Footprint creation for the open-source layout program "PCB"

Author	***e-mail***	***Date***	***Changes made***
Stephen Meier		2003, 2004	Wrote initial document.
Stuart D. Brorson	sdb@cloud9.net	12/27/04	Formatting changes, added more tables & info.
Stuart D. Brorson	sdb@cloud9.net	01/29/05	Included dimensioned pad drawing, incorporated changes suggested by Dan McMahill, updated ElementArc, other improvements.
SDB	sdb@cloud9.net	08/05/07	Updated to reflect current state of PCB file format.
SDB	sdb@cloud9.net	8.11.2007	Incorporated fixes/suggestions from DJ Delorie and Stefan Salewski
SDB	sdb@cloud9.net	8.18.2007	Incorporated error corrections from DJ. Also first release under the GFDL.

Introduction

PCB is an open-source program used for physical design of printed circuit boards (i.e. board layout). PCB is a freely-downloadable, GPL'ed application which runs on Linux, BSD, and other unicies supporting X11. PCB is part of the gEDA suite (http://www.geda.seul.org/). The homepage of PCB is http://pcb.sf.net/. If you are unfamiliar with PCB design or the gEDA suite, you should take time to familiarize yourself with these topics before continuing this document.

When designing a printed circuit board, one of the most important things you need to define are the land patterns -- or footprints -- for each device. Like any layout program, PCB needs a footprint for each device. The footprint tells PCB how to draw the device pads or pin holes, silk screen outline, device name, and other properties associated with an individual component. The footprints used in PCB are usually stored in an external file, and are read into PCB itself when the PCB is created or updated. One of the major advantages of using PCB is that the footprint files are plain ASCII, and are well structured. The purpose of this document is to detail how PCB footprints (land patterns) are defined in the footprint files, and explain how to use them effectively during layout using PCB.

An unusual feature of PCB is that it supports two entirely separate footprint libraries with two entirely different footprint mechanisms. This is because PCB is an old program which has been developed by many different people for many different platforms over more than two decades. Due to its history, PCB has had to handle varying limitations imposed by target platform speed, memory size, and so on.

The first footprint system is referred to as the “oldlib”, or the “M4 library”. This system is historic; it relies upon using the GNU macro language M4 to generate footprints on the fly while PCB is running. Footprints living in the M4 library are prefixed by a tilde (“~”) -- for example, “~geda”. **This document does not cover usage of the M4 library!** Indeed, we recommend that you avoid the M4 library – it is deprecated for creating new footprints. However, the M4 library is large, has many adherents, and it is an integral part of PCB, so it probably won't disappear anytime soon.

The second footprint system for PCB is called the “newlib”. Newlib footprints are defined using ASCII text files which define each graphical primitive which makes up an entire footprint. **This document focuses exclusively on defining and using newlib footprints.** You can use newlib footprints which are distributed with PCB, or you can create your own, and put them in a dedicated directory.

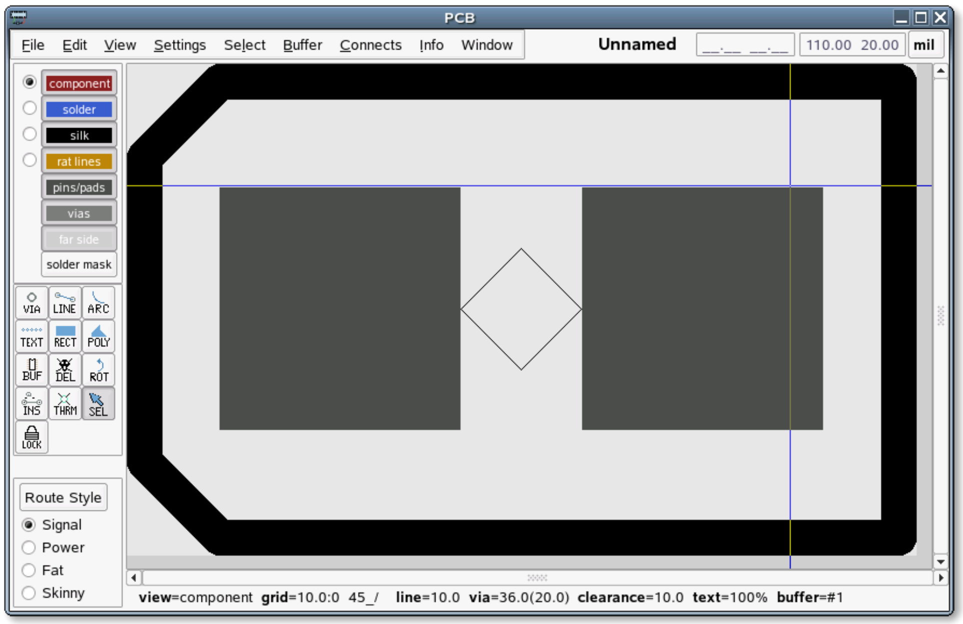
There are several ways to create a newlib footprint using PCB. For example, you may create a footprint graphically within PCB by drawing it, and then saving it out. This procedure is documented in the main PCB documentation; we will not cover it here. This document will concentrate on how footprints are defined within the ASCII footprint file itself. Using this information, you can either create a footprint from scratch using a text editor, or copy an existing footprint, and edit its parameters to correspond exactly to the footprint which you wish to create. Power users of PCB usually prefer the latter method for footprint creation, because it's easy to start with a known symbol, and manually editing the footprint file gives you the most control over your footprint.

Understanding how to correctly set up footprints in a footprint file is important. A footprint can critically effect the manufacturability of the board it lives on. If a footprint's pads are in the wrong place, or the solder mask relief is incorrectly defined, it can be impossible to attach the device to its pads. If the solder mask doesn't cover traces near pads, the traces may become soldered to the pads. Boards using footprints that have the pads in the correct spot but of the wrong size can have a reduced manufacturing yield and possibly a reduced life. Therefore, properly defining your footprints is a critical part of creating a working PCB. See the standards document IPC-SM-782A “Surface Mount Design and Land Pattern Standard” for a more complete discussion of the requirements and impacts of surface mount patterns.

How footprints are used in your .pcb file

In the PCB layout file itself (usually called something like “foo.pcb”), an individual footprint is called an “Element”. If you examine a .pcb file, you will see many Element declarations throughout the file – you should have one declaration for each component you have placed on your layout.

The first line of the Element entry holds top-level information about the footprint itself. Within the element are held the atomic graphical elements making up the footprint. The atomic graphical elements include solder pads, through-holes for pins, lines drawn on the silkscreen layer, and other items which comprise a footprint. For example, here is a simple PCB land pattern for an 0805 chip resistor. First, here's the footprint graphic (which is what is placed in PCB):



view=component grid=10.0:0 45_/ line=10.0 via=36.0(20.0) clearance=10.0 text=100% buffer=#1

Figure 1: *Footprint of two pad 0805 passive.*

Next, here's the footprint file which generates the above footprint:

```
Element["" "" "" "" 1000 1000 -1000 -1000 0 60 ""]
(
        Pad[-3000 0 -3000 0 4000 1200 4600 "" "1" "square"]
        Pad[3000 0 3000 0 4000 1200 4600 "" "2" "square,edge2"]
        ElementLine [-5000 -3750 6250 -3750 600]
        ElementLine [6250 -3750 6250 3750 600]
        ElementLine [6250 3750 -5000 3750 600]
        ElementLine [-5000 3750 -6250 2500 600]
        ElementLine [-6250 2500 -6250 -2500 600]
        ElementLine [-5000 -3750 -6250 -2500 600]
)
```

Within the parentheses after “Element” is information pertaining to the entire footprint, such as its mark position, a placeholder for its refdes, and so on. Following the top-level information, you can see two “Pad” graphical elements. This example 0805 chip resistor requires two pads; other devices may require hundreds of them, leading to hundreds of “Pad” lines in such a part. The information within the “Pad” line specifies the size of the solder pad, any clearance around the pad, each pad's name and number, and other attributes. The “ElementLine” graphical element produces a line drawn on the silkscreen layer. In this example, there are six “ElementLine” elements drawn on the silkscreen layer, corresponding to the rectangular box around the 0805 footprint along with two chamfered edges on the left. The information within the “ElementLine” line specifies the width and position of the lines drawn on the silkscreen layer.

Further details about defining these graphical elements are provided in latter sections of this document.

Peculiarities of PCB

Due to its long history of development, PCB has a number of quirks which you need to be aware of. This section attempts to list some of these quirks relevant to creating and editing footprint files.

- **Coordinate system.** It is particularly important to remember that PCB uses a standard computer graphics coordinate system, and not a Cartesian coordinate system. This means that X increases to the right (as is normal), but Y increases **downwards**. Please keep this in mind when defining the graphical locations of footprint elements.

- **Units.** Originally, PCB used mils (1/1000ths of an inch) as the basic unit of measure. However, recent upgrades to the program have improved its resolution to 1/100 mils (10^{-5} in). Both units are used freely in the footprint files. By definition, units held in **round** parentheses "()" are **mils**. Units contained in **square** braces "[]" are in **1/100 mils**. **Be aware of this dichotomy, and always check how your units are specified!**

  We recommend you to use the new, square brackets and avoid the old, round brackets when defining your footprints!

- **Absolute vs. relative coordinates.** The "Element" declaration calls out the location of the "mark", which specifies the position of the entire footprint in the .pcb file. If the Element declaration's parameters are surrounded by round brackets "()", then the **other** graphical elements used in the footprint are specified using **absolute** units. That is, the footprint's individual parts (ElementLine, Pad, etc.) are positioned relative to the (0, 0) position of the **PC board**. When you move an absolute units footprint in PCB, then PCB will overwrite all position coordinates used by the individual graphical elements with the new position coordinates. **We recommend that you do not use the old round brackets – they are a deprecated, legacy feature of PCB!**

  If the "Element" declaration uses square brackets -- "[]" -- then the **other** graphical elements used in the footprint are specified using **relative** units. That is, the footprint's individual parts (ElementLine, Pad, etc.) are positioned relative to the **mark position** called out in the Element tag. When you move a relative units (square bracket) footprint in PCB, then PCB will only update the position of the mark held in the Element declaration; the other coordinates used in the footprint remain the same. **We recommend you to use the new, square brackets and avoid the old, round brackets when defining your footprint's Element line!**

- **Footprint libraries**. The original PCB footprint library was written using the macro language M4, which is considered by some to be an obscure language. A modern graphical footprint library system – newlib – has been developed for PCB. Most new footprints contributed to the PCB project use the newlib footprint syntax. **Only the newlib library syntax is described in this document.**

- **PCB's Solder Mask Relief Implementation.** PCB only allows for the Pad line to determine the solder mask relief size and shape. Therefore creating gang shadow mask windows (see Glossary) can only happen by setting the Pad sizes and correctly placing the individual components close enough together such that the shadow mask windows merge.

- **Keepouts.** Currently, PCB doesn't have the concept of a keepout. Therefore, you must track any keepout constraints manually. A poor man's component keepout may be produced by encircling your footprint with a boundary drawn on the silkscreen layer. The silkscreen should extend a little bit beyond the part's outline in all directions. Then, during layout manually verify that no silkscreen boundaries touch each other. Note that no DRCs will show up using this method, so you must take care when placing and inspecting your parts.

Developing a new footprint for PCB – work flow

When creating a new footprint file, you will typically follow a work flow like this:

1. Determine the mark to be used by the footprint. This is often either the center of the part, or the location of pin 1. It calls out the position of the entire footprint once the footprint is placed on the larger PCB.

2. Determine the rotational orientation of the text. In PCB, this may be 0, 90, 180, or 270 degrees.

3. Determine the grid placement courtyard and its relationship to the center. The grid courtyard is the total area encompassed by the footprint. When mounted, the component itself will be centered in the courtyard. Determining the grid placement courtyard is particularly important if you plan to assemble your design using a pick-and-place machine since the machine wants to place components at particular positions on the grid. Consult your assembly house for more information about their requirements. If you are hand-assembling your board, you don't need to worry about this.

4. Determine the soldering method to be used. Wave and reflow soldering processes place different requirements upon the pad dimensions, as well as the solder mask clearances. Further information about this topic can be obtained from IPC documents, or from your assembly house.

5. Determine the pad locations and sizes. This is usually found in the materials supplied by the part vendor. Alternately, you can consult IPC documents which specify recommended footprints for many common parts. Keep in mind that pad location and size depend – to some extent – upon your board manufacturer's tolerances as well as your solder method. Again, either use conservative numbers for your pad dimensions or speak to your board house about their recommended design rules.

6. Determine the solder mask application method and its tolerances. Further information about this topic can be obtained from IPC documents, or from your assembly house.

7. Determine the solder mask relief size. This depends upon the type of soldering process you intend on using, the tolerances your board house can meet, and other factors. It is always best to use conservative numbers, or consult with your board manufacturer and assembly house first.

8. Open a footprint file in a directory on your PCB search path. Typically, you will want to name the file something suggestive of the footprint held within it. Typically, Element declaration is placed into a single footprint file. Although, no file name suffix is enforced by the PCB program, modern gEDA convention calls for ".fp" as the suffix used for footprints. Examples footprint names include "Res_0805_large.fp" or "TQFP-44.fp".

9. Create the Element macro within the footprint file.

10. Within the Element body, add a Pad line for each component pad.

11. Within the Element body, add a Pin line for each component through-hole pin. You can also use a Pin line to define a through-hole for component mounting.

12. Within the Element body, add ElementLine or ElementArc lines to create the footprint outline on the silkscreen layer. The footprint outline needn't encircle the grid placement courtyard but doing so can be convenient for correct placement.

13. Save your footprint file.

14. If you are using gschem – the schematic capture program which is part of the gEDA Suite – you attach a footprint attribute to each part which calls out the name of the footprint file you created as its value. (That is, the name=value pair you want is footprint=foo.fp".) The utility program "gattrib" is very useful for efficiently attaching footprint attributes to a design which you have already drawn.

Anatomy of a footprint

To understand the parameters used in defining a footprint, consider the footprint used by SMT resistors. A footprint is shown in the figure below.

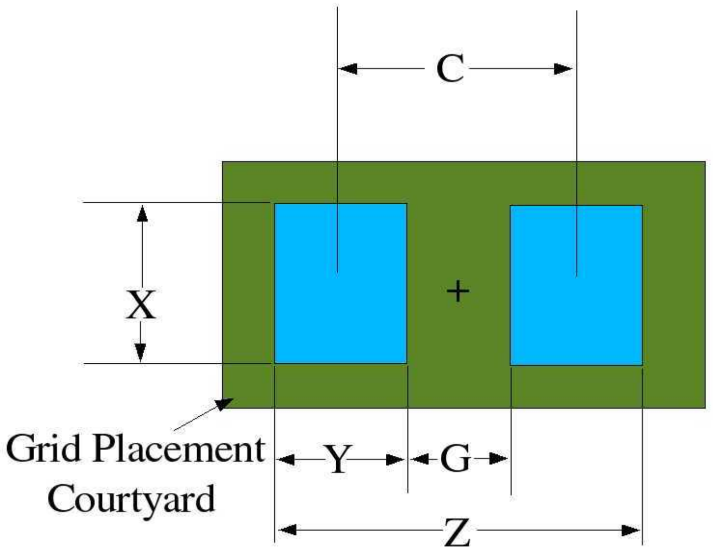


Figure 2: *Example footprint for an SMT resistor.*

The blue areas correspond to bare metal. This is where you will solder the component's pads to your board. The green area is the placement courtyard. This is the area taken by the component itself (along with some additional area to compensate for the component's dimensional variability due to manufacturing tolerances).

Dimensional data for different resistor sizes are presented in the following table.

Type	*C*	*X*	*Y*	*Z*	*G*	*Grid*
'0402	51.2	27.5	35.4	86.6	15.7	39.4x118.1
'0603	66.9	39.4	43.3	110.2	23.6	157.5x118.1
'0805	74.8	59.1	51.2	126.0	23.6	157.5x315.0
1206	110.2	70.9	63.0	173.2	47.2	157.5x393.7
1210	110.2	106.3	63.0	173.2	47.2	118.1x393.7
2010	173.2	106.3	70.9	244.1	102.4	118.1x551.2
2512	220.5	126.0	70.9	291.3	149.6	315.0x629.9

Dimensions C, X, Y, Z, G and Grid are all in mils. Data is derived from the table on page 73 Of IPC-SM-782A “Surface Mount Design and Land Pattern Standard”. Note that this data is example data for the purposes of fixing ideas only. Depending upon the details of your particular manufacturing process (wave vs. reflow solder, assembly tolerances, etc.) the dimensional values you use may differ from those in the above table by several mils or more.

Each PCB Pad impacts several layers. If the Pad is on the component (top) side of the board, it impacts the component layer (where the pad lives), the mask (component-side solder mask relief) and the paste (component-side solder paste) layer. If the Pad lies on the solder (bottom) side of the board, it impacts the solder layer (where the pad lives), the mask (solder-side solder mask relief) and the paste (solder-side solder paste) layer.

Each instance of a macro needs its parameters selected for the manufacturing techniques used to place and solder the components to the board. The standards document (IPC-SM-782A) will cover these in detail. The scope of this paper will be how to use the standards document to generate suitable PCB footprints.

Footprint creation do's and dont's

- Do – Please name your footprint file using ".fp" as a suffix. This is a newly accepted PCB best practice

- Do – Use the **square bracket** footprint definition syntax. Avoid the round bracket footprint syntax; it is deprecated.

- Do – Make sure your solder pads are large enough for SMT devices. The pad should provide sufficient room for development of a solder meniscus between your part and the pad itself. Vendor recommended pads are usually OK. However, it sometimes helps to increase the pad size by a few mils in each dimension if you have the real estate on your board. However, don't go overboard on fine-pitch parts; if the pads get too close together, you run the risk of creating solder bridges between adjacent pads!

- Do – Double check all your footprints (and your layout too). A sizable number of board mistakes arise from simple footprint errors due to carelessness. Things to check for:

    - ✔ Are your pin holes large enough to fit the pins? It doesn't hurt to oversize your holes by at least five mils to ensure that everything will fit together.
    - ✔ Are your pads large enough? Again, oversizing your pad dimensions (w.r.t. the component's foot dimensions) by a few mils is good practice. Also, making the pad extend ten or more mils outward from the end of the foot will give you a place to put your soldering iron while assembling the board.
    - ✔ Is the pad spacing correctly set?
    - ✔ PCB uses mils as the unit of measure for footprints. Sometimes, vendors use metric units in defining footprints. This is particularly true for connectors. Make sure you have converted any metric units into mils in your footprints.
    - ✔ Are you sure you have the right footprint for the package you have specified? (I have personally seen several cases where an SO-16 footprint was placed for an MSOP-16 part. One of these was my fault! Mistakes like this cost money.)
    - ✔ Mechanicals. Make sure all your parts will fit, and that you haven't squeezed them too close together. Also, if your board must fit into a constrained space, or satisfy height restrictions, make sure that you have properly incorporated these constraints into your design. Since PCB doesn't have the concept of keepouts or height restrictions, you need to verify these constraints manually.

- Do – When you are done with your layout, make sure you inspect the Gerber files using a Gerber viewer. This important step will help you catch errors which might not have shown up within PCB. Several free Gerber viewers exist on the net; a quick Google search will identify several for you. On Windows, I use "GCPrevue". On Linux, a decent Gerber viewer is "gerbv".

- Do – Perform a trial placement using your parts. Once you have created your PCB, print it out on a

PostScript printer using 1:1 scaling, and place all your parts onto their footprints. This is a great way to catch footprint (and other layout) errors.

- Do – Use solder mask over bare copper to prevent solder migration. Solder mask tolerances: Screen printed solder masks can be used to produce masks with 15 mil spacing. Photo-imaged solder masks can achieve spacings down to around 3 mils.

- Do – Inspect your layout and verify that all plane regions are connected to their respective nets. Thermals are placed manually in PCB, so it is easy to forget them. This is particularly important if your board has internal plane layers, since you can't easily rework an internal layer. You might also want to verify that the plane layers do have voids (antipads) around non-connected vias or pins.

- Do – Inspect your layout to verify that all text annotations are done on the silkscreen layer. PCB's DRC checker will not identify shorts occurring because of text on a metal layer. Also, verify that your silkscreened text doesn't get too close to metal pads – if your board manufacturer has registration problems, silkscreen can get on your pads, and you won't be able to solder to that pad.

- Don't – Solder masks should not cover a fiducial or the fiducial clearance area since it could cause oxidation and interfere with automated location of the fiducial.

- Don't – Don't allow solder mask contamination on component pads. Solder mask on pads can cause failures since you can't reliably solder a component's foot to a pad through solder mask.

- Don't – Beware of skimping on solder mask. If you allow closely spaced, un-masked copper areas, short circuits via solder bridging can occur when the components are soldered down. This problem can be particularly acute if you are wave soldering your parts.

Element

The “Element” tag defines the top-level information required to define a footprint for a particular part. The Element head holds information pertinent to the footprint as a whole. Within the Element macro body are the individual graphical components of the footprint. The Element body is the code with in the parentheses.

Format

The preferred format of this graphical element is given below in **bold**.

```
Element [SFlags "Desc" "Name" "Value" MX MY TX TY TDir TScale TSFlags] (
Element (NFlags "Desc" "Name" "Value" MX MY TX TY TDir TScale TNFlags) (
Element (NFlags "Desc" "Name" "Value" TX TY TDir TScale TNFlags) (
Element (NFlags "Desc" "Name" TX TY TDir TScale TNFlags) (
Element ("Desc" "Name" TX TY TDir TScale TNFlags) (
   ... element body ...
)
```

Note that either mils or 1/100's of a mil are allowable for the Element tag. This is signaled by the use of round “()” (mils) or square “[]” (1/100 mil) brackets. Also, the distinction between absolute and relative coordinates is specified by the use of round or square brackets.

Detailed description

Item	*Allowed value*	*Explanation*	*Comment*
SFlags	Symbolic flags (strings)	Symbolic flags applying to the element as a whole.	
NFlags	Numeric (hex) flags	Numeric flags applying to the element as a whole.	
Desc	string (surrounded by double quotes)	The name of the footprint or footprint file. This is one of the three strings which can be displayed on the screen.	This field is filled out by gsch2pcb or PCB itself. Leave an empty string (“”) when you create the footprint file.
Name	string (surrounded by double quotes)	The name of the element, usually the reference designator.	This field is filled out by gsch2pcb or PCB itself. Leave an empty string (“”) when you create the footprint file.
Value	string (surrounded by double quotes)	value of component on this particular PCB	This field is filled out by gsch2pcb or PCB itself. Leave an empty string (“”) when you create the footprint file.

Item	*Allowed value*	*Explanation*	*Comment*
MX	Decimal integer (mils or 1/100 mils)	Mark_x. This is the X location of the footprint's mark. It tells PCB where to place the footprint when first read into your layout. Later, when you place the component, PCB will reset this value.	When creating footprint file set to 10 mil so part's initial position is on working area of board.
MY	Decimal integer (mils or 1/100 mils)	Mark_y. This is the X location of the footprint's mark. It tells PCB where to place the footprint when first read into your layout. Later, when you place the component, PCB will reset this value.	When creating footprint file set to 10 mil so part's initial position is on working area of board.
TX	Decimal integer (mils or 1/100 mils)	Text pos X. Refdes initial position X coordinate w.r.t. mark location. Later, PCB will reset this value when you move the refdes.	Must experiment in order to find optimal initial location for text.
TY	Decimal integer (mils or 1/100 mils)	Text Pos Y. Refdes initial position Y coordinate w.r.t. mark location. Later, PCB will reset this value when you move the refdes.	Must experiment in order to find optimal initial location for text.
TDir	decimal integer	The relative direction of the text. 0 = horizontal 1 = CCW 90 deg 2 = 180 deg 3 = CW 90 deg	
TScale	decimal integer	Size of the text, as a percentage of the "default" size of of the font (the default font is about 40 mils high). Default is 100 (40 mils).	Usually set to 100.
TSFlags	string (surrounded by double quotes)	Symbolic flags applying to the text accompanying the footprint.	
TNFlags	unsigned hex value	Hex flags applying to the text accompanying the footprint.	

Example 1 – Old (deprecated) file format

This example shows the old round bracket file format. This usage is now deprecated in favor of the new, square bracket file format (shown in Example 2 below). The old file format is exemplified here for purposes of documentation only.

```
Element(0x00 "Surface Mount Chip Resistor 0603" "" "" 0 0 -31 -82 0 100 0x00)
(
          Pad(-2 0  2 0 39 30 50 "pad 1" "1" 0x00000100)
          Pad(65 0 69 0 39 30 50 "pad 2" "2" 0x00000100)
          ElementLine(-21 -35  87 -35 5)
          ElementLine( 87 -35  87  35 5)
          ElementLine( 87  35 -21  35 5)
          ElementLine(-21  35 -21 -35 5)
)
```

This example defines an 0603 SMT resistor having two solder pads and four ElementLines on the silkscreen layer to define the land pattern. Note that the dimensions held in this example are in **mil** units because they are held in **round** brackets. **This usage is deprecated!** Please use square bracket format shown in example 2.

Example 2 – New file format

This example shows a TO-18 transistor footprint. This element entry uses the new, square bracket file format. It also exemplifies how to use the new, string-based flags. **Please use this format for all new footprints!**

```
Element["" "TO-18" "" "" 65000 287500 -6000 3500 0 100 ""]
(
        Pin[-10000 10000 6000 2000 6600 2800 "" "C" "edge2"]
        Pin[-10000 0 6000 2000 6600 2800 "" "B" "edge2"]
        Pin[0 0 6000 2000 6600 2800 "" "E" "square,edge2"]
        ElementLine [7500 3000 11500 3000 600]
        ElementLine [11500 3000 11500 7000 600]
        ElementLine [11500 7000 7500 7000 600]
        ElementArc [-5000 5000 10500 10500 270 90 600]
        ElementArc [-5000 5000 10500 10500 180 90 600]
        ElementArc [-5000 5000 10500 10500 0 90 600]
        ElementArc [-5001 4999 10501 10501 90 90 600]
        ElementArc [-5000 5000 12500 12500 270 90 600]
        ElementArc [-5001 5001 12501 12501 180 90 600]
        ElementArc [-5000 5000 12500 12500 90 90 600]
        ElementArc [-5001 5001 12499 12499 0 90 600]

)
```

Pad

The Pad element is held within the body of a footprint (Element). It describes a single rectangular metalization serving as a land pattern for an SMT device. Note that either mils or 1/100's of a mil are allowable for the Pad tag. This is signaled by the use of round "()" (mils) or square "[]" (1/100 mil) brackets. The distinction between relative and absolute units is determined by the type of brackets used in the enclosing "Element" tag.

Format

The preferred format of this graphical element is given below in **bold**.

```
Pad [rX1 rY1 rX2 rY2 Thickness Clearance Mask "Name" "Number" SFlags]
Pad (rX1 rY1 rX2 rY2 Thickness Clearance Mask "Name" "Number" NFlags)
Pad (aX1 aY1 aX2 aY2 Thickness "Name" "Number" NFlags)
Pad (aX1 aY1 aX2 aY2 Thickness "Name" NFlags)
```

Detailed description

Item	*Allowed value*	*Explanation*	*Comment*
rX1 aX1	Decimal integer (mils or 1/100 mils)	X coord of first point of line segment. See figure.	r = units relative to the mark. a = absolute units.
rY1 aY1	Decimal integer (mils or 1/100 mils)	Y coord of first point of line segment. See figure	r = units relative to the mark. a = absolute units.
rX2 aX2	Decimal integer (mils or 1/100 mils)	X coord of second point of line segment. See figure	r = units relative to the mark. a = absolute units.
rY2 aY2	Decimal integer (mils or 1/100 mils)	Y coord of second point of line segment. See figure	r = units relative to the mark. a = absolute units.
Thickness	Decimal integer (mils or 1/100 mils)	Width of metal surrounding line segment. See figure	
Clearance	Decimal integer (mils or 1/100 mils)	Separation of pad from other conductors on any layer. See figure.	This is separation – not width. Also note factor of ½ in definition. See figure.
Mask	Decimal integer (mils or 1/100 mils)	Width of solder mask relief. See figure.	The solder mask relief, is the area around the pad where the solder mask is not applied.

Item	*Allowed value*	*Explanation*	*Comment*
Name	string (surrounded by double quotes)	Name of pad. Arbitrary identification string.	Set to pad number.
Number	string (surrounded by double quotes)	Pad number. Used in attaching rats, so it must be consistent with the definition in the netlist.	Set to pad number.
SFlags	string (surrounded by double quotes)	Symbolic flags applying to the pad.	
NFlags	unsigned hex value	Hex flags applying to the pad.	Hex codes described in separate section below.

As shown in the figure below, a pad's dimensions are defined by a line segment with endpoints (x1, y1) and (x2, y2). All other parameters defined in relationship to this line segment.

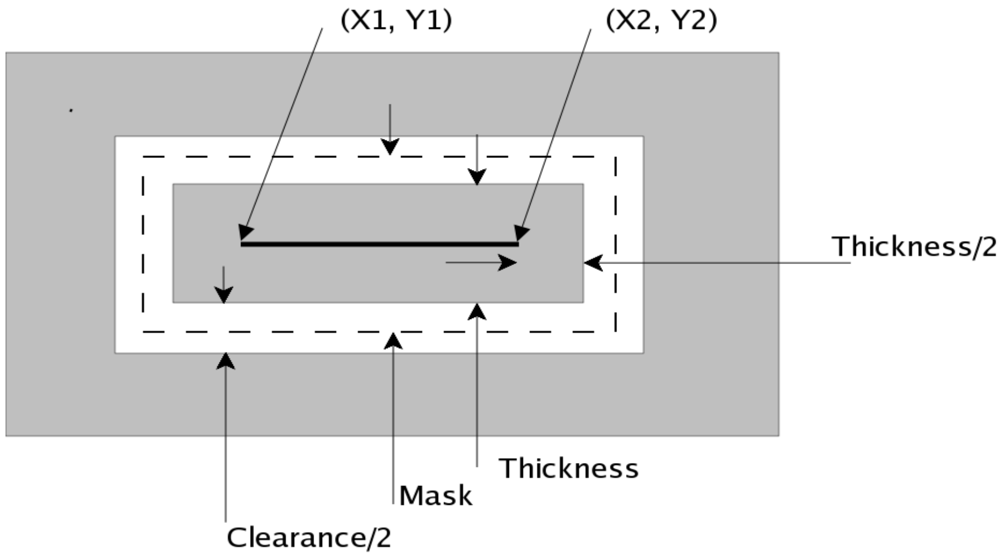


Figure 3: *Explanation of dimensions used in Pad tag.*

Notes

- Pads of zero thickness will not be drawn.

Example

```
Pad[-36284 0 -19881 0 14173 1200 15373 "Positive" "1" "square"]
```

Note that the dimensions held in this example are in **1/100 mil** units because they are held in **square** brackets. Also note that the Pad's tags are called out using strings (e.g. “square”).

Pin

The Pin element is held within the body of a footprint (Element). It defines a single through hole with surrounding metal pad. The Pin tag is usually used to create a footprint for a through-hole part. It can also be used to create a through-hole used for mounting parts to the board, or for mounting the board itself. Note that either mils or 1/100's of a mil are allowable for the Pin tag. This is signaled by the use of round "()" (mils) or square "[]" (1/100 mil) brackets. The distinction between relative and absolute units is determined by the type of brackets used in the enclosing "Element" tag.

Format

The preferred format of this graphical element is given below in **bold**.

```
Pin [rX rY Thickness Clearance Mask Drill "Name" "Number" SFlags]
Pin (rX rY Thickness Clearance Mask Drill "Name" "Number" NFlags)
Pin (aX aY Thickness Drill "Name" "Number" NFlags)
Pin (aX aY Thickness Drill "Name" NFlags)
Pin (aX aY Thickness "Name" NFlags)
```

Detailed description

Item	***Allowed value***	***Explanation***	***Comment***
rX aX	Decimal integer (mils or 1/100 mils)	x coordinate of pin	r = units relative to the mark. a = absolute units.
rY aY	Decimal integer (mils or 1/100 mils)	y coordinate of pin	r = units relative to the mark. a = absolute units.
Thickness	Decimal integer (mils or 1/100 mils)	Outer diameter of copper annulus	
Clearance	Decimal integer (mils or 1/100 mils)	Add to thickness to get clearance diameter	This is separation – not diameter. Also note factor of ½ in definition. See figure.
Mask	Decimal integer (mils or 1/100 mils)	Diameter of solder mask opening	
Drill	Decimal integer (mils or 1/100 mils)	diameter of drill hole	
Name	string (surrounded by double quotes)	Pin name. This is an arbitrary name for the pin.	
Number	string (surrounded by double quotes)	Pin number. This value is used by PCB to attach nets.	

Item	***Allowed value***	***Explanation***	***Comment***
SFlags	string (surrounded by double quotes)	Symbolic flags applying to the pin	
NFlags	unsigned hex value	Numeric (hex) flags applying to the pin	Hex codes described in separate section below.

The various dimensions defining a pin are shown in the illustration below.

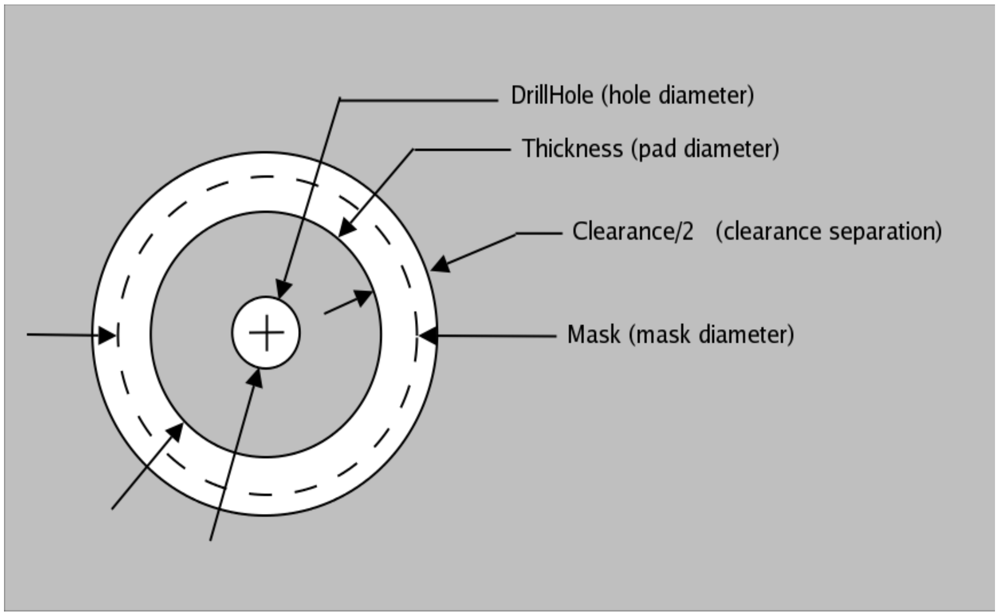


Figure 4: *Definitions of dimensions used in Pin tag.*

Example

```
    Pin[0 0 7000 3000 7000 3500 "1" "1" "square"]
```

Note that the dimensions held in this example are in **1/100 mil** units because they are held in **square** brackets. Also note that the Pin's tags are called out using strings (e.g. “square”).

ElementLine

The ElementLine macro draws line segments on the silk screen layer associated with the layer the device is placed upon (i.e. component or solder side of the board). If the component is placed on the top side of the board, the line is drawn in silkscreen on the top (component) side of the board. If the component is placed on the bottom side of the board, the line is drawn using silkscreen on the bottom (solder) side of the board.

Note that either mils or 1/100's of a mil are allowable for the ElementLine tag. This is signaled by the use of round "()" (mils) or square "[]" (1/100 mil) brackets. The distinction between relative and absolute units is determined by the type of brackets used in the enclosing "Element" tag.

Format

The preferred format of this graphical element is given below in **bold**.

```
ElementLine[rX1 rY1 rX2 rY2 Thickness]
ElementLine(rX1 rY1 rX2 rY2 Thickness)
ElementLine(aX1 aY1 aX2 rY2 Thickness)
```

Detailed description

Item	*Allowed value*	*Explanation*	*Comment*
rX1 aX1	Decimal integer (mils or 1/100 mils)	X coord of first point of segment. See figure.	r = units relative to the mark. a = absolute units.
rY1 aY1	Decimal integer (mils or 1/100 mils)	Y coord of first point of segment. See figure	r = units relative to the mark. a = absolute units.
rX2 aX2	Decimal integer (mils or 1/100 mils)	X coord of second point of segment. See figure	r = units relative to the mark. a = absolute units.
rY2 aY2	Decimal integer (mils or 1/100 mils)	Y coord of second point of segment. See figure	r = units relative to the mark. a = absolute units.
Thickness	decimal integer (mils or 1/100 mils)	Thickness of line segment on silkscreen layer.	

Example

```
ElementLine [-16000 -39100 59200 -39100 1000]
```

Note that the dimensions used in this example are in **1/100 mil** units because they are held in **square** brackets.

ElementArc

An ElementArc is usually used to draw a circle or oval on the silkscreen layer. It can also be used to draw an arc (i.e. incomplete circle) on the silkscreen layer. If the component is placed on the top side of the board, the circle or oval is drawn in silkscreen on the top (component) side of the board. If the component is placed on the bottom side of the board, the circle or oval is drawn in silkscreen on the bottom (solder) side of the board.

Note that either mils or 1/100's of a mil are allowable for the ElementLine tag. This is signaled by the use of round "()" (mils) or square "[]" (1/100 mil) brackets. The distinction between relative and absolute units is determined by the type of brackets used in the enclosing "Element" tag.

Format

The preferred format of this graphical element is given below in **bold**.

```
ElementArc [rX rY Width Height StartAngle DeltaAngle Thickness]
ElementArc (rX rY Width Height StartAngle DeltaAngle Thickness)
ElementArc (aX aY Width Height StartAngle DeltaAngle Thickness)
```

Detailed description

Item	*Allowed value*	*Explanation*	*Comment*
rX aX	Decimal integer (mils or 1/100 mils)	x coordinate of arc's center point.	r = units relative to the mark. a = absolute units.
rY aY	Decimal integer (mils or 1/100 mils)	y coordinate of arc's center point.	r = units relative to the mark. a = absolute units.
Width	Decimal integer (mils or 1/100 mils)	The width from the center to the edge. The bounds of the circle of which this arc is a segment, is thus 2*Width by 2*Height.	For circle, use Width = Height. For oval, Width will be different from Height.
Height	Decimal integer (mils or 1/100 mils)	The height from the center to the edge. The bounds of the circle of which this arc is a segment, is thus 2*Width by 2*Height.	For circle, use Width = Height. For oval, Width will be different from Height.
StartAngle	Decimal integer between 0 and 360 degrees.	The angle of one end of the arc, in degrees. In PCB, an angle of zero points left (negative X direction), and 90 degrees points down (positive Y direction).	

Item	*Allowed value*	*Explanation*	*Comment*
DeltaAngle	Decimal integer between 0 and 360 degrees.	The sweep of the arc. This may be negative. Positive angles sweep counterclockwise.	Usually use 360 for full circle or oval. Incomplete circles – i.e. arcs -- are also possible.
Thickness		Thickness of line segment on silkscreen layer.	

Example

```
ElementArc[0 0 9850 9850 210 300 1000]
```

Note that the dimensions held in this example are in **1/100 mil** units because they are held in **square** brackets.

Important Flags

Two types of flags are supported for use in footprint files: A legacy hex flag, and a newer string flag:

- Hex flags: The older hex flags capture modifiers to the graphical element as individual bits in the “NFlags” field. The flag bits are “or'ed” into a single hex value which is incorporated into the pad or pin declaration. For example: 0x0101 is a square pin.
- String flags: The new string flags capture modifiers to the graphical element as a comma-separated string list in the “SField”. For example: “showname,square” defines a square pin/pad with name visible. Do not place spaces or whitespace between the names!

Detailed description

Mnenomic ***(#define in PCB code)***	***Hex value***	***String value***	***Explanation***	***Comment***
NOFLAG	0x0000	“”	NULL value or empty string.	
PINFLAG	0x0001	“pin”	This is a pin	Set by PCB, not by user.
VIAFLAG	0x0002	“via”	This is a via	Set by PCB, not by user.
HOLEFLAG	0x0008	“hole”	This pin or via is only a hole.	
DISPLAYNAMEFLAG	0x0020	“showname”	Display the names of pins/pads	(Name contained in “pin/pad” tag)
ONSOLDERFLAG	0x0080	“onsolder”	Place this pad on solder side	Use to flag pads on opposite side of board from other graphical elements. Useful for placing copper on opposite side of board (e.g. for defining edge connectors).
SQUAREFLAG	0x0100	“square”	Pin is square, not round.	
OCTAGONFLAG	0x0800	“octagon”	Draw pin or via as octagon instead of round.	

Mnenomic ***(#define in PCB code)***	***Hex value***	***String value***	***Explanation***	***Comment***
EDGE2FLAG	0x4000	“edge2”	For pads, indicates that the second point is closer to the edge. For pins, indicates that the pin is closer to a horizontal edge and thus pinout text should be vertical.	

Glossary

- **Component side –** The top side of the PCB is traditionally called the "component side" since that's where the electronic components were mounted back in the old days of one-sided boards.
- **Footprint** – The pattern of metal, silkscreen, soldermask relief, and drills which defines where you place a component on a circuit board. Footprints are the placed by the user onto the PC board during the "component placement" phase of PCB layout.
- **Gang Solder Mask Window** – A solder mask window large enough to cover more then one pad. Traces not part of the net could become accidentally soldered to a nearby pad.

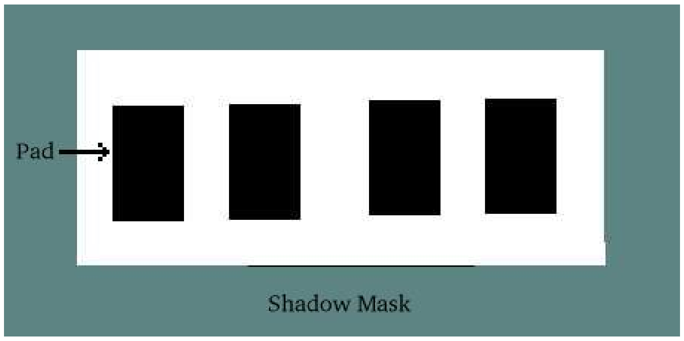


Figure 5: *A solder mask window for four pads. Note that the pads are "ganged" -- there is no solder mask between the individual pads. This part must be carefully hand-soldered so that no solder bridges develop between the pads.*

- **Grid placement courtyard** – A grid placement courtyard is a placement courtyard which assumes that all footprint mark positions lie on a grid. The requirement that all components lie on a grid is driven by the needs of automated board assembly machines. If you are hand assembling your board, then you don't need to worry about this.

- **Gerber file** – The file format used in the industry to convey a board database to the manufacturer is RS-274-X (which replaces the now obsolete RS-274-D format). This file format was originally developed by the Gerber company for their photo plotters and thus RS-274-D and RS-274-X format files are often times referred to as "Gerber" files. PCB exports RS-274-X only.

- **Land pattern** – synonym for "footprint".

- **Placement courtyard** – This is the total area encompassed by a footprint. The placement courtyard's primary purpose is to give you a guideline for placing footprints next to each other while providing enough spacing between them to compensate for component tolerances. Courtyards are used to ensure that your parts will fit into the allotted space, but they do not compensate for assembly machine heads or other manufacturing tolerances. One component's placement courtyard is not meant to overlap with the next one's.

- **Pocket Solder Mask Window** – A window in the solder mask which covers a single pad (see figure below). This requires greater tolerances in creating the solder mask. This may be required in order to run traces between the pads.

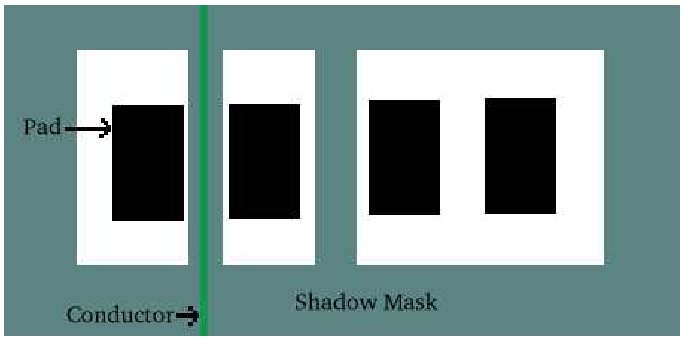


Figure 6: *Pocket solder mask window. Here, the two pads on the left are individually windowed. This prevents solder from bridging between the two pads, but requires higher fabrication tolerances to ensure that solder mask does not accidentally impinge on a pad.*

- **Solder Mask** – Is a coating applied over the surface of the PCB which prevents the covered area from being soldered to. Usually only component pads and pin holes are left exposed. Traces left exposed can be inadvertently soldered to.

- **Solder side –** The bottom side of the PCB is traditionally called the "solder side" since that's where the component leads were soldered to the PCB's traces back in the old days of one-sided boards.
- **Thermal/Thermal relief** – A thermal relief is a way of connecting a pin to a ground or power plane. Instead of directly connecting to the plane, small "spokes" are used to increase the thermal resistance between the pin and the plane. Often times these connections are referred to as simply a thermal. By increasing the thermal resistance to the plane, it becomes easier to solder to the pin. The figure below shows an example thermal relief.

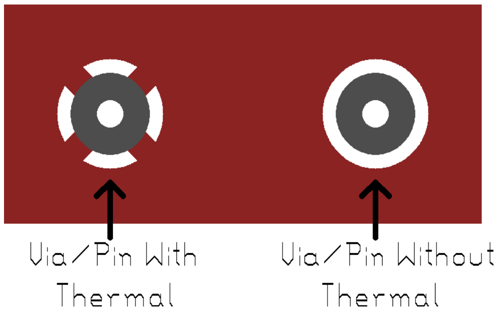


Figure 7: *The hole on the left has a thermal making a connection to a plane. The hole on the right does not connect to the plane at all.*