Below is the formal grammar for the netlist language. This grammar has undefined symbols *ID*, *INT*, and *FLOAT* corresponding to identifiers, integer constants, and floating-point constants, respectively. Literal symbols are in `typewriter` style, and ϵ denotes an empty string. This is the grammar used in the YACC netlist parser, with only minor modifications for readability.

netlist:
    netlistDefs

netlistDefs:
    netlistDefs netlistDef
    netlistDef

netlistDef:
    paramLine
    processLine
    subnet
    codeLine

paramLine:
    `param` *paramList*

paramList:
    param
    paramList `,` *param*

param:def
    ID

processLine:
    `process` *ID* `=` `[` *defs* `]`
    `process` *ID* `:` *ID* `=` `[` *defs* `]`

subnet:
    subnetHead codeBlock

subnetHead:
    `subnet` *ID* `[` *names* `]` `[` *subnetParams* `]`

subnetParams:
    subnetParams subnetParam

subnetParam:
    def
    ID `=` `*`

codeLines:
    codeLines codeLine
    codeLine

codeLine:
    codeBlock
    def
    elementLine
    forLine

codeBlock:
    `[` *codeLines* `]`

elementLine:
    elementHead `[` *names* `]` `[` *defs* `]`

elementHead:
    ID `(` *exprs* `)` *ID ID*
    ID ID ID
    ID `(` *exprs* `)` *ID* `*`
    ID ID `*`
    ID ID
    ID `*`

forLine:
    forHead codeBlock

forHead:
    `for` *ID* `=` *expr* `:` *expr*

def:
    ID `=` *expr*

defs:
    defs def
    ϵ

names:
    names name
    ϵ

name:
    ID
    ID `(` *exprs*`)`

expr:

expr `+` *expr*
expr `-` *expr*
expr `*` *expr*
expr `/` *expr*
`-` *expr*
`(` *expr* `)`
INT
FLOAT
ID
ID `(` *exprs* `)`

exprs:
exprs `,` *expr*
expr

Comments and `uses` statements are implemented with the tokenizer rather than as part of the parser. Comments begin at a `%` character and extend to the end of the line. `uses` statements must appear on a line of their own, and consist of the keyword `uses` followed by the name of the file to be used, optionally including a path.