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.

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 to read 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 (some say archaic); it relies upon using the GNU macro language M4 to generate footprints on the fly. 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's internal workings are obscure, and many of the footprints are unchecked and are possibly wrong (e.g. wrong pin hole sizes). 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 this procedure 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 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, the following is a simple PCB land pattern for an 0603 chip resistor.

```
Element(0x00 "Surface Mount Chip Resistor 0603" "R0" "" 0 0
-31 -82 2 100 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)
)
```

Within the parentheses after “Element” is information pertaining to the entire footprint, such as its position, its refdes, and so on. Following the top-level information, you can see two “Pad” graphical elements. This example 0603 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 four “ElementLine” elements drawn on the silkscreen layer, corresponding to a box around the 0603 footprint. The information within the “ElementLine” line specifies the width and position of the lines which are 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, checkered 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 (1e-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 confusing dichotomy, and always check how your units are specified!**

- **Footprint libraries**. The original PCB footprint library was written using the macro language M4. Common opinion about the M4 footprint library holds that it might have been a clever idea back when the Amiga was considered a cutting-edge machine, but it is now an obscure and hard-to-support boat-anchor. Fortunately, 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. **The newlib library is the footprint library syntax described in this document.**

- **PCB's Solder Mask Relief Implementation.** PCB only allows for the PADS to determine the solder mask relief size and shape. Therefore creating Gang shadow masks 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 (center) of the footprint. This is often either the center of the part, or is pin 1.
2. Determine the rotation of the device around the mark. In PCB, this may be 0, 90, 180, or 270 degrees.
3. Determine the Grid placement courtyard and its relationship to the center. This 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 requirements upon the pad dimensions, as well as the solder mask clearances. In particular, make sure you understand how your mask should be applied. 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. 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.
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(s) held within it. One or more footprints (Elements) may live within a single file. No naming convention is enforced by PCB. Therefore, no file name suffix is required, although you may want to devise your own naming convention. Examples might be "Res_0805_large.fpt" or "TQFP-44.pcbfootprint".
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 lines to create the pattern outline. ElementLine will produce a line on the silkscreen layer. The pattern outline needn't encircle the grid placement courtyard but doing so can be convenient for correct placement.

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 Illustration 1 below.

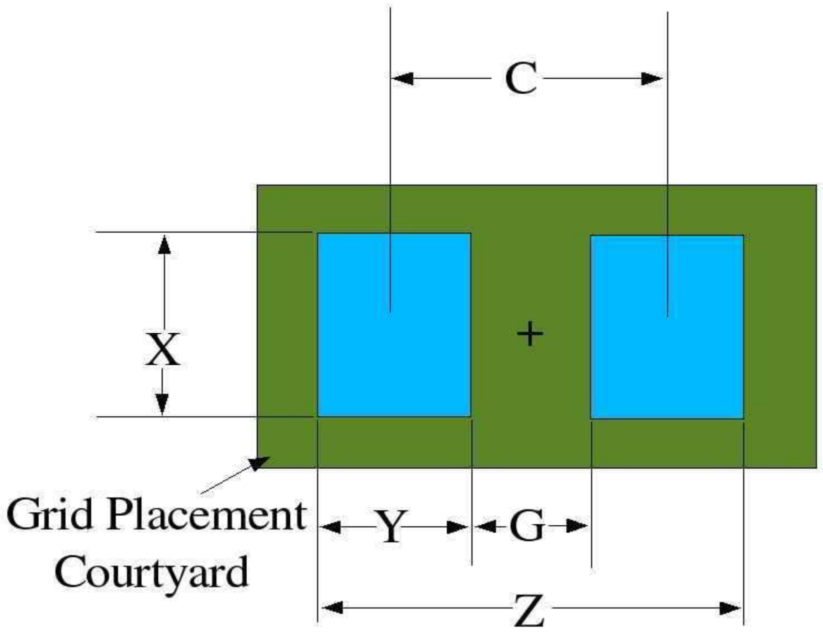


Illustration 1Footprint (land pattern) of SMT resistor.

Data for different resistor sizes are presented in the 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"

Each PCB Pad impacts several layers. If the Pad is on the component surface it impacts the component layer, the component mask (component side solder mask relief) and the component paste. If the component layer has a polygon then the polygon is cleared away from the pad by an amount entered in the Pad macro.

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
land patterns.

Footprint creation do's and dont's

- 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 a few mils to ensure that everything will fit together.
    - Are your pads large enough? 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 – Solder Mask contamination to component pads can cause failures. Insufficient solder mask leaving exposed coper can cause solder to make unintentional connections.

Element

The “Element” tag holds an entire 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. For example, each pad or pin for the device needs a pad or a pin hole. Generally a silkscreen outline is also provided. The body is the code with in the parentheses.

Format

```
        Element  (element_flags,  description,  pcb-name,  value,
                 mark_x,  mark_y,  text_x,  text_y,
                 text_direction,  text_scale,  text_flags)
        (
                 individual graphical components, such as Pad, Pin,
                 or ElementLine.
        )

        Element  [element_flags,  description,  pcb-name,  value,
                 mark_x,  mark_y,  text_x,  text_y,
                 text_direction,  text_scale,  text_flags]
        (
                 individual graphical components, such as Pad, Pin,
                 or ElementLine.
        )
```

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 “()” or square “[]” brackets.

Detailed description

Item	***Allowed value***	***Explanation***	***Comment***
element_flags	unsigned hex value		
description	string	Text description of footprint	Entered by user.
pcb- name	string	Refdes used on this particular PCB	This field is filled out by PCB itself. Leave blank when you define the footprint file.
value	string	value of component on this particular PCB	This field is filled out by PCB itself. Leave blank when you define the footprint file.

Item	***Allowed value***	***Explanation***	***Comment***
mark_x	Decimal integer (mils or 1/100 mils)	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.	Usually set to 10 mil so part's initial position is on working area of board.
mark_y	Decimal integer (mils or 1/100 mils)	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.	Usually set to 10 mil so part's initial position is on working area of board.
text_x	Decimal integer (mils or 1/100 mils)	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.
text_y	Decimal integer (mils or 1/100 mils)	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.
text_direction	decimal integer	0 = horizontal 1 = CCW 90 deg 2 = 180 deg 3 = CW 90 deg	
text_scale	decimal integer		Usually set to 100.
text_flags	unsigned hex value		

Example

```
        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.

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.

Format

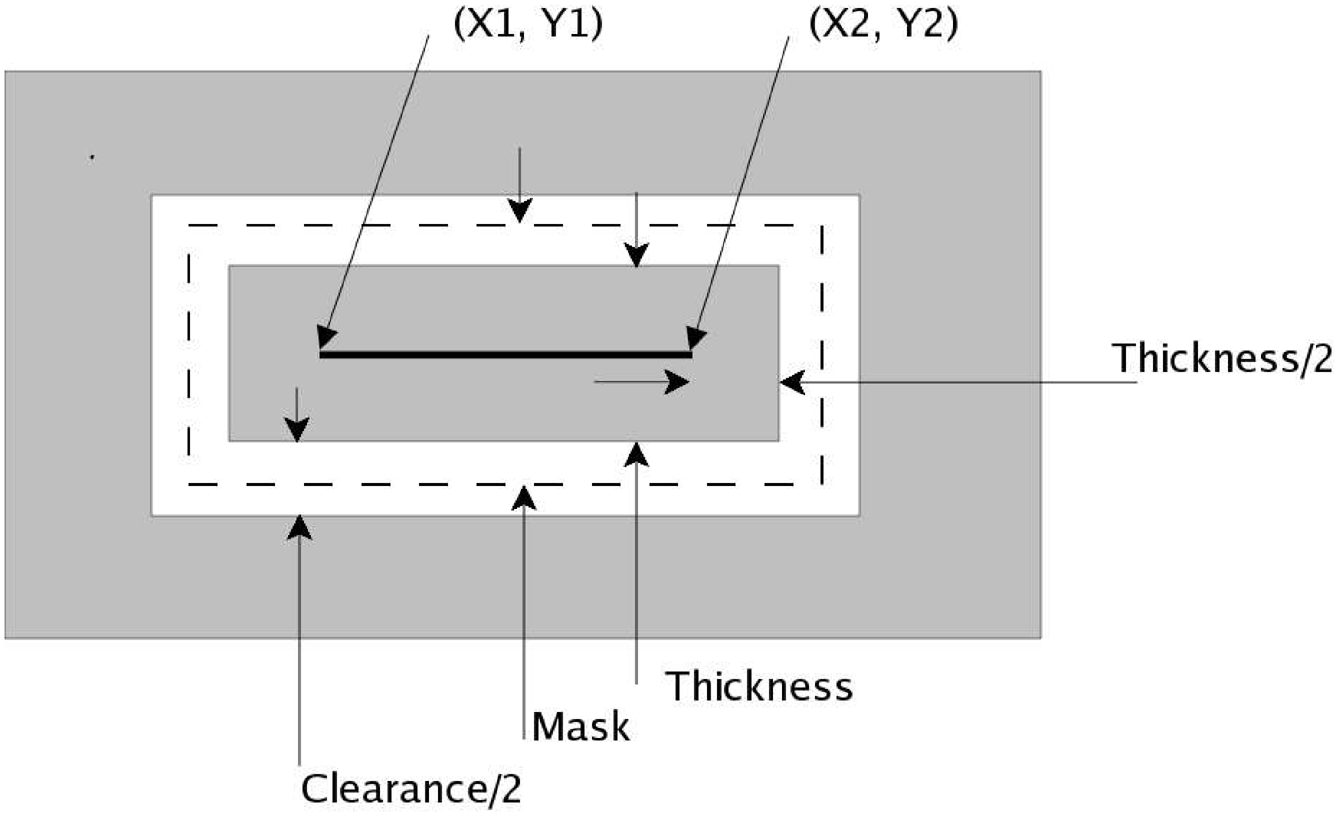
```
Pad (x1 y1 x2 y2 thickness clearance mask name  pad_number
flags)

Pad [x1 y1 x2 y2 thickness clearance mask name pad_number
flags]
```

Detailed description

Item	***Allowed value***	***Explanation***	***Comment***
x1	Decimal integer (mils or 1/100 mils)	X coord of first point of line segment. See figure	
y1	Decimal integer (mils or 1/100 mils)	Y coord of first point of line segment. See figure	
x2	Decimal integer (mils or 1/100 mils)	X coord of second point of line segment. See figure	
y2	Decimal integer (mils or 1/100 mils)	Y coord of second point of line segment. See figure	
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.
name	string	Name of pad. Arbitrary identification string.	Can be e.g. pad_1, plus, or any other string.

Item	***Allowed value***	***Explanation***	***Comment***
pad_number	string	Pad number. Used in attaching rats, so it must be consistent with the definition in the netlist.	
flags	hex value	Defined in “Flag” 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.



Notes

- In the PCB development release 1.99o the entered points (x1,y1) and (x2, y2) are re-arranged such that x1 is the smaller of x1 and x2. Similarly y1 becomes the smaller of y1 and y2.
- Pads of zero thickness will not be drawn.

Example 1

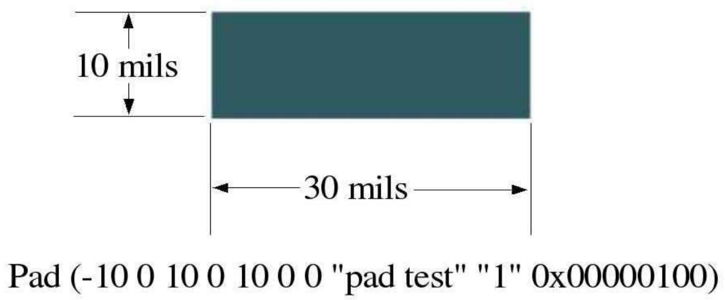
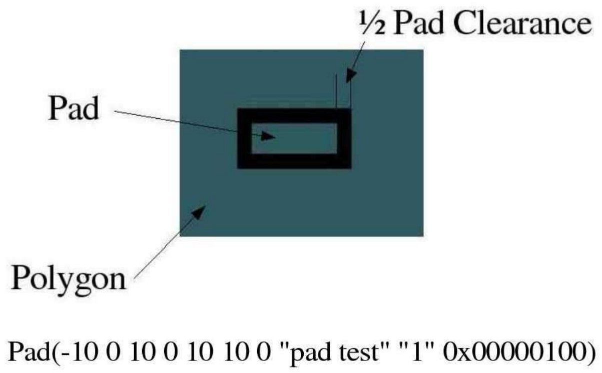


Illustration 2Pad example 1

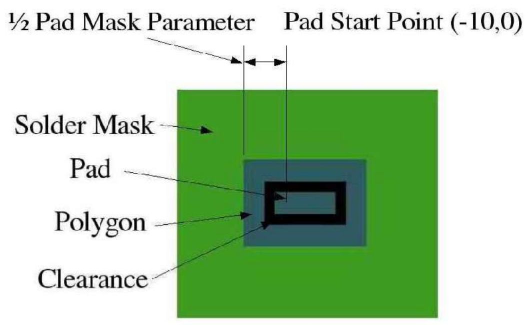
This pad was created along a line 20 mil long which is oriented along the x axis. The completed pad became 10 mils longer do to the thickness parameter. The thickness parameter also made the pad 10 mils wide along the y axis. In order to make a pad a particular length you need to subtract the thickness parameter from the start and end points.

Example 2



Clearance is the area that is cleared from any polygon that the pad is placed within.

Example 3



```
Pad(-10 0 10 0 10 10 40 "pad test" "1" 0x00000100)
```

It is important to note that like the pad metalization itself, the solder mask relief is located with respect to the line segment that the pad is located upon.

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 macro 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.

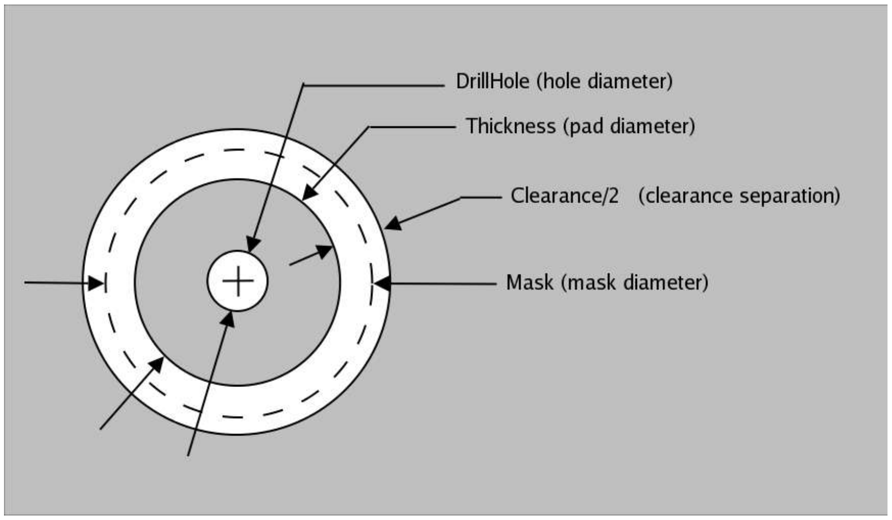
Format

```
Pin(x y Thickness Clearance Mask DrillHole Name  Number
Flags)

Pin[x y Thickness Clearance Mask DrillHole Name  Number
Flags]
```

Detailed description

Item	***Allowed value***	***Explanation***	***Comment***
x	Decimal integer (mils or 1/100 mils)	x coordinate of pin	
y	Decimal integer (mils or 1/100 mils)	y coordinate of pin	
Thickness	Decimal integer (mils or 1/100 mils)	surrounding metalization diameter	
Clearance	Decimal integer (mils or 1/100 mils)	separation of metal from other conductors on any layer	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 relief.	
DrillHole	Decimal integer (mils or 1/100 mils)	diameter of drill hole	
Name	string	Pin name. This is an arbitrary name for the pin.	
Number	decimal integer	Pin number. This value is used by PCB to attach nets.	
Flags	hex value	Defined in “Flag” section below.	

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



Example

```
        Pin[5400 -11200  8000 2000 9000 4300 "Pin_6" "6" 0x02004001]
```

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

ElementLine

The ElementLine macro draws line segments on the silk screen layer associated with the layer the device is placed upon (component or solder).

Format

```
ElementLine(x1 y1 x2 y2 Thickness)

ElementLine[x1 y1 x2 y2 Thickness]
```

Detailed description

Item	***Allowed value***	***Explanation***	***Comment***
x1	decimal integer (mils or 1/100 mils)	x coordinate of segment start point	
y1	decimal integer (mils or 1/100 mils)	y coordinate of segment start point	
x2	decimal integer (mils or 1/100 mils)	x coordinate of segment end point	
y2	decimal integer (mils or 1/100 mils)	y coordinate of segment end point	
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 placed 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 on the bottom (solder) side of the board.

Format

```
ElementArc(x y Width Height StartAngle Delta Thickness)

ElementArc[x y Width Height StartAngle Delta Thickness]
```

Detailed description

Item	***Allowed value***	***Explanation***	***Comment***
x	Decimal integer (mils or 1/100 mils)	X center position of circle or oval.	
y	Decimal integer (mils or 1/100 mils)	Y center position of circle or oval.	
Width	Decimal integer (mils or 1/100 mils)	This is horizontal width of circle or oval.	For circle, use Width = Height. For oval, Width will be different from Height.
Height	Decimal integer (mils or 1/100 mils)	vertical height of circle or oval.	
StartAngle	Decimal integer between 0 and 360 degrees.	Starting angle of arc – measured in degrees clockwise from negative X axis (i.e. degrees clockwise from horizontal ray pointing to the left.)	
Delta	Decimal integer between 0 and 360 degrees.	Angle swept out by arc in degrees. Direction of sweep is clockwise (CW).	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 (350 -410 50 50 180 90 10)
```

Note that the dimensions held in this example are in **mil** units because they are held in

round brackets.

Important Flags

Certain features of the pins and pads making up a footprint are captured as individual bits in the "Flag" field. The flag bits are "or'ed" into a single hex value which is incorporated into the pad or pin declaration. This section lists a few flags which are important for creating footprints. The entire list of flags can be found in the source code under ~pcb/src/const.h.

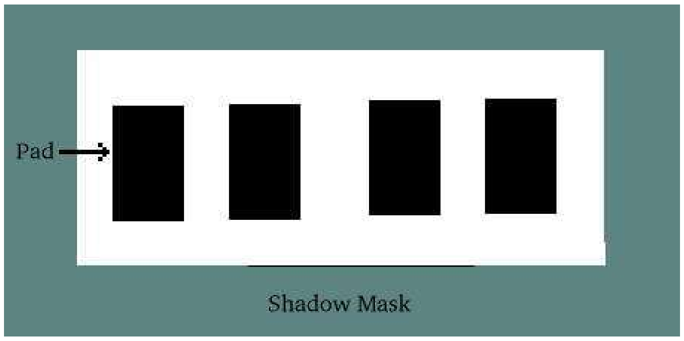
Detailed description

Mnenomic	***Hex value***	***Explanation***	***Comment***
NOFLAG	0x0000	NULL value	
PINFLAG	0x0001	This is a pin	
VIAFLAG	0x0002	This is a via	
HOLEFLAG	0x0008	This pin or via is only a hole.	
DISPLAYNAMEFLAG	0x0020	Display the names of pins/pads	
SQUAREFLAG	0x0100	Pin is square, not round.	
USETHERMALFLAG	0x0400	Draw pin or via with thermal fingers.	
OCTAGONFLAG	0x0800	Draw pin or via as octagon instead of round.	

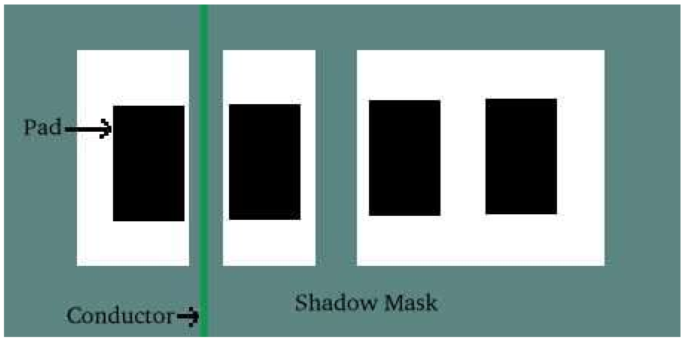
Glossary

- **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.

- **Gang Solder Mask Window** – A window large enough to cover more then one pad. Traces not part of the net could become soldered to a near by pad.



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


Pad
Conductor
Shadow Mask