The DOT Language

The following is an abstract grammar defining the DOT language. Terminals are shown in bold font and nonterminals in italics. Literal characters are given in single quotes. Parentheses (and) indicate grouping when needed. Square brackets [and] enclose optional items. Vertical bars | separate alternatives.

graph : [**strict**] (**graph** | **digraph**) [*ID*] **'{'** *stmt_list* **'}'**
stmt_list : [*stmt* [**';'**] [*stmt_list*]]
stmt : *node_stmt*
| *edge_stmt*
| *attr_stmt*
| *ID* **'='** *ID*
| *subgraph*
attr_stmt : (**graph** | **node** | **edge**) *attr_list*
attr_list : **'['** [*a_list*] **']'** [*attr_list*]
a_list : *ID* [**'='** *ID*] [**','**] [*a_list*]
edge_stmt : (*node_id* | *subgraph*) *edgeRHS* [*attr_list*]
edgeRHS : *edgeop* (*node_id* | *subgraph*) [*edgeRHS*]
node_stmt : *node_id* [*attr_list*]
node_id : *ID* [*port*]
port : **':'** *ID* [**':'** *compass_pt*]
| **':'** *compass_pt*
subgraph : [**subgraph** [*ID*]] **'{'** *stmt_list* **'}'**
| **subgraph** *ID*
compass_pt : (**n** | **ne** | **e** | **se** | **s** | **sw** | **w** | **nw**)

The keywords **node**, **edge**, **graph**, **digraph**, **subgraph**, and **strict** are case-independent.

An *ID* is one of the following:

- Any string of alphabetic characters, underscores or digits, not beginning with a digit;
- a number $[-]^{?}(.[0\text{-}9]^{+} \mid [0\text{-}9]^{+}(.[0\text{-}9]^{*})^{?})$;
- any double-quoted string ("...") possibly containing escaped quotes (\");
- an HTML string (<...>).

Note that in HTML strings, angle brackets must occur in matched pairs, and unescaped newlines are allowed. In addition, the content must be legal XML, so that the special XML escape sequences for ", &, <, and > may be necessary in order to embed these characters in attribute values or raw text.

Both quoted strings and HTML strings are scanned as a unit, so any embedded comments will be treated as part of the strings.

An *edgeop* is `->` in directed graphs and `--` in undirected graphs.

An *a_list* clause of the form *ID* is equivalent to *ID*`=true`.

The language supports C++-style comments: `/* */` and `//`. In addition, a line beginning with a '#' character is considered a line output from a C preprocessor (e.g., # 34 to indicate line 34) and discarded.

Semicolons aid readability but are not required except in the rare case that a named subgraph with no body immediately preceeds an anonymous subgraph, since the precedence rules cause this sequence to be parsed as a subgraph with a heading and a body.

As another aid for readability, dot allows single logical lines to span multiple physical lines using the standard C convention of a backslash immediately preceding a newline character. In addition, double-quoted strings can be concatenated using a '+' operator. As HTML strings can contain newline characters, they do not support the concatenation operator.

Semantic Notes

If a default attribute is defined using a **node**, **edge**, or **graph** statement, or by an attribute assignment not attached to a node or edge, any object of the appropriate type defined afterwards will inherit this attribute value. This holds until the default attribute is set to a new value, from which point the new value is used. Objects defined before a default attribute is set will have an empty string value attached to the attribute once the default attribute definition is made.

Note, in particular, that a subgraph receives the attribute settings of its parent graph at the time of its definition. This can be useful; for example, one can assign a font to the root graph and all subgraphs will also use the font. For some attributes, however, this property is undesirable. If one attaches a label to the root graph, it is probably not the desired effect to have the label used by all subgraphs. Rather than listing the graph attribute at the top of the graph, and the resetting the attribute as needed in the subgraphs, one can simple defer the attribute definition if the graph until the appropriate subgraphs have been defined.

Character encodings

The DOT language assumes at least the ascii character set. Quoted strings, both ordinary and HTML-like, may contain non-ascii characters. In most cases, these strings are uninterpreted: they simply serve as unique identifiers or values passed through untouched. Labels, however, are meant to be displayed, which requires that the software be able to compute the size of the text and determine the appropriate glyphs. For this, it needs to know what character encoding is used.

By default, DOT assumes the UTF-8 character encoding. It also accepts the Latin1 (ISO-8859-1) character set, assuming the input graph uses the **charset** attribute to specify this. For graphs using other character sets, there are usually programs, such as `iconv`, which will translate from one character set to another.

Another way to avoid non-ascii characters in labels is to use HTML entities for special characters. During label evaluation, these entities are translated into the underlying character. This table shows the supported entities, with their Unicode value, a typical glyph, and the HTML entity name. Thus, to include a lower-case Greek beta into a string, one can use the ascii sequence `&beta;`. In general, one should only use entities that are allowed in the output character set, and for which there is a glyph in the font.