Output Formats

The output format is specified with the **-T***lang* flag on the command line, where *lang* is one of the parameters listed below.

The formats actually available in a given Graphviz system depend on how the system was built and the presence of additional libraries. To see what formats **dot** supports, run `dot -T?`. See the description of the -T flag for additional information.

Note that the internal coordinate system has the origin in the lower left corner. Thus, positions in the canon, dot, xdot, plain, and plain-ext formats need to be interpreted in this manner.

Command-line parameter	Format
bmp	Windows Bitmap Format
canon dot xdot	DOT
cmap	Client-side imagemap (deprecated)
dia	Dia format
eps	Encapsulated PostScript
fig	FIG
gd gd2	GD/GD2 formats
gif	GIF
gtk	GTK canvas
hpgl	HP-GL/2
ico	Icon Image File Format
imap cmapx	Server-side and client-side imagemaps
imap_np cmapx_np	Server-side and client-side imagemaps
ismap	Server-side imagemap (deprecated)
jpg jpeg jpe	JPEG
mif	FrameMaker MIF format
mp	MetaPost
pcl	PCL
pdf	Portable Document Format (PDF)
pic	PIC
plain plain-ext	Simple text format
png	Portable Network Graphics format
ps	PostScript
ps2	PostScript for PDF

svg svgz	Scalable Vector Graphics
tga	Truevision Targa Format (TGA)
tif tiff	TIFF (Tag Image File Format)
vml vmlz	Vector Markup Language (VML)
vrml	VRML
vtx	Visual Thought format
wbmp	Wireless BitMap format
xlib	Xlib canvas

Format Descriptions

bmp

Outputs images in the Windows BMP format.

canon ,
dot ,
xdot

These formats produce output in the dot language. Using **canon** produces a prettyprinted version of the input, with no layout performed.

The **dot** option corresponds to attributed dot output, and is the default output format. It reproduces the input, along with layout information for the graph. In particular, a bb attribute is attached to the graph, specifying the bounding box of the drawing. If the graph has a label, its position is specified by the lp attribute.

Each node gets pos, width and height attributes. If the node is a record, the record rectangles are given in the rects attribute. If the node is a polygon and the vertices attribute is defined, this attribute contains the vertices of the node.

Every edge is assigned a pos attribute, and if the edge has a label, the label position is given in lp.

The **xdot** format extends the **dot** format by providing much more detailed information about how graph components are drawn. It relies on additional attributes for nodes, edges and graphs.

The format is preliminary; comments and suggestions for better representations are welcome. To allow for changes in the format, Graphviz attaches the attribute `xdotversion` to the graph.

Additional drawing attributes can appear on nodes, edges, clusters and on the graph itself. There are six new attributes:

draw	Drawing operations	
ldraw	Label drawing	
hdraw	Head arrowhead	Edge only
tdraw	Tail arrowhead	Edge only
hldraw	Head label	Edge only
tldraw	Tail label	Edge only

For a given graph object, one will typically a draw directive before the label directive. For example, for a node, one would first use the commands in **_draw_** followed by the commands in **_ldraw_**.

The value of these attributes consists of the concatenation of some (multi-)set of the following 13 rendering or attribute operations. (The number is parentheses gives the xdot version when the operation was added to the format. If no version number is given, the operation was in the original specification.)

E x_0 y_0 w h	Filled ellipse $((x-x_0)/w)^2 + ((y-y_0)/h)^2 = 1$
e x_0 y_0 w h	Unfilled ellipse $((x-x_0)/w)^2 + ((y-y_0)/h)^2 = 1$
P n x_1 y_1 ... x_n y_n	Filled polygon using the given n points
p n x_1 y_1 ... x_n y_n	Unfilled polygon using the given n points
L n x_1 y_1 ... x_n y_n	Polyline using the given n points
B n x_1 y_1 ... x_n y_n	B-spline using the given n control points
b n x_1 y_1 ... x_n y_n	Filled B-spline using the given n control points (1.1)
T x y j w n -$b_1b_2...b_n$	Text drawn using the baseline point (x,y). The text consists of the n bytes following '-'. The text should be left-aligned (centered, right-aligned) on the point if j is -1 (0, 1), respectively. The value w gives the width of the text as computed by the library.
C n -$b_1b_2...b_n$	Set fill color. The color value consists of the n bytes following '-'. (1.1)
c n -$b_1b_2...b_n$	Set pen color. The color value consists of the n bytes following '-'. (1.1)
F s n -$b_1b_2...b_n$	Set font. The font size is s points. The font name consists of the n bytes following '-'. (1.1)
S n -$b_1b_2...b_n$	Set style attribute. The style value consists of the n bytes following '-'. The syntax of the value is the same as specified for a **styleItem** in style. (1.1)
I x y w h n -$b_1b_2...b_n$	Externally-specified image drawn in the box with lower left corner (x,y) and upper right corner (x+w,y+h). The name of the image consists of the n bytes following '-'. This is usually a bitmap image. Note that the image size, even when converted from pixels to points, might be different from the required size (w,h). It is assumed the renderer will perform the necessary scaling. (1.2)

Note that the filled figures (ellipses, polygons and B-Splines) imply two operations: first, drawing the filled figure with the current fill color; second, drawing an unfilled figure with the current pen color, pen width and pen style. the

Style values which can be incorporated in the graphics model do not appear in xdot output. In particular, the style values `filled`, `rounded`, `diagonals`, and `invis` will not appear. Indeed, if style contains `invis`, there will not be any xdot output at all.

In handling text alignment, the application may want to recompute the string width using its own rendering primitives.

The text operation is only used in the label attributes. Normally, the non-text operations are only used in the non-label attributes. If, however, the decorate attribute is set on an edge, its label attribute will also contain a polyline operation. In addition, if a label is a complex, HTML-like label, it will also contain non-text operations.

All coordinates and sizes are in points. Note though that if an edge or node is invisible, no drawing operations are attached to it.

Version info:

Xdot version	Graphviz version
1.0	1.9
1.1	2.8
1.2	2.13

cmap

Produces map files for client-side image maps. The cmap format is mostly identical to cmapx, but the latter is well-formed XML amenable to processing by XML tools. In particular, the cmapx output is wrapped in <map></map>.

See Note.

dia

Produces Dia output.

eps

Produces Encapsulated PostScript output. At present, this is only guaranteed to be correct for a single input graph since the Bounding Box information has to appear at the beginning of the output, and this will be based on the first graph.

fig

Outputs graphs in the FIG graphics language.

gd ,
gd2

Output images in the GD and GD2 format. These are the internal formats used by the gd library. The latter is compressed.

gif

Outputs GIF bitmap images.

gtk

Creates a GTK window and displays the output there.

hpgl

Produces output in the HP-GL/2 vector graphic printer language.

ico

Outputs images in the Windows ICO format.

imap ,
cmapx

Produces map files for server-side and client-side image maps, These can be used in a web page with a graphical form of the output, e.g. in JPEG or GIF format, to attach links to nodes and edges. For example, to create a server-side map given the dot file

```
/* x.dot */
digraph mainmap {
  URL="http://www.research.att.com/base.html";
  command [URL="http://www.research.att.com/command.html"];
  command -> output [URL="colors.html"];
}
```

one would process the graph and generate two output files:

```
dot -Timap -ox.map -Tgif -ox.gif x.dot
```

and then refer to it in a web page:

```
<A HREF="x.map"><IMG SRC="x.gif" ismap="ismap" /></A>
```

For client-side maps, one again generates two output files:

```
dot -Tcmapx -ox.map -Tgif -ox.gif x.dot
```

and uses the HTML

```
<IMG SRC="x.gif" USEMAP="#mainmap" />
... [content of x.map] ...
```

URLs can be attached to the root graph, nodes and edges. If a node has a URL, clicking in the node will activate the link. If an edge has a URL, various points along the edge (but not necessarily the head or tail) will link to it. In addition, if the edge has a label, that will link to the URL. As for the head of the edge, this is linked to the headURL, if set. Otherwise, it is linked to the edge's URL if that is defined. The analogous description holds for the tail and the tailURL. A URL associated with the graph is used as a default link.

If the URL of a node contains the escape sequence "\N", it will be replaced by the node's name. If the headURL is defined and contains the escape sequence "\N", it will be replaced by the headlabel, if defined. The analogous result holds for the tailURL and the taillabel.

See Note.

imap_np ,
cmapx_np

These are identical to the imap and cmapx formats, except they rely solely on rectangles as active areas.

ismap

Produces HTML image map files. This is a predecessor (circa 1994) of the IMAP format. Most servers now use the latter. URLs can be attached to the root graph, nodes and edges. Since edge links are attached to edge labels, an edge must have a label for its URL to be used. For both nodes and edges, if the URL has the escape sequence "\N" embedded in its string, this will be replaced with the node or edge name.

jpg ,
jpeg ,
jpe

Output JPEG compressed image files.

mif

Generates Frame Maker MIF files.

mp

Produces MetaPost output.

pcl

Produces output in the PCL printer language. HP-GL is a subset of PCL, so that PCL output is the same as HP-GL, wrapped with some initial and final commands to set the printer to and from HP-GL mode.

pdf

Produces PDF output. (This option assumes Graphviz includes the Cairo renderer.) Alternatively, one can use the ps2 option to produce PDF-compatible PostScript, and then use a ps-to-pdf converter.

Note: At present, this option does not support anchors, etc. To get these included in your PDF output, use ps2.

pic

Outputs in PIC, the picture description language in the troff-family

plain ,
plain-ext

The plain and plain-ext formats produce output using a simple, line-based language. The latter format differs in that, on edges, it provides port names on head and tail nodes when applicable.

There are four types of statements.

```
graph scale width height
node name x y width height label style shape color fillcolor
edge tail head n x₁ y₁ .. xₙ yₙ [label xl yl] style color
stop
```

graph

The *width* and *height* values give the width and height of the drawing. The lower left corner of the drawing is at the origin. The *scale* value indicates how the drawing should be scaled if a size attribute was given and the drawing needs to be scaled to conform to that size. If no scaling is necessary, it will be set to 1.0. Note that all graph, node and edge coordinates and lengths are given unscaled.

node

The *name* value is the name of the node, and *x* and *y* give the node's position. The *width* and *height* are the width and height of the node. The *label*, *style*, *shape*, *color* and *fillcolor* give the node's label, style, shape, color and fillcolor, respectively, using attribute default values where necessary. If the node does not have a style attribute, "solid" is used.

edge

The *tail* and *head* values give the names of the head and tail nodes. In plain-ext format, the head or tail name will be appended with a colon and a portname if the edge connects to the node at a port. *n* is the number of control points defining the B-spline forming the edge. This is followed by 2**n* numbers giving the x and y coordinates of the control points in order from tail to head. If the edge has a label, this comes next followed by the x and y coordinates of the label's position. The edge description is completed by the edge's style and color. As with nodes, if a style is not defined, "solid" is used.

Note: The control points given in an edge statement define the body of the edge. In particular, if the edge has an arrowhead to the head or tail node, there will be a gap between the last or first control points and the boundary of the associated node. There are at least 3 possible ways of handling this gap:

- Arrange that the input graph uses `dir=none`, `arrowhead=none`, or `arrowtail=none` for all edges. In this case, the terminating control points will always touch the node.
- Consider the line segment joining the control point and the center of the node, and determine the point where the segment intersects the node's boundary. Then use the control point and the intersection point as the main axis of an arrowhead. The problem with this approach is that, if the edge has a port, the edge will not be pointing to the center of the node. In this case, rather than use the control point and center point, one can use the control point and its tangent.
- Arrange that the input graph uses `headclip=false` or `tailclip=false`. In this case, the edge will terminate at the node's center rather than its boundary. If arrowheads are used, there will still be a gap, but normally this will occur within the node. The application will still need to clip the spline to the node boundary. Also, as with the previous item, if the edge points to a node port, this technique will fail.

The output consists of one **graph** line, a sequence of **node** lines, one per node, a sequence of **edge** lines, one per edge, and a final **stop** line. All units are in inches, represented by a floating point number.

Note that the plain formats provide minimal information, really giving not much more than node positions and sizes, and edge spline control points. These formats are usually most useful to applications wanting just this geometric information, and willing to fill in all of the graphical details. The only real advantages to these formats is their terseness and their ease of parsing. In general, the dot and xdot are preferable in terms of the quantity of information provided.

png

Produces output in the PNG (Portable Network Graphics) format.

ps

Produces PostScript output.

Note: The default PostScript renderer can only handle the Latin-1 character set. To get non-Latin-1 characters into PostScript output, use `-Tps:cairo`, assuming your version was built with the Cairo renderer.

ps2

Produces PostScript output with PDF notations. It is assumed the output will be directly converted into PDF format. The notations include PDF bounding box information, so that the resulting PDF file can be correctly used with pdf tools, such as **pdflatex**. In addition, if a node has a URL attribute, this gets translated into PDF code such that the node, when viewed in a PDF-viewer, e.g., **acroread**, is a link to the given URL. If a URL is attached to the graph, this serves as a base, such that relative URLs on nodes are derived from it.

svg ,
svgz

Produce SVG output, the latter in compressed format.

See Note.

tga

Produces Targa output.

tif ,
tiff

Produces TIFF output.

vml ,
vmlz

Produces VML output, the latter in compressed format.

See Note.

vrml

Outputs graphs in the VRML format. To get a 3D embedding, nodes must have a z attribute. These can either be supplied as part of the input graph, or be generated by neato provided dim`=3` and at least one node has a **z** value.

Line segments are drawn as cylinders. In general, VRML output relies on having the PNG library to produce images used to texture-fill the node shapes. However, if shape`=point`, a node is drawn as a 3D sphere.

vtx

Generates graph diagrams in the format for Confluents's Visual Thought.

wbmp

Produces output in the Wireless BitMap (WBMP) format, optimized for mobile computing.

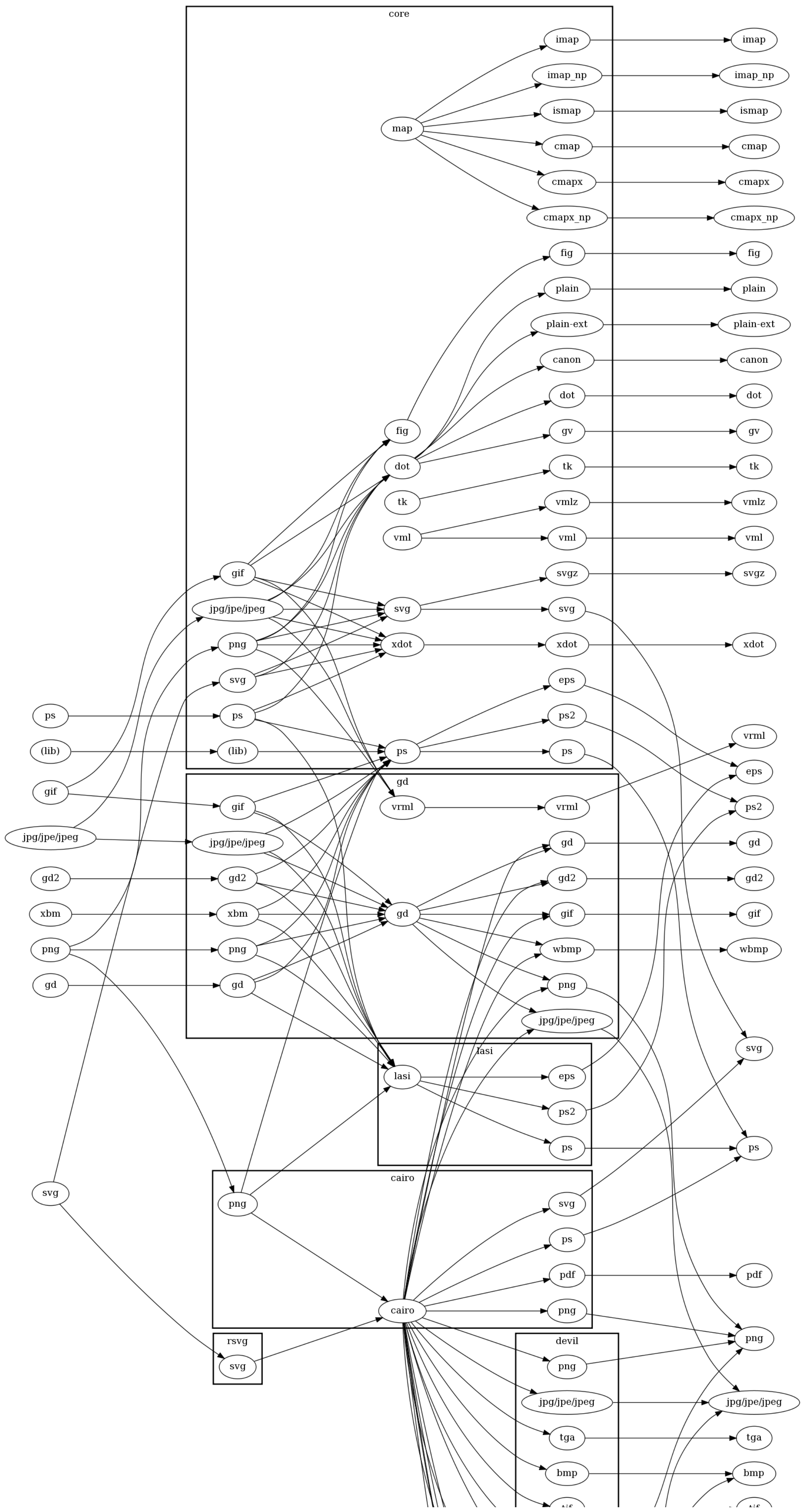
xlib

Creates an Xlib window and displays the output there.

Image Formats

The image and shapefile attributes specify an image file to be included as part of the final diagram. Not all image formats can be read. In addition, even if read, not all image formats can necessarily be used in a given output format.

The graph below shows what image formats can be used in which output formats, and the required plugins. On the left are the supported image formats. On the right are the supported output formats. In the middle are the plugins: image loaders, renderers, drivers, arranged by plugin library. This presents the most general case. A given installation may not provide one of the plugins, in which case, that transformation is not possible.


core
imap
imap_np
ismap
cmap
cmapx
cmapx_np
map
fig
plain
plain-ext
canon
dot
gv
tk
vmlz
vml
svgz
svg
xdot
eps
ps2
ps
gif
jpg/jpe/jpeg
png
(lib)
gd
vrml
gd2
xbm
wbmp
lasi
cairo
pdf
rsvg
devil
tga
bmp
tif

Notes

1. In the formats: -Tcmap, -Tcmapx, -Tsvg, -Tvml, the output generates 'id="node#"' properties for nodes, 'id="edge#"' properties for edges, and 'id="cluster#"' properties for clusters, with the '#' replaced by an internally assigned integer. These strings can be provided instead by an externally provided "id=xxx" attribute on the object. Normal "\N" "\E" "\G" substitutions are applied. Externally provided id values are not used internally, and it is the use's reponsibilty to ensure that they are sufficiently unique for their intended downstream use. Note, in particular, that "\E" is not a unique id for multiedges.