# Big Java Generator ReadMe

## Overview

Big Java Generator (B.J.G., pronounced “big”) is a source code generation tool written in Java. It can function with any programming language, and is a text-to-text command line application.

BJG functions by searching for specifically formatted text which serve as anchor points for modification. These are called flags, and their format can be specified in any way in accordance with the language-independent  requirement.

In fact, anything can be generated, not just source code.

## How It Works

Everything in BJG works from the command line, so its behavior is based on the parameters it receives to it’s main method. A single input file is given and an output file is produced (see the documentation on commands to see more details. The input file is read in and searched for flags. These flags are formatted by `[prefix][long ID][suffix]`. An example will be `/*__10__*/`. And in fact, if no specific prefix and suffix are given, the prefix for flags defaults to `/*__`, and the suffix defaults to `__*/`.

For each of these flags, you can provide a task to occur on it. The tasks that can be performed are `Prepend`, `Replace`, `Remove`, `Insert`, and `Append`. Each of these takes a specific number of arguments, the types and formatting of which is explained below.

## Commands

Below are all commands you can pass to the application. Those in a code typeface `such as this` are to be input directly as they appear. The various inputs to a command are separated by spaces.

| Argument Name | Argument Desc. | Argument Input | Input 1 | Input 2 | Input 3 |
|---|---|---|---|---|---|
| Input File | File path to the input file | `-in`, `-input` | Absolute or local file path | | |
| Output File | File path for output file | `-o`, `-output` | Absolute or local file path | | |
| Replace Token | Replaces a token with something else | `-rp`, `-replace` | ID of token to replace | String to replace it with | |
| Remove Token | Removes a token | `-rm`, `-remove` | ID of a token to remove | | |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| Prepend To Token | Prepends some text to a token | `-p` , `-prepend` | ID of a token to prepend to | String to prepend |  |
| Insert Into Token | Inserts some text into a token | `-i` , `-insert` | ID of a token to insert to | Offset to insert from the first character of the token | String to insert |
| Append To Token | Appends some text to the end of a token | `-a` , `-append` | ID of a token to append to | String to append |  |
| License | Appends a license text to the top of the output file | `-l` , `-license` | Text to append to the top of the output file |  |  |
| Note | Appends some text to the top of the output file, below the license | `-n` , `-note` | Text to append to the top of the output file but below the license |  |  |
| File Type | File type added to output file | `-f` , `-filetype` | String representing a file type |  |  |
| Set Flag Prefix | Sets the prefix to look for when identifying flags in text | `-sfp` , `-setflagprefix` | String containing no spaces to search for as the flag prefix |  |  |
| Set Flag Suffix | Sets the suffix to look for when identifying flags in text | `-sfs` , `-setflagsuffix` | String containing no spaces to search for as the flag suffix |  |  |

## Example

In the ‘in’ folder an example file is found called `Referencer.java`. An empty folder called ‘out’ is found along side the ‘in’ folder.

To test the program, run the program with the arguments:

```
-in in/Referencer.java -license "/* Licensed Under MIT No Attribution. */" -f .java -rm 2 -rp 3 int -rp 4 0;
-o out/IntReferencer -rp 5 value; -rp 6 "== value;" -i 7 1 "Volatile " -a 8 Int -a 9 Int
```

After it runs, look in the out folder and you should see a new file called `IntReferencer.java`.