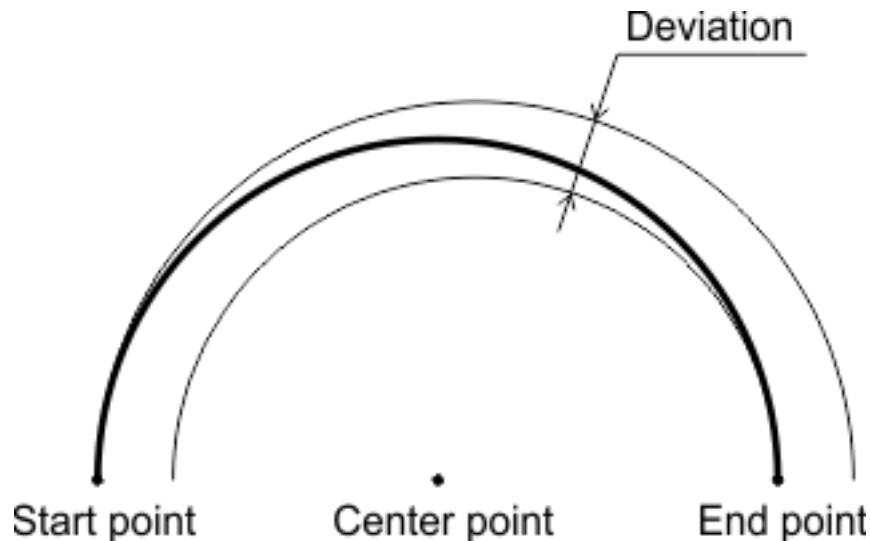
25. Arc with a non-zero deviation

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 26.
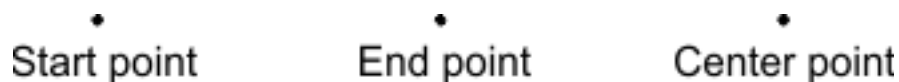
26. Nonsensical center point

Note that self-intersecting contours are not allowed, see 4.12.2. 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 4.1.1.

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

4.10.2 **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*
```

4.10.3 **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*
```

4.10.4 **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*
```

4.10.5 **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*
```

4.10.6 **D01 Command**

In G02 or G03 mode the interpolate command D01 creates an arc. After the D01 command the current point is moved to the end point of the arc. The D01 command syntax is:

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

Syntax	Comments
X<Number>	<Number> is coordinate data – see section 3.6.3. It defines the X coordinate of the circular arc.
	The default is the X coordinate of the old current point.
Y<Number>	<Number> is coordinate data – see section 3.6.3. It defines the Y coordinate of the circular arc.
	The default is the Y coordinate of the old current point.
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.
	The default is a 0 distance.
	<Number> is coordinate data – 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.
	The default is a 0 distance.
	<Number> is coordinate data – see section 3.6.3.
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.

Copyright Ucamco NV   93

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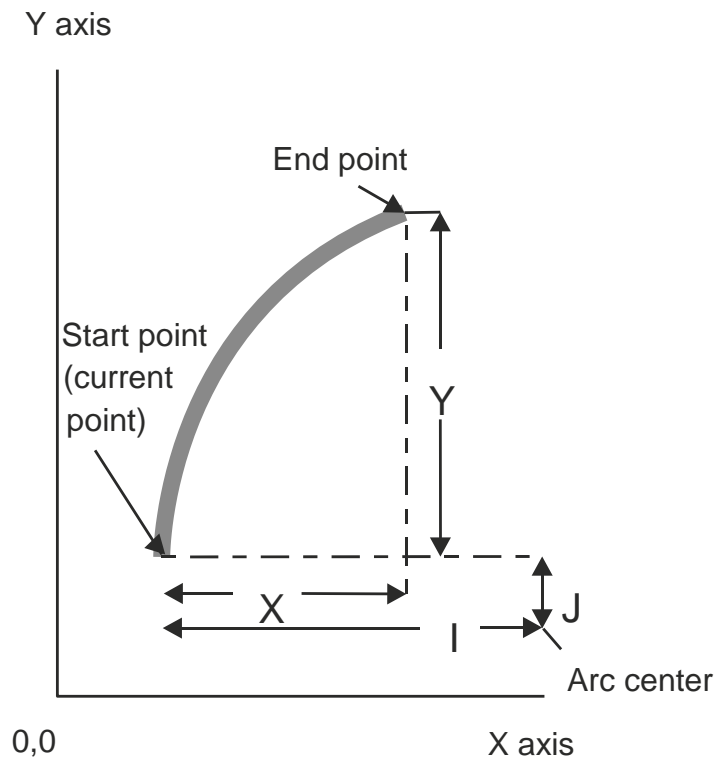
27. Circular interpolation example

ⓘ **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.

4.10.7 **Example: Single Quadrant Mode**

Syntax	Comments

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

`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)
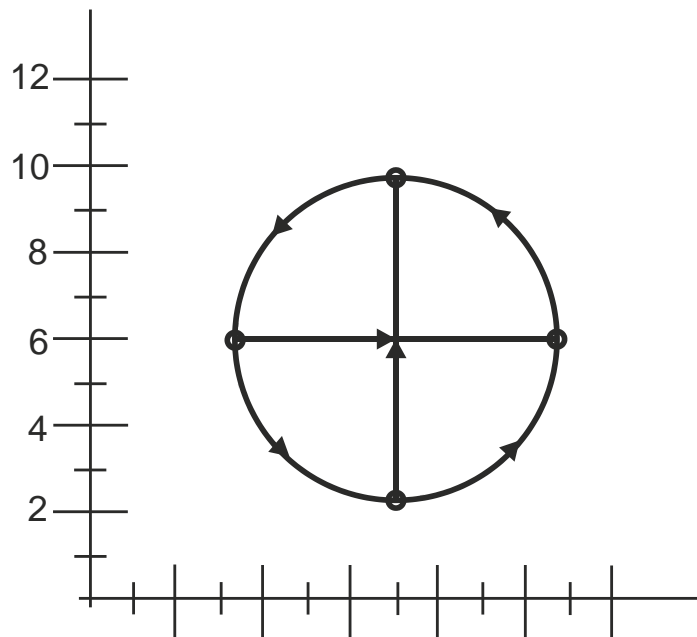
28. Single quadrant mode example: arcs and draws
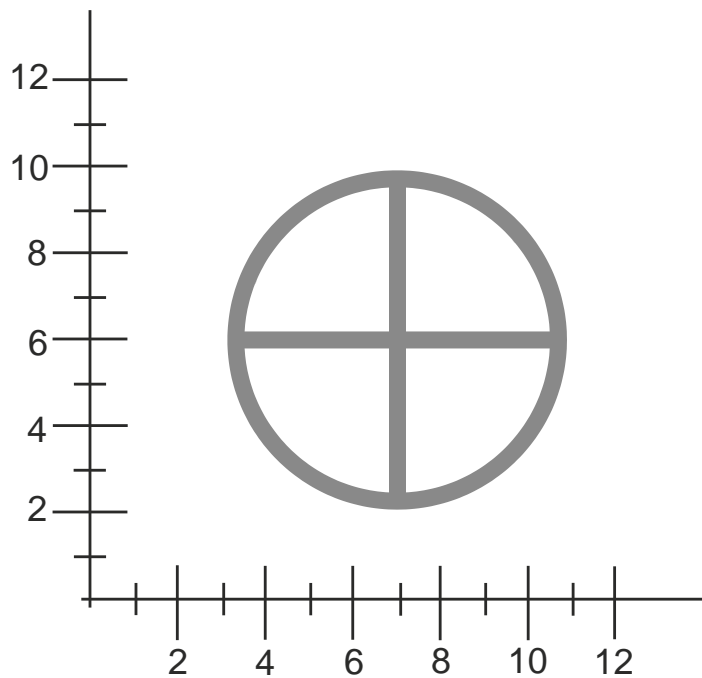
29. Single quadrant mode example: resulting image

4.10.8 **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)

Copyright Ucamco NV   96

Having a question or remark about the spec? Please contact us at gerber@ucamco.com
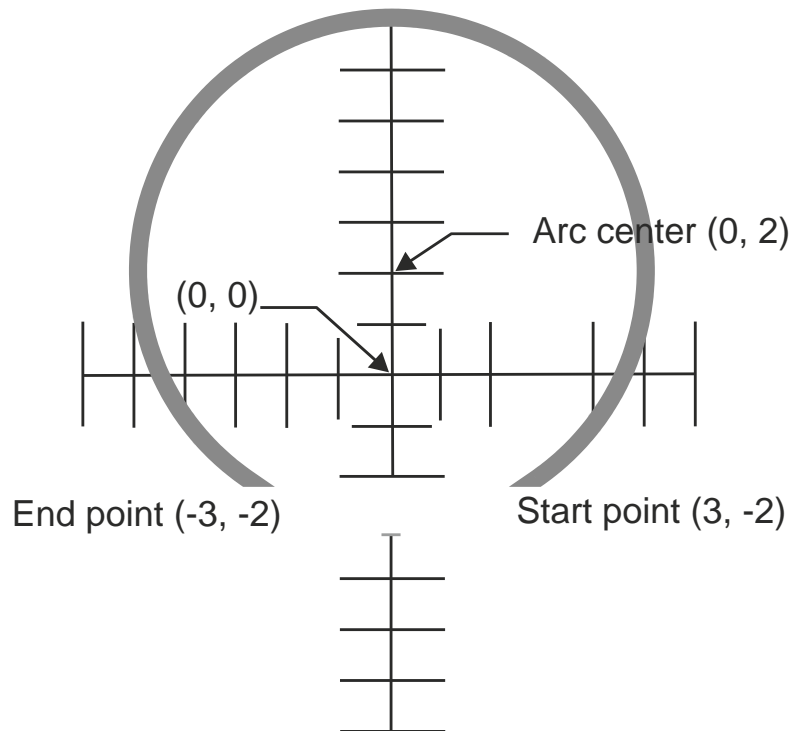
30. Multi quadrant mode example: resulting image

4.10.9 **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 4.10.1. 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 4.1.1.

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.

4.10.10 **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.

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

Syntax	Comments
D10*	Set the current aperture to D10 aperture
G01*	Set linear interpolation mode
X0Y600D02*	Set the current point to (0, 6)
G74*	Set single quadrant mode
G02*	Set clockwise circular interpolation mode
X0Y600I500J0D01*	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*	Set the current aperture to D10 aperture
G01*	Set linear interpolation mode
X0Y600D02*	Set the current point to (0, 6)
G75*	Multi quadrant mode
G02*	Set clockwise circular interpolation mode
X0Y600I500J0D01*	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.

Copyright Ucamco NV

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

98

4.11 Object Option Parameters (LP, LM, LR, LS)

4.11.1 Overview

The commands LP, LM, LR and LS load the object option graphics state parameters:

Object option parameter commands	
Command	**Option Parameter**
LP	Polarity
LM	Mirror
LR	Rotate
LS	Scaling

An object option parameter transforms the objects when adding them to the graphics object stream. The polarity option directly sets the polarity of the objects. They affect the shape of the objects by *temporarily* transforming the current aperture before the object is created, and restoring it to the original state afterwards. Consequently, as the current aperture does not affect regions shape, the object parameters do not affect it either. Flashes (D03) are mirrored/rotated/scaled around the flash point according to the option parameters. interpolations (D01) have their shape altered; however, the only useful applications of object parameters on interpolations seems to rotate a square aperture to align it with a draw.

Option parameters become effective immediately after loading and remain in effect until a new value is loaded. No other command alters the option parameters. The option parameters do not affect apertures or attributes.

👁 **Example on how the parameter changes affect the image**

`D123*`	Select D123
`X5000Y7000D03*`	Flash D123
`%LR90.0*%`	Set object rotation to 90 degrees
`X6000Y8000D03*`	Flash D123 rotated 90 degrees
`D124*`	Select D124
`X6000Y8000D03*`	Flash D124 rotated 90 degrees
`%LR0.0*%`	Reset object rotation to 0 degrees
`X7000Y9000D03*`	Flash D124, not rotated
`D123*`	Select D123
`X1000Y2000D03*`	Flash D123, this is the original, not rotated

👁 **Example of the effect on interpolations (draws and arcs)**

`%MOMM*%`	
`%FSLAX26Y26*%`	
`%AD1C,1*%`	Define D1 as a 1mm circle
`G01*`	Set linear interpolation
`X00000000Y00000000D02*`	Move to origin
`X01000000D01*`	Draw a 1mm thick line
`%LS1.5*%`	Set scale factor to 1.5

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

```
X02000000D01*                    Draw a 1.5mm thick line
M02%
```

Copyright Ucamco NV   100

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

4.11.2 Load Polarity (LP)

The LP command sets the *object polarity*. This command can be used multiple times in a file. The polarity option remains in effect until modified by another LP command.

Polarity can be either *dark* or *clear*. Its effect is explained in 2.7. Section 4.12.5.7 gives an example of its use.

The syntax for the LP command is:

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

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

4.11.3 Load Mirroring (LM)

The LM command sets the *object mirroring*. This command can be used multiple times in a file. The mirroring option remains in effect until modified by another LM command.

The mirroring option defines the mirroring axis. The object is mirrored around its *origin* (which may not be its geometric center). Mirroring is set as is, it is not cumulative.

The syntax for the LM command is:

<LM command> = LM(N|X|Y|XY)*

Syntax	Comments
LM	LM for Load Mirroring
N|X|Y|XY	N – No mirroring X – Mirroring *along* the X axis; mirror left to right; the signs of the x coordinates are inverted Y – Mirroring *along* the Y axis; mirror top to bottom; the signs of the y coordinates are inverted XY – Mirroring *along* both axes; mirror left to right and top to bottom; the signs of both the x *and* y coordinates are inverted

Mirroring is performed *before* the rotation.

The LM command was introduced in revision 2016.12.

4.11.4 Load Rotation (LR)

The LR command sets the *object rotation*. This command can be used multiple times in a file. The mirroring option remains in effect until modified by another LR command.

The rotation value defines the rotation angle. The object is rotated around its *origin* (which may not be its geometric center). Rotation is set as is, it is not cumulative.

The syntax for the LR command is:

<LR command> = LR<Rotation>*

Syntax	Comments
LR	LR for Load Rotation
<Rotation>	The rotation angle, in degrees, counterclockwise. A decimal.

Mirroring is performed *before* the rotation.

The LR command was introduced in revision 2016.12. .

4.11.5 **Load Scaling (LS)**

The LS command sets the *object scaling*. This command can be used multiple times in a file. The mirroring option remains in effect until modified by another LM command.

The value defines the scale factor applied to objects before they are added to the stream. The object is scaled centered on its *origin* (which may not be its geometric center). Rotation is set as is, it is not cumulative.

The syntax for the LS command is:

<LS command> = LS<Scale>*

Syntax	Comments
LS	LS for Load Scaling
<Scale>	The scale factor is specified by a decimal.

The LM command was introduced in revision 2016.12.

4.11.6 **Examples**

Syntax	Comments
%LPD*%	Sets the object polarity to dark
%LPC*%	Sets the object polarity to clear
%LMX*%	Sets object mirroring to mirroring along the X axis
%LMN*%	Sets object mirroring to no mirroring
%LR45.0*%	Sets object rotation to 45 degrees counterclockwise
%LR-90*%	Sets object rotation to 90 degrees clockwise
%LS0.8*%	Sets object scaling to 80%

Example of a block flashed in different mirroring, orientation, scaling

```
G04 Ucamco copyright*
%FSLAX26Y26*%
%MOMM*%
%ADD10C,1*%
%LPD*%
G04 Define block aperture D12
%ABD12*%
%ADD11C,0.5*%
D10*
G01*
X-2500000Y-1000000D03*
Y1000000D03*
%LPC*%
D11*
X-2500000Y-1000000D03*
%LPD*%
X-500000Y-1000000D02*
X2500000D01*
G75*
G03*
X500000Y1000000I-2000000J0D01*
G74*
G01*
%AB*%
G04 Flash block aperture D12 in four different orientation
D12*
X0Y0D03*
%LMX*%
X10000000D03*
%LMY*%
%LR30.0*%
X0Y8000000D03*
%LMXY*%
%LR45.0*%
%LS0.8*%
X10000000D03*
M02*
```
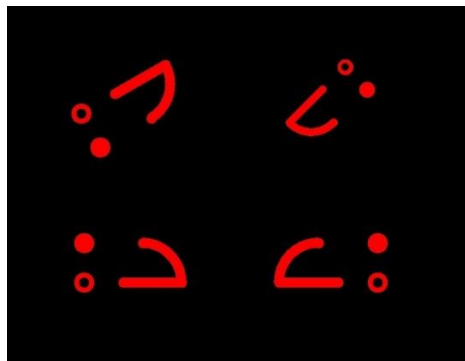

31. Block flashed in different orientations

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

4.12 Region Statement (G36/G37)

4.12.1 Region Overview

A region is a graphics object defined by its contour(s) - see 4.12.2.

The G36 command begins a region statement and G37 ends it. In a region statement the D01 and D02 commands create the contour segments. The first D01 encountered in a region statement 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 a G37 command is encountered the region statement ends and a set of region graphics objects is added to the object stream 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 finishes the region statement. It finishes the last contour in the absence of a finishing D02. Note that a G37 is only allowed when all contours are properly closed.

Using contours with horizontal or vertical fully coincident linear segments (see 4.12.2) it is possible to create holes in a region with cut-ins (see 4.12.5.8).

D01 and D02 are the *only* D code commands allowed in a region statement; in other words D03 and Dnn (nn≥10) are *not* allowed. Extended commands 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.

⚠️ **Warning:** A*perture* attributes can be attached to a region. See **Attributes on regions.**

⚠️ **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 0 and 4.6.14 for examples on how *not* to use cut-ins.

ⓘ **Note:** In the 1960's and 1970s, the era of vector plotters, when the region statement was not available, the only way to produce copper pours was by stroking it with a large 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 must be converted to proper regions laboriously. A file with stroked areas and/or stroked pads is not suited for PCB production.

4.12.2 Valid Contours

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 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 coincide with the start point of the first segment. A

contour thus defines a closed curve in the 2D plane. Zero-length linear and circular segments are allowed but have no effect. Readers can remove them before processing the contour. Note that full arcs are not zero-length! (Avoid zero-length segments as they are useless and can only cause confusion.)

There are two classes of valid contours.

S*imple contours* are the class of basic valid contours. A contour is said to be *simple* if all its segments are disjoint, except for consecutive segments sharing their connection points only; a simple contour does not self-intersect or self-touch. It is obvious which part lies inside. Note that if one travels along the contour in the direction of the segments the inside is always to the same side – always to the left or always to the right; depending on the orientation of the contour. This orientation is not significant. The inside of the contour constitutes the region object. Note that a simple contour can only define a simple region, without holes.

Simple c*ut-in contours* are the second class of valid contours. It allows to define a region with holes. A cut-in connects the contour defining the hole to its enclosing contour, thus joining them into a single contour. If one travels along the contour in the direction of the segments and the inside must always to the same side, just as for simple contours. See the illustration below; see 4.12.5.8 for a fully worked out example.
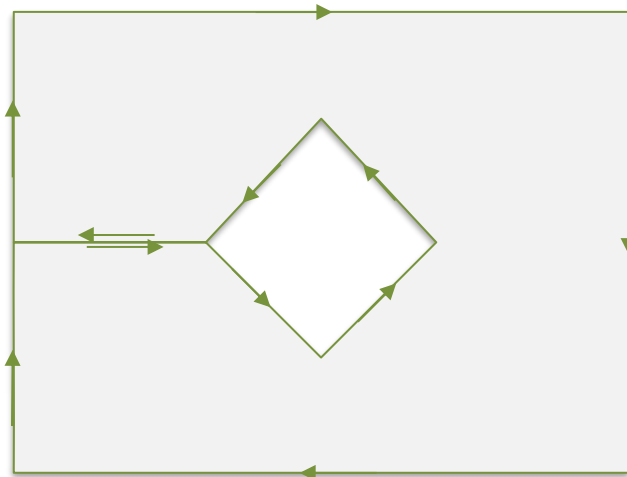
32. A contour with a cut-in

A cut-in is subject to strict requirements: it must consist of two fully coincident linear segments; a pair of linear segments are said to be fully coincident if the segments coincide completely, with the second segment starting where the first one ends; cut-ins must be either horizontal or vertical; all cut-ins in a contour must have the same direction; cut-ins cannot intersect the contour in any other location than their start and end points.

All contours except simple contours and simple cut-in contours are called *self-intersecting* and are *not allowed*. Segments *cannot* cross, overlap or touch except

- ❑ connected segments in their end points.
- ❑ cut-ins in their end points.

Any other form of self-touching or self-intersection is *not allowed*. For the avoidance of doubt, not allowed are, amongst others: 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, points where more than two segments end. If a contour violates any of these restrictions, it is invalid. An invalid contour does not represent an image; its interpretation is unpredictable.

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

This paragraph is for the mathematically inclined. A contour is said the be *weakly simple* if there exists an arbitrarily small perturbation of the vertices changing it in a simple contour. Simple contours with cut-ins are weakly simple. The winding number for valid Gerber contours is for the outside 0 and for the inside everywhere either +1 or -1, depending on the orientation. However, not all weakly simple contours or contours with these winding numbers are valid.

Contours are also used to define outline primitives in macro apertures (see 4.5.4.5).

Processing Gerber files is inevitably subject to rounding errors. Contours must be constructed robustly so that allowed perturbations due to this rounding do not turn an otherwise valid contour in a self-intersecting one. See 4.18.2.

⚠️ **Warning:** Use maximum resolution. Low file coordinate resolution brings uncontrolled rounding and often results in self-intersecting contours, see 4.1.

4.12.3 **G36 Command**

The G36 commands begins a region statement. The syntax is:

<G36 command> = G36*

Syntax	Comments
G36	Begins a region statement.

Example:

```
G36*
```

4.12.4 **G37 Command**

Ends a region statement. The syntax is:

<G37 command> = G37*

Syntax	Comments
G37	Ends a region statement.
	This creates the set of region graphics object by filling the contours created since the previous G36 command.

Example:

```
G37*
```

4.12.5 **Examples**

4.12.5.1 A Simple Contour

Syntax	Comments
`G36*`	Begins a region statement
`X200Y300000D02*`	Set the current point to (2, 3)
`G01*`	Set linear interpolation mode
`X700000D01*`	Create linear segment to (7, 3)
`Y100000D01*`	Create linear segment to (7, 1)
`X1100000Y500000D01*`	Create linear segment to (11, 5)
`X700000Y900000D01*`	Create linear segment to (7, 9)
`Y700000D01*`	Create linear segment to (7, 7)
`X200000D01*`	Create linear segment to (2, 7)
`Y300000D01*`	Create linear segment to (2, 3)
`G37*`	Create the region by filling the contour
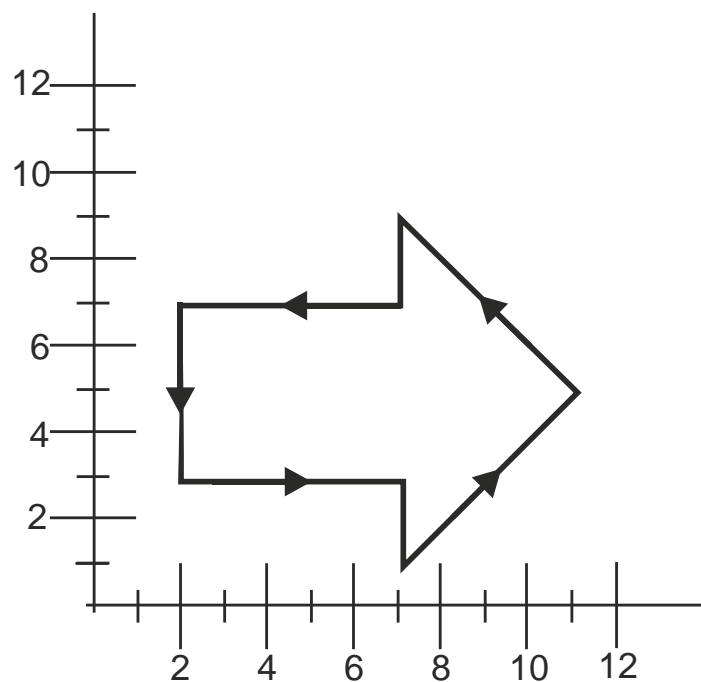
33. Simple contour example: the segments
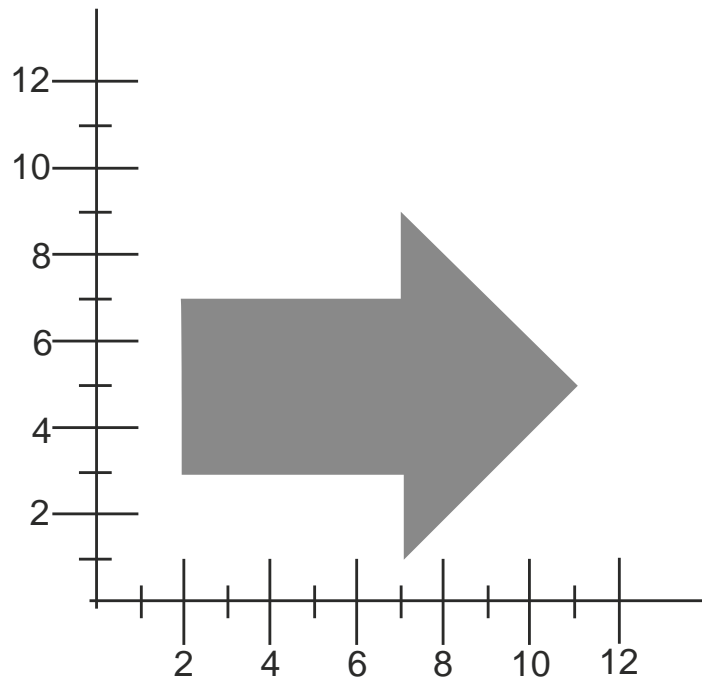
34. Simple contour example: resulting image

4.12.5.2   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.

Copyright Ucamco NV

109

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

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*

4.12.5.3 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:
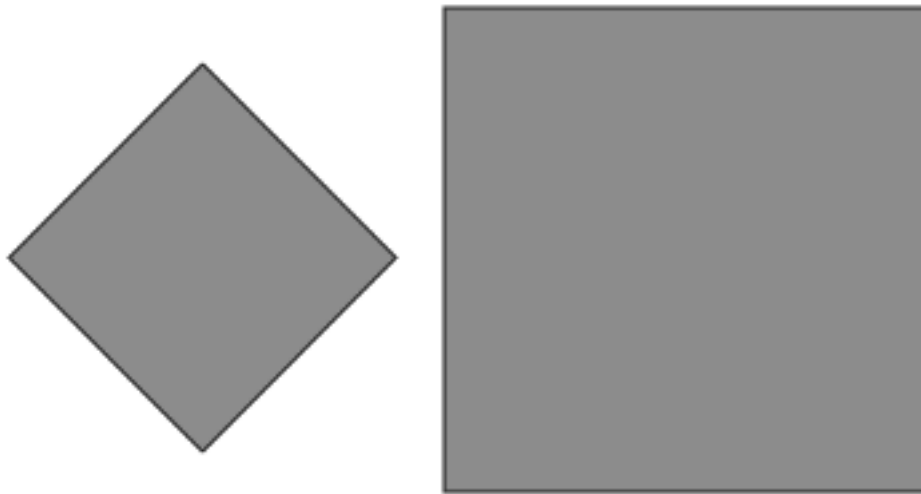
35. Use of D02 to start an new non-overlapping contour

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

4.12.5.4 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:
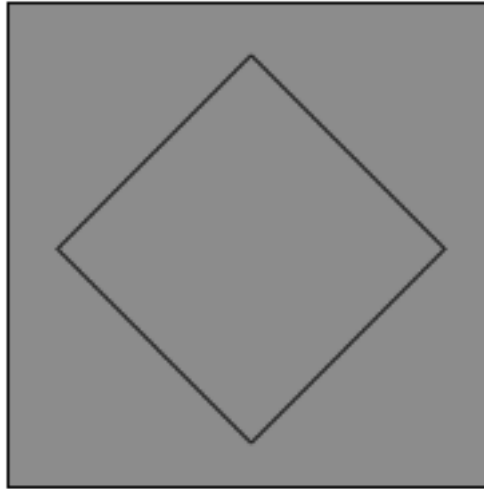
36. Use of D02 to start an new overlapping contour

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.

4.12.5.5 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:

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

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

4.12.5.6 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:

Having a question or remark about the spec? Please contact us at gerber@ucamco.com
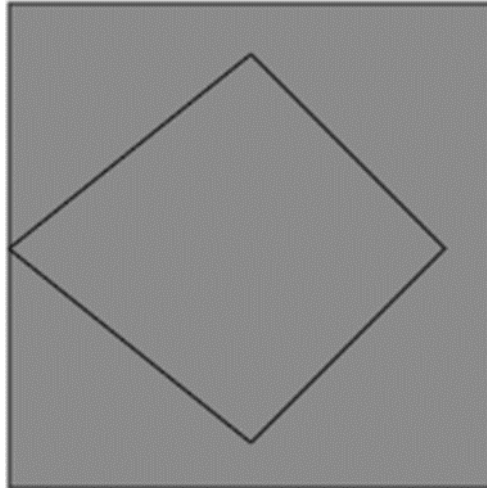
38. Use of D02 to start an new overlapping and touching contour

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

4.12.5.7 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*%
G01*
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.

39. Resulting image: first object only
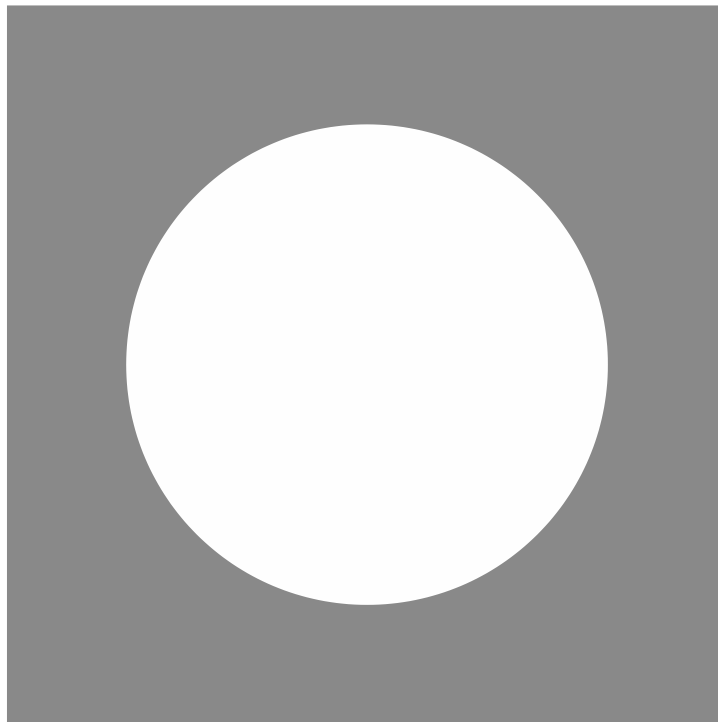
40. Resulting image: first and second objects
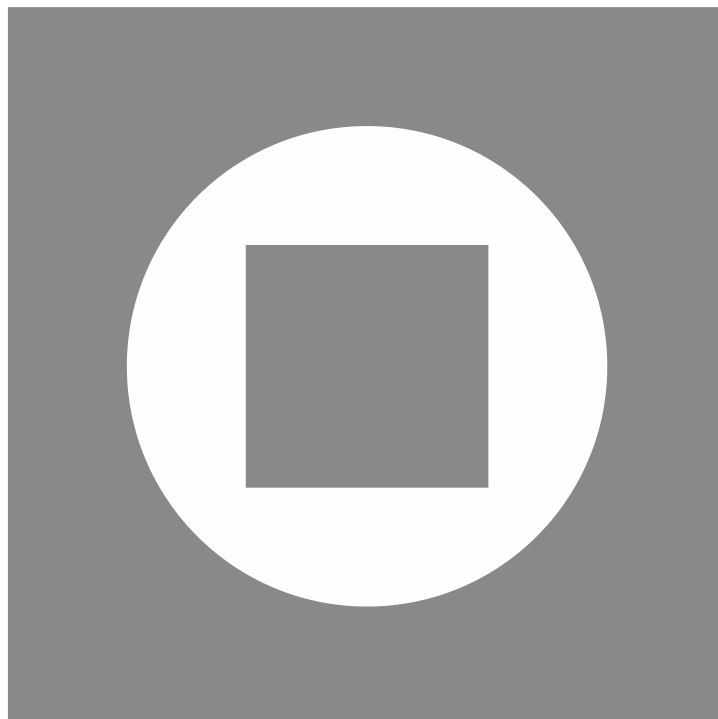
41. Resulting image: first, second and third objects

42. Resulting image: all four objects

4.12.5.8 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 4.12.2.

Syntax	Comments
`%FSLAX26Y26*%`	Format statement
...	Set multi quadrant mode
`G75*`	Begins a region statement
`G36*`	Set the current point to (2,10)
`X20000Y10000000D02*`	Set linear interpolation mode
`G01*`	Create linear contour segment to (12,10)
`X12000000D01*`	Create linear contour segment to (12, 2)
`Y20000000D01*`	Create linear contour segment to (2, 2)
`X2000000D01*`	Create linear contour segment to (2, 6)
`Y6000000D01*`	Create linear contour segment to (5, 6),1st fully coincident segment
`X5000000D01*`	Set counterclockwise circular interpolation mode
`G03*`	Create counterclockwise circle with radius 3
`X50000Y60000I30000J0D01*`	Set linear interpolation mode
`G01*`	Create linear contour segment to (2, 6), 2nd fully coincident segment
`X20000D01*`	Create linear contour segment to (2, 10)
`Y100000D01*`	Create the region by filling the contour
`G37*`	
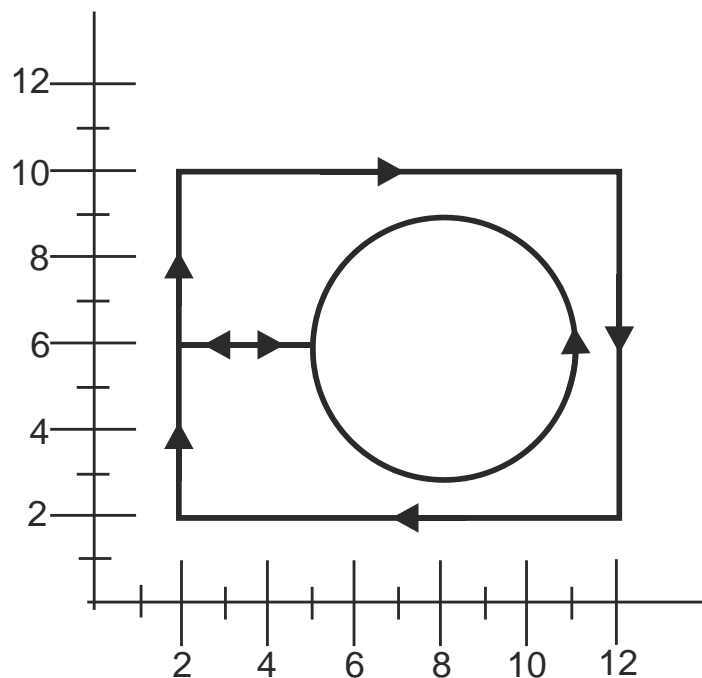
43. Simple cut-in: the segments

Copyright Ucamco NV

118

Having a question or remark about the spec? Please contact us at gerber@ucamco.com
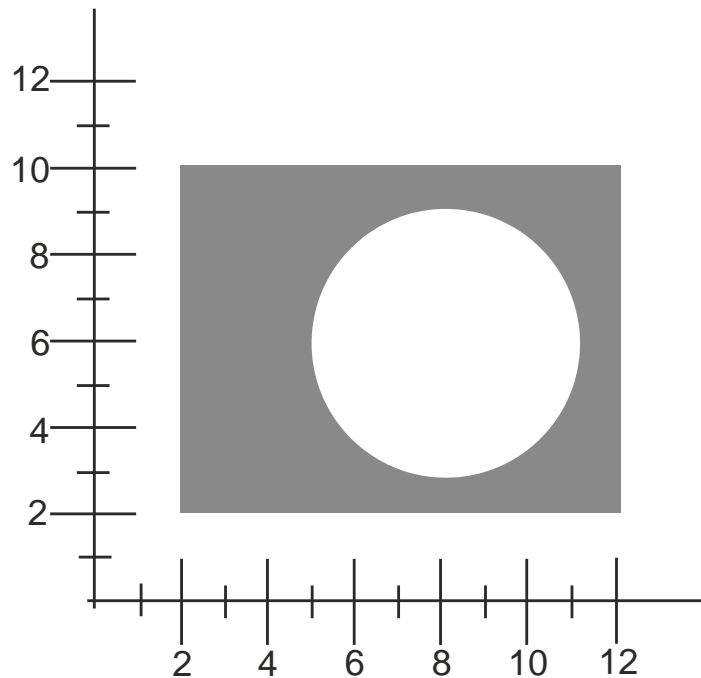
44. Simple cut-in: the image

Note the orientation of the inner circle. If the orientation would be different the contour would be self-intersecting. This becomes immediately apparent if you try to perturb the contour to convert it to a simple contour.

4.12.5.9 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.
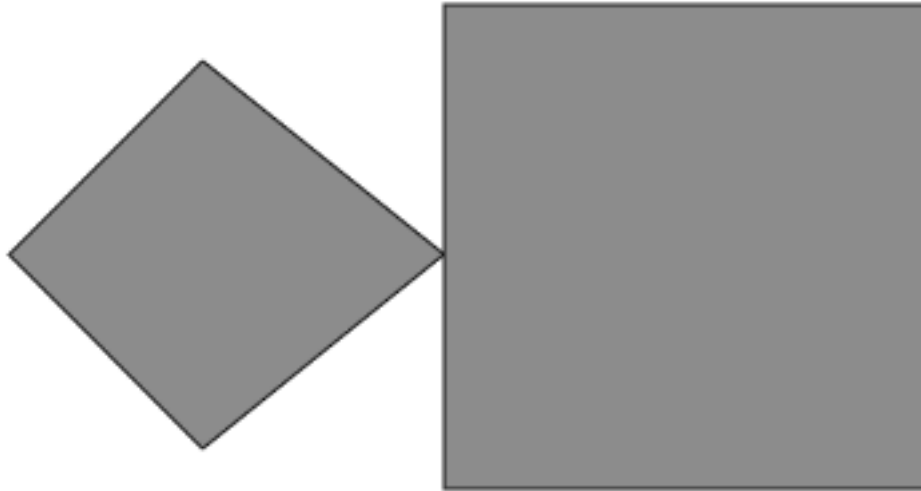
Example:

```
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:
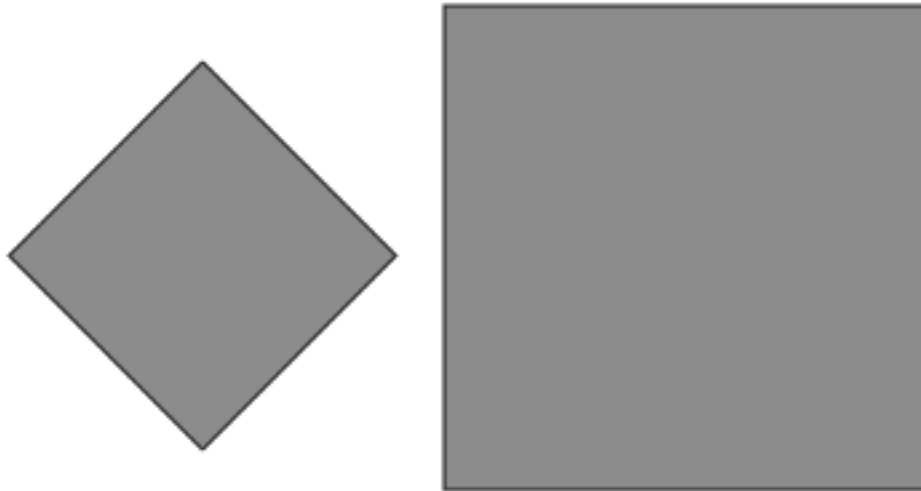
45. Fully coincident segments in contours: two regions

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:
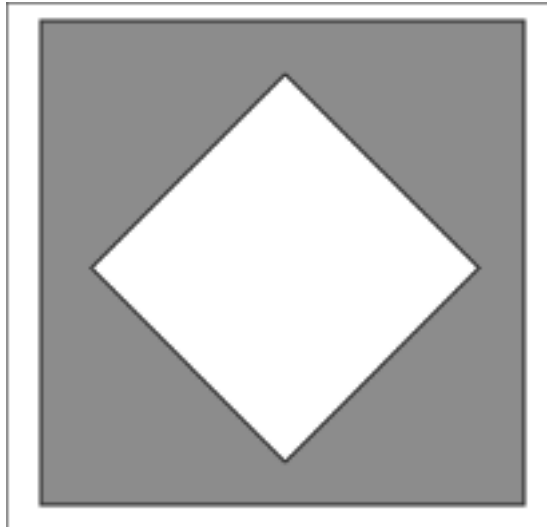
46. Fully coincident segments in contours: region with hole

4.12.5.10 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:

X1310000D01*

X1250000D01*

Y2670000D01*   Y2700000D01*

X1290000D01*

Y2670000D01*

Y2640000D01*

X1250000D01*

Y2700000D01*   Y2570000D01*

Y2600000D01*

Y2640000D01*

X1290000D01*

Y2600000D01*

Y2570000D01*

X1220000D01*

X1250000D01*

X1220000Y2570000D02*
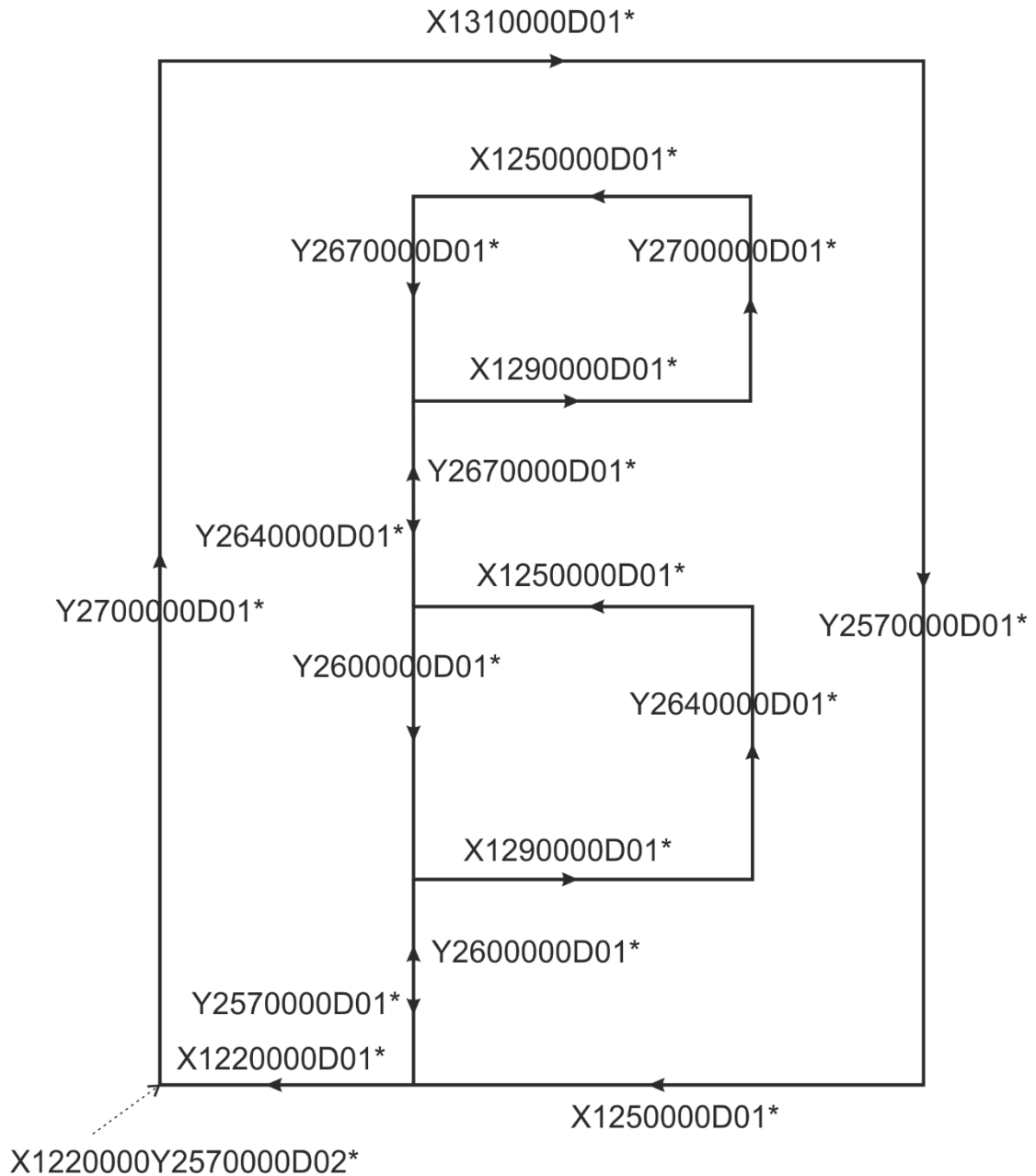
47. Valid cut-in: fully coincident segments

This creates the following image:
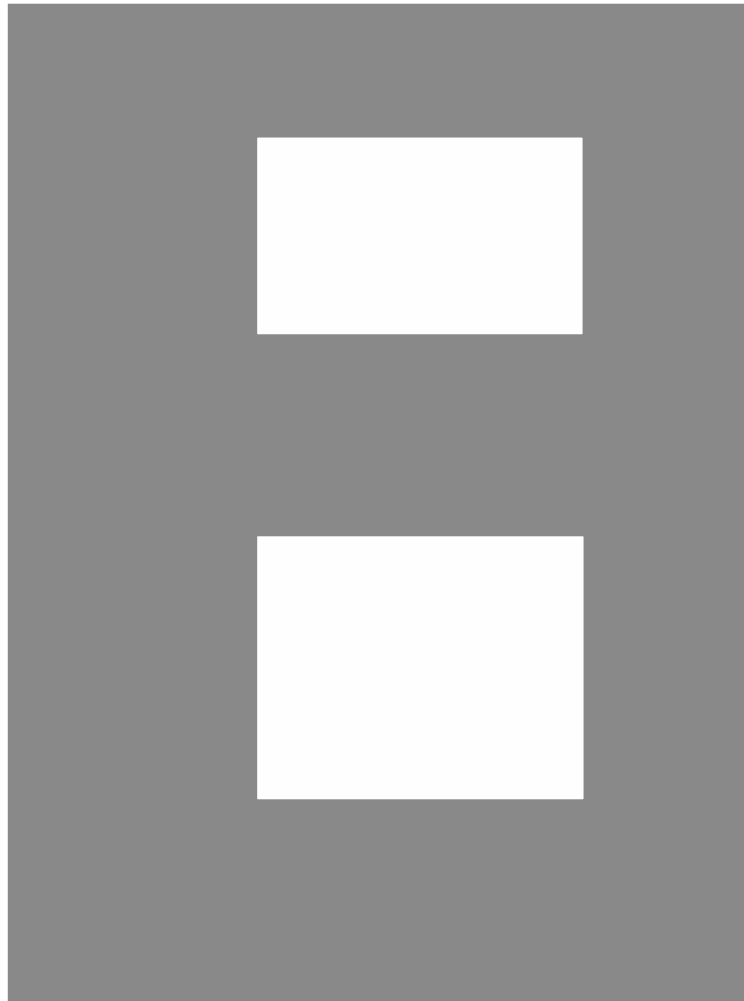
48. Valid cut-in: resulting image

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 4.12.2). 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:

X1170000D01*

X1110000D01*

Y2700000D01*

X1140000D01*

Y2720000D01*   Y2670000D01*   Y2570000D01*

X1110000D01*

Y2570000D01*

Y2640000D01*

X1140000D01*

Y2600000D01*

X1090000D01*

X1110000D01*

X1110000Y2570000D02*
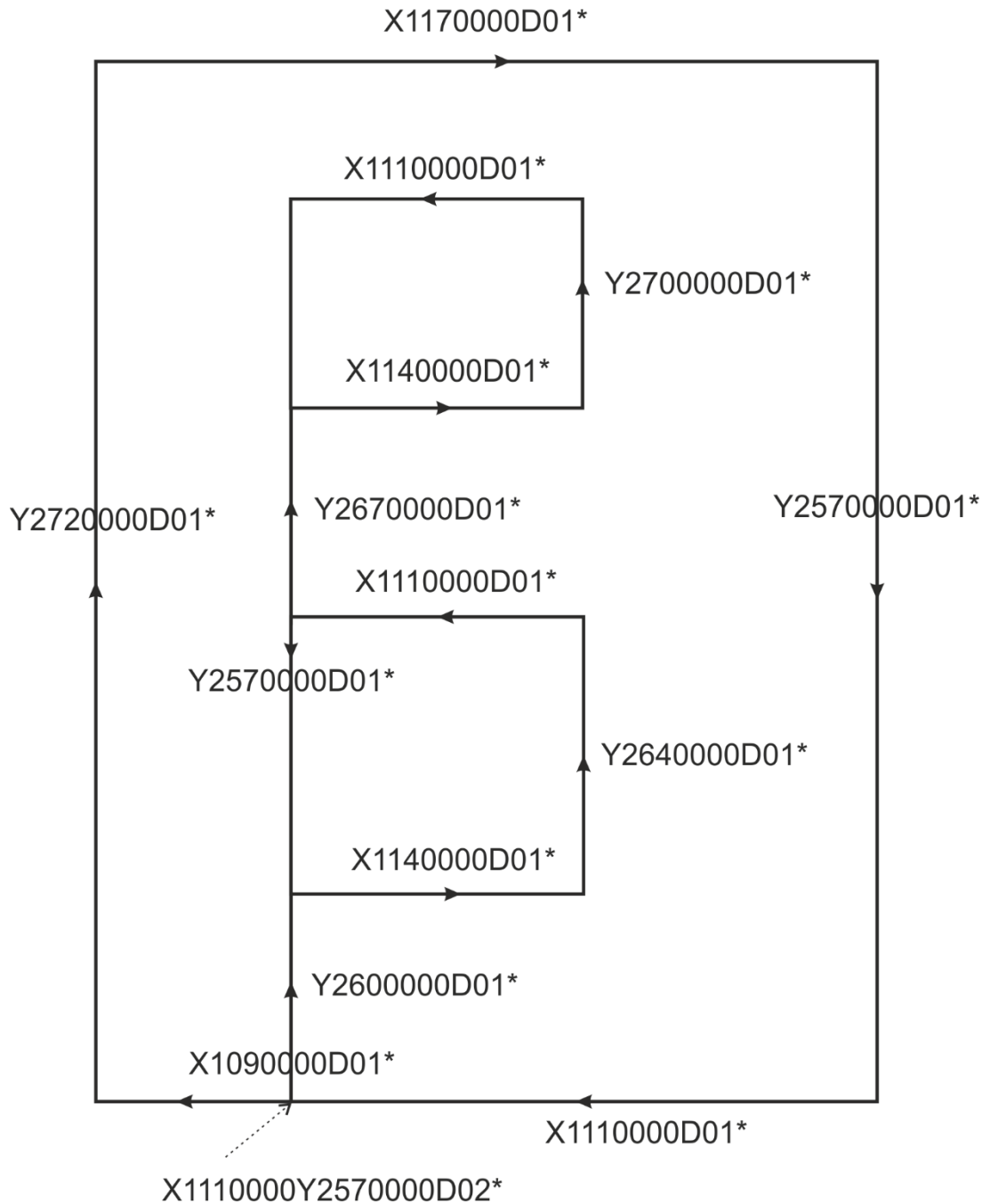
49. Invalid cut-in: overlapping segments

4.12.6 **Power and Ground Planes**

The simplest way to construct power and ground planes is first to 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*
%TA.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. With negative anti-pads this information is transferred directly to CAM in a simple way.

Clearances in power and ground planes can also be constructed with cut-ins, as below.
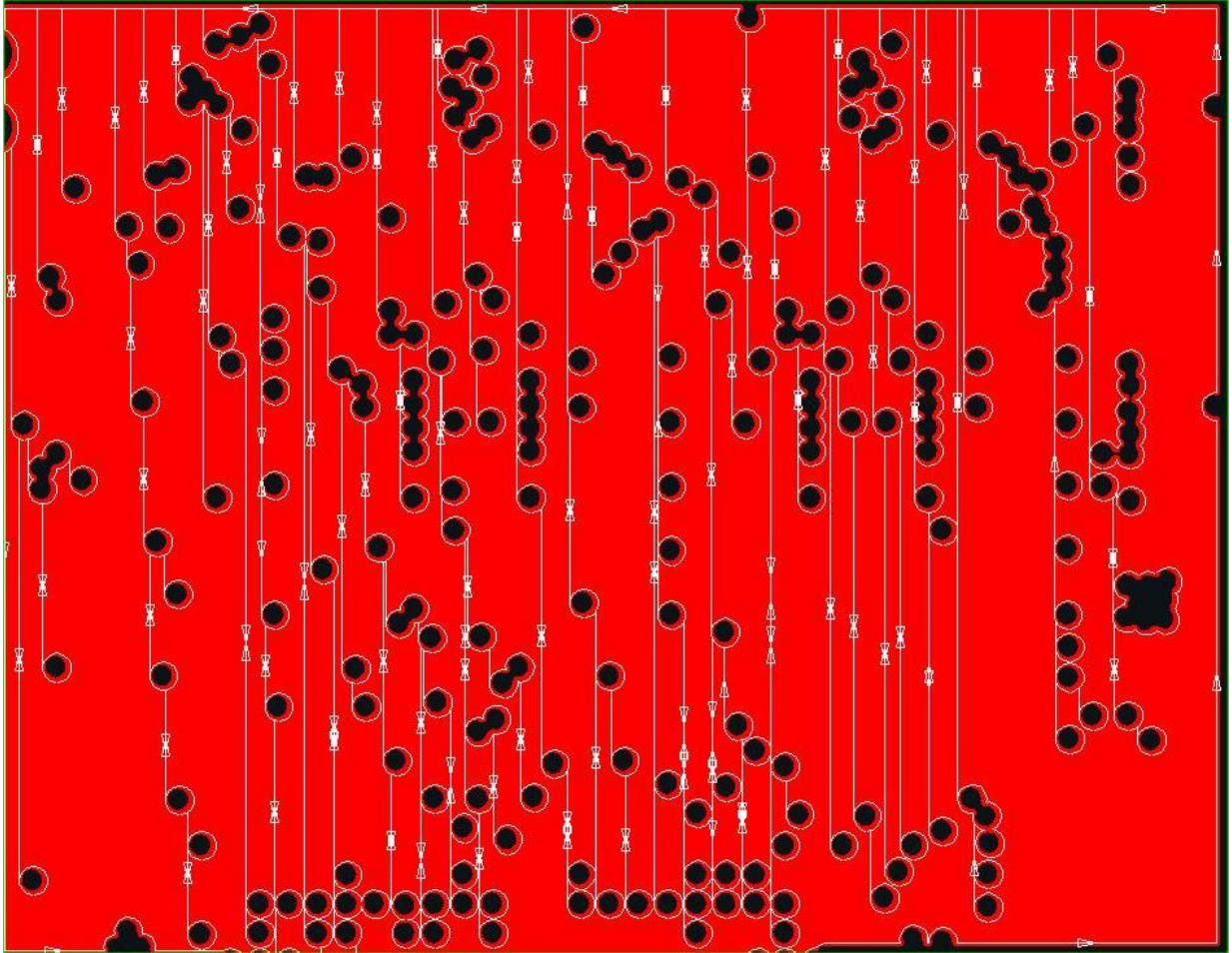
50. Power and ground planes with cut-ins.

The cut-ins are rather complex to create on output; on input in CAM the cut-ins must be removed and the original clearances restored, again rather complex. Use this more complex construction only if there is a good reason not to use the anti-pad method.

Care must be taken to only create valid cut-ins. Sloppy cut-ins are the most frequent cause of scrap due to faulty Gerber files, causing a self-intersecting contour and a missing clearance. Below is an example of such sloppy cut-in; it is a real-life example that lead to expensive scrap. Watch out for rounding errors. Make sure that coincident points indeed are coincident in the file. Using the highest resolution on outputs reduces rounding errors.

X-4.013790 Y-2.667940, X-4.013788 Y-2.667940

X-4.013790 Y-3.166338, X-4.013788 Y-3.166338
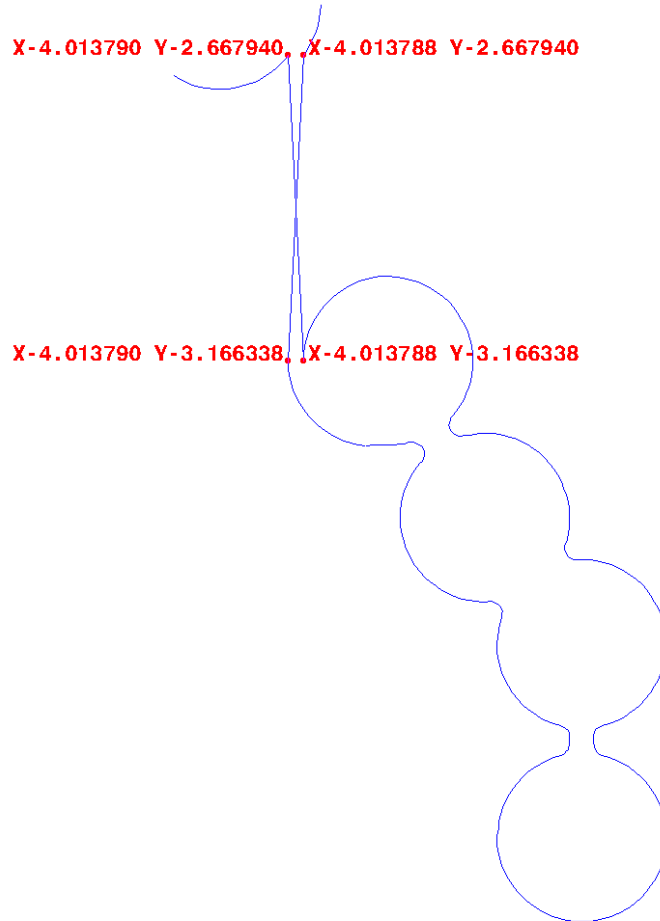
51. Power plane with invalid with cut-in.

It is sometimes recommended to avoid the cut-ins altogether by splitting the plane in separate pieces, where each piece has no holes. Do not follow this bad advice. The remedy is worse than the disease. Splitting the single contour in separate contours without holes is as complex as adding cut-ins. All clearance boundaries must be cut in pieces and spit over different contours; not much of an improvement over finding cut-in points. Rounding errors still lurk, and can lead to pieces that are no longer connected; not much of an improvement over invalid cut-ins. The situation is far worse on input. If the plane consists of a single contour it is clear it is a single plane. When planes are split in pieces the coherence is lost. The file reader must figure out from a bewildering set of contours that a single plane is intended. It must recover clearances which boundaries are scattered over different contours. Cutting a plane in pieces to avoid clearances is bad practice. It is asking for problems. See also 4.18.

4.13 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*
```
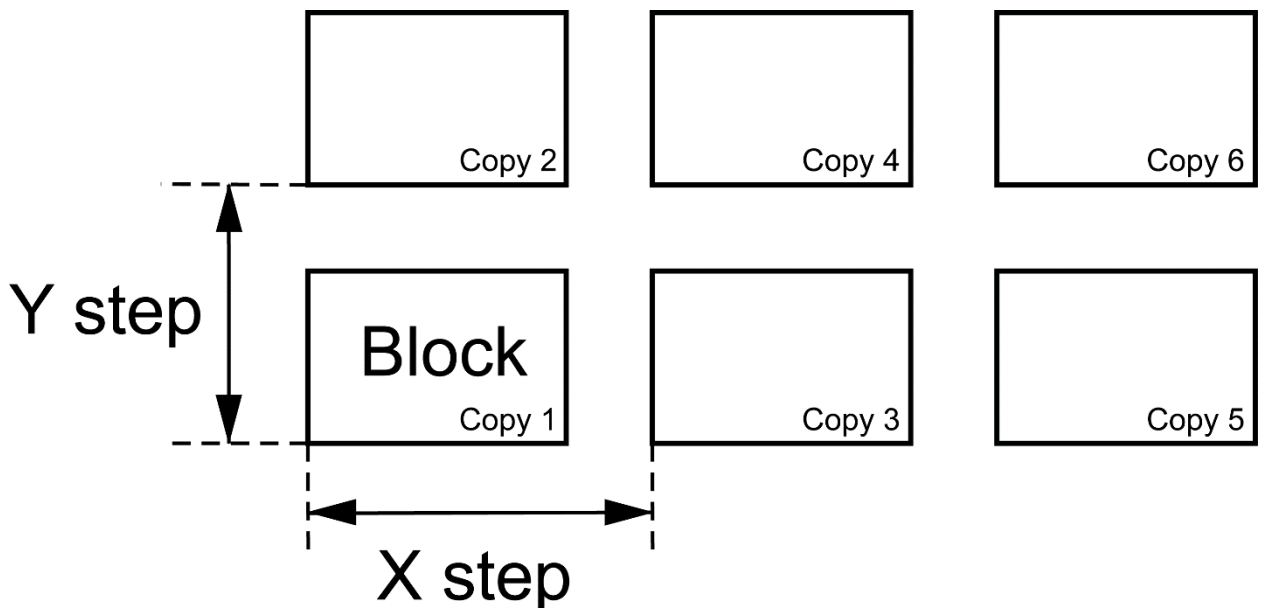

52. Blocks replication with SR command

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. After closing a block the current point is at its position after the last command in the block definition.

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; it is then recommended to set the step value to 0 for this axis.

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

A step & repeat block can contain different polarities (LPD and LPC – see **Error! Reference s ource not found.**).

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 using overlapping blocks is safe.)

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

ℹ **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.

Copyright Ucamco NV 129

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

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

4.14 Syntax of SR and AB nesting

We first informally define three variables:

<single command> = all commands except the block commands SR and AB

<sr parameters> = X<Repeats>Y<Repeats>I<Step>J<Step>

<D-code> = D<integer ≥ 10>

Now we are ready to specify the block commands:

<AB open> = %AB<D-code>*%

<AB close> = %AB*%

<AB statement> = <AB open><section><AB close>

<SR set> = %SR<sr parameters>*%

<SR close> = %SR*%

<SR statement> = <SR set><section>{<SR set><section>}<SR close>

<block statement> = <AB statement>|<SR statement>

<section> = {<single command>}|{<single command>}<block statement>{<single command>}

This syntax defines how block statements can and cannot be nested.

The graphics state is not affected by the SR or AB commands in itself; it is not so that the graphics state is pushed to and popped from a stack by these commands; there is one instance of the graphics state. Of course, a block can contain commands changing the graphics state; the effect of these commands persists after the block is closed.

Note that the M02 command must be the last command in a file. Consequently, an M02 command cannot be issued within such a statement.

Copyright Ucamco NV

131

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

4.15 Comment (G04)

The G04 command is used for human readable comments. It does not affect the image. Gerber readers must ignore the command when generating 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.

Content starting with " #@! " is reserved for *standardized* comments. They can only be used as defined in the specification. Gerber readers must of course also ignore such comments when generating the image

Example:

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

4.16 End-of-file (M02)

The M02 command indicates the end of the file.

The syntax for M02 is as follows:

<M02 command> = M02*

👁 **Example:**

```
M02*
```

The last command in a Gerber file *must* be the M02 command. No data is allowed after an M02. Note that a block or region statement must be explicitly be closed. Consequently, an M02 command *cannot* be issued within such a statement.

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

Copyright Ucamco NV

133

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

4.17 Text in the Image

Gerber has no native font support – this adds too much complication in relation to the modest text requirements in PCB fabrication data. Text must be represented as pure image data, best as regions (contours), see 4.12. Painting the text area is anathema!

Font definitions often contain splines and Gerber does has linear and circular segments but no splines. The splines must therefore be approximated to either linear or circular segments. Circular is more precise with less objects than linear, but mathematically more complicated.

An issue with representing text as image is that the information is lost that this image is text, and which string it represents. This is easily solved with the attributes .AperFunction (5.6.4.1) and .FlashText (5.6.4.3). (It may be counterintuitive, but these aperture attributes can be associated with regions; see 5.3, attributes on regions.) An example; suppose one needs to add the text 'Intercept' on the bottom copper layer. Here is how it goes:

```
%TA.AperFunction,NonConductor*%   <- Indicates the copper is not a conductor,
                                     typically text and graphics
%TA.Flashtext,Intercept,C,M*%     <- Indicates the copper represents the string
                                     'Intercept', as characters and mirrored
G36*

…                                  <- Draws creating the contours
G37*
%TD.AperFunction*%                <- Deletes the attribute
%TD.FlashText*%                   <- Deletes the attribute
```

4.18 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 4.10.1.). 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.

4.18.1 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.

4.18.2 **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] and that coincident points remain coincident. The writer can assume that the perturbation is within this limit. If higher accuracy than 1 µm is required, it must be checked that the applications downstream can handle this. Higher accuracy cannot be blindly assumed.

The perturbation has some noteworthy consequences:

- Contours that are not self-intersecting by a margin of ≤1µm can become self-intersecting under a valid perturbation. Such contours are therefore invalid; see section 4.12.2. Contours must be constructed robustly so that allowed processing perturbations do not turn an otherwise valid contour in a self-intersecting one. See 4.18.2. Consequently, points and segments that are not coincident must be separated by at least 1µm. Furthermore, circular segments add their own intrinsic fuzziness. A circular segment can be validly interpreted by a range of curves, see 4.10.1. If any valid interpretation of the arc violates the requirement of 1µm separation the contour is invalid and the result is unpredictable. Construct contours defensively. Observe sufficient clearances. Marginal contours can and do lead to problems

- 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.

- Avoid objects much smaller than 1 µm. They do not add precision but can cause problems.

Construct files robustly.

5 Attributes

5.1 Attributes Overview

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

❑ The function of the file in the layer structure. Is the file the top solder mask, the bottom copper layer, …?

❑ The function of a pad. Is the pad is an SMD pad, or a via pad, or a fiducial, ...

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

Attributes do *not* affect the image. A Gerber reader will generate the correct image even if it simply ignores the attributes.

Each attribute consists of an *attribute name* and an optional *attribute value:*

<Attribute> = <AttributeName>[,<AttributeValue>]*

Attribute names follow the name syntax in section 3.6.5.

The attribute value consists of one or more comma-separated fields, see section 3.6.7.

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

Attributes are defined with the commands TF, TA, TO or TD. (The attribute commands start with a T as the more obvious A is already taken by the aperture commands.)

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

❑ File attributes attaching metadata to the file as a whole.

❑ Aperture attributes attaching metadata to an aperture or a region.

❑ Object attributes attaching metadata to graphics objects

There are two types of attributes by the *scope* of their use:

❑ *Standard attributes*. Standard attribute names, values and semantics are defined in this specification and are part of it. As they are standardized they can exchange meta-information between all applications.

❑ *User attributes*. User attributes can be chosen freely by users to extend the format with meta-information for proprietary workflows. The users must agree on the names, values and semantics. Use these attributes only for unequivocally defined machine-readable information, use G04 for mere human-readable comments.

In accordance with the general rule in 3.6.5 standard attribute names *must* begin with a dot '.' while user attribute names *cannot* begin with a dot. 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 5.6.

👁 **Example of a user attributes:**

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

During the processing of a Gerber file an *attribute dictionary* is maintained. Dictionary entries consist of the attribute name, its domain and its value. The attribute name is the key of the entry; it must consequently be unique, even of attributes attached to different types of objects.

The current aperture dictionary is defined after each command by the following rules:

- ❑ Initially the attribute dictionary is empty
- ❑ File attributes are added with the TF command
- ❑ Aperture attributes are added or updated with the TA command
- ❑ Object attributes are added or updated with the TO command
- ❑ Attributes are deleted with the TD command

When an aperture or a graphics object is created all attributes with the proper domain present in the dictionary at the time of creation are attached to it. They remain fixed and cannot be changed.

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*%
```

5.2 File Attributes (TF)

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.

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 (TA)

An *aperture attribute* is attached to an aperture or a region. They typically provide information about the graphics objects that will be created with the aperture; for example, a via attribute on an aperture means that all pads flashed with it are via pads. Providing information about graphics objects via their apertures is elegant, compact and efficient.

When an AD command creates an aperture all aperture attributes in the attribute dictionary at that moment are attached to it. Once an aperture is defined its attributes are fixed and cannot be changed.

Attributes on regions. Counterintuitively, aperture attributes *can* be attached to regions. When a G36/G37 creates a region all aperture attributes in the attribute dictionary at that moment are attached to it. The reason for this specification is that aperture attributes on regions are necessary: for example, the function 'conductor' on an aperture tells that all draws with that apertures are conductive tracks; but the function of a region can also be 'conductor'; so we either need a double set of attributes with same meaning, one for regions and one for apertures, or re-use the aperture attributes on regions; clearly the latter is cleaner. A way to view this is that the G36 command defines a virtual region aperture and attaches attributes to it in the same way as an AD does for a real attribute; the only difference is that the AD defines an aperture while a G36 defines a virtual aperture and the single object linked with it.

The TA only command adds an aperture attribute into the attributes dictionary. It has the same syntax:

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

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

The attribute name must follow the naming syntax in section 3.6.5. It cannot be used for any other attribute. The fields composing the attribute value follow the field syntax in section 3.6.7.

The value of an aperture attribute can be modified by a new TA command with the same attribute name.

The example below defines several attributes.

👁 **Example:**
```
%TA.AperFunction,ComponentPad*%
%TAMyApertureAttributeWithValue,value*%
%TAMyApertureAttributeWithoutValue*%
```

Copyright Ucamco NV   138

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

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

👁 **Example:**
```
%TA.AperFunction,ComponentPad*%
%TA.AperFunction,ViaPad*%
```

5.4 Object Attributes (TO)

An *object attribute* is attached to graphics objects. When a D01, D03 or G36/G37 creates an object all object attributes present in the attribute dictionary at that moment are attached to it. They remain fixed and cannot be changed.

The TO command adds an aperture attribute into the attributes dictionary. It has the same syntax as the TF command:

> **<TO command> = %TO<AttributeName>[,<AttributeValue>]*$**

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

The attribute name must follow the naming syntax in section 3.6.5. The name is unique and cannot be used for any other attribute. The fields composing the attribute value must follow the field syntax in section 3.6.7.

The value of an object attribute can be modified by a new TO command with the same attribute name.

👁 **Example:**
```
%TOMyObjectAttribute,Niceobject*%
```

5.5 Delete Attribute (TD)

The TD command deletes an attribute from the 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.

5.6 Standard Attributes

5.6.1 Overview

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, and not just render the image. For example, the fabricator needs to know what are the via pads to handle the solder mask properly. The standard attributes transfer the design intent from CAD to CAM in an unequivocal and standardized manner. This is sometimes rather grandly called "adding intelligence to the image". Without these attributes the fabricator must reverse engineer the design intent of the features in the file, a time-consuming and error-prone process.

Name	Usage
File Attributes	
`.Part`	Identifies the part the file represents, e.g. a single PCB
`.FileFunction`	Identifies the file's function in the PCB, e.g. top copper layer
`.FilePolarity`	Defines whether the file represents the presence or absence of material in the PCB layer, expressed by positive or negative
`.SameCoordinates`	All files in a fabrication data set with this attribute use the same coordinates, in other words align.
`.CreationDate`	Defines the creation time of the file
`.GenerationSoftware`	Identifies the software creating the file.
`.ProjectId`	Defines project and revisions
`.MD5`	Sets the MD5 file signature or checksum
Aperture Attributes	
`.AperFunction`	Function objects created with the apertures, e.g. SMD pad
`.DrillTolerance`	Tolerance of drill holes
`.FlashText`	The meaning of a flash representing text
Object Attributes	
`.C`	The component reference designator linked to an object, e.g C2
`.N`	The CAD net name of a conducting object, e.g. Clk13
`.P`	The pin number (or name) and reference descriptor of a component pad on an outer layer, e.g. IC3,7

Table with the Standard Attributes

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 .FileFunction and .FilePolarity.

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

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

⚠ **Warning: Do not invent your own standard attribute names** (names starting with a dot), your own standard attribute values and semantics. This would defeat the purpose of standardization. User attributes cater to specific needs that are not covered by the standard attributes. Feel free to invent any user attribute you wish.

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

5.6.2 .Part

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

.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,Array*%
```

5.6.3 .FileFunction

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

👁 Example:

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

The attribute value consists of a number of fields. The first field always identifies the type of material, e.g. copper, solder mask, etc. The following fields identify its position in the layer structure, e.g if the solder mask is top or bottom. Subsequent fields may carry extra information. The precise field structure depends on the material. The field values are listed in the tables below. Be very careful when implementing this, spelling errors are easily made. Verify your output carefully.

The attribute must be defined in the header.

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

The existence of a file function does not mean that it must be included in each PCB fabrication data sets. Include the files that are required: no more, no less.

All files with the .FileFunction attribute in a fabrication data set must be scaled 1/1 and align perfectly – same offset, no mirroring, seen from the top to bottom.

The drill/route file image represents the *end diameter* of the hole, after plating, aka inner diameter. 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. PTH and NPTH must be split in two separate files, even if have the same span.

ⓘ Note: It may come as a surprise that in CAD to CAM workflows drill information can and is indeed better represented in Gerber than in an NC format. 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 the appropriate NC format, including feeds and speeds and all the information exactly as needed by the driller. To transfer data from CAD to CAM Gerber is more suitable. 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 do not have attributes.

Copyright Ucamco NV   143

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

.FileFunction value	Usage
Data files	
`Copper,L<p>,(Top|Inr|Bot)[,<type>]`	A conductor or copper layer.
	L<p> (p is an integer>0) specifies the physical copper layer number. Numbers are consecutive. The top layer is always L1. (L0 does *not* exist.) The mandatory field (*Top|Inr|Bot*) specifies it as the top, an inner or the bottom layer; this redundant information helps in handling partial data. The specification of the top layer is "`Copper,L1,Top[,label]`", of the bottom layer of an 8 layer job it is `Copper,L8,Bot[,label`
	The <type> field is optional. If present it must take one of the following values: *Plane, Signal, Mixed* or *Hatched*.
`Plated,i,j,(PTH|Blind|Buried)` `[,<label>]`	Plated drill/rout data, span from copper layer i to layer j. The from/to order is not significant. The (PTH|Blind|Buried) field 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, span from copper layer i to layer j. The from/to order is not significant. The (NPTH|Blind|Buried) field is mandatory.
	The optional label is explained in the row above.
`Profile,(P|NP)`	A file containing the board profile (or outline) and only the board profile. Such a file is mandatory in a PCB fabrication data set.
	The mandatory (P|NP) label indicates whether board is edge-plated or not.
`Soldermask,Top|Bot)[,<index>]`	Solder mask or solder resist.
	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.
	Usually the image represents the solder mask *openings;* it then has negative polarity, see 5.6.4.

.FileFunction value	Usage
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.
Paste,(Top|Bot)	
Glue,(Top|Bot)	Glue spots used to fix components to the board prior to soldering.
Carbonmask,(Top|Bot)[,<index>]	See Soldermask for an explanation <index>.
Goldmask,(Top|Bot)[,<index>]	See Soldermask for an explanation <index>.
Heatsinkmask,(Top|Bot)[,<index>]	See Soldermask for an explanation <index>.
Peelablemask,(Top|Bot)[,<index>]	See Soldermask for an explanation <index>.
Silvermask,(Top|Bot)[,<index>]	See Soldermask for an explanation <index>.
Tinmask,(Top|Bot)[,<index>]	See Soldermask for an explanation <index>.
Depthrout,(Top|Bot)	
Vcut[,(Top|Bot)]	Contains the lines that must be v-cut. (V-cutting is also called scoring.) If the optional attachment (Top|Bot) is not present the scoring lines are identical on top and bottom – this is the normal case. In the exceptional case scoring is different on top and bottom two files must be supplied, one with Top and the other with Bot.
Viafill	Containts the via's that must be filled. It is however recommended to specify the filled via's with the optional field in the .AperFunction ViaDrill.
Drawing files	
ArrayDrawing	A drawing of the array (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.
Drillmap	A drawing with the locations of the drilled holes. It often also contains the hole sizes, tolerances and plated/non-plated info.

Copyright Ucamco NV

145

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

.FileFunction value	Usage
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.
Vcutmap	A drawing with v-cut or scoring information.
OtherDrawing,<mandatory field>	Any other drawing than the 4 ones above. The mandatory field informally describes its topic.

Copyright Ucamco NV

146

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

Other files	
`Keep-out,(Top|Bot)`	Not needed in a fabrication data set.
`Pads,Top|Bot)`	A file containing only the pads (SMD, BGA, component, …). Not needed in a fabrication data set.
`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*%
```

Copyright Ucamco NV

147

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

5.6.4 .FilePolarity

The .FilePolarity specifies whether the image represents the *presence or absence* of material.

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*.

The attribute must be defined in the header.

Solder mask images usually represent solder mask *openings* and are therefore *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*%
```

Note: It is recommended always to use positive for copper layers. Copper plane layers are sometimes output in negative for historic reasons, to get around the limitations in vector photoplotters used in the 1970s and 1980s. They no longer offer a single benefit. They have however a problem. A negative Gerber file cannot represent the main copper pour in a plane. Hence it is impossible to define its extent, on other words to define how close the copper gets to the outline of the PCB. But if you insist on outputting negative copper, please make it clear they are negative by setting the .FilePolarity attributes.

5.6.5 .SameCoordinates

All layers in a PCB fabrication data set- a folder or archive - must use the same coordinate system. In other words, they must align or register as the layers is the physical PCB do. For example, the pads, drill holes, pads and solder mask openings of a pad stack must all have the same coordinates.

Layers must not be offset, mirrored of flipped versus one another. This may seem obvious but unfortunately in 1/3 of the data sets layers do not align. The fabricator consequently cannot trust the alignment of the incoming data. This uncertainty interferes with fast and automatic processing; worse, it can lead to scrap with boards where wrong alignment is not obvious, such as very symmetric of HF boards.

The .SameCoordinates attribute allows CAD to inform the fabricator that the alignment is correct and instruct him to use the alignment in the data.

The attribute must be defined in the header. The syntax is as follows:

```
%TF.SameCoordinates[,<ident>]*%
```

Example – without ident:

```
%TF.SameCoordinates%
```

If in a PCB fabrication data set this attribute is present in a number of Gerber files then they are in alignment with each other.

Example – with a GUID as ident:

```
%TF.SameCoordinates,f81d4fae-7dec-11d0-a765-00a0c91e6bf6%
```

The ident is optional. Its purpose is the following. There may be situations where files in the fabrication data are output at different times, with different coordinate systems. In that situation simply outputting the attribute would wrongly signal the layers do align. Adding the ident avoids this error by distinguish between these different coordinate systems at output time. If the attribute with the same ident is present in a number of Gerber files then they align. If the ident is different they may not align – which would be an error because all files must align. For the fabricator the only important fact is whether the ident, if present, are identical or not; otherwise the ident has no meaning. A very safe ident is a GUID. When all files are output at the same time, with the same coordinates, the ident is not needed.

Note that anyhow all data files must align, attribute or no attribute, ident or not.

5.6.6 .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 5.6.8

Example:

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


5.6.7 .GenerationSoftware

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

The attribute – if present - must be defined in the header. The syntax is as follows:

```
%TF.GenerationSoftware,<vendor>,<application>[,<version>]*%
```

👁 **Example:**

```
%TF.GenerationSoftware,Ucamco,UcamX,2017.04*%
```

5.6.8 .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 syntax is as follows:

```
%TF.ProjectId,<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 string representation of a UUID conforming to RFC4122 version 1 or version 4.

👁 **Examples:**

```
%TF.ProjectId,My PCB,f81d4fae-7dec-11d0-a765-00a0c91e6bf6,2*%


%TF.ProjectId,project#8,68753a44-4D6F-1226-9C60-0050E4C00067,/main/18*%
```

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

5.6.9 .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 .MD5 checksum is not intended for CAD to CAM data transfer which is probably sufficiently protected by the checksum of the zip, but rather for individual files used within fabrication, with a bewildering collection of legacy systems and protocols, and where file transmission errors sometimes occur.

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
```

Copyright Ucamco NV   151

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

The following Pearl script specifies precisely how the .MD5 is calculated:

```perl
#script expects one parameter (original Gerber X2 file)
use Digest::MD5;

local $_ = shift;
local *IN;
my $content;

local $/;
open(IN, "<$_") or die "Cannot open $_ due to $!";
$content = <IN>;          #read file to the variable
close IN;
$content =~ s/\r|\n//g; #remove all CRLF (end of line) characters
$content =~ s/M02\*//;    #remove M02 from the end of file

#calculate MD5
$md5 = Digest::MD5->new; #init MD5
$md5->add($content);      #send content of the file to MD5 engine

print "Add 2 following lines to the Gerber file, please.\n";
print "%TF.MD5,";
print $md5->hexdigest;    #print correct MD5 hash
print "*%\nM02*\n\n";
```

Copyright Ucamco NV

152

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

5.6.10 .AperFunction

This aperture attribute defines the function or purpose of an aperture, or rather the graphics objects created with it. PCB CAM needs to know this information. If it is not defined by this attribute CAM has to reverse engineer it. Gerber file creators are encouraged to identify the function of all apertures to avoid this error-prone reverse engineering. If this is not possible define it for those where it is possible - partial information is better than no information. The bare minimum is to identify via pads: the PCB can only be fabricated properly if the via pads are identified; without via function attribute the fabricator must reverse engineer this information, with all attending risks.

One function, one aperture. Objects with different functions must have separate apertures, even if they are of the same shape and size. Mixed function attributes are hard to handle in CAM to start with. The function of mixed attributes cannot be identified as an aperture can only carry one function attribute. It is *invalid* to put one of the values only on a mixed function pad in an attempt to convey partial information. For example, if edge connector and SMD pads are created with the same aperture it is invalid 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. If an aperture carries a function attribute all its objects must have that function.

Stroking.(aka painting)

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

Regions. Counterintuitively, regions can carry aperture attributes, see Attributes on regions. Use this to define the function of the regions.
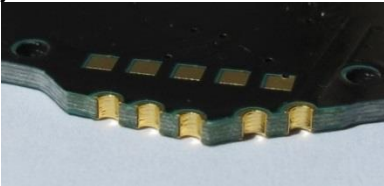
Note that the applicable attribute values depend on the layer - for example, an SMD pad can only be defined on an outer copper layer. The values are defined in the tables below. The header of each table identifies in which layers the values can be used – using them in other layers is invalid.

Drill and rout files.

The values in this table for use only in files with .FileFunction Plated or NonPlated.

AperFunction value	Usage
`ViaDrill[,Filled|NotFilled]`	A via hole. This is reserved for holes whose *sole* function is to connect different layers. Not to be used for holes for component leads.
	An optional field Filled|NotFilled.

`BackDrill`	***5.6.10.1 A hole to remove plating in a hole over a depth by drilling with a slightly larger diameter. See Required Attributes*** A single Ge***rber file defines a single layer image.*** A PCB, however, is not a set of ***independent layers.*** T***o define the complete PCB image one must*** not only define the in***dividual layer images but also*** their position in the layer structure. This is done with the file attributes .FileFunction and .FilePolarity. Considering the complete PCB these attributes are not meta-information but image ***definition data without*** which the ***PCB image is not fully defined. These*** two attributes can be viewed as mandatory in PCB fabrication ***data.*** ***5.6.10.2 The Profile*** The profile defines the physical extent or the ***area*** covered by the PCB. An area is described by its contour (see ***4.12.1) in the Gerber format. A fabrication*** data ***set must contain a separate file with the*** profile, defined ***be a contour (G36/G37). This file*** is identified by the ***file function value*** Profile,***(P|NP).*** Normally a profile will not have holes. Slots are part of the PCB ***and are defined in the drill/rout files. A*** bi***g hole that is not considered part of the PCB*** can be defi***ned by a contour with a cut-in (see*** 4.12.2***).*** S***ometimes one defines the profile layer by a*** line drawn with a sm***all or a zero-size aperture*** instead of the profile***. Such a line does not*** represent an area as ***defined in Gerber date but is*** a drawing***, useful for human reading. While not*** correct, ***it is not difficult to convert it to a*** contour if the line is drawn accurately, obeying the same rules as ***for contour segments (see 4.12.2).*** So if for whatever reason you ***must represent the*** profile by a line ensure that the line is properly constructed. ***Essential is that the profile is clearly*** defined, ***in a separate file, so that it is clear it ís a*** profile, and is not hidden in the copper layers or drawing***s. Corner marks are not good enough as*** they are only suitable for manual processing. Backdrilling

Copyright Ucamco NV

155

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

`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. The label can only be applied on PTH holes. See ComponentPad.
`Slot`	PCB slots.

`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:
	
	• `Tooling`: Tooling holes to attach the board or panel temporarily to test fixtures during assembly and test. Also called mounting holes.
	
	• `BreakOut`: Non-plated holes forming a break-out tab used in break routing.
	
	• `Other`
	Example:
	`.AperFunction,MechanicalDrill,Breakout`
	`.AperFunction,MechanicalDrill`

`CastellatedDrill`	Plated holes cut- through by the board edge; used to join PCBs. Image courtesy Eurocircuits.
`CutOut`	PCB cut-outs. This is the generic term for a hole other than a drill hole.
`Slot`	PCB slots. This is a subset of the cut-outs. Which cut-outs are called slots is subjective. In case of doubt use the value CutOut
`Cavity`	Cavities in a PCB.
`OtherDrill,<mandatory field>`	A hole, but none of the above. The mandatory field informally describes the type.

The values in All layers

can also be used

Copper layers

The values in this table are for use only in files with .FileFunction Copper. Some can be used on outer layers only; this is then mentioned in the usage column.

AperFunction value	Usage
`ComponentPad[,PressFit]`	A pad belonging to the footprint of a *through-hole component*. They are soldered, press-fitted or otherwise electrically connected to the PCB. The purpose of these pads is normally to connect the component circuitry to the PCB but for specific components some pads may not connected to the component inside the package. 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 which have their own dedicated type.
`SMDPad,(CuDef|SMDef)`	A pad belonging to the footprint of an *SMD component.* They are pasted or otherwise electrically connected to the PCB. The purpose of these pads is normally to connect the component circuitry to the PCB but for specific components some pads may not connected to the component inside the package. Excludes BGA pads which have their own type. This function is valid only valid for the normal electrical pads, thermal pads have their own function; see HeatsinkPad.
	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 connectors are *not* SMDs. Surely one does not want solder paste on an edge connector. Use ConnectorPad.

Copyright Ucamco NV

158

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

`BGAPad,(CuDef|SMDef)`	A pad belonging to the footprint of an *BGA component*. They are soldered or otherwise electrically connected to the PCB. The purpose of these pads is normally to connect the component circuitry to the PCB but for specific components some pads may not connected to the component inside the package. The specifier (CuDef|SMDef) is *mandatory*. CuDef stands for *copper defined*, SMDef for *solder mask defined*; see SMDPad. Only applicable for outer layers.
`ConnectorPad`	An *edge connector* pad. Only applicable for outer layers.
`HeatsinkPad`	Heat sink or thermal 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: Component pads often also have a via function; however their main function is component pad and they must have this function; similar for test pads etc.
`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.

Copyright Ucamco NV   159

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

WasherPad	A pad around a non-plated hole without electrical function. 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 poorly by stroking – see **Regions** and **Stroking**.

Copyright Ucamco NV

160

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

EtchedComponent	Etched components are embedded inductors, transformers and capacitors which are etched into the PCB copper. The following illustration shows two etched inductors: For the CAD netlist these are components like others: the net names are different on both sides. (However, for bare-board electrical test they may be conducting copper and connect the net on both sides.)
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.
The values in All layers can also be used	

All layers

The values in this table can be used on all layers, *including* plated, non-plated and copper.

AperFunction value	Usage
`Profile`	Profile of outline. Identifies the draws and arcs that precisely describe the shape of the PCB.

Profiles can but do not have to be present in all PCB layers. However, a dedicated profile file defining the profile must be present.

Note that the profile is viewed as a material layer, as digital data, not as a drawing. It identifies where the extent of the PCB. Sometimes drawing elements are confusingly mixed with the true profile. These must be identified by NonMaterial below. |
| `NonMaterial` | Identify objects that do not represent a physical, material part of the PCB, typically drawing elements mixed in with the digital data represented by the file. For example, in a copper layer the copper pattern is sometimes surrounded by a border containing information as in a technical drawing. This border is not part of the copper pattern, and does not represent copper; it is non-material. It is strongly recommended to put this information in a separate document rather than confusingly mixing it with the true copper pattern. But if you insist on mixing a drawing with a data file use the NonMaterial function to identify drawing elements so that CAM can delete them. |
| `Material` | Identifies the proper part of the data file.

Solder masks are traditionally negative. The image represents the solder mask openings. The apertures take the value 'Material' – they define solder mask material, but in a negative way. Similar for all negative extra layers.

For copper and drill layers this function is split into more detailed functions. The value 'Material' therefore *cannot* be used for copper and drill layers. |
| `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.

The following file creates a circular hole in the solder mask.

```
%FSLAX25Y25*%
%MOIN*%
%TF.FileFunction,Soldermask,Top*%
%TF.Part,Single*%
%TF.FilePolarity,Negative*%
%TA.AperFunction,Material*%
%ADD10C,0.000070*%
%LPD*%
D10*
X123500Y001250D03*
...
M02*
```

Whereas the following files creates a circle consisting of solder mask.

```
%FSLAX25Y25*%
%MOIN*%
%TF.FileFunction,Soldermask,Top*%
%TF.Part,Single*%
%TF.FilePolarity,Positive*%
%TA.AperFunction,Material*%
%ADD10C,0.000070*%
%LPD*%
D10*
X123500Y001250D03*
...
M02*
```

5.6.11 .DrillTolerance

.DrillTolerance defines the plus and minus tolerance of a drill hole end diameter. 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*%
```

5.6.12 .FlashText

Gerber intentionally does not contain fonts or typographic text – this would introduce a complexity out of proportion to its benefits. Any text can be represented with the available graphic constructs, especially with contours. However, such generic graphic constructs do not maintain the information which text string is represented; this is sometimes a disadvantage.

The .FlashText aperture attribute transfers this otherwise lost information. .FlashText is designed for text image created with a flash. If the text is created with draws and arcs rather than a flash, it can always be flashed by collecting them in a block aperture.

Bar codes are handled as text – one can view a barcode as a special font.

Syntax and semantic of the attribute value is as follows:

<Text>,(B|C),[(R|M)],[<Font>],[Size],[<Comment>]

.FlashText fields	Usage
<text>	The text string represented by the aperture image.
(B|C)	Indicates if the text is represented by a *barcode* – B -or by *characters* - C.
(R|M)	Indicates if the text is readable or mirrored. Optional.
<Font>	Font name. Content not standardized. Optional.
<Size>	Font size. Content not standardized. Optional.
<Comments>	Any extra information one wants to add. Optional.

An empty field means that the corresponding meta-data is not specified.

Examples:

```
%TA.FlashText,L1,C,R,Courier,10,Layer number (L1 is top)%
```

Text:	L1
B|C:	Characters,
(R|M):	Readable
Font:	Courier
Size:	10
Comment:	Layer number (L1 is top)

```
%TA.FlashText,XZ12ADF,B,,Code128,,Project identifier *%
```

Text:	XZ12ADF
B|C:	Barcode
(R|M)	Not specified
Font:	Code128
Size:	Not specified
Comment:	Project identifier

Copyright Ucamco NV   165

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

5.6.13 .N (Net)

The .N object attribute attaches a CAD netlist name to any conducting object. The attribute can be attached to objects on any copper layer. It indicates the object is part of the given net. The .N attribute is intended to allow quick visualization of nets and, more importantly, to define the CAD netlist.

Normally an object is fully connected and consequently belongs to a single net. However, if an object consists of different disconnected parts or is split in several disconnected parts by clear (LPC) objects it may belong to different nets. Then the .N attribute value must include all net names involved. It is recommended to avoid creating disconnected objects: one object, one net.

The syntax is:

<.N Attribute> = .N,<netname>{,<netname>}

.N field	Usage
<netname>	The CAD net name. It can take any value conforming to the field syntax.

Example:

```
%TA.aperFunction,Conductor*%
%ADD21C,1*%              Create aperture 21, for conductive tracks
…
D21*                    Select aperture 21
%TO.N,Clk3*%            Select net Clk3
X5600000Y1200000D02*    Move to the start of a track
X5600000Y1202500D01*    Draw the tracks with Clk3 is attached
X5605000Y1205000D01*
X5605000Y1220000D01*
…
```

There are two reserved net names:

1) The empty string, defined by `%TO.N,*%` identifies objects not connected to a net, such as tooling holes, text, logos, pads for component leads not connected to the component circuitry.
2) The name N/C, defined by `%TO.N,N/C*%`, identifies a single pad net, as an alternative to giving each such net a unique name.

Except the reserved names all net names must be unique.

It is recommended to attach a .N attribute to all copper objects, also those without net. The absence of the .N attribute does not mean there is no net; the absence is therefore ambiguous.

Normally an object is fully connected and consequently belongs to a single net. However, if an object consists of different disconnected parts or is split in several disconnected parts by clear (LPC) objects it may belong to different nets. Then the .N attribute value must include all net names involved. It is recommended to avoid creating disconnected objects: one object, one net.

5.6.14 .C (Component)

The .C object attribute attaches the component reference descriptor of a component to an object. It indicates that the object belongs to the given component. The attribute can be attached to objects on any layer. It is intended to identify e.g. which objects on a legend belong to which components.

The syntax is:

<.C Attribute> = .C,<component>

.C field	Usage
`<component>`	The component reference descriptor. It can take any value conforming to the field syntax.

👁 **Example in a legend layer:**

`D21*`	Select aperture 13, used for component symbols
`%TO.C,R2*%`	Select reference descriptor R2
`X5600000Y1200000D02*`	Move to the start of the symbol
`X5600000Y1202500D01*`	Draw the symbol
`X5605000Y1205000D01*`	
`X5605000Y1220000D01*`	
…	

The attribute .C,R2 is attached to all tracks drawing the resistor symbol.

5.6.15 .P (Pin)

The .P object attribute attaches the reference descriptor and pin name of a component pin to a pad.

The syntax is:

<.P Attribute> = .P,<component>,<pin>

.P fields	Usage
<component>	The component reference descriptor. It can take any value conforming to the field syntax.
<pin>	The pin name. It can take any value conforming to the field syntax. It is very often a number.

Example :

```
D13*                     Select aperture for the pads for R5
%TO.P,R5,1*%
X5600000Y1200000D03*     The pad for R5, pin 1
%TO.P,R5,2*%
X5600000Y1202500D03*     The pad for R5, pin 2
```

The .P attribute can be attached to any pad belonging to a component and only those. The pin name must be a non-empty field, with one exception: if the pad is part of the component footprint but not connected to the component circuitry the name may be an empty field, e.g. defined by %TO.P,U3,*%; pads with an empty name are normally not part of a net and therefore also have an empty net name attached defined by %TO.N,*%.

Consequently, it can only be used in copper layers that carry the components. Typically, these are the outer layers and then the .P attribute can only be used in the outer layers. Embedded or etched components can be on inner layers and then these inner layers can also carry the .P attribute.

The .P attribute can be attached to *flashed pads only*. It can therefore not be used on painted pads.

A single component pad can consist of multiple flashes. Each flash then carries the same reference descriptor and pin number.

It is technically possible to create a single object representing multiple pads. For example, a single macro aperture can describe a complete component footprint. This is possible but *not allowed*. (It would be a neat way to make life miserable for CAM engineers though.)

Copyright Ucamco NV   168

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

5.7 Using Attributes in PCB Fabrication Data

5.7.1.1 Required Attributes

A single Gerber file defines a single layer image. A PCB, however, is not a set of independent layers. To define the complete PCB image one must not only define the individual layer images but also their position in the layer structure. This is done with the file attributes .FileFunction and .FilePolarity. Considering the complete PCB these attributes are not meta-information but image definition data without which the PCB image is not fully defined. These two attributes can be viewed as mandatory in PCB fabrication data.

5.7.1.2 The Profile

The profile defines the physical extent or the area covered by the PCB. An area is described by its contour (see 4.12.1) in the Gerber format. A fabrication data set must contain a separate file with the profile, defined be a contour (G36/G37). This file is identified by the file function value Profile,(P|NP).

Normally a profile will not have holes. Slots are part of the PCB and are defined in the drill/rout files. A big hole that is not considered part of the PCB can be defined by a contour with a cut-in (see 4.12.2).

Sometimes one defines the profile layer by a line drawn with a small or a zero-size aperture instead of the profile. Such a line does not represent an area as defined in Gerber date but is a drawing, useful for human reading. While not correct, it is not difficult to convert it to a contour if the line is drawn accurately, obeying the same rules as for contour segments (see 4.12.2). So if for whatever reason you must represent the profile by a line ensure that the line is properly constructed. Essential is that the profile is clearly defined, in a separate file, so that it is clear it ís a profile, and is not hidden in the copper layers or drawings. Corner marks are not good enough as they are only suitable for manual processing.

5.7.1.3 Backdrilling

An example explains best how to structure a job with backdrilling. Suppose we have an 8 layer job with backdrilling to remove via plating between layers 8 and 7. We need two files:

One file contains the via's. It has .FileFunction *Plated,1,8,PTH*. The drill tool has the via end diameter. Its .AperFunction value is *Via*.

The second file contains the backdrills. It has .FileFunction *NonPlated,8,7,Blind.* The drill tool has the same diameter as the via – the manufacturer will determine the tool diameter. Its .AperFunction value is *Backdrill*.

5.7.1.4 The CAD Netlist

The CAD netlist describes which component pins are connected. Pins are identified by their component reference descriptor and pin number or name. Nets are identified by the net name. Here is an example of a CAD netlist; The first line lists all pins of net Clk3.

```
Clk3: U1-4,U2-3,U5-9,U6-9,U7-9
Sig11: U1-3,U5-12
Data8: U2-4,U5-10,U6-10,U7-8,U8-1
GND: U1-1,U2-1,U3-8,U4-16,U5-16,U6-1,U7-8,U8-8
…
```

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

The CAD netlist links component pads to nets. It specifies the function of the PCB and is the basis for the layout.

In Gerber the CAD netlist is defined by attaching both the pin and the net attribute to each component pad. This defines a pad - net entry in the CAD netlist, and at the same time associates a pad location and shape to it. For instance, the first entry in the CAD netlist above is Gerber is given by the following sequence:

```
%TO.P,U1,4*%
%TO.N,Clk3*%
X…Y…D03*      The flash that creates the pad
…
```

If the .P attribute is present then the *complete* CAD netlist must be present, including all edge connectors test points and etched components. In other words, all the end points of the nets must be included, all the pads, *not only* the pads of the physical components that are part of the BOM. Although vias are part of a net, they are not component pads and cannot have a .P attached. Washer pads or any pads that are not part of an component cannot have a .P attached.

5.7.1.4.1 Etched Components

Etched components are embedded inductors, transformers and capacitors which are etched into the PCB copper. The following illustration shows two etched inductors.

They are identified by the .AperFunction attribute value 'EtchedComponent' on to the aperture used to create them. See `EtchedComponent`.

For the CAD netlist these are components like others: the net names are different on both sides. (However, for bare-board electrical test they may be conducting copper and connect the net on both sides.)

Etched components do not need and normally do not have pads. T there is no .P associated with them. The net on each side is different however.

5.7.1.4.2 Benefits of Including the CAD Netlist.

For the assembly process the location and orientation of each component must be known. This can easily be extracted from the Gerber file.

The netlist and component names facilitate the communication between the parties involved in design and fabrication. Viewers show more complete PCB information.

More importantly, the netlist information dramatically increases the security of the design to fabrication data transfer. If the image CAM reads from a Gerber file significantly differs from the image intended by CAD, due to bugs, operator errors or transmission errors, the inevitable result is scrap. Such a difference in image results in a difference in board netlist, which can be detected by comparing the calculated board netlist with the supplied CAD netlist. The CAD netlist therefore provides a very powerful redundancy check against image errors. To be precise, the following assert must be valid:

- Interpret all N/C's as unique names

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

- Flashes with the same reference descriptor and pad are deemed connected
- Etched components are removed before connection is calculated
- Assert 1: pads with the same net name must be connected
- Assert 2: pads with different net names must be isolated.

Lastly, a bare board PCB fabricator is expected to perform an electrical test on the bare PCB and guarantee the PCB conforms to the CAD netlist. It then is logical to provide him with the netlist to test against. Without providing the netlist the fabricator is expected to reverse engineer the netlist and must test against a reverse engineered net – hardly a secure procedure.

5.7.1.4.3 IP Considerations

Net names such as Clk13 provide information about the design. This may be a concern. A solution is to replace the meaningful names with obfuscated names such a sequential number. This still allows to compare the design netlist with the image netlist as a redundancy check – meaningful names are not needed for that. The obfuscated names are a little less convenient when communicating between creator and receiver of the Gerber file, but both can still identify the same net as long as the creator can identify the net corresponding to the obfuscated name he created. Obfuscated names are sometimes a sensible balance between IP protection and data transfer security.

It is sometimes alleged that even a net list with obfuscated names pose an IP security risk as it still shows the connections between the pads. This is an obvious fallacy as the connections between the pads can be worked out from the image. In fact, if this were not true, a fabricator would be unable to perform a bare board electrical test without netlist information.

5.8 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 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.

Copyright Ucamco NV   172

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

👁 **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 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
                                  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*
%TA.AperFunction,Conductor*%      G04 Changes the value of .AperFunction
                                  attribute to define conductors*
G36*                              G04 Start a conductive region. IT TAKES ON
                                  THE ATTRIBUTE VALUES FROM THE DICTIONARY*
….
G37*
%TD*%                             G04 Clear attribute dictionary*
G36*                              G04 Start a region, without attributes*
….
G37*
```

Copyright Ucamco NV   173

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

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 2.6 and 4.10.9.*
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 4.5.4.4.*
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 4.4.1.5.*
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 4.5.2.*
Clearances in planes disappear.	This is often the consequence of invalid cut-ins resulting in self-intersecting contours. The root cause is usually sloppy rounding and low resolution output. *See **Error! Reference source not found.** and 4.12.2.*

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

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* 4.4.1.4.
The result of the MI command is not as expected.	The MI command mirrors coordinate data but not apertures. A number of implementations unfortunately also mirror apertures making MI unsafe to use. It is therefore deprecated. Do not use the MI command but apply the transformation directly in the coordinate data. *See* 7.1.7.
The result of the SF command is not as expected.	The SF command scales coordinate data but no other sizes in the file. A number of implementations unfortunately also scale other elements making SF unsafe to use. It is therefore deprecated. Do not use the SF command but apply the transformation directly in the coordinate data. *See* 7.1.9
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.
Standard Gerber or RS-274-D	Standard Gerber is revoked. It is therefore *no longer valid* Gerber. 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. See 7.8. **Always use Gerber X!**
Sending a PCB layer as several positive/negative files that must be merged together.	This is a method dating that belongs to Standard Gerber, which is now invalid. It is a method dating from the days of the vector photoplotter, and such data is as obsolete as paper tape. There is no more reason for this obnoxious practice: Gerber now allows positive/negative layers in the same file – such data is processed automatically. There is no standard governing how to split the files. It requires manual work. This practice is *invalid* in Gerber X. See 2.1 **One file = one layer**
Error message.	Some files contain %ICAS*%. One wonders what this pseudo-command is supposed to achieve. Anyhow, it is invalid.

Error message; not the intended image.	Invalid format statement %FS<mark>D</mark>….*% The only valid zero omission options in the %FS are L and T. D is invalid. See 7.4.1.
Error message.	*Presence of %FSLA<mark>N2</mark>X26Y26*% The N2 in the format statement is invalid. See 4.1 One wonders what it is supposed to do.
Error message.	…X5555Y5555<mark>I</mark>J001 .. Missing zero after the "I". The number after I must have at least one digit, see 4.1.1.

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 in a PCB fabrication data set. A netlist can be provided in IPC-D-356A file or with Gerber attributes – see 5.7.1.4.**
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 4.1. Do not sacrifice precision to save a few bytes.
Cutting a single plane in several sections through the clearances to avoid regions with holes.	The information what the plane is and where the clearances are is lost. The reader must attempt to reverse engineer that information but may be thwarted by rounding errors. Risk of scrap. See section 4.12.6.	**Construct planes with clearances using dark polarity for the plane and clear polarity for the clearances (anti-pads).**
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 4.10.1	**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.**
Using a non-standard file extension	When you use a non-standard file extension the reader must open the file to know what format it is and which application to use. See 3.2.	**Please use ".gbr" or ".GBR" as file extension for all your Gerber files.**

Poor/good practices

Copyright Ucamco NV

178

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

7 Deprecated Format Elements

7.1 Deprecated Commands

7.1.1 Overview

Current Gerber writers must not use the deprecated commands. Gerber readers may implement them to support legacy applications and files. 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. Sometimes used.
G55	Prepare for flash	This historic code optionally precedes D03 code. It has no effect. Very rarely used nowadays.
G70	Set the 'Unit' to inch	These historic codes perform a function handled by the MO command. See 4.2. Sometimes used.
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 4.1. They are superfluous and deprecated. Very rarely used nowadays.
G91	Set the 'Coordinate format' to 'Incremental notation'	
M00	Program stop	This historic code has the same effect as M02. See 4.15. Very rarely used nowadays.
M01	Optional stop	This historic code has no effect. Very rarely used nowadays.
IP	Sets the 'Image polarity' graphics state parameter	IP can only be used once, at the beginning of the file. Sometimes used, and then usually as %IPPOS*% to confirm the default – a positive image; it then has no effect. *As it is not clear how %IPNEG*% must be handled it is probably a waste of time to try to fully implement it, and sufficient to give a warning if an image is negative.*
AS	Sets the 'Axes correspondence' graphics state parameter	These commands can only be used once, at the beginning of the file. The order of execution is always MI, SF, OF, IR and AS, independent of their order of appearance in the file.
IR	Sets 'Image rotation' graphics state parameter	

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

MI	Sets 'Image mirroring' graphics state parameter	Rarely used nowadays. If used it is nearly always to confirm the default value; they have no effect. *It is probably a waste of time to fully implement these commands; simply ignoring them but issue a warning when used with a non-default value will handle the overwhelming majority of Gerber files correctly.*
OF	Sets 'Image offset' graphics state parameter	
SF	Sets 'Scale factor' graphics state parameter	
IN	Sets the name of the file image. Has no effect on the image. It is no more than a comment.	These commands can only be used once, at the beginning of the file. Use G04 for comments. See 4.15. Sometimes used.
LN	Has no effect on the image. It is no more than a comment	Can be used many times in the file. Use G04 for comments. See 4.15. Sometimes used.

Deprecated Gerber Commands

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

7.1.2 **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.

7.1.2.1 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. This is the default.
AYBX	Assign output device axis A to data axis Y, output device axis B to data axis X.

7.1.2.2 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

7.1.3 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 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.

7.1.3.1 IN Command

The syntax for the IN command is:

<IN command> = IN<Name>*

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

7.1.3.2 Examples

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

7.1.4 **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.

7.1.4.1 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.)

7.1.4.2 Negative Image Polarity

The purpose of negative image polarity 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 encountered *must* have dark polarity.

7.1.4.3 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

7.1.4.4 Examples

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

7.1.5 **Image Rotation (IR)**

The IR command is deprecated.

IR rotates the entire image counterclockwise in increments of 90° around the image origin (0, 0). All image objects are rotated.

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

7.1.5.1 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

7.1.5.2 Examples

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

7.1.6 **Load Name (LN)**

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

LN assigns 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.

7.1.6.1 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.

7.1.6.2 *Examples*

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

7.1.7 **Mirror Image (MI)**

The MI command is deprecated as unsafe.

MI sets the mirroring for the coordinate data. All the *coordinate data* – and *only* coordinate data - are mirrored according to the specified factor. **Step and repeat distances are not coordinate data – see 4.11 - and hence are *not* mirrored. Apertures are *not* mirrored.**

This command affects the entire image. It can only be used once, at the beginning of the file. The default is no mirroring.

Quite a number of implementations incorrectly also mirror apertures and/or step and repeat distances. These incorrect implementations make the MI too risky to use. We strongly recommend *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.

7.1.7.1 *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

7.1.7.2 *Examples*

Syntax	Comments

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

Syntax	
`%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

7.1.8 **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.

7.1.8.1 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.

7.1.8.2 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

Copyright Ucamco NV

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

185

7.1.9 **Scale Factor (SF)**

The SF command is deprecated.

SF sets a scale factor for the A- and/or B-axis coordinate data. All the *coordinate data* – and *only* coordinate data - are multiplied by the specified factor for the corresponding axis. **Step and repeat distances – see 4.11 - are not coordinate data and hence are not scaled. Apertures are *not* scaled.**

This command affects the entire image. It can only be used once, at the beginning of the file. The default scale factor is '1'. The factor values must be between 0.0001 and 999.99999. The scale factor can be different for A and B axes.

Quite a number of implementation incorrectly also scale step and repeat distances. These incorrect implementations make the SF too risky to use. We strongly recommend *not* to use SF on output as you do know how the reader will interpret the file. If an image must be scaled, write out the scaled coordinates.

7.1.9.1 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.

7.1.9.2 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

---

Copyright Ucamco NV

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

187

7.3 Rectangular Hole in Standard Apertures

In addition to the round hole described in the section 4.4 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:
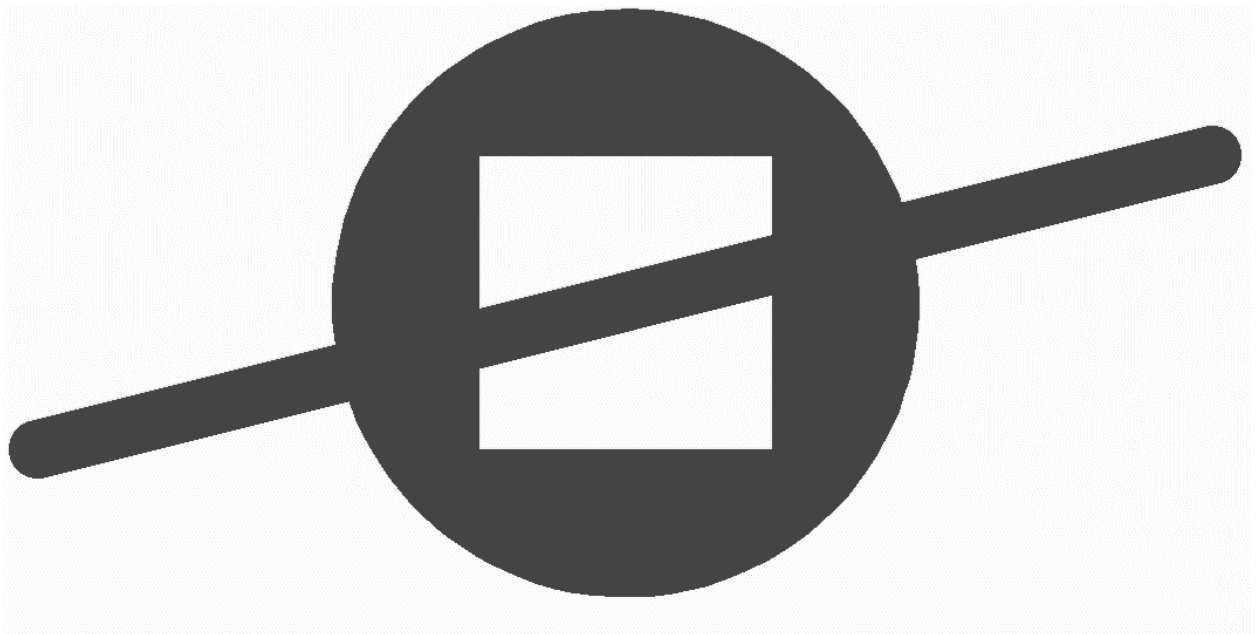
<Hole> = <X-axis hole size>X<Y-axis hole size>

The modifiers specify the X and Y sizes of the hole. Decimals >0.

The hole must strictly 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*
```

53. 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 4.1).

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.

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

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*
```

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

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

❑ *Mass parameters* was the original term for extended commands. Mass parameters made sense at the moment of its introduction but became awkward over time.

❑ *Polygon fill* was used for contour fill.

❑ *Coordinate date block* was used of operation.

❑ 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 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 commandss

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.

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

8 **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

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

http://www.rfc-base.org/txt/rfc-4122.txt

https://msdn.microsoft.com/en-us/library/windows/desktop/aa378623(v=vs.85).aspx

https://developer.apple.com/library/mac/documentation/Foundation/Reference/NSUUID_Class/

9 **History**

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*.

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.

During 2012, Ucamco reviewed the entire format in depth in "the great reform". Over 10.000 files from all over the world were gathered into a representative library to help establish current practice. Rarely used and historic format elements were deprecated. Format elements with conflicting interpretations in the market were either deprecated or clarified. The specification document itself was re-organized, the quality of the text and the drawings improved and many new drawings were added. 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 result was a powerfully clear and simple format, without needless embellishments, focused on the current needs of the PCB industry. 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 are due to Paul Wells-Edwards whose insightful comments contributed substantially to the revision.

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.

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

In August 2015, Ucamco published a draft specification *adding nested step and repeat* and block apertures to make panel descriptions more efficient, calling for comments from the user community. In November 2016 the review process was closed after substantial input and modifications and the final version included in revision 2016.12. This revision was developed by Karel Tavernier and Rik Breemeersch.

Early in 2015, the entire specification was reviewed once again by Karel Tavernier, Thomas Weyn and Artem Kostyukovich focused 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 and were deprecated. 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.

In July 2016 Karel Tavernier from Ucamco published a draft specification to include netlist information in Gerber for public review. A number of revisions of the draft were triggered by input from users. The final version was in included in revision 2016.11 from November 2, 2016.

In March 2017 Karel Tavernier from Ucamco published a draft specification to include fab documentation in Gerber for public review.

10 **Revisions**

10.1 Revision 2017.05

Added the new file attribute .UsesSameCoordinates, see 5.6.5.

Added file functions Depthrout, Viafill, Vcut, and Vcutmap.

Created section 5.7 with guidelines on the use of attributes in fabrication data; added guidelines on how to define the PCB profile in 5.7.1.1.

Reorganized and edited the chapter Overview. Clarified section on zero-size apertures. Corrected an error in the comment in example 2.13.3 pointed out by Nav Mohammed.

Revision 2017.05 was developed by Karel Tavernier.

10.2 Revision 2017.03

Added section 4.17, specifying how to put text in the image.

We now allow upper case 'X' as well as the multiply operator on macro's, next to the lower case 'x'. This conforms to wide-spread practice. This was triggered by a question from Oliver Broad.

Changed file function *Gluemask* to *Glue;* added explanation; see 5.6.3.

Reorganized chapter 4. Extended section 4.12.6.

Corrections in 4.11.1, 4.11.5, 4.12.1 and in **Attributes on regions.** triggered by remarks from Remco Poelstra. Corrected an error in example 2.13.2 pointed out by Danilo Bargen.

Revision 2017.03 was developed by Karel Tavernier.

10.3 Revision 2016.12

This is a major revision with powerful new imaging functions: 4.6, 4.11.3, 4.11.4 and 4.11.5. These allow nested step and repeat to define panels efficiently, see 4.6.3 and 4.6.4.

Thera are fixes for errors in examples, pointed out by Danilo Bargen and Urban Bruhin.

Revision 2016.12, and especially the new imaging function for panels was developed by Karel Tavernier and Rik Breemeersch. The first draft of these functions was published in August 2016. During the public review process. Thomas Weyn, Bruce McKibben, Masao Miyashita and Remco Poelstra provided essential input.

10.4 Revision 2016.11

This major revision allows to include the CAD netlist to Gerber files by adding three new standard object attributes – see 5.7 above. The goal of the Gerber CAD netlist is to facilitate upfront communication between the different parties involved in design, assembly and automation. The X2 attributes proposed include CAD netlists in Gerber fabrication data and allow to:

- Attach the component reference designator, pin number and net name to the component pads in the outer copper layers. This information is essential for a complete board display and for a complete board display. More importantly, the netlist provides a powerful checksum to guarantee PCB fabrication data integrity.

- Attach the netlist name to any conducting object on any copper layer. Lightweight viewers can then display netlists without the need for an algorithm to compute connectivity
- Attach the component reference to any object, e.g. to identify all the legend objects belonging to a given component, for example.

Several text improvements. Section 4.12.2 on regions clarified triggered by deep questions asked by Remco Poelstra.

Revision 2016.11 was developed by Karel Tavernier. Jean-Pierre Charras provided essential input on the CAD netlist, and further remarks by Remco Poelstra and Wim De Greve were included.

10.5   Revision 2016.09

New or modified attribute values – see 5.6.10:

- Replaced file function *Drawing* with *OtherDrawing*.
- Added the optional field *Filled|NotFilled* to ViaDrill.
- Added aperture function *EtchedComponent*.

Added object attributes – see 5.4. Object attributes attach information to individual graphics objects.

Corrected an error in example 4.12.5.7. The error was pointed out by Thomas van Soest and Siegfried Hildebrand. Clarified the syntax of attaching aperture attributes to regions. Added Perl script to show precisely how to calculate the .MD5. Several other clarifications.

Revision 2016.09 was developed by Karel Tavernier.

10.6   Revision 2016.06

Added a section on back-drilling job triggered by questions from Alexey Sabunin. See Required Attributes

A single Gerber file defines a single layer image. A PCB, however, is not a set of independent layers. To define the complete PCB image one must not only define the individual layer images but also their position in the layer structure. This is done with the file attributes .FileFunction and .FilePolarity. Considering the complete PCB these attributes are not meta-information but image definition data without which the PCB image is not fully defined. These two attributes can be viewed as mandatory in PCB fabrication data.

10.6.1.1   *The* Profile

The profile defines the physical extent or the area covered by the PCB. An area is described by its contour (see 4.12.1) in the Gerber format. A fabrication data set must contain a separate file with the profile, defined be a contour (G36/G37). This file is identified by the file function value Profile,(P|NP).

Normally a profile will not have holes. Slots are part of the PCB and are defined in the drill/rout files. A big hole that is not considered part of the PCB can be defined by a contour with a cut-in (see 4.12.2).

Sometimes one defines the profile layer by a line drawn with a small or a zero-size aperture instead of the profile. Such a line does not represent an area as defined in Gerber date but is a drawing, useful for human reading. While not correct, it is not difficult to convert it to a contour if the line is drawn accurately, obeying the same rules as for contour segments (see 4.12.2). So if for whatever reason you must represent the profile by a line ensure that the line is properly

constructed. Essential is that the profile is clearly defined, in a separate file, so that it is clear it ís a profile, and is not hidden in the copper layers or drawings. Corner marks are not good enough as they are only suitable for manual processing.

Backdrilling.

The .ProjectID UUID was changed to RFC4122; section rewritten by Remco Poelstra. See 5.6.8 .

Aperture function attributes were clarified triggered by remarks from John Cheesman. Drill sizes were clarified triggered by remarks from Jeff Loyer.

10.7 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 the name 'Level Polarity' by 'Load Polarity. Various other text improvements.

10.8 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.

10.9 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.

10.10 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

10.11 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 (0, 4.12.2, 4.7, 7.2, 7.4) and subsections (4.9, 4.10, 4.12, 5) were added.

The use 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 now can 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* (5.6.5), *.CreationDate* (5.6.5) and *.ProjectId* (5.6.8).

As of now the revision numbering follows the year.month scheme as in 2015.06.

10.12 Revision J4, February 2015

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.

10.13 Revision J3, October 2014

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

10.14 Revision J2, July 2014

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

10.15 Revision J1, February 2014

This revision created Gerber X2 by adding attributes to what was hitherto a pure image format. See chapter 5. X2 is Gerber version 2, with "X1" being Gerber version 1, without attributes. Gerber X2 is backward compatible as attributes do not affect image generation.

10.16 Revision I4, October 2013

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

The regions overview section 4.12.1 was expanded and examples were added different places in 4.12 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 command'.

10.17 Revision I3, June 2013

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.

10.18 Revision I2, April 2013

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.

10.19 Revision I1, December 2012

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 2.6 and 2.4. *We urge all providers of Gerber software to review their implementation in the light of these sections.*

Defaults. In previous revisions the definitions of the default values for the modes were scattered throughout the text, or were sometimes omitted. All default values are now unequivocally specified in an easy-to-read table. See 2.6. *We urge all providers of Gerber software to review their handling of defaults.*

Rotation of macro primitives. The rotation center of macro primitives was clarified. *See 4.5.3. 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 4.12. *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 can lead to 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.