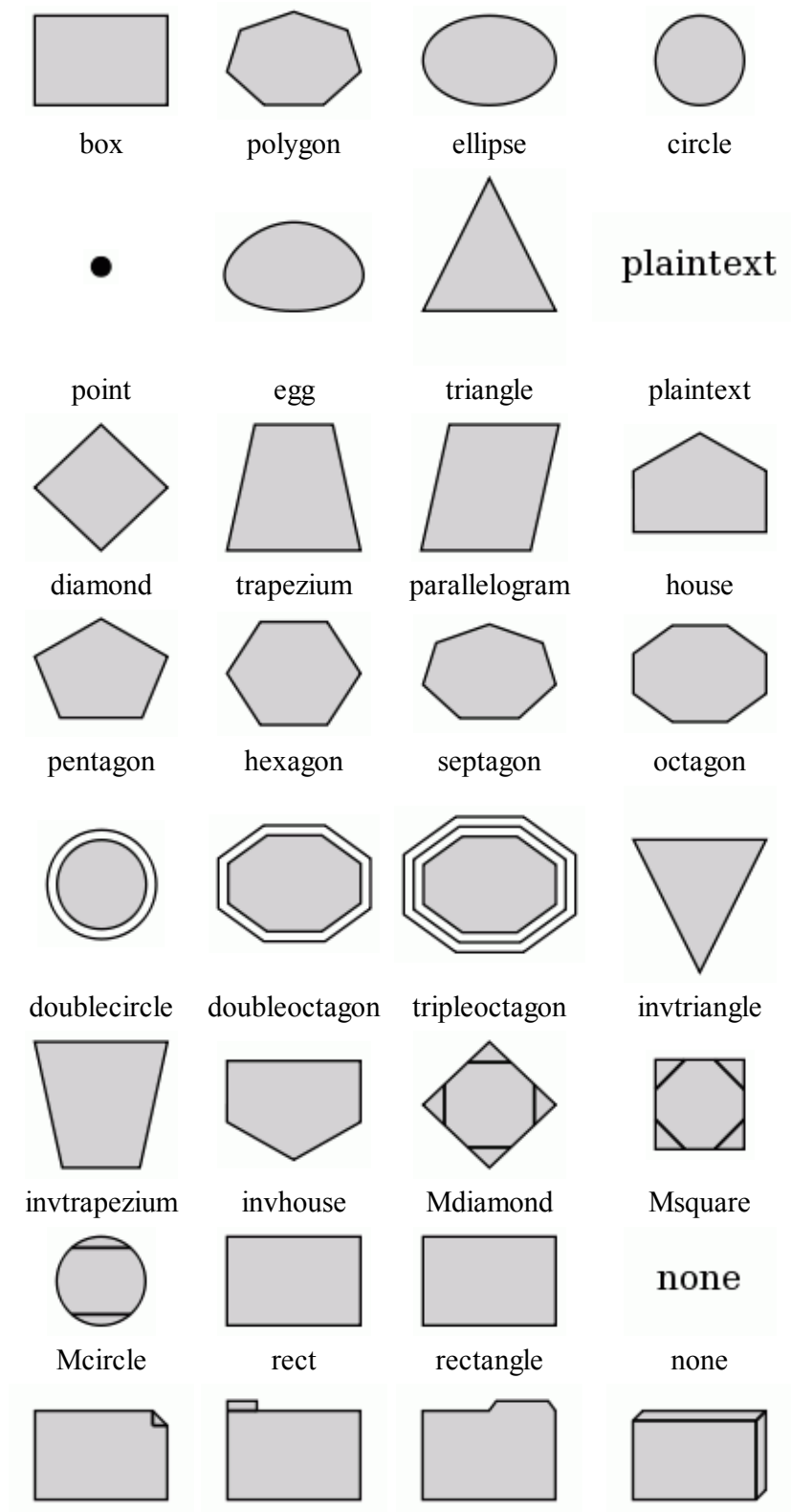


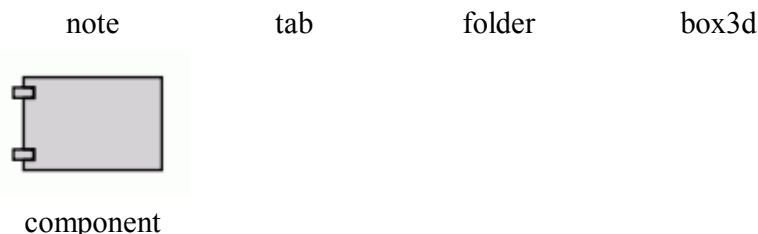
Node Shapes

The geometries of all node shapes are affected by the node attributes [fixedsize](#), [fontname](#), [fontsize](#), [height](#), [label](#), [style](#) and [width](#).

Polygon-based Nodes

The possible polygon-based shapes are displayed below.





As the figures suggest, the shapes `rect` and `rectangle` are synonyms for `box`, and `none` is a synonym for `plaintext`. Also, unlike the rest, we have shown these last two without `style=filled` to indicate the normal use. If `fill` were turned on, the label text would appear in a filled rectangle.

The geometries of polygon-based shapes are also affected by the node attributes [regular](#), [peripheries](#) and [orientation](#). If `shape="polygon"`, the attributes [sides](#), [skew](#) and [distortion](#) are also used. If unset, they default to 4, 0.0 and 0.0, respectively. In addition, the 3 M* shapes support auxiliary labels using the [toplabel](#) and [bottomlabel](#) attributes. On the other hand, the point shape is special in that it is only affected by the [peripheries](#), [width](#) and [height](#) attributes.

Record-based Nodes

These are specified by shape values of `"record"` and `"Mrecord"`. The structure of a record-based node is determined by its [label](#), which has the following schema:

rlabel = *field* ('|' *field*)*

where *field* = *fieldId* or '{' *rlabel* '}'

and *fieldId* = ['<' *string* '>'] [*string*]

Literal braces, vertical bars and angle brackets must be escaped. Spaces are interpreted as separators between tokens, so they must be escaped if you want spaces in the text.

The first string in *fieldId* assigns a portname to the field and can be combined with the node name to indicate where to attach an edge to the node. (See [portPos](#).) The second string is used as the text for the field; it supports the usual [escape sequences](#) `\n`, `\l` and `\r`.

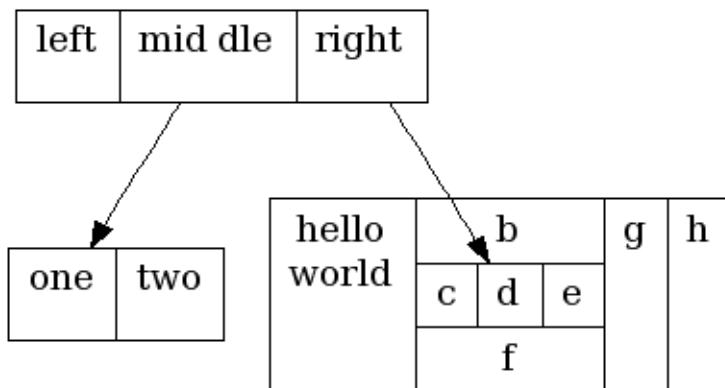
Visually, a record is a box, with fields represented by alternating rows of horizontal or vertical subboxes. The `Mrecord` shape is identical to a record shape, except that the outermost box has rounded corners. Flipping between horizontal and vertical layouts is done by nesting fields in braces `"{...}"`. The top-level orientation in a record is horizontal. Thus, a record with label `"A | B | C | D"` will have 4 fields oriented left to right, while `"{A | B | C | D}"` will have them from top to bottom and `"A | { B | C } | D"` will have "B" over "C", with "A" to the left and "D" to the right of "B" and "C".

The initial orientation of a record node depends on the [rankdir](#) attribute. If this attribute is `TB` (the default) or `TB`, corresponding to vertical layouts, the top-level fields in a record are displayed horizontally. If, however, this attribute is `LR` or `RL`, corresponding to horizontal layouts, the top-level fields are displayed vertically.

As an example of a record node, the dot input

```
digraph structs {
  node [shape=record];
  struct1 [label="<f0> left|<f1> mid\ dle|<f2> right"];
  struct2 [label="<f0> one|<f1> two"];
  struct3 [label="hello\nworld |{ b |{c|<here> d|e}| f}| g | h"];
  struct1:f1 -> struct2:f0;
  struct1:f2 -> struct3:here;
}
```

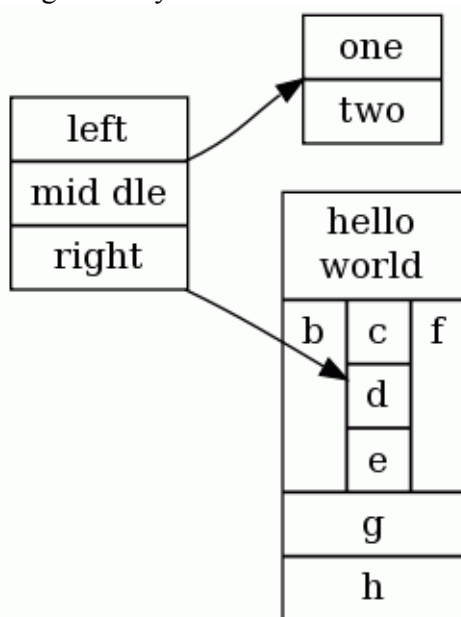
yields the figure



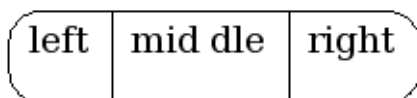
If we add the line

```
rankdir=LR
```

we get the layout



If we change node `struct1` to have shape `Mrecord`, it then looks like:



Styles for Nodes

The [style](#) attribute can be used to modify the appearance of a node. At present, there are 8 style values recognized: `filled`, `invisible`, `diagonals`, `rounded`, `dashed`, `dotted`, `solid` and `bold`. As usual, the value of the [style](#) attribute can be a comma-separated list of any of these. If the style contains conflicts (e.g, `style="dotted, solid"`), the last attribute wins.

`filled`

This value indicates that the node's interior should be filled. The color used is the node's `fillcolor` or, if that's not defined, its `color`. For unfilled nodes, the interior of the node is transparent to whatever color is the current graph or cluster background color. Note that `point` shapes are always filled.

Thus, the code

```
digraph G {
    rankdir=LR
```

```

node [shape=box, color=blue]
node1 [style=filled]
node2 [style=filled, fillcolor=red]
node0 -> node1 -> node2
}

```

yields the figure



invisible

Setting this style causes the node not to be displayed at all. Note that the node is still used in laying out the graph.

diagonals

The diagonals style causes small chords to be drawn near the vertices of the node's polygon or, in case of circles and ellipses, two chords near the top and the bottom of the shape. The special node shapes [Msquare](#), [Mcircle](#), and [Mdiamond](#) are simply an ordinary square, circle and diamond with the diagonals style set.

rounded

The rounded style causes the polygonal corners to be smoothed. Note that this style also applies to record-based nodes. Indeed, the `Mrecord` shape is simply shorthand for setting this style. Also, prior to 26 April 2005, the rounded and filled styles were mutually exclusive.

As an example of rounding, dot uses the graph

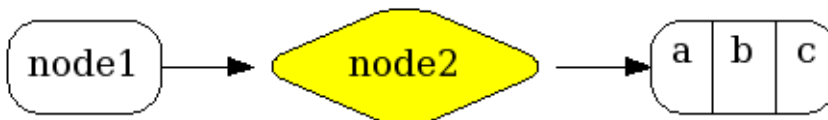
```

digraph R {
  rankdir=LR
  node [style=rounded]
  node1 [shape=box]
  node2 [fillcolor=yellow, style="rounded,filled", shape=diamond]
  node3 [shape=record, label="{ a | b | c }"]

  node1 -> node2 -> node3
}

```

to produce the figure



dashed

This style causes the node's border to be drawn as a dashed line.

dotted

This style causes the node's border to be drawn as a dotted line.

solid

This style causes the node's border to be drawn as a solid line, which is the default.

bold

This style causes the node's border to be drawn as a bold line. See also [setlinewidth](#).

Additional styles may be available with a specific code generator.

HTML-Like Labels

NOTE: This feature is only available on versions of Graphviz that are newer than mid-November 2003. In particular, it is not part of release 1.10.

If the value of a label attribute ([label](#) for nodes, edges, clusters, and graphs, and the [headlabel](#) and

[taillabel](#) attributes of an edge) is given as an [HTML string](#), that is, delimited by `<...>` rather than `"..."`, the label is interpreted as an HTML description. At their simplest, such labels can describe multiple lines of variously aligned text as provided by ordinary [string labels](#). More generally, the label can specify a table similar to those provided by HTML, with different graphical attributes at each level.

NOTE: The features and syntax supported by these labels are modeled on HTML. However, there are many aspects that are relevant to Graphviz labels that are not in HTML and, conversely, HTML allows various constructs which are meaningless in Graphviz. We will generally refer to these labels as "HTML labels" rather than the cumbersome "HTML-like labels" but the reader is warned that these are not really HTML. The grammar below describes precisely what Graphviz will accept.

Although HTML labels are not, strictly speaking, a shape, they can be viewed as a generalization of the record shapes described above. In particular, if a node has set its [shape](#) attribute to `plaintext`, the HTML label will be the node's shape. On the other hand, if the node has any other shape (except `point`), the HTML label will be embedded within the node the same way an ordinary label would be.

The following is an abstract grammar for HTML labels. Terminals, corresponding to elements, are shown in bold font, and nonterminals in italics. Square brackets `[` and `]` enclose optional items. Vertical bars `|` separate alternatives. Note that, as in HTML, element and attribute names are case-insensitive. (cf. sections 3.2.1 and 3.2.2 of the [HTML 4.01 specification](#)).

```

label : text
      | table
text : textitem
      | text textitem
textitem : string
          | <BR/>
          | <FONT> text </FONT>
table : [ <FONT> ] <TABLE> rows </TABLE> [ </FONT> ]
rows : row
      | rows row
row : <TR> cells </TR>
cells : cell
       | cells cell
cell : <TD> label </TD>
      | <TD> <IMG/> </TD>

```

Above, a *string* is any collection of printable characters, including all spaces. Note that outside of the body of a `<TD>` element, whitespace characters are ignored; within a `<TD>` element, spaces are preserved but all other white space characters are discarded. HTML comments are allowed within an HTML string. They can occur anywhere provided that, if they contain part of an HTML element, they must contain the entire element.

As is obvious from the above description, the interpretation of white space characters is one place where HTML-like labels is very different from standard HTML. In HTML, any sequence of white space characters is collapsed to a single space. If the user does not want this to happen, the input must use non-breaking spaces ` `. This makes sense in HTML, where text layout depends dynamically on the space available. In Graphviz, the layout is statically determined by the input, so it is reasonable to treat ordinary space characters as non-breaking. In addition, ignoring tabs and newlines allows the input text to be formatted for easier reading.

Each of the HTML elements has a set of optional attributes. Attribute values must appear in double quotes.

```

<TABLE
  ALIGN="CENTER|LEFT|RIGHT"
  BGCOLOR="color"
  BORDER="value"

```

```

CELLBORDER="value"
CELLPADDING="value"
CELLSPACING="value"
COLOR="color"
FIXEDSIZE="FALSE | TRUE"
HEIGHT="value"
HREF="value"
PORT="portName"
TARGET="value"
TITLE="value"
TOOLTIP="value"
VALIGN="MIDDLE | BOTTOM | TOP"
WIDTH="value"
>

<TR
  <!-- No attributes -->
>

<TD
  ALIGN="CENTER | LEFT | RIGHT | TEXT"
  BALIGN="CENTER | LEFT | RIGHT"
  BGCOLOR="color"
  BORDER="value"
  CELLPADDING="value"
  CELLSPACING="value"
  COLOR="color"
  COLSPAN="value"
  FIXEDSIZE="FALSE | TRUE"
  HEIGHT="value"
  HREF="value"
  PORT="portName"
  ROWSPAN="value"
  TARGET="value"
  TITLE="value"
  TOOLTIP="value"
  VALIGN="MIDDLE | BOTTOM | TOP"
  WIDTH="value"
>

<FONT
  COLOR="color"
  FACE="fontname"
  POINT-SIZE="value"
>

<BR
  ALIGN="CENTER | LEFT | RIGHT"
>

<IMG
  SCALE="FALSE | TRUE | WIDTH | HEIGHT | BOTH"
  SRC="value"
>

```

ALIGN

specifies horizontal placement. When an object is allocated more space than required, this value determines where the extra space is placed left and right of the object.

- CENTER aligns the object in the center. (Default)
- LEFT aligns the object on the left.
- RIGHT aligns the object on the right.
- (<TD> only) TEXT aligns lines of text using the full cell width. The alignment of a line is determined by its (possibly implicit) associated
 element.

The contents of a cell are normally aligned as a block. In particular, lines of text are first aligned as a text block based on the width of the widest line and the corresponding
 elements. Then,

the entire text block is aligned within a cell. If, however, the cell's **ALIGN** value is "TEXT", and the cell contains lines of text, then the lines are justified using the entire available width of the cell. If the cell does not contain text, then the contained image or table is centered.

BALIGN

specifies the default alignment of
 elements contained in the cell. That is, if a
 element has no explicit **ALIGN** attribute, the attribute value is specified by the value of **BALIGN**.

BGCOLOR="color"

sets the color of the background. This color can be overridden by a **BGCOLOR** attribute in descendents.

BORDER="value"

specifies the width of the border around the object in points. A value of zero indicates no border. The default is 1. The maximum value is 255. If set in a table, and **CELLBORDER** is not set, this value is also used for all cells in the table. It can be overridden by a **BORDER** tag in a cell.

CELLBORDER="value"

specifies the width of the border for all cells in a table. It can be overridden by a **BORDER** tag in a cell. The maximum value is 255.

CELLPADDING="value"

specifies the space, in points, between a cell's border and its content. The default is 2. The maximum value is 255.

CELLSPACING="value"

specifies the space, in points, between cells in a table and between a cell and the table's border. The default is 2. The maximum value is 127.

COLOR="color"

sets the color of the font within the scope of ..., or the border color of the table or cell within the scope of <TABLE>...</TABLE>, or <TD>...</TD>. This color can be overridden by a **COLOR** attribute in descendents. By default, the font color is determined by the [fontcolor](#) attribute of the corresponding node, edge or graph, and the border color is determined by the [color](#) attribute of the corresponding node, edge or graph.

COLSPAN="value"

specifies the number of columns spanned by the cell. The default is 1. The maximum value is 65535.

FACE="fontname"

specifies the font to use within the scope of This can be overridden by a **FACE** attribute in descendents. By default, the font name is determined by the [fontname](#) attribute of the corresponding node, edge or graph.

FIXEDSIZE

specifies whether the values given by the **WIDTH** and **HEIGHT** attributes are enforced.

- FALSE allows the object to grow so that all its contents will fit. (Default)
- TRUE fixes the object size to its given **WIDTH** and **HEIGHT**. Both of these attributes must be supplied.

HEIGHT="value"

specifies the minimum height, in points, of the object. The height includes the contents, any spacing and the border. Unless ***FIXEDSIZE*** is true, the height will be expanded to allow the contents to fit. The maximum value is 65535.

HREF="value"

attaches a URL to the object.

POINT-SIZE="value"

sets the size of the font, in points, used within the scope of This can be overridden by a ***POINT-SIZE*** attribute in descendents. By default, the font size is determined by the [fontsize](#) attribute of the corresponding node, edge or graph.

PORT="value"

attaches a portname to the object. (See [portPos](#).) This can be used to modify the head or tail of an edge, so that the end attaches directly to the object.

ROWSPAN="value"

specifies the number of rows spanned by the cell. The default is 1. The maximum value is 65535.

SCALE

specifies how an image will use any extra space available in its cell. Allowed values are

- FALSE : keep image its natural size. (Default)
- TRUE : scale image uniformly to fit.
- WIDTH : expand image width to fill
- HEIGHT : expand image height to fill
- BOTH : expand both image width height to fill

If this attribute is undefined, the image inherits the [imagescale](#) attribute of the graph object being drawn. As with the [imagescale](#) attribute, if the cell has a fixed size and the image is too large, any offending dimension will be shrunk to fit the space, the scaling being uniform in width and height if *SCALE="true"*.

SRC="value"

specifies the image file to be displayed in the cell. Note that if the software is used as a web server, file system access to images is more restricted. See [GV_FILE_PATH](#) and [SERVER_NAME](#).

TARGET="value"

determines which window of the browser is used for the URL if the object has one. See [W3C documentation](#).

TITLE="value"

sets the tooltip annotation attached to the element. This is used only if the element has a HREF attribute.

TOOLTIP="value"

is an alias for ***TITLE***.

VALIGN

specifies vertical placement. When an object is allocated more space than required, this value determines where the extra space is placed above and below the object.

- MIDDLE aligns the object in the center. (Default)
- LEFT aligns the object on the left.
- RIGHT aligns the object on the right.

WIDTH="value"

specifies the minimum width, in points, of the object. The width includes the contents, any spacing and the border. Unless **FIXEDSIZE** is true, the width will be expanded to allow the contents to fit. The maximum value is 65535.

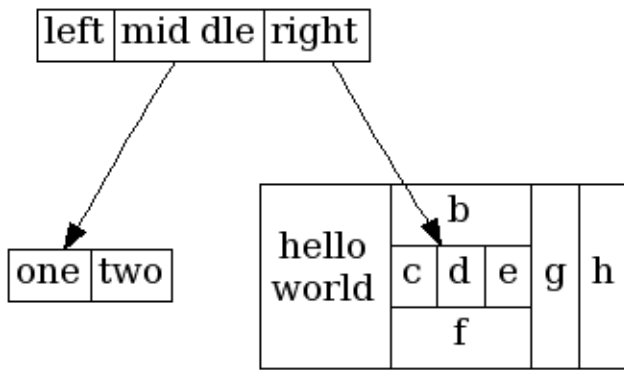
There is some inheritance among the attributes. If a table specifies a **CELLPADDING**, **CELLBORDER** or **BORDER** value, this value is used by the table's cells unless overridden. If a cell or table specifies a **BGCOLOR**, this will be the background color for all of its descendents. Of course, if a background or fill color is specified for the graph object owning the label, this will be the original background for the label. The object's fontname, fontcolor and fontsize attributes are the default for drawing text. These can be overridden by using **FONT** to set new values. The new font values will hold until overridden by an enclosed **FONT** element. Finally, the pencolor or color of the graph object will be used as the border color.

Because of certain limitations in handling tables in a device-independent manner, when **BORDER** is 1 and both table and cell borders are on and **CELLSPACING** is less than 2, anomalies can arise in the output, such as gaps between sides of borders which should be abutting or even collinear. The user can usual get around this by increasing the border size or the spacing, or turning off the table border.

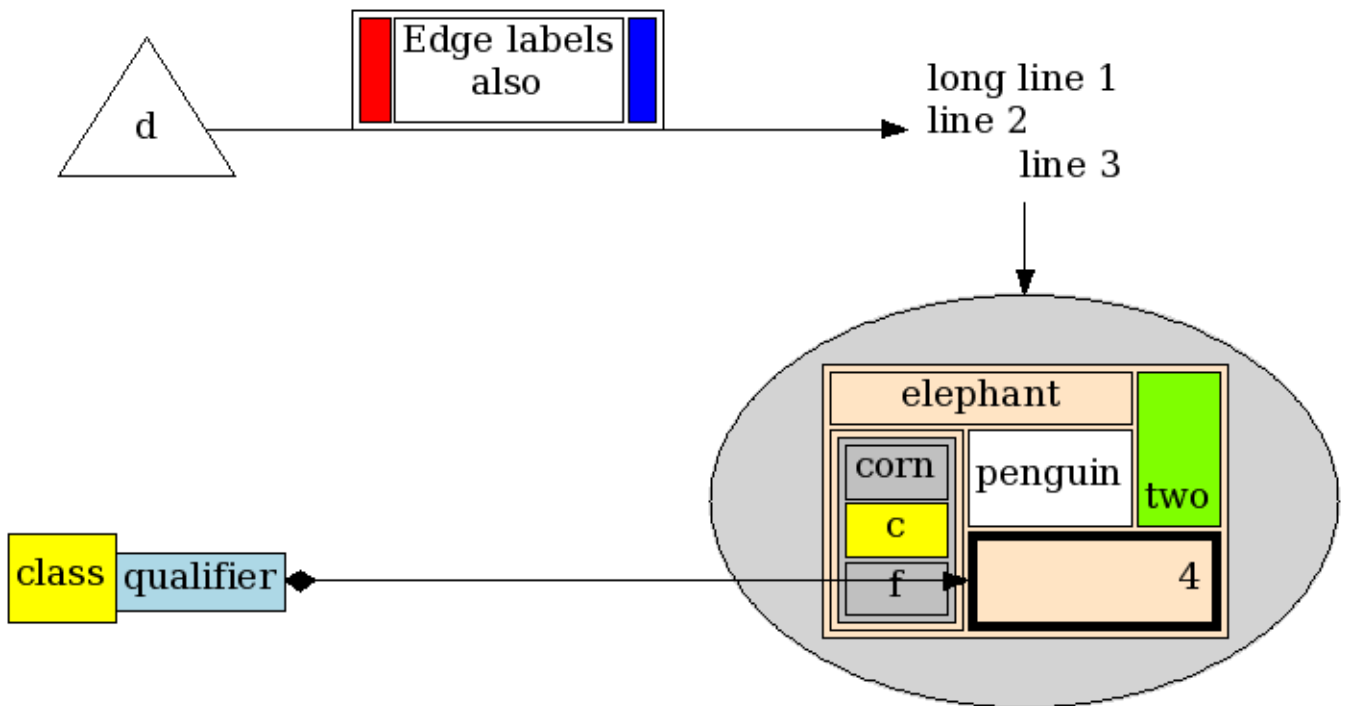
As an example of HTML labels, the dot input

```
digraph structs {
    node [shape=plaintext]
    struct1 [label=<
TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0">
  <TR><TD>left</TD><TD PORT="f1">mid dle</TD><TD PORT="f2">right</TD></TR>
</TABLE>>];
    struct2 [label=<
TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0">
  <TR><TD PORT="f0">one</TD><TD>two</TD></TR>
</TABLE>>];
    struct3 [label=<
TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0" CELLPADDING="4">
  <TR>
    <TD ROWSPAN="3">hello<BR/>world</TD>
    <TD COLSPAN="3">b</TD>
    <TD ROWSPAN="3">g</TD>
    <TD ROWSPAN="3">h</TD>
  </TR>
  <TR>
    <TD>c</TD><TD PORT="here">d</TD><TD>e</TD>
  </TR>
  <TR>
    <TD COLSPAN="3">f</TD>
  </TR>
</TABLE>>];
    struct1:f1 -> struct2:f0;
    struct1:f2 -> struct3:here;
}
```

produces the HTML analogue of the record example above



As usual, an HTML specification is more verbose. On the other hand, HTML labels are much more general, as the following example shows:



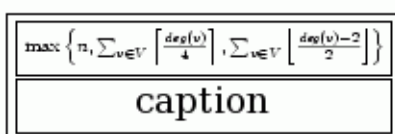
The source for this graph can be found [here](#).

Here is an example using elements



with the [input graph](#).

Here is an example using an element



with the [input graph](#).

User-defined Nodes

There is a third type of node shape which is specified by the user. Typically, these shapes rely on the details of a concrete graphics format. At present, shapes can be described using PostScript, via a file or add-on library, for use in PostScript output, or shapes can be specified by a bitmap-image file for use with SVG or bitmap (jpeg, gif, etc.) output. More information can be found on the page [How to create custom shapes](#).

SDL Shapes for PostScript

One example of user-defined node shapes is provided by Mark Rison of CSR. These are the [SDL](#) shapes. These are available as PostScript functions whose use is described in [External PostScript procedures](#). The necessary PostScript library file and sample use can be found in the `contrib/sdlshapes` directory in the release. Please note the COPYRIGHT AND PERMISSION NOTICE contained in the library file `sdl.ps`.

The table below gives the shape names and the corresponding node shapes. 