UNIVERSITÀ DEGLI STUDI DI BERGAMO | Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

G-CODE PARSER AND SHAPER

Project documentation

Course: Linguaggi formali e compilatori

Students: Ghislotti Luca (matr. 1052975)

Mazzola Alessandro (matr. 1053121)

Parimbelli Luca (matr. 1053142)

Marinò Andrea (matr. 1053230)

Professor: Giuseppe Psaila

Università degli Studi di Bergamo

A.A. 2021/2022

What are GCODE Parser and Shaper

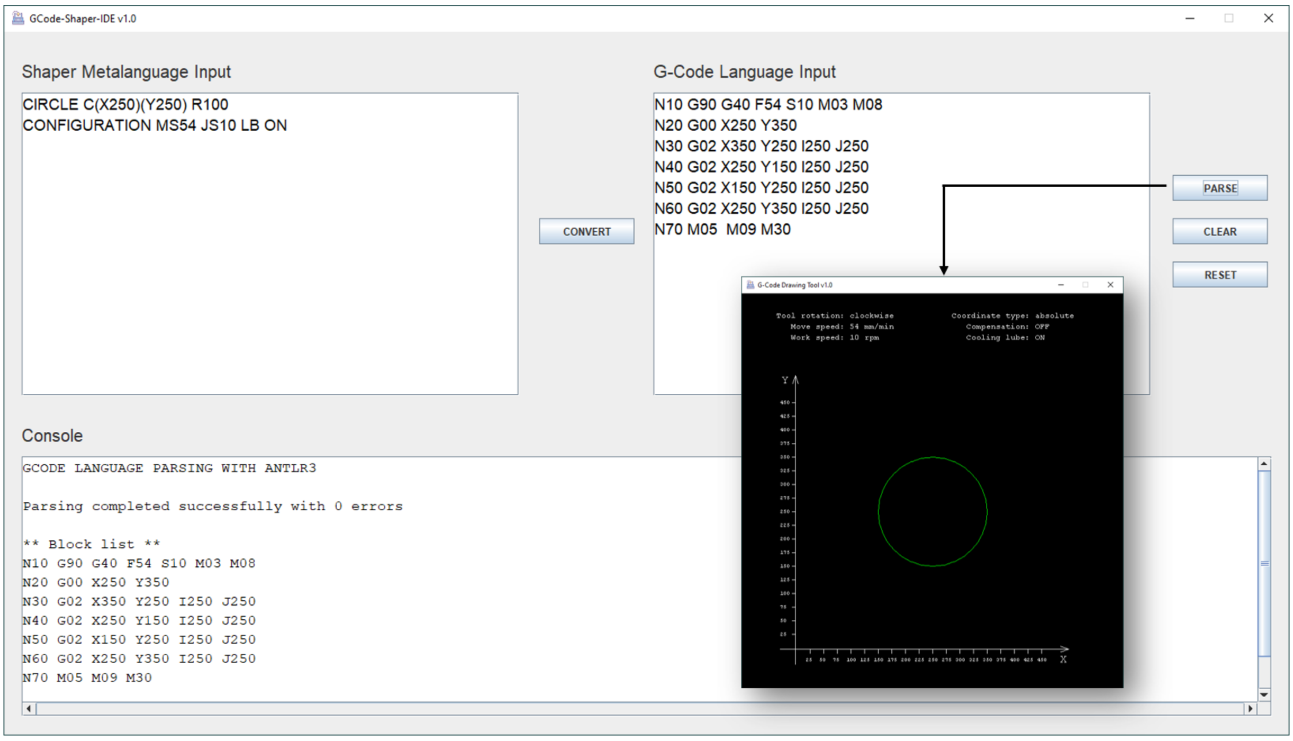
GCode-Shaper-Parser aims to provide students a useful tool for understanding and practicing with G-code, a programming language for CNC (*Computer Numerical Control*) machines. This project is composed by two main parts that are integrated each other and work together to provide to the final user the best approach to the industrial automation duties. As specifed below, the GCode programming language used by this software is a simplifed version of the real GCode used on the industrial machines and, in particular, is composed by:

- ***GCODE Parser*** is the main compiler developed for parsing G-code language and provide a graphical representation of the written code.
- ***Shaper*** is a metalanguage built upon G-code in order to simplify the definition of CNC commands and help the users understand the rules of G-code language.

For further information on GCode programming language plese refere to this link.

We want to underline that the GCode Shaper works with a new metalanguage created from scratch by the developers of this tool while the GCode Parser works with the "ligth GCode", that is a simplified version of the GCode which already exist and is widespread in the industrial sector.

Both GCODE Parser and Shaper are written in Java using ANTLR package. GCode-Shaper-IDE is a GUI developed for helping users in the usage of *GCode-Shaper-Parser*.



```
CIRCLE C(X250)(Y250) R100
CONFIGURATION MS54 JS10 LB ON
```

```
N10 G90 G40 F54 S10 M03 M08
N20 G00 X250 Y350
N30 G02 X350 Y250 I250 J250
N40 G02 X250 Y150 I250 J250
N50 G02 X150 Y250 I250 J250
N60 G02 X250 Y350 I250 J250
N70 M05  M09 M30
```

```
GCODE LANGUAGE PARSING WITH ANTLR3

Parsing completed successfully with 0 errors

** Block list **
N10 G90 G40 F54 S10 M03 M08
N20 G00 X250 Y350
N30 G02 X350 Y250 I250 J250
N40 G02 X250 Y150 I250 J250
N50 G02 X150 Y250 I250 J250
N60 G02 X250 Y350 I250 J250
N70 M05 M09 M30
```

```
Tool rotation: clockwise          Coordinate type: absolute
   Move speed: 54 mm/min             Compensation: OFF
   Work speed: 10 rpm                Cooling lube: ON
```

Repository structure

Repository tree diagram

```
GCode-Shaper-Parser
├── code
├── docs
├── errors
├── jars
├── libraries
├── UML
├── LICENSE
└── README.md
```

This repository is basically organized as follows:

code

It contains all the code related to this project. It is further organized into packages, each of which refers to a specific feature or features subset. Please refer below for subfolder organization details.

docs

It contains the documentation both for the G-Code parser and the Shaper metalanguage. You can navigate it using links to the errors subfolder.

```
docs
├── gcode parser
│   ├── GCode Parser Syntax Grammar.pdf
│   ├── README.md
│   └── tokenList.md
├── pdf_readme.pdf
├── pptx_presentation.pdf
└── shaper metalanguage
    ├── README.md
    ├── Shaper Parser Syntax Grammar.pdf
    └── tokenList.md
```

errors

In this folder are located the specification for the errors, both for G-Code and Shaper.

```
errors
├── gcode parser
│   └── README.md
└── shaper metalanguage
    └── README.md
```

jars

This folder containts the antlrworks-1.5.2-complete.jar. If you'd like to develop further, please refer to this jar in order to update and modify the language specification (both for G-Code and Shaper).

```
jars
└── antlrworks-1.5.2-complete.jar
```

libraries

This project uses antlr-3.4-complete.jar. When importing the Java code, please use this library version to build the project.

```
├── antlr-3.4-complete.jar
├── antlr-4.8-complete.jar
└── antlr-runtime-4.8.jar
```

UML

This folder contains all UML diagrams, namely class diagrams for all classes and the package diagram for the entire project.

```
UML
├── gcodeCompiler_package.png
├── gcodeCompilerUtil_package.png
├── gcodeDrawingTool_package.png
├── gcodeIDE.png
├── gcodeMain_package.png
├── package_diagram.png
├── shaperCompiler_package.png
├── shaperCompilerUtil_package.png
├── shaperMain_package.png
└── README.md
```

Code management and organization

Code tree diagram

```
code
├── GCODE
├── IDE
├── SHAPER
├── resources
└── temp_files
```

The code folder is organized as follows:

GCODE

This source folder contains all the code related to the G-Code specification language. gcodeGrammarHandler.java is responsible for managing all G-Code main functions, mainly responsible for language-related data acquisition, data structures manipulation and population. In the util package are located all support classes that allow to build all the language associated objects. The gcodeDrawingTool package contains the classes that manage the graphical interface for the G-Code drawing tool, while gcodeMain contains the error manager class for G-Code.

```
GCODE
├── gcodeCompiler
│   ├── gcodeGrammar.g
│   ├── gcodeGrammarHandler.java
│   ├── gcodeGrammarLexer.java
│   ├── gcodeGrammarParser.java
│   ├── gcodeGrammar.tokens
│   └── util
│       ├── BlockDescriptor.java
│       ├── CircularMove.java
│       ├── Coordinate.java
│       ├── GCodeError.java
│       ├── InfoGeometriche.java
│       ├── InfoTecnologiche.java
│       ├── InfoTecnologicheM.java
│       ├── LinearMove.java
│       └── Tool.java
├── gcodeDrawingTool
│   ├── GCodeDrawingViewer.java
│   └── StaticDrawingController.java
└── gcodeMain
    └── GcodeErrorManager.java
```

IDE

In this source folder are located the viewer and controller classes for the IDE.

```
IDE
└── gcodeIDE
    ├── GCodeIDEMain.java
    └── GCodeIDEWindow.java
```

SHAPER

This source folder has the exact same structure as GCODE but refers to the Shaper Metalanguage.

```
SHAPER
├── shaperCompiler
│   ├── shaperGrammar.g
│   ├── shaperGrammarHandler.java
│   ├── shaperGrammarLexer.java
│   ├── shaperGrammarParser.java
│   ├── shaperGrammar.tokens
│   └── util
│       ├── Circle.java
│       ├── Rectangle.java
│       ├── Shape.java
│       ├── ShaperError.java
│       ├── Square.java
│       └── Triangle.java
└── shaperMain
```

```
    └── ShaperErrorManager.java
```

resources

This folder contains some debugging file useful for future developments, mainly involving the possibilty of direct parsing of *.gcode* files, manually written or automatically generated by CAM (*Computer Aided Manufacturing*) software.

```
resources
├── input.gcode
├── sampleInputDrawing.gcode
└── shaperInput.shaper
```

temp_files

In this folder are located the temporari swap files used by the IDE for to store the user input (both Shaper and G-Code) or mid-processing data, namely the G-Code specification generated by parsing the Shaper user input.

```
temp_files
├── gcode_temp.txt
└── shaper_temp.txt
```

Javadoc

Please refer to this link in order to browse the javadoc for all the packages, classes, methods and fields.

Installation

GCode-Shaper-IDE v1.0 executable program can be downloaded from the Releases section of this repo ("GCodeShaperIDE.exe").

Docs

Helpful docs with syntax, examples and errors for understanding both *GCODE Parser* and *Shaper* are here provided:

1. G-code Parser docs
2. Shaper docs

Semantic analysis

GCode parser

Block structure

Following the GCode definition, each block of the source code can contain one or more instruction and, each of them, has further informations about the movement, the speed, the tecnology and other possibile details of the CNC machine tool.

Block build

An important difference between the GCode and our "light GCode" is that in the reality all the instruction informations can be listed randomly while in our solution a specified a fix order to be followed is specified. The GCode grammar implemented in this software is based on this assumption and it is built via a bottom-up approach that combine two main methods: the first one is used to create the instruction informations while the other one is used to assembly them in several blocks.

Block ordering check

During the costruction is also verified that the identificator of each block (N###) follows an increasing order (not necessarily sequential) and, after this check, all the components are finally merged in the main object called "Blocks" and built as a SortedMap. It represent the top-chain object which is passed and processed by the GCode parser.

GCode shaper

Shape structure

As done for the GCode specification, a bottom-up approach is applied. All the shape informations are collected (`shape` and `configuration`) and the shape object is created via the `createShape()` method. In this case no lists are used beacuse only one shape is built.

Errors

All errors in G-code Parser and Shaper are here listed:

1. G-code Parser error list
2. Shaper error list

Contributors

- Luca Ghislotti
- Luca Parimbelli
- Andrea Marinò
- Alessandro Mazzola

G-Code parser Overview

G-code Parser is built with the idea of being able to parse a simplified version of the original G-code programming language used to program CNC machines. The are some differences between the real G-code and the G-code which is parsed by this compiler, like:

- G-code Parser requires a stricter order for the directives defined inside the N-blocks (*info_geometriche-info_tecnologiche-info_3M*)
- G-code Parser is not parsing any rules concerning unit of measurement (like G94, G95, G96, G97). The parser is built considering G94 and G97 as default and they can't be changed
- G-code Parser can't deal with arcs which are not exactly equal to 90 degrees

A G-code specification is composed by an infinite number of N-blocks, each of them with same structure. Each block must begin with an **increasing** N-block number and the last block must containt the **M30** directive. The structure of a block is composed by 3 main structures that, if defined, must follow the following order:

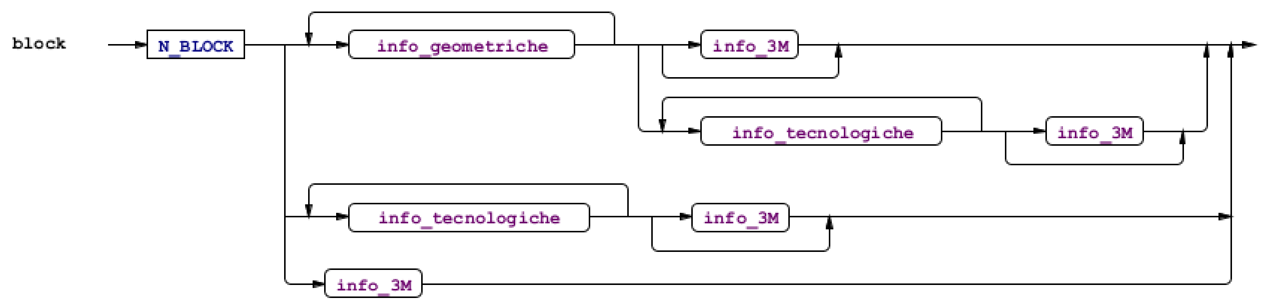
1. *info_geometriche*
2. *info_tecnologiche*
3. *info_3M*

EBNF Notation

```
block ::= N_BLOCK (
            ( info_geometriche )+  (
                                  ( info_3M )?
                                 | ( info_tecnologiche )+ ( info_3M )?
                                )
            | ( info_tecnologiche )+ ( info_3M  )?
            | info_3M
```

)

Syntax Diagram

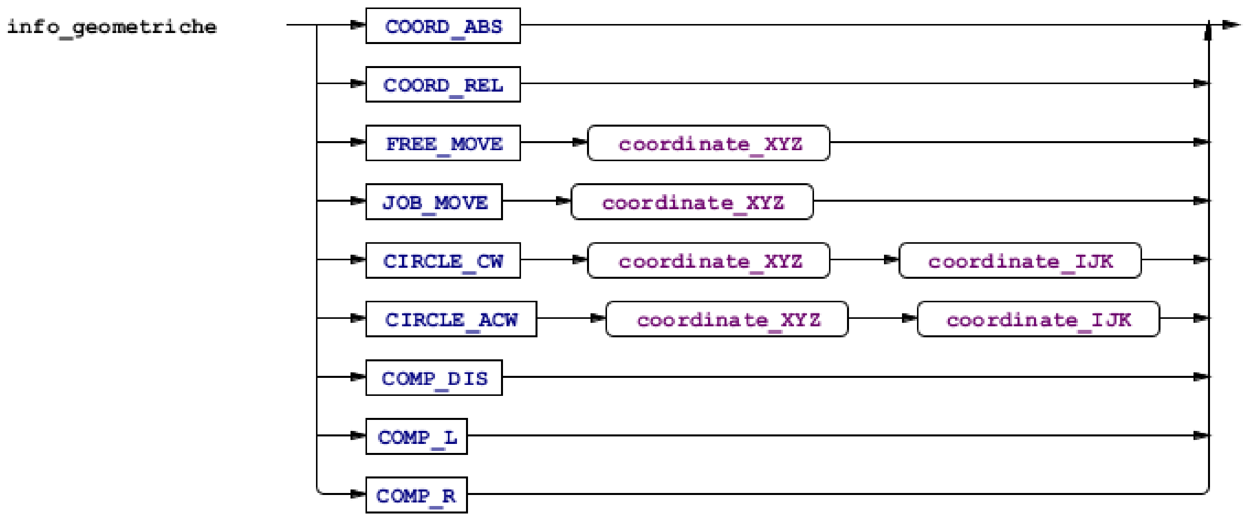


info_geometriche

EBNF Notation

info_geometriche ::= (COORD_ABS
| COORD_REL
| FREE_MOVE coordinate_XYZ
| JOB_MOVE coordinate_XYZ
| CIRCLE_CW coordinate_XYZ coordinate_IJK
| CIRCLE_ACW coordinate_XYZ coordinate_IJK
| COMP_DIS
| COMP_L
| COMP_R
)

Syntax Diagram

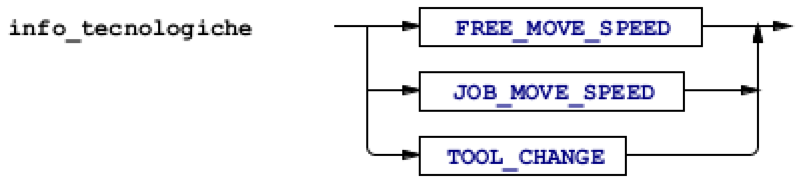


info_tecnologiche

EBNF Notation

info_tecnologiche ::= FREE_MOVE_SPEED
| JOB_MOVE_SPEED
| TOOL_CHANGE

Syntax Diagram

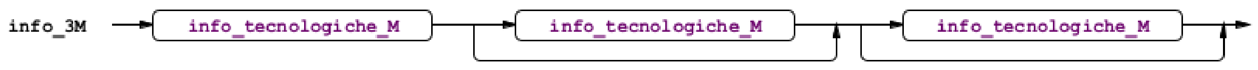


info_3M

EBNF Notation

info_3M::=info_tecnologiche_M (info_tecnologiche_M)? (info_tecnologiche_M)?

Syntax Diagram



info_tecnologiche_M

EBNF Notation

info_tecnologiche_M ::= ROT_TOOL_CW

| ROT_TOOL_ACW

| STOP_TOOL

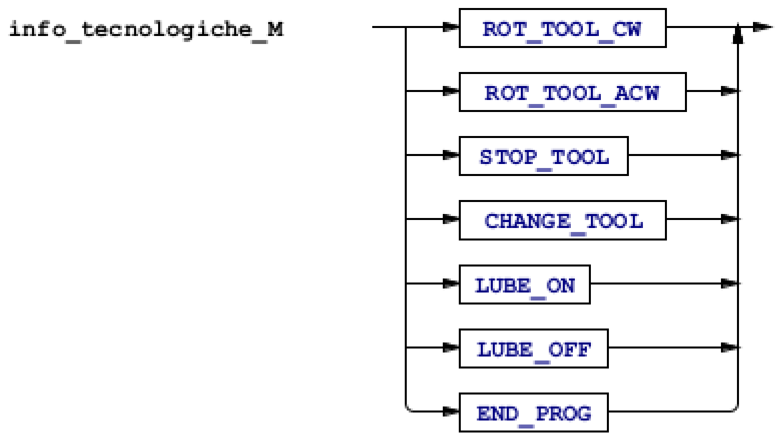
| CHANGE_TOOL

| LUBE_ON

| LUBE_OFF

| END_PROG

Syntax Diagram



Examples

Example 1

G-code Specification

N10 G90 G42 F300 S1000 T0101 M06 M03 M08

N20 G00 X0 Y-10

N30 G01 Y250

N40 G01 X100

N50 G03 X200 Y150 I200 J250

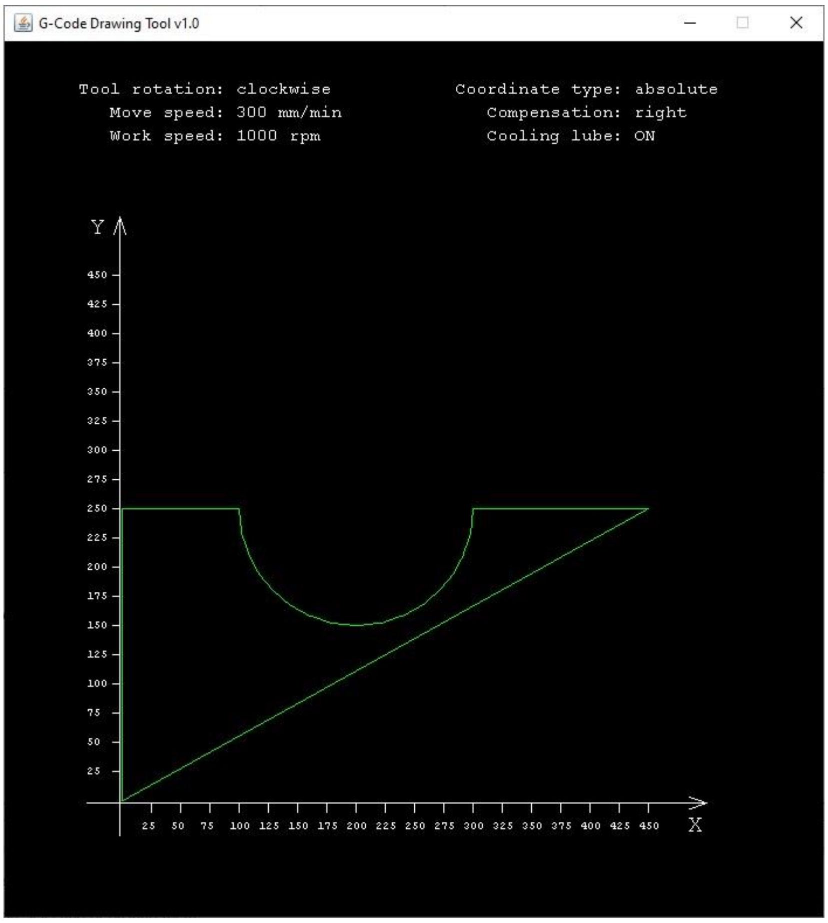
N60 G03 X300 Y250 I200 J250

N70 G01 X450

N80 G01 X0 Y0

N90 G00 X0 Y-10 M05 M09 M30

Output



```
Tool rotation: clockwise           Coordinate type: absolute
   Move speed: 300 mm/min             Compensation: right
   Work speed: 1000 rpm               Cooling lube: ON
```

Example 2

G-code Specification

N0 G90 F100 S100 M03 M08

N1 G00 X0 Y50

N2 G01 X0 Y200

N3 G02 X50 Y250 I50 J200

N4 G01 X200 Y250

N5 G01 X200 Y200

N6 G01 X250 Y200

N7 G01 X250 Y50

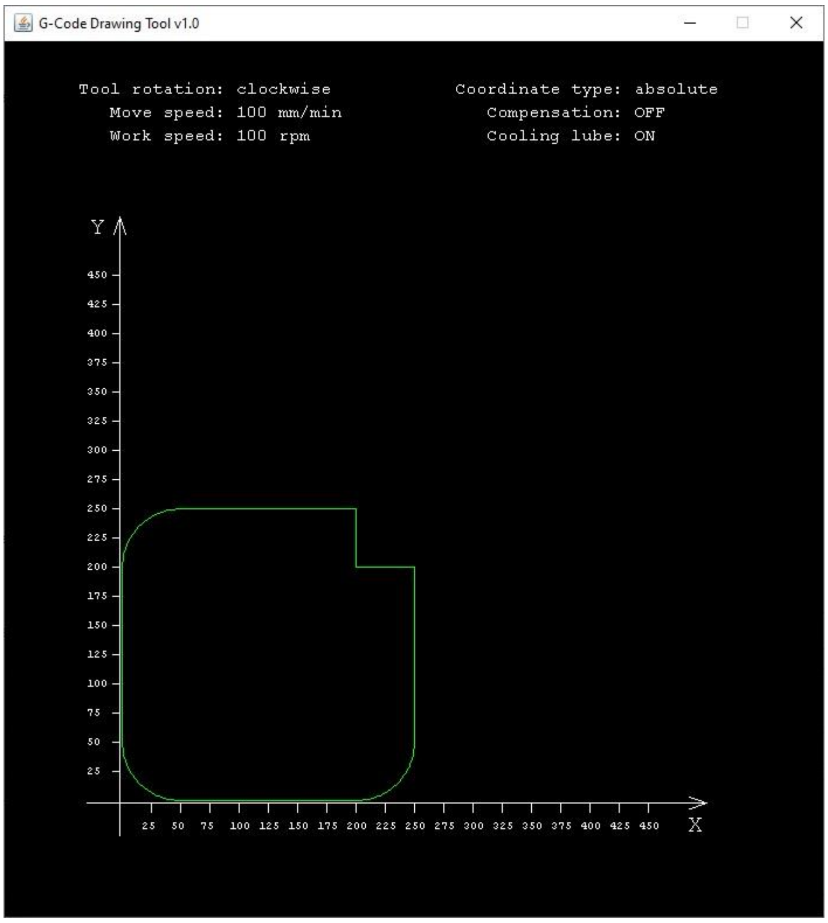
N8 G02 X200 Y0 I200 J50

N9 G01 X50 Y0

N10 G02 X0 Y50 I50 J50

N11 M30

Output



```
Tool rotation: clockwise            Coordinate type: absolute
   Move speed: 100 mm/min              Compensation: OFF
   Work speed: 100 rpm                 Cooling lube: ON
```

Example 3

G-code Specification

N10 G90 G42 T0301 F400 S500 M03 M06 M08

N20 G01 X0 Y300

N30 G01 X100 Y300

N40 G03 X150 Y250 I150 J300

N50 G03 X200 Y300 I150 J300

N60 G01 X300 Y300

N70 G01 X300 Y200

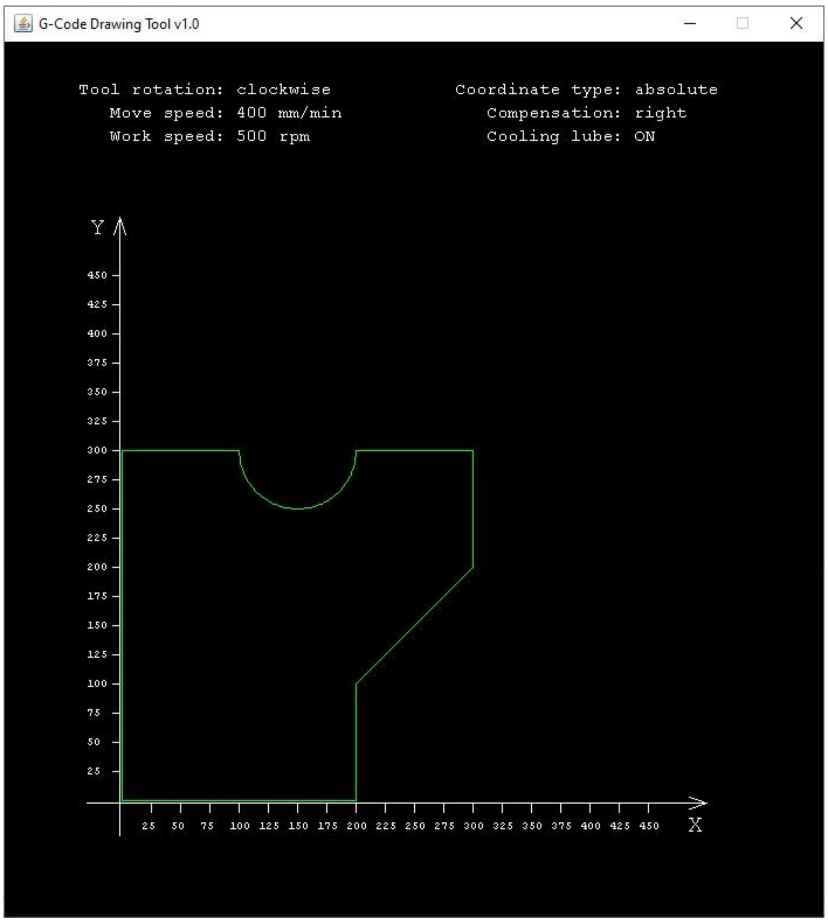
N80 G01 X200 Y100

N90 G01 X200 Y0

N100 G01 X0 Y0

N110 M05 M09 M30

Output



```
Tool rotation: clockwise          Coordinate type: absolute
   Move speed: 400 mm/min            Compensation: right
   Work speed: 500 rpm               Cooling lube: ON
```

References

For the *token list specification* see description tokenList.md file. For the full syntax grammar of GCode Parser check "GCode Parser Syntax Grammar.pdf" file.

List of tokens of the G-CODE Language

Token	**Definition**
Macro	
LETTER	['A'..'Z' \ 'a'..'z']
DIGIT	['0'..'9']
WS	[' ' \ '\t' \ '\r' \ '\n']+
COMMENT	['//' ~('\n'\ '\r')* '\r'? '\n' \ *'/' (options {greedy=false;} : .)* '*/']
Reserved Words	
END_PROG	'M30'
LUBE_OFF	'M09'
LUBE_ON	'M08'
CHANGE_TOOL	'M06'
STOP_TOOL	'M05'
ROT_TOOL_ACW	'M04'
ROT_TOOL_CW	'M03'
TOOL_CHANGE	'T0' ('1' .. '9') '0' ('1' .. '9')
JOB_MOVE_SPEED	'S' ('1' .. '9')(DIGIT)*
FREE_MOVE_SPEED	'F' ('1' .. '9')(DIGIT)*
COMP_R	'G42'
COMP_L	'G41'

Token	Definition
COMP_DIS	'G40'
CIRCLE_ACW	'G03'
CIRCLE_CW	'G02'
JOB_MOVE	'G01'
FREE_MOVE	'G00'
COORD_REL	'G91'
COORD_ABS	'G90'
N_BLOCK	'N' ('0' .. '9')(DIGIT)*
K_CORD	'K'CORD_DIGIT
J_CORD	'J'CORD_DIGIT
I_CORD	'I'CORD_DIGIT
Z_CORD	'Z'CORD_DIGIT
Y_CORD	'Y'CORD_DIGIT
X_CORD	'X'CORD_DIGIT
CORD_DIGIT	('-')?(DIGIT)+
I_CORD	'I'CORD_DIGIT
I_CORD	'I'CORD_DIGIT
I_CORD	'I'CORD_DIGIT

List of errors of G-code Parser

Here are listed all the errors that G-code parser is designed to throw. For more info check docs section.

Error number	**Error name**	**Description**
Lexical errors		
0	SCAN_ERROR	Invalid token
Syntax errors		
1	ERR_ON_SYNTAX	Invalid token order
Semantic errors		
2	BLOCK_NUMBERING_ERROR	Invalid sequence of N_i (N_i must be greater than N_{i-1})
3	NO_M30_ERROR	'M30' token (end program)
4	CHANGE_TOOL_ERROR	'M06' and 'T[][]' are not used together
5	NO_COORDINATE_TYPE_ERROR	'G90' or 'G91' is missing while using 'G00', 'G01', 'G02' or 'G03'
6	NO_SPINDLE_ROTATION_ERORR	'M03' or 'M04' is missing while using 'G01', 'G02' or 'G03'
7	DUPLICATED_COMMAND_ERROR	Duplicated command within a single block
8	END_ROTATION_ERROR	Spindle turned off before being turned on
9	NO_MOVE_SPEED_ERROR	Movement speed 'F0 not defined before command 'G00'
11	NO_JOB_SPEED_ERROR	Working speed 'S' not defined before command 'G01'

12	NO_COORD_TYPE_SPEED_ERROR	Speed 'F' or 'S' defined before setting the ordinate type 'G90' or 'G91'
13	NO_ABS_BEFORE_REL_ERROR	'G91' defined before setting an absolute reference point using 'G90'
14	NOT_90_DEGREE_ERROR	Circular interpolation is not equal to 90 degrees

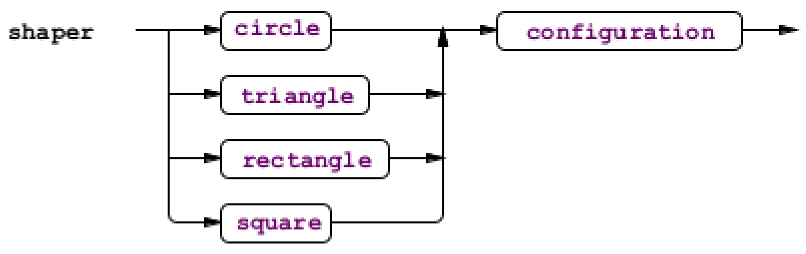
Shaper Parser Overview

A Shaper specification is composed by a *shape* followed by the job *configuration* machine parameters. The *configuration* definition must always be defined in each Shaper directive and it must follow the *shape* definition.

EBNF Notation

```
shaper ::= (  circle
         | triangle
         | rectangle
         | square
           ) configuration
```

Syntax Diagram



Configuration

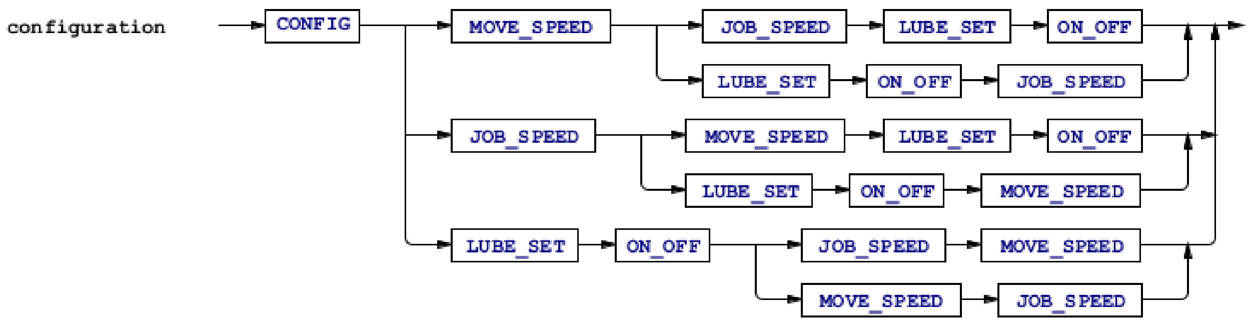
The *configuration* directive is indipendent from the particular *shape* defined and it requires to specify the following parameters:

- *movement speed*, that is the speed associated to the tool while not in use;
- *job speed*, defined as the speed associated to the tool while in use;
- *lube power option*, it is used to set the lube on or off (M08 or M09 G-code directive respectively)

EBNF Notation

```
configuration ::= CONFIG  (
          ( MOVE_SPEED (
                   JOB_SPEED LUBE_SET ON_OFF
               | LUBE_SET ON_OFF JOB_SPEED
              )
          )
          | ( JOB_SPEED (
                 MOVE_SPEED LUBE_SET ON_OFF
               | LUBE_SET ON_OFF MOVE_SPEED
              )
           )
          | ( LUBE_SET ON_OFF (
                  JOB_SPEED MOVE_SPEED
                   | MOVE_SPEED JOB_SPEED
                  )
           )
          )
```

Syntax Diagram



Examples

Examples of *configuration* definition are shown in the shapes paragraph.

Shapes

Shaper currently provides support for 4 different type of *shape* figures:

- circle
- triangle
- rectangle
- square

Circle

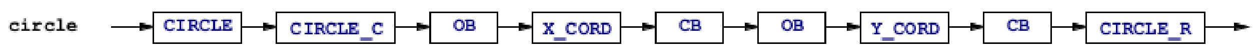
The *circle* command allows to draw a circle in the Cartesian plane with a given center and radius. It requires to specify the following parameters:

- *center coordinates*, that are the spatial coodinates of the circle's center
- *radius*, that is the distance between the center and any point belonging to the circle's circumference

EBNF Notation

```
circle ::= CIRCLE CIRCLE_C
                   OB
               X_CORD
              CB
              OB
               Y_CORD
              CB
         CIRCLE_R
```

Syntax Diagram



Examples

```
CIRCLE C(X250)(Y250) R100
CONFIGURATION MS54 JS10 LB ON
```

Errors

The *circle* command can raise the following *semantic errors*:

- *MAX_COORD_ERROR*: this error can be thrown due to huge radius lenght or due to center coordinates too close to the Cartesian plan limits

```
CIRCLE C(X400)(Y400) R300
CONFIGURATION MS54 JS10 LB ON

** Error list **
1 - Semantic Error (3) at [0, 0]: Found MAX_COORD_ERROR - all coordinates must be positive and lower
than 500 pixel to be displayed
```

Triangle

The *triangle* command allows to draw a triangle in the Cartesian plane with three given points. It requires to specify the following parameters:

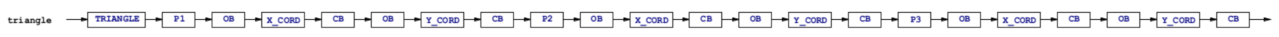
- *first vertex coordinates*, that are the spatial coodinates of the first triangle's vertex
- *second vertex coordinates*, that are the spatial coodinates of the second triangle's vertex
- *third vertex coordinates*, that are the spatial coodinates of the third triangle's vertex

EBNF Notation

```
triangle ::= TRIANGLE P1
           OB
      X_CORD
     CB
     OB
      Y_CORD
     CB
      P2
     OB
```

```
 X_CORD
CB
OB
 Y_CORD
CB
 P3
OB
 X_CORD
CB
OB
 Y_CORD
CB
```

Syntax Diagram



Examples

```
//isosceles triangle
TRIANGLE P1(X100)(Y100)P2(X150)(Y250)P3(X200)(Y100)
CONFIGURATION MS54 JS10 LB ON

//scalene triangle
TRIANGLE P1(X100)(Y100)P2(X150)(Y200)P3(X300)(Y100)
CONFIGURATION MS54 JS10 LB ON

//rectangle triangle
TRIANGLE P1(X100)(Y100)P2(X100)(Y200)P3(X300)(Y100)
CONFIGURATION MS54 JS10 LB ON
```

Errors

The *triangle* command can raise the following *semantic errors*:

- *MAX_COORD_ERROR*: this error can be thrown due to point coordinates too close to the Cartesian plan limits

```
TRIANGLE P1(X400)(Y400)P2(X650)(Y650)P3(X600)(Y400)
CONFIGURATION MS54 JS10 LB ON
```

```
 ** Error list **
 1 - Semantic Error (3) at [0, 0]: Found MAX_COORD_ERROR - all coordinates must be positive and lower
than 500 pixel to be displayed
```

Rectangle

The *rectangle* command allows to draw a rectangle in the Cartesian plane with three given points. It requires to specify the following parameters:

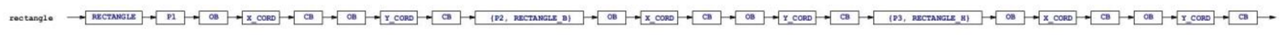
- *first point coordinates*, that are the spatial coordinates of the first rectangle's point
- *second point coordinates*, that are the spatial coordinates of the second rectangle's point
- *third point coordinates*, that are the spatial coordinates of the third rectangle's point

EBNF Notation

```
rectangle ::= RECTANGLE P1
       OB
        X_CORD
       CB
       OB
          Y_CORD
       CB
      ( P2 | RECTANGLE_B )
       OB
        X_CORD
         CB
       OB
        Y_CORD
         CB
      ( P3 | RECTANGLE_H )
         OB
        X_CORD
         CB
       OB
        Y_CORD
```

CB

Syntax Diagram



Examples

```
//rectangle generated by bottom-left vertex
RECTANGLE P1(X100)(Y200) P2(X400)(Y200) P3(X100)(Y400)
CONFIGURATION MS54 JS56 LB ON

//rectangle generated by bottom-right vertex
RECTANGLE P1(X400)(Y200) P2(X400)(Y400) P3(X100)(Y200)
CONFIGURATION MS54 JS56 LB ON

//oblique rectangle generated by top vertex
RECTANGLE P1(X300)(Y200) P2(X250)(Y250) P3(X200)(Y100)
CONFIGURATION MS54 JS56 LB ON

//oblique rectangle generated by bottom vertex
RECTANGLE P1(X200)(Y100) P2(X300)(Y200) P3(X150)(Y150)
CONFIGURATION MS54 JS56 LB ON
```

Errors

The *rectangle* command can raise the following *semantic errors*:

- *MAX_COORD_ERROR*: this error can be thrown due to point coordinates too close to the Cartesian plan limits
- *NOT_RECT_PERP_ERROR*: this error can be thrown due to non-perpendicular shape's sides

```
RECTANGLE P1(X400)(Y200) P2(X500)(Y200) P3(X600)(Y500)
CONFIGURATION MS54 JS56 LB ON
```

** Error list **

```
1 - Semantic Error (3) at [0, 0]: Found MAX_COORD_ERROR - all coordinates must be positive and lower than 500 pixel to be displayed
2 - Semantic Error (4) at [0, 0]: Found NOT_RECT_PERP_ERROR - sides of the rectangle must be perpendicular
```

Square

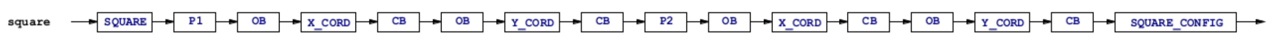
The *square* command allows to draw a square in the Cartesian plane with three given points. It requires to specify the following parameters:

- *first point coordinates*, that are the spatial coordinates of the first rectangle's point
- *second point coordinates*, that are the spatial coordinates of the second rectangle's point
- *square orientation*, that is the spatial orientation of the shape (UP, DOWN)

EBNF Notation

```
square ::= SQUARE P1
        OB
         X_CORD
          CB
          OB
         Y_CORD
          CB
         P2
        OB
         X_CORD
        CB
        OB
         Y_CORD
        CB
         SQUARE_CONFIG
```

Syntax Diagram



Examples

//up square
SQUARE P1(X150)(Y150) P2(X350)(Y150) CONFIG UP
CONFIGURATION MS54 JS56 LB ON

//down square
SQUARE P1(X150)(Y350) P2(X350)(Y350) CONFIG DOWN
CONFIGURATION MS54 JS56 LB

//oblique suqare
SQUARE P1(X150)(Y150) P2(X300)(Y200) CONFIG UP
CONFIGURATION MS54 JS56 LB ON

Errors

The *square* command can raise the following *semantic errors*:

- *MAX_COORD_ERROR*: this error can be thrown due to point coordinates too close to the Cartesian plan limits
- SQUARE P1(X150)(Y350) P2(X350)(Y350) CONFIG UP
- CONFIGURATION MS54 JS56 LB ON
- ** Error list **
- 1 - Semantic Error (3) at [0, 0]: Found MAX_COORD_ERROR - all coordinates must be positive and lower than 500 pixel to be displayed

References

For the *token list specification* see description tokenList.md file. For the full syntax grammar of Shaper metalanguage check "Shaper Parser Syntax Grammar.pdf" file.

List of tokens of the Shaper Language

Token	**Definition**			
Macro				
LETTER	['A'..'Z' \	'a'..'z']		
DIGIT	['0'..'9']			
WS	[' ' \	'\t' \	'\r' \	'\n']+
COMMENT	['//' ~('\n'\	'\r')* '\r'? '\n' \ '/' *(options {greedy=false;} : .)* '*/']		
OB	'('			
CB	')'			
Reserved Words				
ON_OFF	'ON' \	'OFF'		
LUBE_SET	'LB'			
JOB_SPEED	'JS' DIGIT+			
MOVE_SPEED	'MS' DIGIT+			
CONFIG	'CONFIGURATION'			
Y_CORD	'Y' DIGIT+			
X_CORD	'X' DIGIT+			
SQUARE_CONFIG	'UP' \	'DOWN'		
P1	'P1'			
P2	'P2'			
P3	'P3'			
RECTANGLE_H	'H'			

Token	**Definition**
RECTANGLE_B	'B'
RECTANGLE_P	'P'
SQUARE_L	'L'
CIRCLE_R	'R' DIGIT+
CIRCLE_C	'C'
TRIANGLE	'TRIANGLE'
RECTANGLE	'RECTANGLE'
SQUARE	'SQUARE'
CIRCLE	'CIRCLE'

List of errors of Shaper Metalanguage

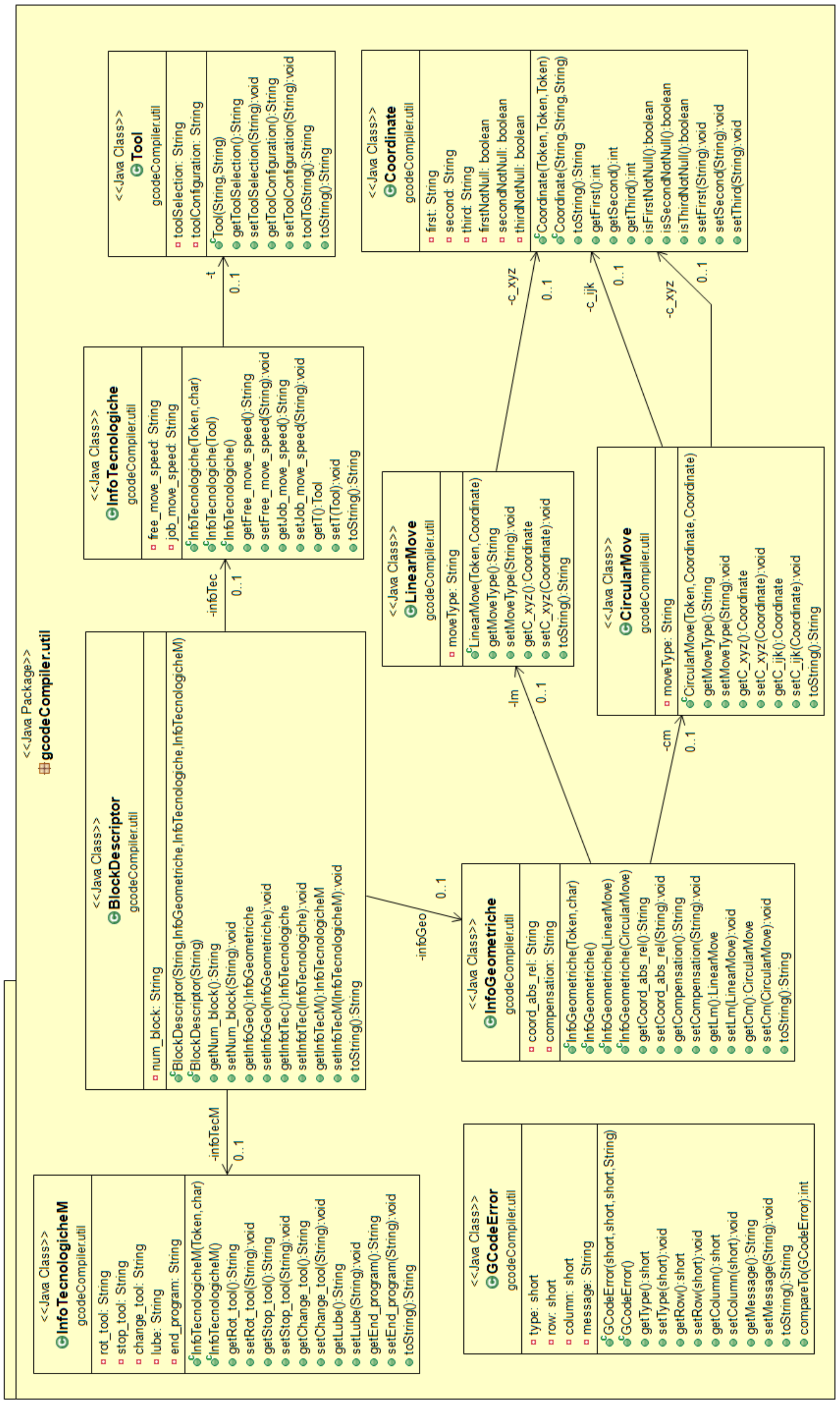
Here are listed all the errors that Shaper parser is designed to throw. For more info check docs section.

Error number	Error name	Description
Lexical errors		
0	SCAN_ERROR	Invalid token
Syntax errors		
1	ERR_ON_SYNTAX	Invalid token order
Semantic errors		
3	MAX_COORD_ERROR	Input coordinates does not respect X-Y axis limits
4	NOT_RECT_PERP_ERROR	Rectangle sides are not perpendicular

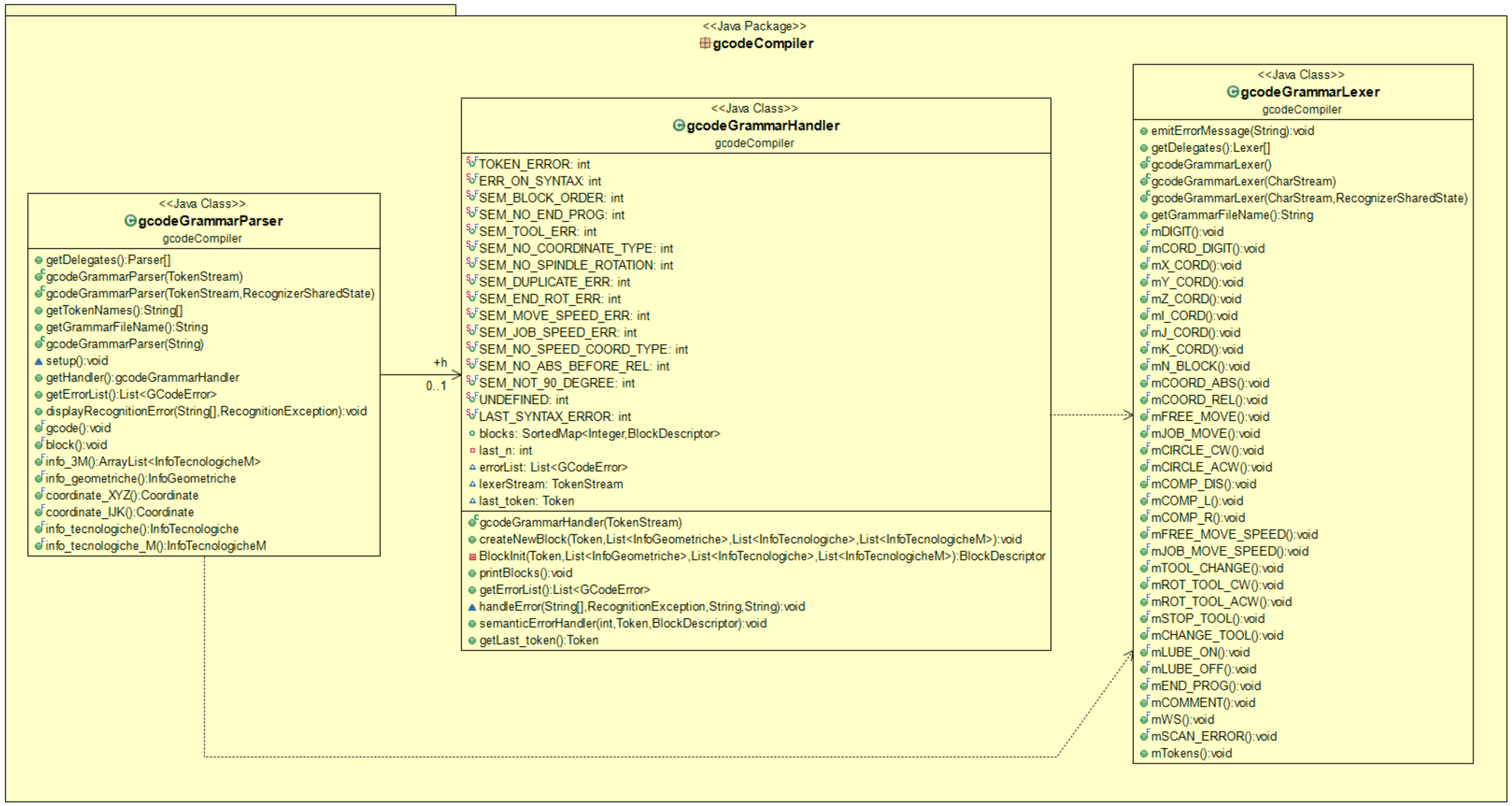
Attachments

In this section are provided the complete project package diagram and all class diagrams for each package.

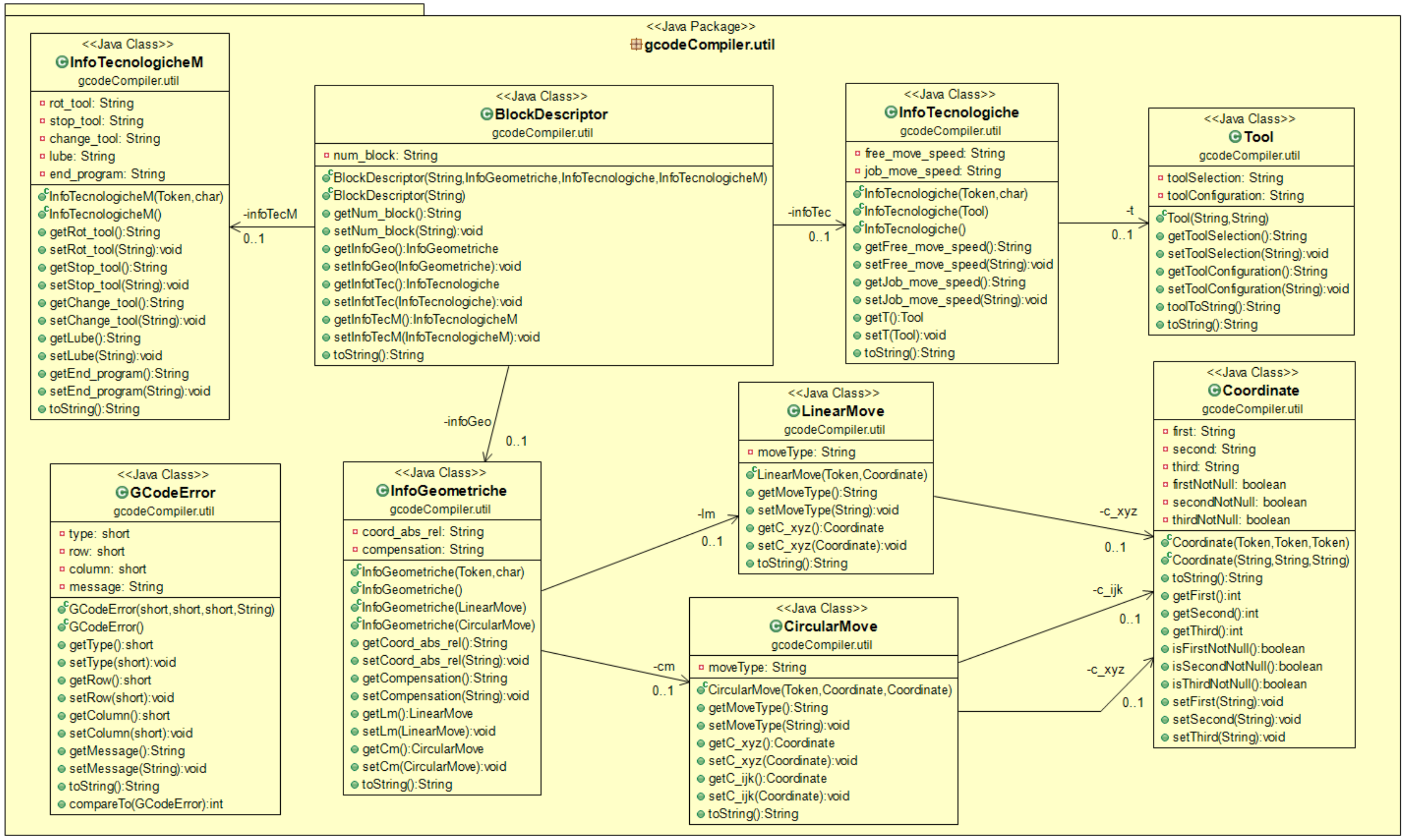
Package diagram



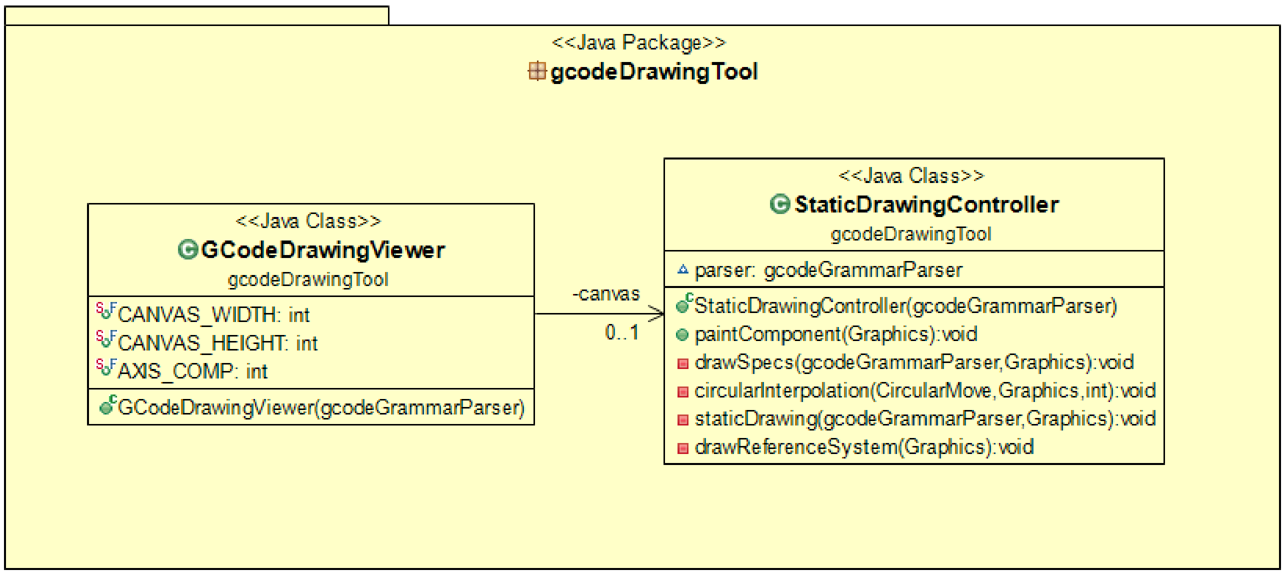
gCodeCompiler class diagram


<<Java Package>>
gcodeCompiler
<<Java Class>>
gcodeGrammarParser
gcodeCompiler
getDelegates():Parser[]
gcodeGrammarParser(TokenStream)
gcodeGrammarParser(TokenStream,RecognizerSharedState)
getTokenNames():String[]
getGrammarFileName():String
gcodeGrammarParser(String)
setup():void
getHandler():gcodeGrammarHandler
getErrorList():List<GCodeError>
displayRecognitionError(String[],RecognitionException):void
gcode():void
block():void
info_3M():ArrayList<InfoTecnologicheM>
info_geometriche():InfoGeometriche
coordinate_XYZ():Coordinate
coordinate_IJK():Coordinate
info_tecnologiche():InfoTecnologiche
info_tecnologiche_M():InfoTecnologicheM
+h
0..1
<<Java Class>>
gcodeGrammarHandler
gcodeCompiler
TOKEN_ERROR: int
ERR_ON_SYNTAX: int
SEM_BLOCK_ORDER: int
SEM_NO_END_PROG: int
SEM_TOOL_ERR: int
SEM_NO_COORDINATE_TYPE: int
SEM_NO_SPINDLE_ROTATION: int
SEM_DUPLICATE_ERR: int
SEM_END_ROT_ERR: int
SEM_MOVE_SPEED_ERR: int
SEM_JOB_SPEED_ERR: int
SEM_NO_SPEED_COORD_TYPE: int
SEM_NO_ABS_BEFORE_REL: int
SEM_NOT_90_DEGREE: int
UNDEFINED: int
LAST_SYNTAX_ERROR: int
blocks: SortedMap<Integer,BlockDescriptor>
last_n: int
errorList: List<GCodeError>
lexerStream: TokenStream
last_token: Token
gcodeGrammarHandler(TokenStream)
createNewBlock(Token,List<InfoGeometriche>,List<InfoTecnologiche>,List<InfoTecnologicheM>):void
BlockInit(Token,List<InfoGeometriche>,List<InfoTecnologiche>,List<InfoTecnologicheM>):BlockDescriptor
printBlocks():void
getErrorList():List<GCodeError>
handleError(String[],RecognitionException,String,String):void
semanticErrorHandler(int,Token,BlockDescriptor):void
getLast_token():Token
<<Java Class>>
gcodeGrammarLexer
gcodeCompiler
emitErrorMessage(String):void
getDelegates():Lexer[]
gcodeGrammarLexer()
gcodeGrammarLexer(CharStream)
gcodeGrammarLexer(CharStream,RecognizerSharedState)
getGrammarFileName():String
mDIGIT():void
mCORD_DIGIT():void
mX_CORD():void
mY_CORD():void
mZ_CORD():void
mI_CORD():void
mJ_CORD():void
mK_CORD():void
mN_BLOCK():void
mCOORD_ABS():void
mCOORD_REL():void
mFREE_MOVE():void
mJOB_MOVE():void
mCIRCLE_CW():void
mCIRCLE_ACW():void
mCOMP_DIS():void
mCOMP_L():void
mCOMP_R():void
mFREE_MOVE_SPEED():void
mJOB_MOVE_SPEED():void
mTOOL_CHANGE():void
mROT_TOOL_CW():void
mROT_TOOL_ACW():void
mSTOP_TOOL():void
mCHANGE_TOOL():void
mLUBE_ON():void
mLUBE_OFF():void
mEND_PROG():void
mCOMMENT():void
mWS():void
mSCAN_ERROR():void
mTokens():void

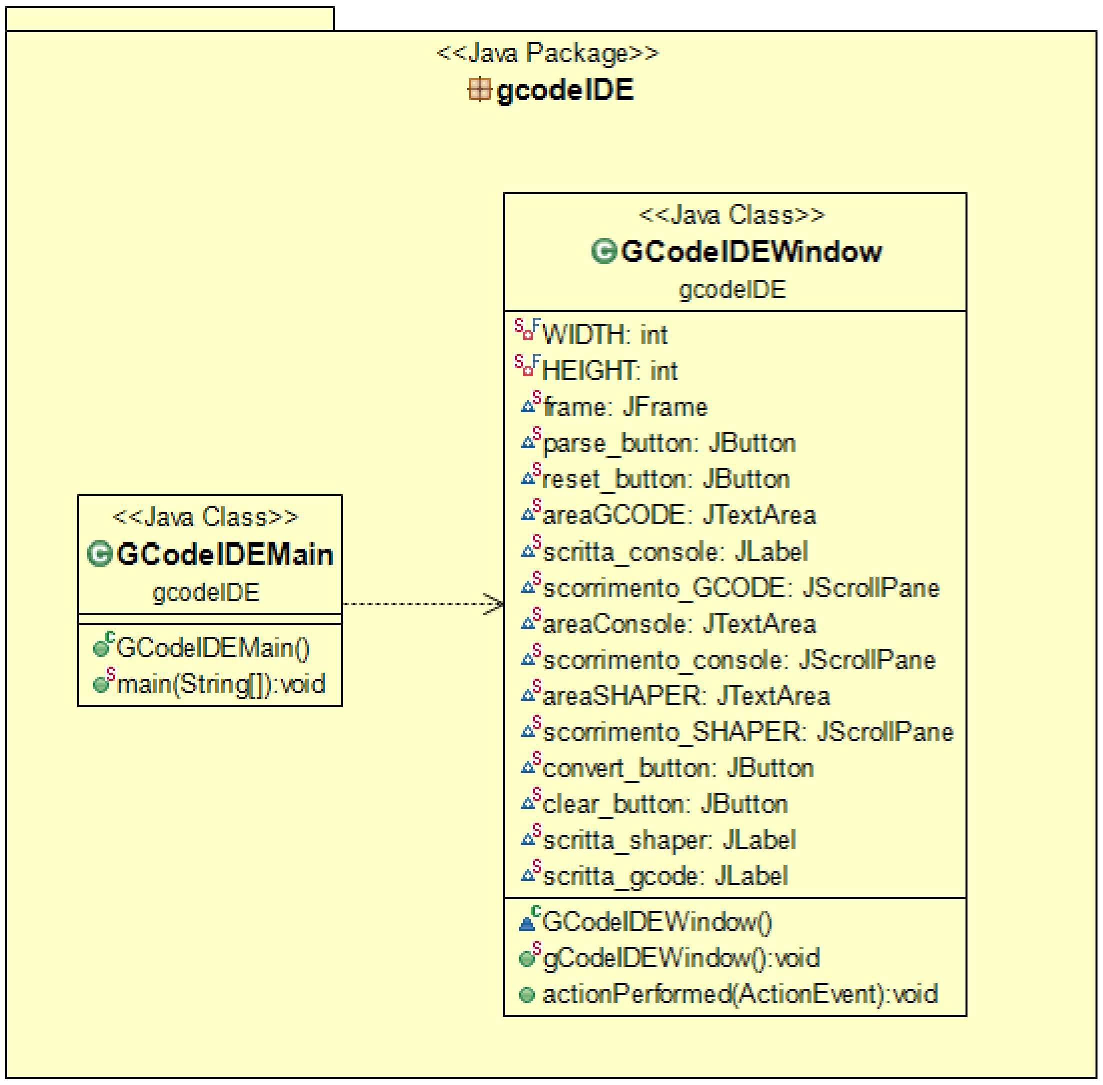
gCodeCompiler.util class diagram



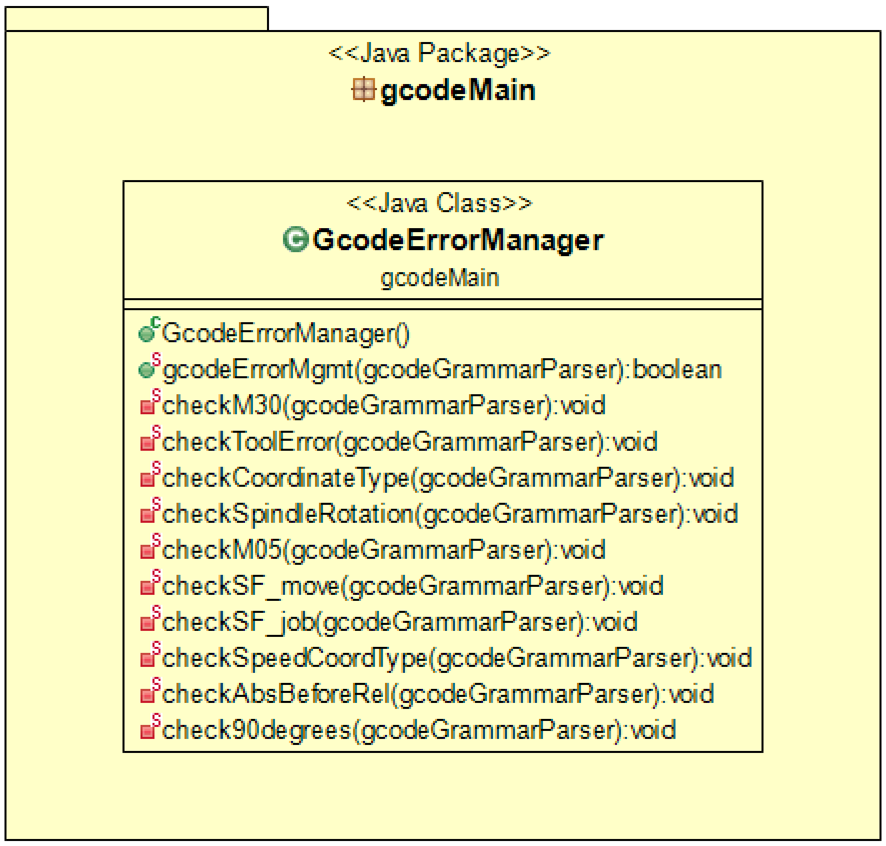
gCodeDrawingTool class diagram


<<Java Package>>
gcodeDrawingTool
<<Java Class>>
GCodeDrawingViewer
gcodeDrawingTool
CANVAS_WIDTH: int
CANVAS_HEIGHT: int
AXIS_COMP: int
GCodeDrawingViewer(gcodeGrammarParser)
-canvas
0..1
<<Java Class>>
StaticDrawingController
gcodeDrawingTool
parser: gcodeGrammarParser
StaticDrawingController(gcodeGrammarParser)
paintComponent(Graphics):void
drawSpecs(gcodeGrammarParser,Graphics):void
circularInterpolation(CircularMove,Graphics,int):void
staticDrawing(gcodeGrammarParser,Graphics):void
drawReferenceSystem(Graphics):void

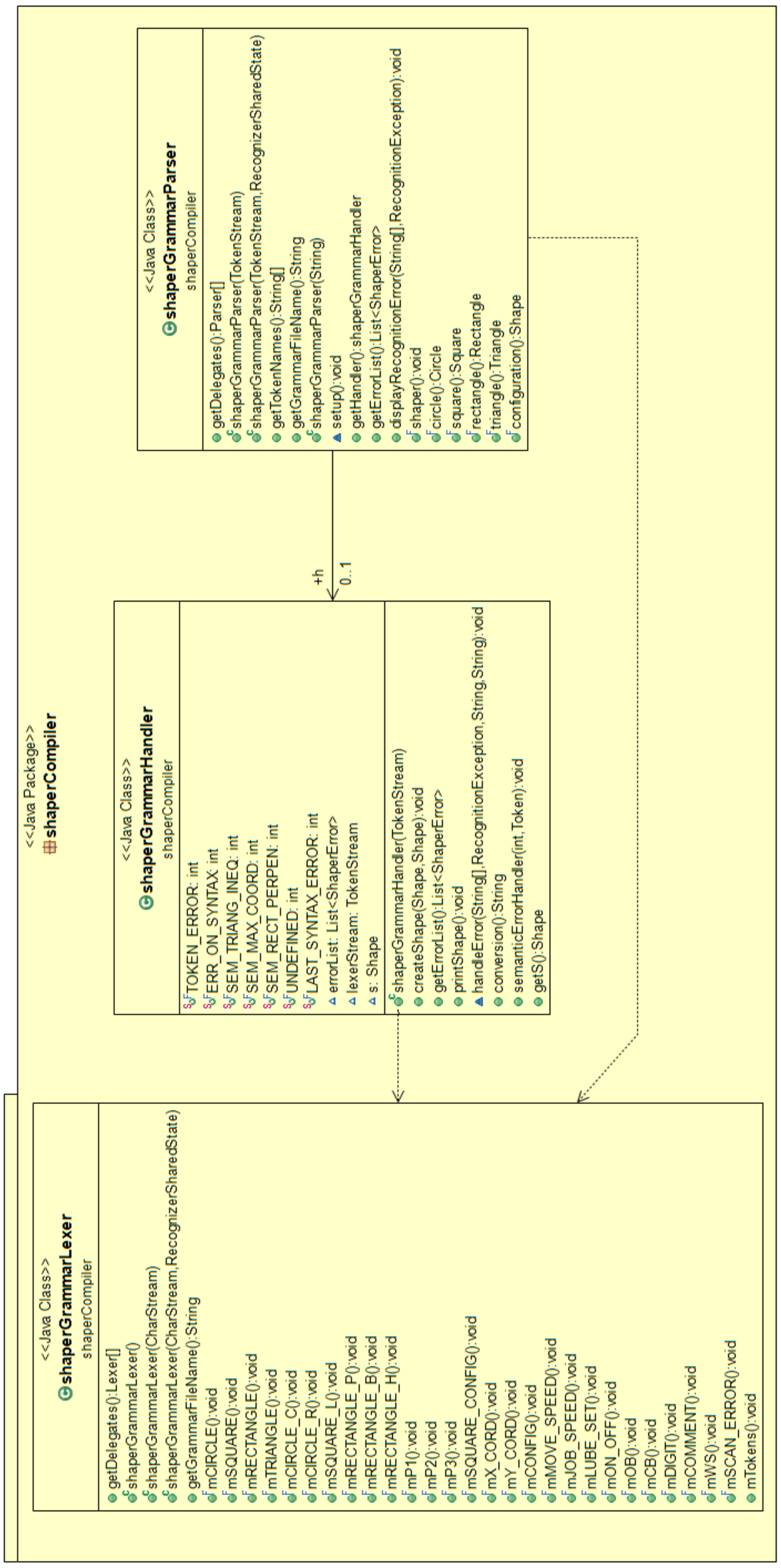
gCodeIDE class diagram



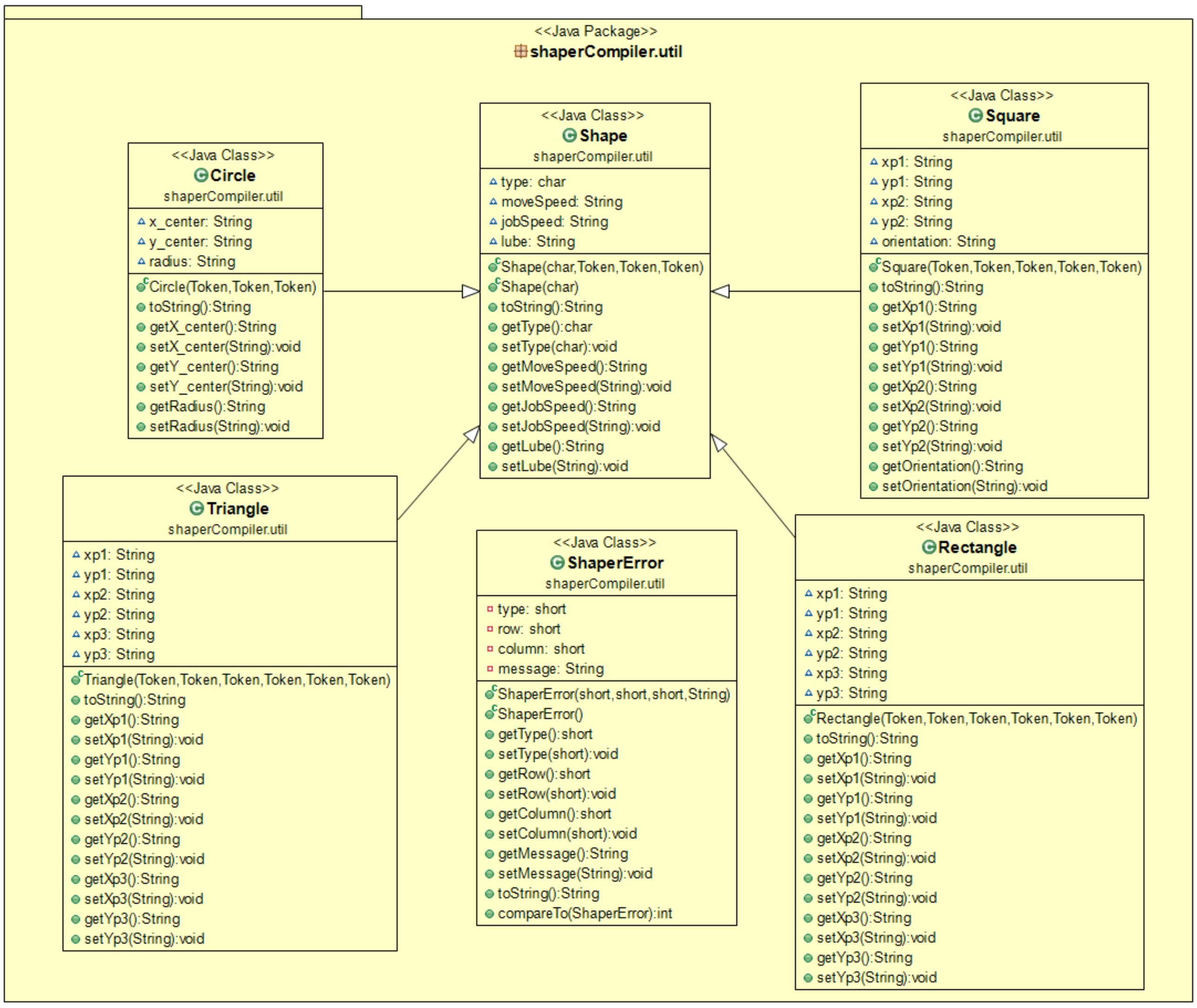
gCodeMain class diagram


<<Java Package>>
gcodeMain
<<Java Class>>
GcodeErrorManager
gcodeMain
GcodeErrorManager()
gcodeErrorMgmt(gcodeGrammarParser):boolean
checkM30(gcodeGrammarParser):void
checkToolError(gcodeGrammarParser):void
checkCoordinateType(gcodeGrammarParser):void
checkSpindleRotation(gcodeGrammarParser):void
checkM05(gcodeGrammarParser):void
checkSF_move(gcodeGrammarParser):void
checkSF_job(gcodeGrammarParser):void
checkSpeedCoordType(gcodeGrammarParser):void
checkAbsBeforeRel(gcodeGrammarParser):void
check90degrees(gcodeGrammarParser):void

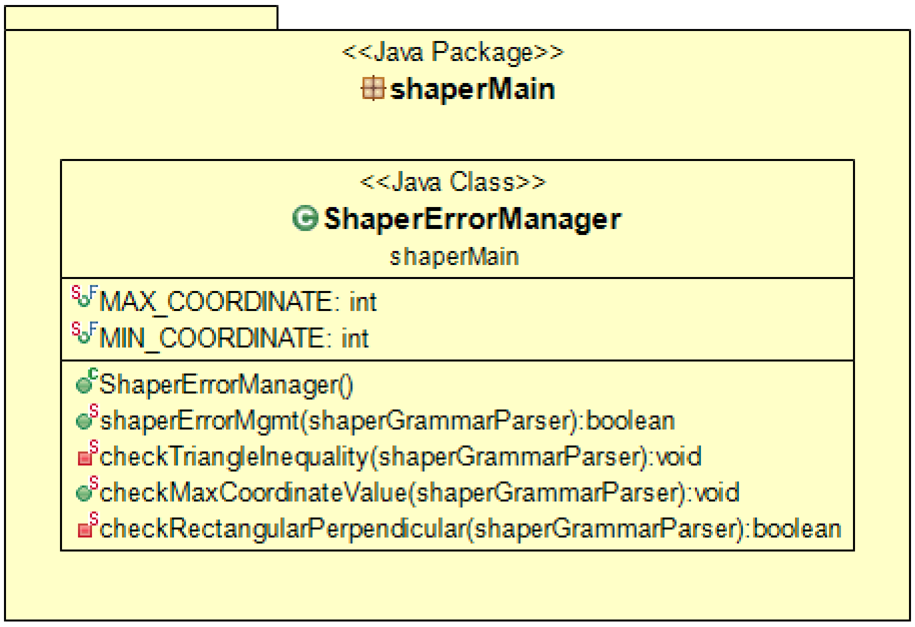
shaperCompiler class diagram



shaperCompilerUtil class diagram



shaperMain class diagram


<<Java Package>>
shaperMain
<<Java Class>>
ShaperErrorManager
shaperMain
MAX_COORDINATE: int
MIN_COORDINATE: int
ShaperErrorManager()
shaperErrorMgmt(shaperGrammarParser):boolean
checkTriangleInequality(shaperGrammarParser):void
checkMaxCoordinateValue(shaperGrammarParser):void
checkRectangularPerpendicular(shaperGrammarParser):boolean