Ucamco

Parsing Expression Grammar for the Gerber Format

A format developed by Ucamco

Revision 2020.09

Contents

1 Grammar Syntax

The grammar of the Gerber file format is specified in a variant of the Extended Backus-Naur Form used by the 竜 TatSu PEG parser generator. For simplicity, only a subset of the rules in the very powerful TatSu grammar was used – after all Gerber is a simple format.

Below is a description of the subset used, largely taken from the TatSu documentation.

1.1 Rules

A grammar consists of a sequence of one or more rules of the form:

```
name = <expression> ;
```

The expressions are constructed from the following operators, in reverse order of precedence.

The root name is `start`.

1.1.1 # comment

Comments have no effect on the grammar.

1.1.2 e1 | e2

Choice. Match either `e1` or `e2`. A `|` can be used before the first option as a layout aid:

```
choices
    =
    | e1
    | e2
    | e3
    ;
```

1.1.3 e1 e2

Sequence. Match e1 and then match e2.

1.1.4 (e)

Grouping. Match `e`. For example: `('a' | 'b')`.

1.1.5 [e]

Optionally match `e`.

1.1.6 { e } or { e }*

Closure. Match `e` zero or more times.

1.1.7 { e }+

Positive closure. Match `e` one or more times.

1.1.8 &e

Positive lookahead. Succeeds if `e` can be parsed, and does not consume any input.

1.1.9 !e

Negative lookahead. Fails if `e` can be parsed, and does not consume any input.

1.1.10 'text' or "text"

Match the token *text* within the quotation marks.

Note that if *text* is alphanumeric it will only parse if the character following the token is not alphanumeric. This is done to prevent tokens like *IN* matching when the text ahead is *INITIALIZE*. This feature can be turned setting the grammar directive `@@nameguard=False.`

1.1.11 /regexp/

The pattern expression. Match the Python regular expression **regexp** at the current text position. Unlike other expressions, this does not advance over whitespace or comments. For that, place the regexp as the only term in its own rule.

The regex is interpreted as a Python's raw string literal and passed with regexp MULTILINE | regexp.UNICODE options to the Python re module (or to regex, if available), using match() at the current position in the text.

Consecutive patterns are concatenated to form a single one.

1.1.12 @:e

The override operator. It is used to separate language content from the delimiters. This is a typical use of the override operator:

```
subexp = '(' @:expre ')' ;
```

From the complete rule only e is entered in the abstract syntax tree.

1.1.13 $

The *end of text* symbol. Verify that the end of the input text has been reached.

1.2 Grammar Directives

Directives in the grammar control the behavior of the parsers. All directives are of the form @@name :: <value>. The following directives are used:

1.2.1 @@grammar :: <word>

Specifies the name of the grammar.

1.2.2 @@nameguard :: <bool>

When set to True, avoids matching tokens when the next character in the input sequence is alphanumeric. Defaults to True. See the 'text' expression for an explanation.

```
@@nameguard :: False
```

1.2.3 @@whitespace :: <regexp>

Provides a regular expression for the whitespace to be ignored by the parser. It defaults to /(?s)\s+/:

```
@@whitespace :: /[\t ]+/
```

2 The Parsing Expression Grammar

```
@@grammar    :: Gerber_2020_09
@@nameguard  :: False
@@whitespace :: /\n/

start = {statement}* M02 $;
statement =
    | single_statement
    | compound_statement
    ;

single_statement =
    | operation
    | interpolation_state_command
    | Dnn
    | G04
    | attribute_command
    | AD
    | AM
    | coordinate_command
    | transformation_state_command
    ;
compound_statement =
    | region_statement
    | SR_statement
    | AB_statement
    ;

coordinate_command =
    |FS
    |MO
    ;
operation =
    |D01
    |D02
    |D03
    ;
interpolation_state_command =
    |G01
    |G02
    |G03
    |G74
    |G75
    ;
transformation_state_command =
    |LP
    |LM
    |LR
    |LS
    ;
attribute_command =
```

```
    |TO
    |TD
    |TA
    |TF
    ;


# Graphics commands
#-----------------------------

FS = '%' @:('FS' 'LA' 'X'/[1-9][56]/ 'Y'/[1-9][56]/) '*%';
MO = '%' @:('MO' ('MM'|'IN')) '*%';

D01 = @:(['X' coordinate] ['Y' coordinate] ['I' coordinate 'J' coordinate] 'D01') '*';
D02 = @:(['X' coordinate] ['Y' coordinate] 'D02') '*';
D03 = @:(['X' coordinate] ['Y' coordinate] 'D03') '*';

G01 = @:('G01') '*';
G02 = @:('G02') '*';
G03 = @:('G03') '*';
G74 = @:('G74') '*';
G75 = @:('G75') '*';

Dnn = @:(aperture_ident) '*';

G04 = @:('G04' string) '*';

M02 = @:('M02') '*';

LP = '%' @:('LP' ('C'|'D')) '*%';
LM = '%' @:('LM' ('N'|'XY'|'Y'|'X')) '*%';
LR = '%' @:('LR' decimal) '*%';
LS = '%' @:('LS' decimal) '*%';

AD = '%' @:('AD' aperture_ident AD_body) '*%';
AD_body =
   |'C' fst_par [nxt_par]
   |'R' fst_par nxt_par [nxt_par]
   |'O' fst_par nxt_par [nxt_par]
   |'P' fst_par nxt_par [nxt_par [nxt_par]]
   |!(('C'|'R'|'O'|'P')(','|'*')) name [fst_par {nxt_par}*]
   ;
fst_par = ',' @:decimal;
nxt_par = 'X' @:decimal;

AM = '%' @:('AM' macro_name macro_body) '%';
macro_name = @:name '*';
macro_body = {in_macro_block}+;
in_macro_block =
    |primitive
                |variable_definition
                ;
variable_definition = @:(macro_variable '=' expression) '*';
macro_variable =      '$' pos_integer;
primitive =
```

```
    |@:('0'  string) '*'
    |@:('1'  par par par par [par]) '*'
    |@:('20' par par par par par par par) '*'
    |@:('21' par par par par par par) '*'
    |@:('4'  par par par par {par par}+ par) '*'
    |@:('5'  par par par par par par) '*'
    |@:('6'  par par par par par par par par par) '*'
    |@:('7'  par par par par par par) '*'
    ;
par = ',' @:(expression);


# Compound statements

region_statement = G36 {contour}+ G37;
contour =          D02 {D01|interpolation_state_command}*;
G36 = @:('G36') '*';
G37 = @:('G37') '*';

AB_statement = AB_open {in_block_statement}* AB_close;
AB_open  =     '%' @:('AB' aperture_ident) '*%';
AB_close =     '%' @:('AB') '*%';

SR_statement = SR_open {in_block_statement}* SR_close;
SR_open =      '%' @:('SR' 'X' pos_integer 'Y' pos_integer 'I' decimal 'J' decimal) '*%';
SR_close =     '%' @:('SR') '*%';

in_block_statement =
    |single_statement
    |region_statement
    |AB_statement
    ;


# Attribute commands
#---------------------------

TF = '%' @:('TF' TF_atts) '*%';
TA = '%' @:('TA' TA_atts) '*%';
TO = '%' @:('TO' TO_atts) '*%';
TD = '%' @:('TD' [all_atts]) '*%';

TF_atts =
    |'.Part'               nxt_field
    |'.FileFunction'       {nxt_field}*
    |'.FilePolarity'       nxt_field
    |'.SameCoordinates'    [nxt_field]
    |'.CreationDate'       nxt_field
    |'.GenerationSoftware'  nxt_field nxt_field [nxt_field]
    |'.ProjectId'          nxt_field nxt_field  nxt_field
    |'.MD5'                nxt_field
    |user_name
    ;
TA_atts =
    |'.AperFunction'   {nxt_field}*
```

```
    |'.DrillTolerance'  nxt_field   nxt_field
    |'.FlashText'      {nxt_field}*
    |user_name         {nxt_field}*
    ;
TO_atts =
    |'.N'     nxt_field   {nxt_field}*
    |'.P'     nxt_field    nxt_field   [nxt_field]
    |'.C'     nxt_field
    |'.CRot'  nxt_field
    |'.CMfr'  nxt_field
    |'.CMPN'  nxt_field
    |'.CVal'  nxt_field
    |'.CMnt'  nxt_field
    |'.CFtp'  nxt_field
    |'.CPgN'  nxt_field
    |'.CPgD'  nxt_field
    |'.CHgt'  nxt_field
    |'.CLbN'  nxt_field
    |'.CLbD'  nxt_field
    |'.CSup'  nxt_field nxt_field {nxt_field nxt_field}*
    |user_name {nxt_field}*
    ;
all_atts =
    |TF_atts
    |TA_atts
    |TO_atts
    ;
nxt_field = ',' @:field;


# Expressions
#-----------------

unsigned_expression =
    |term term_operation unsigned_expression
    |term
    ;
term =
    |factor factor_operation term
    |factor
    ;
factor =
    |'(' expression ')'
    |unsigned_decimal
    |macro_variable
    ;
expression = [sign] unsigned_expression;


# Numbers
#-------------

unsigned_integer =
    |'0'
    |pos_integer
```

```
    ;
integer = [sign] unsigned_integer;
unsigned_decimal =
    |unsigned_decimal_integer_part_pos
    |unsigned_decimal_integer_part_zero
    |unsigned_decimal_integer_part_none
    |unsigned_integer
    ;
decimal = [sign] unsigned_decimal;

# Terminals, by regex
#-------------------------

sign =              /[+-]/;
term_operation =    /[+-]/;
factor_operation = /[x\/]/;

pos_integer =     /[1-9][0-9]*/;
aperture_ident = /D[1-9][0-9]+/;
coordinate =      /[+-]{0,1}[0-9]+/;
unsigned_decimal_integer_part_pos =  /[1-9][0-9]*\.[0-9]*/;
unsigned_decimal_integer_part_zero = /0\.[0-9]*/;
unsigned_decimal_integer_part_none = /\.[0-9]+/;

name =          /[a-zA-Z_.$][a-zA-Z_.0-9]*/;
standard_name = /\.[a-zA-Z_.0-9]+/;
user_name =     /[a-zA-Z_$][.a-zA-Z_0-9]*/;
string =        /[^%*]*/;
field =         /[^%*,]*/;
```

3 References

Documentation for the TatSu PEG parser generator: https://tatsu.readthedocs.io/en/stable/

Wikipedia on the Parsing Expression Grammar: https://en.wikipedia.org/wiki/Parsing_expression_grammar

Bryan Fords page on PEGs and its parsers: https://bford.info/packrat/

4 Revisions

4.1 Revision 2020.09

Initial version

5 Copyright and Intellectual Property