4.5 Aperture Macro (AM)

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

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

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

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

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

4.5.1 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. The name must be unique, i.e. a name once given cannot be reused for another 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.

Syntax	Comments
<Primitive code>	A code specifying the primitive (e.g. polygon).
<Modifier>	Modifier can be a decimal number (e.g. 0.050), a variable (e.g. $1) or an arithmetic expression based on numbers and variables. The actual value for a variable is either provided by an AD command or defined within the AM by some previous <Variable definition>.
<Comment>	Comment does not affect the image.
<Text>	Single-line text string

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’

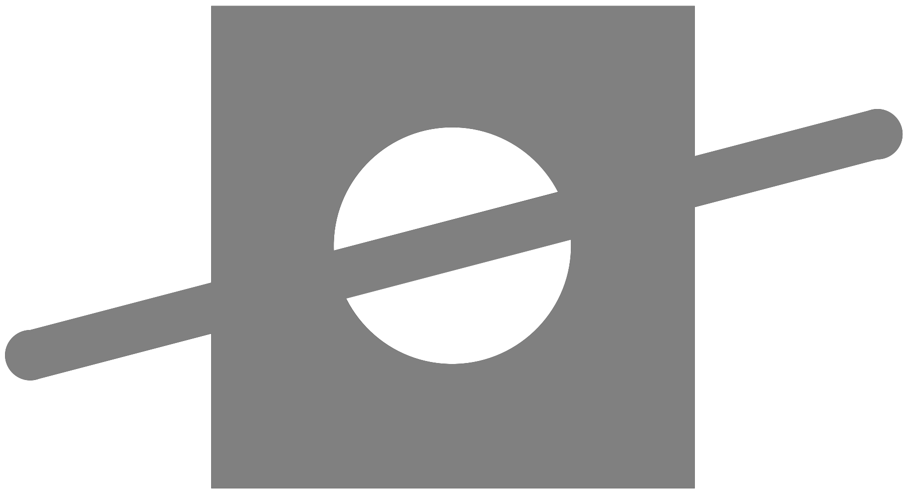
Primitives with exposure ‘on’ create the solid part of the macro aperture. Primitives with exposure ‘off’ erase the solid part created earlier *in the same macro*. Exposure off is used to create a hole in the aperture – see also 4.4.6.

The erasing action of exposure off only acts on other primitives within the same macro definition. When a macro is flashed the hole does not clear objects in the final image – the hole is transparent. Another way of expressing it is that the macro definition is flattened before it is used, and the result is a positive image.

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*
```



13. Macro aperture with a hole above a draw

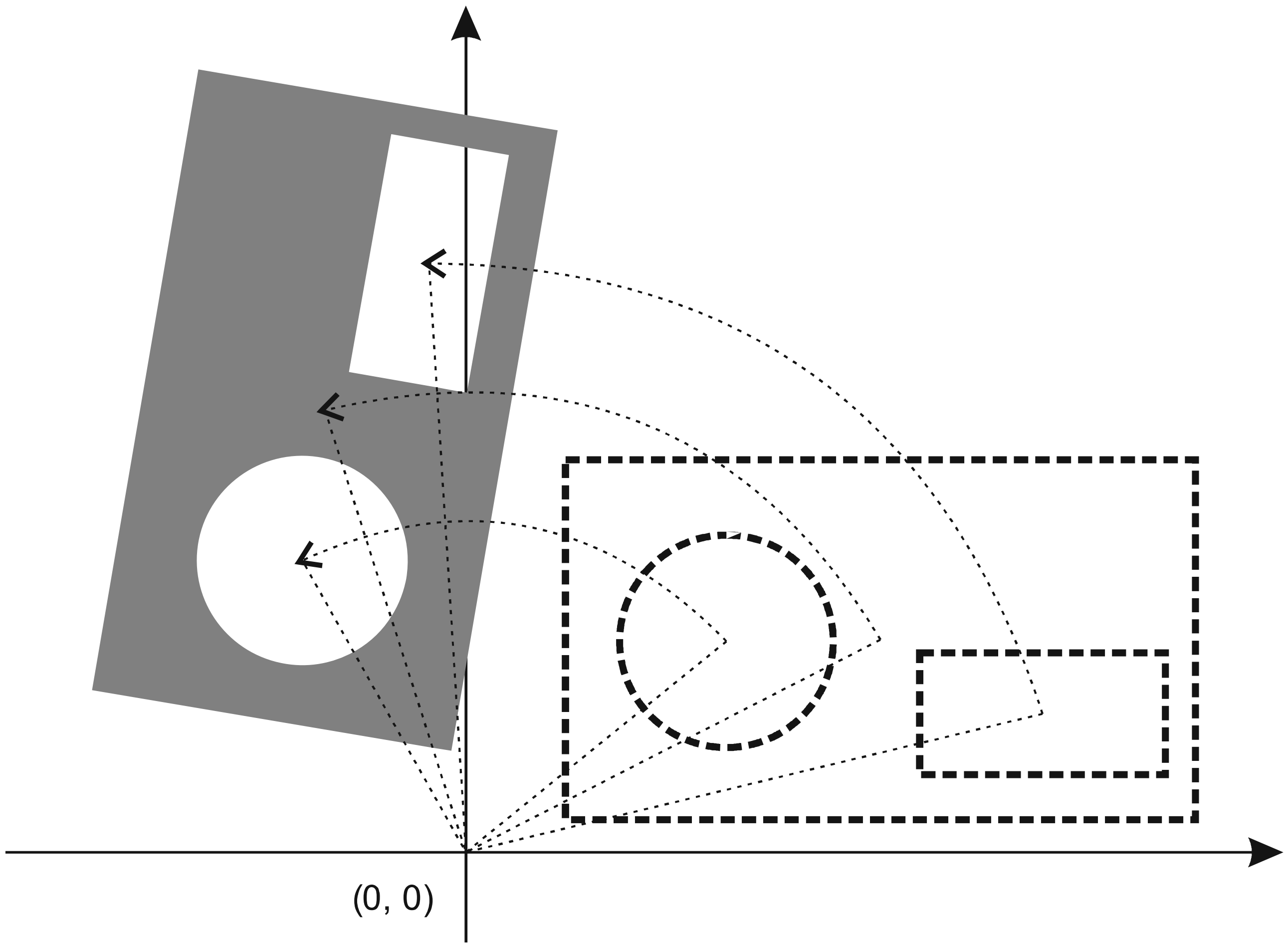
Note that the draw is still visible through the hole.

4.5.3 Rotation Modifier

All primitives can be rotated around the *origin* of the macro definition, i.e. its point (0, 0). (Make no mistake: rotation is *not* around the geometric center of the primitive, unless of course it coincides with the origin.)

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

To rotate a macro composed of several primitives it is sufficient to rotate all primitives by the same angle. See illustration below.



14. Rotation of an aperture macro composed of several primitives

⚠ **Warning:** Rotation is around the origin of the macro definition, not around the geometric center of the primitive – unless the two coincide of course. The reason is obvious: if rotation were about the center of each primitive a composite aperture like the one above would fall apart under rotation.

4.5.4 **Primitives**

4.5.4.1 Overview

Macro Primitives			
Code	**Name**	**Modifiers**	**Section**
0	Comment		4.5.4.2
1	Circle	Exposure, Diameter, Center X, Center Y, Rotation	4.5.4.3
20	Vector Line	Exposure, Width, Start X, Start Y, End X, End Y, Rotation	4.5.4.4
21	Center Line	Exposure, Width, Hight, Center X, Center Y, Rotation	4.5.4.5
4	Outline	Exposure, # vertices, Start X, Start Y, Subsequent points..., Rotation	4.5.4.6
5	Polygon	Exposure, # vertices, Center X, Center Y, Diameter, Rotation	4.5.4.7
6	Moiré	Center X, Center Y, Outer diameter rings, Ring thickness, Gap, Max # rings, Crosshair thickness, Crosshair length, Rotation	4.5.4.8
7	Thermal	Center X, Center Y, Outer diameter, Inner diameter, Gap, Rotation	4.5.4.9

Table with macro primitives

4.5.4.2 Comment, Code 0

The comment primitive has no effect on the image but adds human-readable comments in an 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 for strings in section 3.6.6.

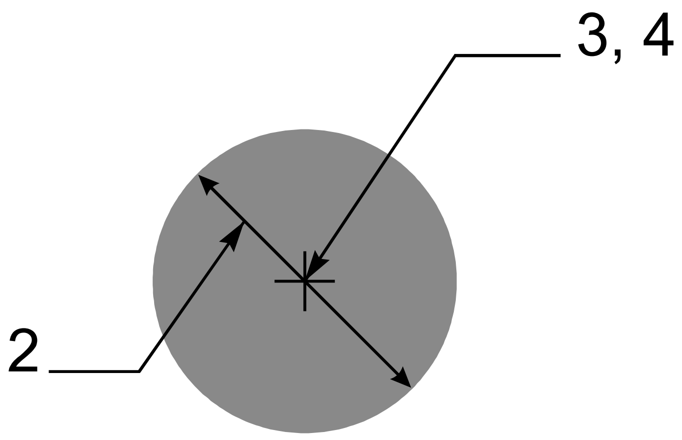
Example:

```
%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*%
```

4.5.4.3 Circle, 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.



15. Circle primitive

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

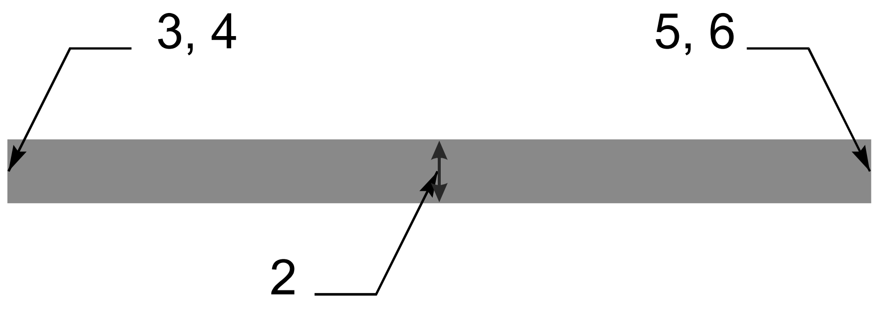
Example:

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

4.5.4.4 Vector Line, 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



16. Vector line primitive

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

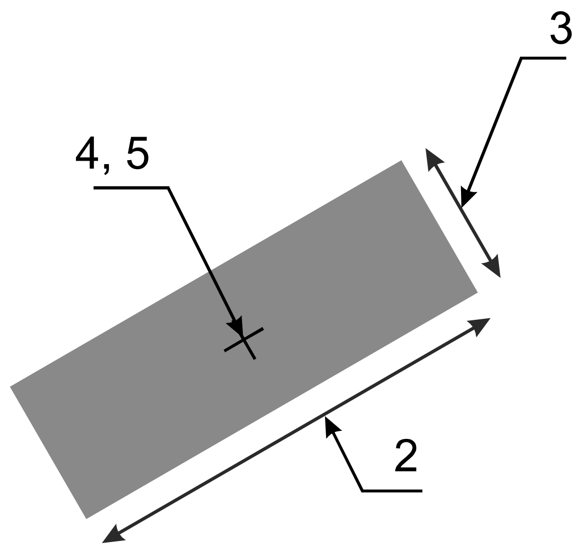
Example:

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

4.5.4.5 Center Line, 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. ⚠ **Warning**: The rotation is *not* around the center point. (Unless the center point happens to be the macro origin.)



17. Center line primitive

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

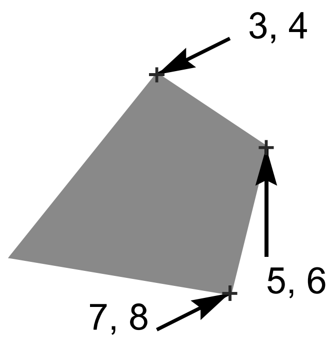
Example:

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

4.5.4.6 Outline, Code 4

An outline primitive is an area defined by its outline or contour. The outline is a polygon, consisting of linear segments only, defined by its start vertex and n subsequent vertices. The outline must be closed, i.e. the last vertex must be equal to the start vertex. The outline must comply with all the requirements of a contour according to 4.12.3.

Modifier number	**Description**
1	Exposure off/on (0/1)
2	The number of vertices of the outline = the number of coordinate pairs *minus one*. An integer ≥3.
3, 4	Start point X and Y coordinates. Decimals.
5, 6	First subsequent X and Y coordinates. Decimals.
...	Further subsequent X and Y coordinates. Decimals. The X and Y coordinates are *not* modal: both X *and* Y must be specified for all points.
3+2n, 4+2n	Last subsequent X and Y coordinates. Decimals. *Must* be equal to the start coordinates.
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.



18. Outline primitive

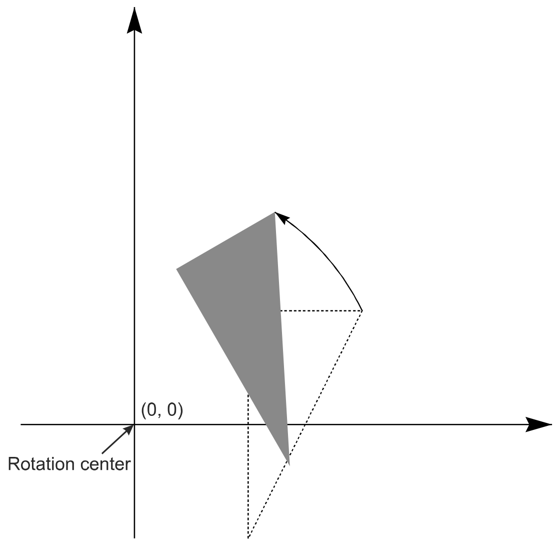
The maximum number of vertices is 5000. The purpose of this primitive is to create apertures to flash *pads* with special shapes. The purpose is not to create copper pours. Use the region statement for copper pours; see 4.12.

Example:

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
AM Triangle _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

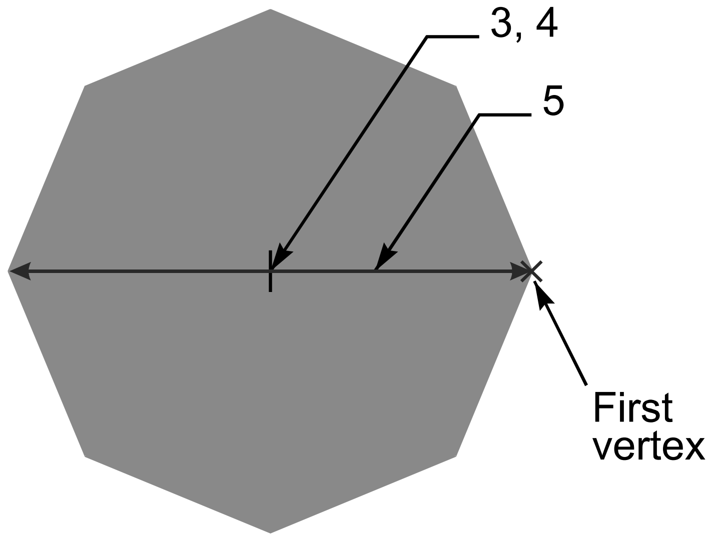


19. Rotated triangle

4.5.4.7 Polygon, 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. An integer. 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.



20. Polygon primitive

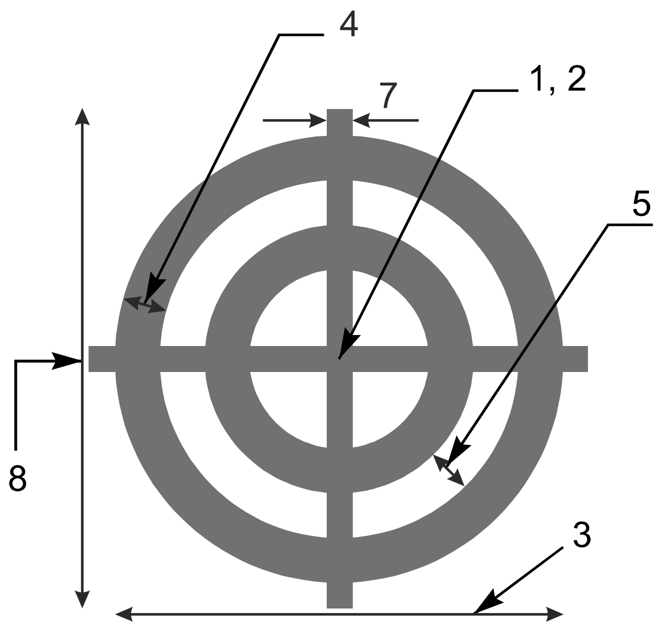
Example:

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

4.5.4.8 *Moiré, 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. The effective number of rings can be less if the center is reached. If there is not enough space for the inner ring it becomes a full disc.
7	Crosshair thickness. A decimal ≥ 0. If the thickness is 0 there are no crosshairs.
8	Crosshair length. A decimal ≥ 0. If the length is 0 there are no crosshairs.
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.



21. Moiré primitive

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

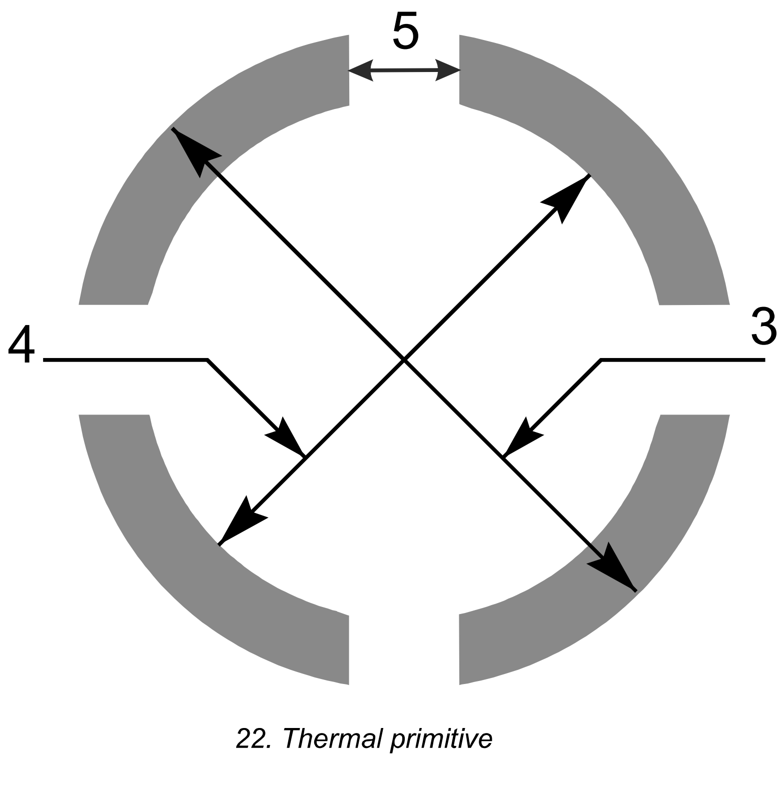
Example:

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

4.5.4.9 Thermal, 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. Note that if the (gap thickness)*√2 ≥ (inner diameter) the inner circle disappears. This is not invalid.
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.



22. Thermal primitive

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 4.5.4. 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 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 an arithmetic expression consisting of arithmetic operators, constants, with or without decimal point, and other variables. The standard arithmetic precedence rules apply. The brackets '(' and ')' are available to overrule the standard precedence.

The following arithmetic operators are available:

Operator	Function
+	Unary plus (expressions are positive even without the unary plus)
-	Unary minus (negates an expression)
+	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

Their precedence is as follows:

- ‘(’ and ‘)’
- The unary ‘+’ 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.

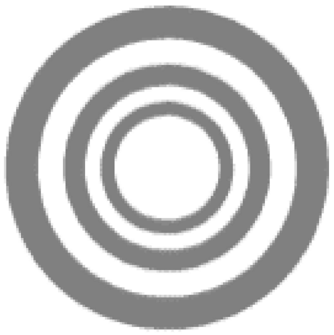
```
%ADD35DONUTCAL,0.020X0X0X0.03*%
```

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*%
```



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:

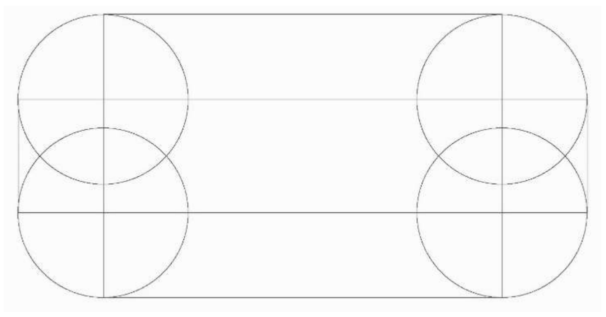
```
%ADD35DONUTCAL,0.020X0X0*%
```

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:



23. 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*%
```