

```

1  /*
2   * Copyright (c) 2005 Perception Software. All Rights Reserved.
3   */
4  package jgsl;
5
6
7  import org.apache.log4j.BasicConfigurator;
8  import org.apache.log4j.Logger;
9  import org.apache.log4j.PropertyConfigurator;
10
11 import java.io.File;
12 import java.net.URL;
13
14 import jgsl.controller.script.ScriptEngine;
15
16 /**
17  * JGSL program main class. This is main entry point for executing the JGSL in both command line and GUI modes. This
18  * class configures 2 loggers using the Log4J API. The Log4J properties file is bundled with the jgsl.jar
19  * file and is located in the <pre>jgsl/resources/jgsl_log.prop</pre> file.
20  * <p/>
21  * After the loggers are configured the control flow is passed to the ScriptEngine class.
22  *
23  * @author zenarchitect
24  * @version $Id: JGSL.java,v 1.8 2005/05/16 00:54:23 zenarchitect Exp $
25  */
26 public class JGSL {
27     static Logger jgslLogger;
28     static Logger sysLogger;
29
30     /**
31      * @link aggregationByValue
32      * @directed
33      */
34     /*# ScriptEngine lnkScriptEngine; */
35
36     public static void main(String[] args) {
37         try {
38             File logFileDir = new File(System.getProperty("user.home") + File.separator + ".jgsl" + File.separator +
"logs");
39             logFileDir.mkdirs();
40             URL props = Thread.currentThread().getContextClassLoader().getResource("jgsl/resources/jgsl_log.prop");
41             if (props != null) {
42                 PropertyConfigurator.configure(props);
43             } else {
44                 BasicConfigurator.configure();
45             }
46             jgslLogger = Logger.getLogger("jgsl_log");
47             sysLogger = Logger.getLogger("jgsl_sys_log");

```

```
48
49     jgslLogger.info("Starting JGSL");
50     ScriptEngine se = new ScriptEngine();
51     se.processCommandLine(args);
52     jgslLogger.info("Ending JGSL");
53
54     } catch (Throwable t) {
55         jgslLogger.error("JGSL Error\n", t);
56     }
57
58 }
59 }
```

```

1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.controller.script;
5
6  import org.apache.commons.cli.CommandLine;
7  import org.apache.commons.cli.CommandLineParser;
8  import org.apache.commons.cli.HelpFormatter;
9  import org.apache.commons.cli.Option;
10 import org.apache.commons.cli.Options;
11 import org.apache.commons.cli.ParseException;
12 import org.apache.commons.cli.PosixParser;
13 import org.apache.log4j.Level;
14 import org.apache.log4j.Logger;
15
16 import java.io.File;
17 import java.io.FileNotFoundException;
18 import java.io.FileWriter;
19 import java.io.IOException;
20 import java.util.Arrays;
21 import java.util.ResourceBundle;
22
23 import jgsl.io.ScriptParser;
24 import jgsl.io.ScriptParserException;
25 import jgsl.model.JGSLScript;
26 import jgsl.util.JarPackager;
27 import jgsl.util.JarPackagerException;
28 import jgsl.view.swing.JGSLSwingFrame;
29 import jgsl.view.swing.SwingScriptViewer;
30
31 /**
32 * The ScriptEngine class is the controller for the JGSL application. It contains the command line processor for the
33 * command console interface. The *Interactive methods are the controller interfaced for the Interactive GUI.
34 *
35 * @author zenarchitect
36 * @version $Id: ScriptEngine.java,v 1.8 2005/05/21 01:42:06 zenarchitect Exp $
37 */
38 public class ScriptEngine {
39     static Logger jgslLogger = Logger.getLogger("jgsl_log");
40     static Logger sysLogger = Logger.getLogger("jgsl_sys_log");
41
42     // create Options object
43     private Options options = new Options();
44
45     /** @link dependency */
46     /**# JGSLScript lnkJGSLScript; */
47
48     /** @link dependency */

```

```

49  /*# ScriptEngineException lnkScriptEngineException; */
50
51  /** @link aggregationByValue
52  * @directed*/
53  /*# ScriptParser lnkScriptParser; */
54
55  /** @link aggregationByValue
56  * @directed*/
57  /*# SwingScriptViewer lnkSwingScriptViewer; */
58
59  /** @directed
60  * @link aggregation*/
61  /*# JGSLScript lnkJGSLScript1; */
62
63  /**
64  * Parse the script supplied in fileName and return the JGSLScript containing the JGSL object model for the script.
65  *
66  * @param fileName Name of the JGSL script file
67  * @return JGSLScript object containing the script object model
68  * @throws ScriptParserException If a problem is encountered during parsing a ScriptParser exception will be
69  * thrown.
70  * @see jgsl.model.JGSLScript
71  */
72  public JGSLScript parseInteractive(File fileName) throws ScriptParserException {
73      ScriptParser sp = new ScriptParser();
74      JGSLScript script = sp.execScript(fileName);
75      return script;
76  }
77
78  /**
79  * This method will parse the script contained in fileName and then display the result in the JGSL viewer.
80  *
81  * @param fileName Name of the JGSL script file
82  * @return JGSLScript object containing the script object model
83  * @throws ScriptParserException If a problem is encountered during parsing a ScriptParser exception will be
84  * thrown.
85  * @see jgsl.model.JGSLScript
86  */
87  public JGSLScript viewInteractive(File fileName, String saveToFileName) throws ScriptParserException {
88      JGSLScript script = parseInteractive(fileName);
89      String fullClassName = script.generateImplementation();
90      if (fullClassName != null) {
91          SwingScriptViewer ssv = new SwingScriptViewer();
92          ssv.renderScript(fullClassName, saveToFileName);
93      } else {
94          sysLogger.debug("Unable to generated implementation, full class name for file: " + fileName.getAbsolutePath
95  ());
96          throw new ScriptParserException("Unable to generated implementation, full class name for file: " + fileName.
97  getAbsolutePath());
96      }
97      return script;

```

```

98     }
99
100    /**
101     * The method will parse the script contained in scriptFileName and then create an executable JAR file with name of
102     * jarFileName containing the Java class for the JGSL script.
103     *
104     * @param scriptFileName Name of JGSL script to create the JAR from.
105     * @param jarFileName Name of JAR file to generate
106     * @return JGSLScript object containing the script object model
107     * @throws ScriptParserException If a problem is encountered during parsing a ScriptParser exception will be
108     *         thrown.
109     * @throws ScriptEngineException If a problem occurs during the creation of the JAR file.
110     * @see jgsl.model.JGSLScript
111     */
112    public JGSLScript jarInteractive(File scriptFileName, File jarFileName) throws ScriptEngineException,
ScriptParserException {
113        JGSLScript script = parseInteractive(scriptFileName);
114        script.generateImplementation();
115        try {
116            JarPackager.makeJar(jarFileName, script.getClassFileName(), script.getFullClassName());
117        } catch (JarPackagerException e) {
118            sysLogger.error(e.getMessage(), e);
119            throw new ScriptEngineException("Unable to generated JAR: " + scriptFileName.getAbsolutePath());
120        }
121        return script;
122    }
123
124    /**
125     * Process the command line arguments and perform the requested actions. The set of available options is listed
126     * below.
127     * <p/>
128     * <pre>
129     * usage: jgsl.JGSL
130     * -d,--doc jgsl script filt script doc file  Generate script documenation
131     * -e,--exec script file                      Execute the script file
132     * -h,--help                                  Print this message
133     * -j,--jar jgsl script file JAR file        Generate JAR file for script
134     * -l,--logLevel user log level              Set user logging level
135     *                                           to one of: LOG, DEBUG, ERROR, WARNING
136     * -p,--parse script file                    Parse the script file and print the results
137     * -s,--sysLogLevel system log level         Set system logging
138     *                                           level to one of: LOG, DEBUG, ERROR, WARNIN
139     * -v,--view Type of viewer, supports: swing Parse, execute and view the script
140     * </pre>
141     *
142     * @param args Array of String references to valid arguments
143     * @throws ScriptParserException If parsing is requested and problem is found this script this exception will be
144     *         thrown.
145     * @throws ScriptEngineException If execution or JAR is requested and a problem occurs this exception will be
146     *         thrown.
147     */

```

```

148 public void processCommandLine(String[] args) throws ScriptParserException, ScriptEngineException {
149     ResourceBundle res = ResourceBundle.getBundle("jgsl.resources.JGSL");
150
151     sysLogger.info("JGSL arguments: " + Arrays.toString(args));
152
153     // create the command line parser
154     CommandLineParser parser = new PosixParser();
155
156     // create the Options
157
158     // help option
159     Option help = new Option(res.getString("app.option.help.short"), res.getString("app.option.help.long"), false, res.
getString("app.option.help.message"));
160     options.addOption(help);
161
162     // log level script option
163     Option logLevel = new Option(res.getString("app.option.loglevel.short"), res.getString("app.option.loglevel.
long"), true, res.getString("app.option.loglevel.message"));
164     logLevel.setArgName(res.getString("app.option.loglevel.level"));
165     options.addOption(logLevel);
166
167     // log level script option
168     Option syslogLevel = new Option(res.getString("app.option.sysloglevel.short"), res.getString("app.option.
sysloglevel.long"), true, res.getString("app.option.sysloglevel.message"));
169     syslogLevel.setArgName(res.getString("app.option.sysloglevel.level"));
170     options.addOption(syslogLevel);
171
172     // parse script option
173     Option parseScript = new Option(res.getString("app.option.parsescript.short"), res.getString("app.option.
parsescript.long"), true, res.getString("app.option.parsescript.message"));
174     parseScript.setArgName(res.getString("app.option.parsescript.filearg"));
175     options.addOption(parseScript);
176
177     // execute script option
178     Option execScript = new Option(res.getString("app.option.execscript.short"), res.getString("app.option.
execscript.long"), true, res.getString("app.option.execscript.message"));
179     execScript.setArgName(res.getString("app.option.execscript.filearg"));
180     options.addOption(execScript);
181
182     // view script option
183     Option viewScript = new Option(res.getString("app.option.viewscript.short"), res.getString("app.option.
viewscript.long"), true, res.getString("app.option.viewscript.message"));
184     viewScript.setArgName(res.getString("app.option.viewscript.viewertype"));
185     viewScript.setOptionalArg(true);
186     options.addOption(viewScript);
187
188     // save to file
189     Option saveToFile = new Option(res.getString("app.option.savetofiletype.short"), res.getString("app.option.
savetofiletype.long"), true, res.getString("app.option.savetofiletype.message"));
190     saveToFile.setArgName(res.getString("app.option.savetofiletype.type"));
191     options.addOption(saveToFile);
192

```

```

193     // generate docs script option
194     Option genDoc = new Option(res.getString("app.option.gendoc.short"), res.getString("app.option.gendoc.long"),
true, res.getString("app.option.gendoc.message"));
195     genDoc.setArgs(2);
196     genDoc.setArgName(res.getString("app.option.gendoc.files"));
197     options.addOption(genDoc);
198
199     // generate JAR option
200     Option genJar = new Option(res.getString("app.option.genjar.short"), res.getString("app.option.genjar.long"),
true, res.getString("app.option.genjar.message"));
201     genJar.setArgs(2);
202     genJar.setArgName(res.getString("app.option.genjar.files"));
203     options.addOption(genJar);
204
205     String saveToFileName = null;
206     // parse the command line arguments
207     CommandLine line = null;
208     try {
209         line = parser.parse(options, args);
210     } catch (ParseException e) {
211         throw new ScriptEngineException(new StringBuffer().append("app.exception.program.args\n").append(e.
getMessage()).toString());
212     }
213
214     if (line.hasOption(help.getOpt())) {
215         // automatically generate the help statement
216         HelpFormatter formatter = new HelpFormatter();
217         formatter.printHelp(res.getString("app.command.line.name"), options);
218         return;
219     }
220
221     if (line.hasOption(logLevel.getOpt())) {
222         String level = line.getOptionValue(logLevel.getOpt());
223         jgslLogger.setLevel(Level.toLevel(level));
224     }
225
226     if (line.hasOption(sysLogLevel.getOpt())) {
227         String level = line.getOptionValue(sysLogLevel.getOpt());
228         sysLogger.setLevel(Level.toLevel(level));
229     }
230
231     if (line.hasOption(parseScript.getOpt())) {
232         ScriptParser sp = new ScriptParser();
233         String fileName = line.getOptionValue(parseScript.getOpt());
234         String result = sp.parseScript(new File(fileName));
235         jgslLogger.info(result);
236         return;
237     }
238
239     if (line.hasOption(execScript.getOpt())) {
240         ScriptParser sp = new ScriptParser();
241         String fileName = line.getOptionValue(execScript.getOpt());

```



```

242     JGSLScript script = sp.execScript(new File(fileName));
243
244     jgslLogger.info(script.getParseStatus());
245
246     if (script.hasErrors()) {
247         return;
248     }
249
250     String fullClassName = script.generateImplementation();
251     if (line.hasOption(viewScript.getOpt())) {
252         // TODO: currently only swing is supported so the type is not checked here
253         SwingScriptViewer ssv = new SwingScriptViewer();
254         if (line.hasOption(saveToFile.getOpt())) {
255             saveToFileName = line.getOptionValue(saveToFile.getOpt());
256         }
257         ssv.renderScript(fullClassName, saveToFileName);
258     }
259     return;
260 }
261
262 if (line.hasOption(genDoc.getOpt())) {
263     ScriptParser sp = new ScriptParser();
264     String filenames[] = line.getOptionValues(genDoc.getOpt());
265     JGSLScript script = sp.execScript(new File(filenames[0]));
266     jgslLogger.info(script.getParseStatus());
267
268     if (script.hasErrors()) {
269         System.exit(1);
270     }
271     String docStr = script.getDocumentation();
272     File docFile = new File(filenames[1]);
273     try {
274         FileWriter fw = new FileWriter(docFile);
275         fw.write(docStr);
276         fw.close();
277     } catch (FileNotFoundException e) {
278         jgslLogger.error(e.getMessage());
279         sysLogger.error(e.getMessage(), e);
280         System.exit(1);
281     } catch (IOException e) {
282         jgslLogger.error(e.getMessage());
283         sysLogger.error(e.getMessage(), e);
284         System.exit(1);
285     }
286     return;
287 }
288
289
290 if (line.hasOption(genJar.getOpt())) {
291     ScriptParser sp = new ScriptParser();
292     String filenames[] = line.getOptionValues(genJar.getOpt());
293     JGSLScript script = sp.execScript(new File(filenames[0]));

```



```
294     jgslLogger.info(script.getParseStatus());
295
296     if (script.hasErrors()) {
297         System.exit(1);
298     }
299     script.generateImplementation();
300     try {
301         JarPackager.makeJar(new File(filenamees[1]), script.getClassFileName(), script.getFullClassName());
302     } catch (JarPackagerException e) {
303         jgslLogger.error(e.getMessage());
304         sysLogger.error(e.getMessage(), e);
305         System.exit(1);
306     }
307     return;
308 }
309
310 // if we get here then show the GUI
311 JGSLSwingFrame.startJGSL(args);
312 }
313
314 }
315
```

/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/io/ScriptParser.java

```

1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.io;
5
6      import java.io.File;
7
8      import jgsl.model.JGSLScript;
9      import jgsl.parser.JGSL\_Parser;
10     import jgsl.parser.ParseException;
11
12     /**
13      * Parse the specified script file using the JGSL_Parser and report the status of the parse.
14      *
15      * @author zenarchitect
16      * @version $Id: ScriptParser.java,v 1.6 2005/05/16 00:54:16 zenarchitect Exp $
17      */
18
19     public class ScriptParser {
20         /**
21          * @link aggregationByValue
22          */
23         JGSL\_Parser parser;
24
25         /** @link dependency */
26         /**# ScriptParserException lnkScriptParserException; */
27
28         /** @link dependency */
29         /**# Message lnkMessage; */
30
31         /**
32          * @directed
33          * @link aggregation
34          */
35         /**# JGSLScript lnkJGSLScript; */
36
37         public JGSLScript execScript(File scriptFile) throws ScriptParserException {
38             parseScript(scriptFile);
39             if (parser == null) {
40                 throw new ScriptParserException("jgsl parser: Execution failed, unable to create script parser.");
41             }
42             JGSLScript script = parser.getScript();
43             script.setScriptName(scriptFile.getName());

```

```

44     return script;
45 }
46
47 public String parseScript(File scriptFile) throws ScriptParserException {
48     String result = "jgsl parser: Reading from file " + scriptFile + "...\n";
49     double initTime = 0;
50     double parseTime = 0;
51     double startTime = 0;
52     double stopTime = 0;
53     try {
54         startTime = (double) System.currentTimeMillis();
55         parser = new JGSL\_Parser(new java.io.FileInputStream(scriptFile));
56         stopTime = (double) System.currentTimeMillis();
57         initTime = (double) stopTime - startTime;
58     } catch (java.io.FileNotFoundException e) {
59         if (parser != null) {
60             JGSLScript script = parser.getScript();
61             script.addError(new ScriptError(result + "\njgsl parser: File " + scriptFile + " not found.\n"));
62         } else {
63             throw new ScriptParserException("jgsl parser: Execution failed, unable to create script
64 parser.");
65         }
66     } try {
67         startTime = (double) System.currentTimeMillis();
68         parser.parseScript();
69         stopTime = (double) System.currentTimeMillis();
70         parseTime = stopTime - startTime;
71         result += "jgsl parser:\n";
72         result += "\tparsed " + scriptFile + " in " + (initTime + parseTime) / 1000.0 + " sec\n";
73         result += "\tinitialization time = " + initTime / 1000.0 + " sec\n";
74         result += "\tparse time = " + parseTime / 1000.0 + " sec\n";
75         JGSLScript script = parser.getScript();
76         script.addMessage(new ScriptMessage(result));
77     } catch (ParseException e) {
78         String ex = "jgsl parser: Reading from file " + scriptFile + "...\n"
79             + "jgsl parser: Encountered errors during parse...\n";
80         ex += e.getMessage();
81         if (parser != null) {
82             JGSLScript script = parser.getScript();
83             script.addError(new ScriptError(ex));
84         } else {
85             throw new ScriptParserException(ex);
86         }
87     }
88     return result;
89 }

```

```
90
91  }
92
93
```

```
1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.io;
5
6  /**
7   * Report script parsing exceptions.
8   *
9   * @author Joe Chavez
10  * @version $Id: ScriptParserException.java,v 1.2 2005/05/16 00:54:16 zenarchitect Exp $
11  */
12  public class ScriptParserException extends Exception {
13      /**
14       * Constructs a new exception with the specified detail message. The cause is not initialized, and may subsequently
15       * be initialized by a call to {@link #initCause}.
16       *
17       * @param message the detail message. The detail message is saved for later retrieval by the {@link #getMessage
18       *      ()}
19       *      method.
20       */
21      public ScriptParserException(String message) {
22          super(message);
23      }
24  }
25
26
```

```

1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4
5  package jgsl.model;
6  import javaxassist.CannotCompileException;
7  import javaxassist.ClassClassPath;
8  import javaxassist.ClassPool;
9  import javaxassist.CtClass;
10 import javaxassist.CtConstructor;
11 import javaxassist.CtMethod;
12 import javaxassist.NotFoundException;
13 import org.apache.log4j.Logger;
14
15 import java.io.BufferedInputStream;
16 import java.io.BufferedOutputStream;
17 import java.io.File;
18 import java.io.FileOutputStream;
19 import java.io.IOException;
20 import java.io.InputStream;
21 import java.io.Serializable;
22 import java.util.ArrayList;
23 import java.util.LinkedList;
24 import java.util.ResourceBundle;
25
26 import jgsl.io.Message;
27 import jgsl.io.ParseStatus;
28 import jgsl.io.ScriptError;
29 import jgsl.io.ScriptMessage;
30 import jgsl.io.ScriptWarning;
31
32 /**
33  * A JGSLScript contains an ordered collection of objects that implement the statement interface.
34  *
35  * @author zenarchitect
36  * @version $Id: JGSLScript.java,v 1.6 2005/05/16 00:54:18 zenarchitect Exp $
37  */
38 public class JGSLScript implements Serializable, Script, ParseStatus {
39     static Logger jgsLogger = Logger.getLogger("jgs_log");
40     static Logger sysLogger = Logger.getLogger("jgs_sys_log");
41
42     ResourceBundle res = ResourceBundle.getBundle("jgs.view.swing.resources.BaseFrame");
43
44     /**
45      * @link aggregationByValue
46      * @supplierCardinality 1..*
47      * @clientCardinality 1

```

```

48      * @label is composed of
49      */
50      /*# Statement lnkStatement; */
51
52      private LinkedList<Statement> statements = new LinkedList<Statement>();
53
54      private ArrayList<Message> messages = new ArrayList<Message>();
55      int errorCount = 0;
56      int warningCount = 0;
57      int messageCount = 0;
58
59      private String scriptName;
60      private Documentation doc;
61
62      private String className;
63      private String fullClassName;
64      private String classFileName;
65
66
67      public JGSLScript() {
68      }
69
70      public String getClassName() {
71          return className;
72      }
73
74      public String getFullClassName() {
75          return fullClassName;
76      }
77
78      public String getClassFileName() {
79          return classFileName;
80      }
81
82      /**
83       * Get the script name
84       *
85       * @return String containing the script name
86       */
87      public String getScriptName() {
88          return scriptName;
89      }
90
91      /**
92       * Set the script name
93       *
94       * @param scriptName name of the script file
95       */
96      public void setScriptName(String scriptName) {
97          this.scriptName = scriptName;
98      }

```



```

99
100 public String getJavaForInit() {
101     StringBuffer strBuff = new StringBuffer(1024);
102
103     for (Statement s : statements) {
104         if (s.getType().equals(Commands.CANVAS.getName())) {
105             strBuff.append(s.getJava());
106         }
107     }
108     String title = res.getString("jgsl.title") + " - " + scriptName;
109     strBuff.append("setTitle(\"\" + title + "\");");
110
111     return strBuff.toString();
112 }
113
114 /**
115  * Return the Java implementation of this script
116  *
117  * @return the Java language implementation of this script
118  */
119 public String getJava() {
120     StringBuffer strBuff = new StringBuffer(1024);
121
122     strBuff.append("super.paint(g);");
123     strBuff.append("java.awt.Graphics2D g2 = (java.awt.Graphics2D) g;");
124     strBuff.append("java.awt.Container canvas = this.getContentPane();");
125
126     for (Statement s : statements) {
127         if (s.getType().equals(Commands.CANVAS.getName())) {
128             continue;
129         }
130         strBuff.append(s.getJava());
131     }
132
133     return strBuff.toString();
134 }
135
136 public void addDocumentation(String d) {
137     if (doc == null) {
138         doc = new Documentation();
139     }
140     if (d.startsWith("\'")) {
141         d = d.substring(1);
142     }
143     if (d.endsWith("\'")) {
144         d = d.substring(0, d.length() - 1);
145     }
146     if (!d.endsWith("\n")) {
147         d += "\n";
148     }
149     doc.addDoc(d);

```

```

150     }
151
152     /**
153      * Returns the JGSL script documentation as specified in the DOC keyword by the script author.
154      *
155      * @return The script documentation
156      */
157     public String getDocumentation() {
158         return doc.getJava();
159     }
160
161     /**
162      *
163      */
164     public void add(Statement s) {
165         statements.add(s);
166     }
167
168     /**
169      * Generate the implementation class and return the name of the class
170      *
171      * @return returns a String containing the full name of the implementation class
172      */
173     public String generateImplementation() {
174
175         className = scriptName.substring(scriptName.lastIndexOf("/") + 1, scriptName.lastIndexOf("."));
176         fullClassName = "jgsl.generated." + className;
177         //TODO: get destination dir from command line or config file
178         String dirName = System.getProperty("user.home") + File.separator + "jgsl" + File.separator + "cache";
179
180         File dir = new File(dirName);
181         if (!dir.exists()) {
182             dir.mkdirs();
183         }
184         // now check for the jar files
185         // if not there then write them from the resources
186         File jgslJar = new File(dir, "jgsl_rt.jar");
187         try {
188             InputStream is = Thread.currentThread().getContextClassLoader().getResourceAsStream("lib/jgsl_rt.jar");
189             BufferedInputStream bis = new BufferedInputStream(is);
190             FileOutputStream fos = new FileOutputStream(jgslJar);
191             BufferedOutputStream bos = new BufferedOutputStream(fos);
192             byte buff[] = new byte[1024];
193             int bytesRead = 0;
194             while ((bytesRead = bis.read(buff)) != -1) {
195                 bos.write(buff, 0, bytesRead);
196             }
197             bis.close();
198             bos.close();
199         } catch (IOException e) {
200             e.printStackTrace(); //TODO handle exception

```

```

201     }
202
203     File log4Jar = new File(dir, "log4j-1.2.9.jar");
204     try {
205         InputStream is = Thread.currentThread().getContextClassLoader().getResourceAsStream("lib/log4j-1.2.9.
jar");
206         BufferedInputStream bis = new BufferedInputStream(is);
207         FileOutputStream fos = new FileOutputStream(log4Jar);
208         BufferedOutputStream bos = new BufferedOutputStream(fos);
209         byte buff[] = new byte[1024];
210         int bytesRead = 0;
211         while ((bytesRead = bis.read(buff)) != -1) {
212             bos.write(buff, 0, bytesRead);
213         }
214         bis.close();
215         bos.close();
216     } catch (IOException e) {
217         e.printStackTrace(); //TODO handle exception
218     }
219
220     File f = new File(dirName + File.separator + fullClassName);
221     if (f.exists()) {
222         f.delete();
223     }
224
225     ClassPool pool = ClassPool.getDefault();
226     pool.insertClassPath(new ClassClassPath(this.getClass()));
227     CtClass cc = null;
228     try {
229         cc = pool.get("jgsl.view.swing.BaseFrame");
230         cc.setName(fullClassName);
231
232         // modify the constructor
233         CtConstructor cd[] = cc.getDeclaredConstructors();
234
235         String initStr = getJavaForInit();
236         cd[0].insertBeforeBody(initStr);
237
238         // modify the paint method
239         CtMethod m = cc.getDeclaredMethod("paint");
240
241         String paintStr = getJava();
242         m.insertAfter("{ " +
243             paintStr +
244             "}");
245
246         cc.writeFile(dirName);
247         cc.defrost();
248         classFileName = dirName + File.separator + fullClassName.replace('.', '/');
249         return fullClassName;
250

```

```

251     } catch (NotFoundException e) {
252         e.printStackTrace(); //TODO: To change body of catch statement use File | Settings | File Templates.
253     } catch (IOException e) {
254         e.printStackTrace(); //TODO: To change body of catch statement use File | Settings | File Templates.
255     } catch (CannotCompileException e) {
256         e.printStackTrace(); //TODO: To change body of catch statement use File | Settings | File Templates.
257     }
258     return null;
259 }
260
261
262 public String toString() {
263     return "JGSLScript{" +
264         "statements=" + statements +
265         ", scriptName=" + scriptName + "" +
266         "}";
267 }
268
269 /**
270  * Add a ScriptError to the parse status
271  *
272  * @param se
273  */
274 public void addError(ScriptError se) {
275     messages.add(se);
276     errorCount++;
277 }
278
279 /**
280  * Add a ScriptWarning to the parse status
281  *
282  * @param sw
283  */
284 public void addWarning(ScriptWarning sw) {
285     messages.add(sw);
286     warningCount++;
287 }
288
289 /**
290  * Add a ScriptMessage to the parse status
291  *
292  * @param sm
293  */
294 public void addMessage(ScriptMessage sm) {
295     messages.add(sm);
296     messageCount++;
297 }
298
299
300 /**
301  * Return the error state of the script

```

```

302     *
303     * @return true of the script contains errors or false otherwise
304     */
305     public boolean hasErrors() {
306         return errorCount > 0;
307     }
308
309     /**
310     * Return the warning state of the script
311     *
312     * @return true of the script contains warnings or false otherwise
313     */
314     public boolean hasWarnings() {
315         return warningCount > 0;
316     }
317
318     /**
319     * Return the message state of the script * @return true of the script contains messages or false otherwise
320     */
321     public boolean hasMessages() {
322         return messageCount > 0;
323     }
324
325     public int getErrorCount() {
326         return errorCount;
327     }
328
329     public int getWarningCount() {
330         return warningCount;
331     }
332
333     public int getMessageCount() {
334         return messageCount;
335     }
336
337     public String getParseStatus() {
338         String status = "Parse Status\n";
339
340         for (Message m : messages) {
341             status += m.getMessage();
342         }
343
344         String messageString = "messages";
345         String warningString = "warnings";
346         String errorString = "errors";
347         if (messageCount == 1) {
348             messageString = "message";
349         }
350         if (warningCount == 1) {
351             warningString = "warning";
352         }

```

```
353     if (errorCount == 1) {
354         errorString = "error";
355     }
356
357     status += String.format("%d %s, %d %s, %d %s encountered during script processing.\n", messageCount,
messageString, warningCount, warningString, errorCount, errorString);
358     if (hasErrors()) {
359         status += "Please examine and correct any errors.";
360     }
361     return status;
362 }
363 }
364
```

```

1  /*
2   * Copyright (c) 2005 Perception Software. All Rights Reserved.
3   */
4  package jgsl.util;
5
6  import org.apache.log4j.Logger;
7
8  import java.io.File;
9  import java.io.FileInputStream;
10 import java.io.FileNotFoundException;
11 import java.io.FileOutputStream;
12 import java.io.IOException;
13 import java.util.ResourceBundle;
14 import java.util.jar.Attributes;
15 import java.util.jar.JarEntry;
16 import java.util.jar.JarInputStream;
17 import java.util.jar.JarOutputStream;
18 import java.util.jar.Manifest;
19
20 /**
21  * Create an executable JAR file for a JGSL script Java class.
22  *
23  * @author jchavez
24  */
25 public class JarPackager {
26     static Logger jgslLogger = Logger.getLogger("jgsl_log");
27     static Logger sysLogger = Logger.getLogger("jgsl_sys_log");
28
29     /**
30      * Create a jar file for JGSL distribution. The className parameter will be used to set the main class attribute.
31      * <p>
32      * Main-Class: className
33      *
34      * @param jarFileName Name of JAR to create
35      * @param classFileName Full path to the class file to add to the jar
36      * @param className Name of the class with full package specification. The "." will be replaced with "/".
37      * @throws JarPackagerException
38      */
39     public static void makeJar(File jarFileName, String classFileName, String className) throws JarPackagerException
40     {
41         jgslLogger.info("Creating " + jarFileName.getAbsolutePath() + " for JGSL script " + className);
42         JarOutputStream targetJar;
43         FileOutputStream fos;
44         Manifest manifest = new Manifest();
45         Attributes attrs = manifest.getMainAttributes();
46
47         ResourceBundle res = ResourceBundle.getBundle("jgsl.resources.JGSL");
48         String version = res.getString("jgsl.app.name");

```



```

48
49     attrs.putValue("Manifest-Version", "1.0");
50     attrs.putValue("Created-By", version);
51     attrs.putValue("Main-Class", className);
52
53
54     // Need the following files in the JAR
55     //     the JGSL class
56     //     jgsl_rt.jar
57     //     log4j-1.2.9.jar
58
59     try {
60         fos = new FileOutputStream(jarFileName);
61         targetJar = new JarOutputStream(fos, manifest);
62         mergeJar(targetJar, "jgsl_rt.jar");
63         mergeJar(targetJar, "log4j-1.2.9.jar");
64
65         addEntry(targetJar, classFileName, className);
66
67         targetJar.flush();
68         targetJar.close();
69         jgslLogger.info("JAR creation completed.");
70     } catch (FileNotFoundException e) {
71         jgslLogger.error(e.getMessage(), e);
72         throw new JarPackagerException("Unable to create JAR file: " + jarFileName.getAbsolutePath());
73     } catch (IOException e) {
74         jgslLogger.error(e.getMessage(), e);
75         throw new JarPackagerException("Unable to create JAR file: " + jarFileName.getAbsolutePath());
76     }
77
78 }
79
80 /**
81  * Add an entry to a jar file
82  *
83  * @param targetJar    Output stream of the jar to add the entry to
84  * @param classFileName Full path to the class file to add to the jar
85  * @param className    Name of the class with full package specification. The "." will be replaced with "/".
86  * @throws IOException If reading/writing encounters an error
87  */
88 private static void addEntry(JarOutputStream targetJar, String classFileName, String className) throws IOException
89 {
90     sysLogger.debug("BEGIN - JarPackager.addEntry");
91     String jarEntryName = className.replace('.', '/') + ".class";
92     JarEntry entry = new JarEntry(jarEntryName);
93
94     targetJar.putNextEntry(entry);
95
96     FileInputStream fis = new FileInputStream(classFileName + ".class");
97
98     byte[] buf = new byte[4096];
99     int bytesRead = 0;

```

```

99     while ((bytesRead = fis.read(buf)) != -1) {
100         targetJar.write(buf, 0, bytesRead);
101     }
102     targetJar.closeEntry();
103     sysLogger.debug("END - JarPackager.addEntry");
104 }
105
106 /**
107  * Merge the contents of a JAR file into another
108  * <p/>
109  * Looks for jarName in the JGSL cache: System.getProperty("user.home") + File.separator + ".jgsl" + File.
separator
110  * + "cache";
111  *
112  * @param jarOut Output stream of the jar to merge into
113  * @param jarName Name of the JAR file to merge will taken from the JGSL cache
114  * @throws IOException Thrown if reading/writing fails.
115  */
116 private static void mergeJar(JarOutputStream jarOut, String jarName) throws IOException {
117     sysLogger.debug("BEGIN - JarPackager.mergeJar");
118     JarInputStream jarIn;
119     String jgslCache = System.getProperty("user.home") + File.separator + ".jgsl" + File.separator + "cache";
120     File jarFile = new File(jgslCache + File.separator + jarName);
121
122     jarIn = new JarInputStream(new FileInputStream(jarFile));
123     // Create a read buffer to be used for transferring data from the input
124
125     byte[] buf = new byte[4096];
126
127     // Iterate the entries
128
129     JarEntry entry;
130     while ((entry = jarIn.getNextJarEntry()) != null) {
131         // Exclude the Manifest file from the old JAR
132
133         if (entry.getName().equals("META-INF/MANIFEST.MF")) {
134             continue;
135         }
136
137         // Write out the entry to the output JAR
138
139         jarOut.putNextEntry(entry);
140         int read;
141         while ((read = jarIn.read(buf)) != -1) {
142             jarOut.write(buf, 0, read);
143         }
144
145         jarOut.closeEntry();
146     }
147
148     // Flush and close all the streams
149

```

```
150
151     jarIn.close();
152     sysLogger.debug("END - JarPackager.mergeJar");
153 }
154
155 }
156
```

/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/util/JarPackagerException.java

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.util;
5
6      /**
7      * Report JarPackageer exceptions.
8      *
9      * @author jchavez
10     */
11     public class JarPackagerException extends Exception {
12         public JarPackagerException(String string) {
13             super(string);
14         }
15     }
16
```

```
1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.view.swing;
5
6  import com.intellij.uiDesigner.core.GridConstraints;
7  import com.intellij.uiDesigner.core.GridLayoutManager;
8  import org.apache.log4j.Logger;
9
10 import javax.swing.BorderFactory;
11 import javax.swing.DefaultListModel;
12 import javax.swing.JButton;
13 import javax.swing.JCheckBox;
14 import javax.swing.JFileChooser;
15 import javax.swing.JFrame;
16 import javax.swing.JList;
17 import javax.swing.JMenu;
18 import javax.swing.JMenuBar;
19 import javax.swing.JMenuItem;
20 import javax.swing.JPanel;
21 import javax.swing.JScrollPane;
22 import javax.swing.JTextArea;
23 import javax.swing.JTextField;
24 import javax.swing.event.ListSelectionEvent;
25 import javax.swing.event.ListSelectionListener;
26 import java.awt.Dimension;
27 import java.awt.HeadlessException;
28 import java.awt.Insets;
29 import java.awt.Toolkit;
30 import java.awt.event.ActionEvent;
31 import java.awt.event.ActionListener;
32 import java.awt.event.ItemEvent;
33 import java.awt.event.ItemListener;
34 import java.awt.event.WindowAdapter;
35 import java.awt.event.WindowEvent;
36 import java.io.BufferedReader;
37 import java.io.File;
38 import java.io.FileReader;
39 import java.io.FileWriter;
40 import java.io.IOException;
41 import java.io.InputStreamReader;
42 import java.util.Locale;
43 import java.util.ResourceBundle;
44
45 import jgsl.controller.script.ScriptEngine;
46 import jgsl.controller.script.ScriptEngineException;
47 import jgsl.io.ImageFileFilter;
48 import jgsl.io.JARFileFilter;
```

```

49 import jgsl.io.JGSLFileFilter;
50 import jgsl.io.ScriptParserException;
51 import jgsl.model.JGSLScript;
52
53 /**
54  * The JGSLSwingFrame class is the main class for the interactive GUI.
55  *
56  * @author zenarchitect
57  * @version $Id: JGSLSwingFrame.java,v 1.5 2005/05/21 01:42:11 zenarchitect Exp $
58  */
59 public class JGSLSwingFrame implements ActionListener, ItemListener, ListSelectionListener {
60     static Logger jgslLogger = Logger.getLogger("jgsl_log");
61     static Logger sysLogger = Logger.getLogger("jgsl_sys_log");
62
63     ResourceBundle res = ResourceBundle.getBundle("jgsl.view.swing.resources.JGSLSwingFrame", new Locale("en",
64 "US"), Thread.currentThread().getContextClassLoader());
65
66     private File currentFileName = new File("newfile.jgsl");
67     private File newFileName;
68     private boolean isNew = false;
69
70     private JFrame frame;
71
72     private JPanel mainPanel;
73     private JTextArea scriptTextArea;
74     private JPanel scriptPanel;
75     private JButton viewButton;
76     private JButton preferencesButton;
77     private JButton quitButton;
78     private JPanel actionPanel;
79     private JPanel quitPanel;
80
81
82     private JPanel statusPanel;
83     private JList statusList;
84     private JScrollPane statusScrollPane;
85
86     JMenuBar menuBar;
87
88     JMenu fileMenu;
89     JMenuItem newMenuItem;
90     JMenuItem openMenuItem;
91     JMenuItem saveMenuItem;
92     JMenuItem saveAsMenuItem;
93     JMenuItem exitMenuItem;
94
95     JMenu viewMenu;
96
97     JMenu helpMenu;
98     JMenuItem helpMenuItem;
99     JMenuItem aboutMenuItem;

```

```

100
101 DefaultListModel statusListModel = new DefaultListModel();
102 private int MAX_STATUS_LIST_SIZE = 250;
103 private JScrollPane scriptScrollPane;
104 private String frameTitle;
105 private JButton jarButton;
106 private JTextField scriptOutputFileName;
107 private JButton selectScriptOutputButton;
108 private JCheckBox saveScriptOutputCheckBox;
109 private boolean saveScriptOuput;
110
111 /**
112  * Constructs a new frame that is initially invisible.
113  * <p/>
114  * This constructor sets the component's locale property to the value returned by
115  * <code>JComponent.getDefaultLocale</code>.
116  *
117  * @throws java.awt.HeadlessException if GraphicsEnvironment.isHeadless() returns true.
118  * @see java.awt.GraphicsEnvironment#isHeadless
119  * @see java.awt.Component#setSize
120  * @see java.awt.Component#setVisible
121  * @see javax.swing.JComponent#getDefaultLocale
122  */
123 public JGSLSwingFrame(JFrame frame) throws HeadlessException {
124     sysLogger.debug("BEGIN: JGSLSwingFrame");
125
126     this.frame = frame;
127
128     //Center frame
129     Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
130     Dimension size = frame.getSize();
131     screenSize.height = screenSize.height / 2;
132     screenSize.width = screenSize.width / 2;
133     size.height = size.height / 2;
134     size.width = size.width / 2;
135     int y = screenSize.height - size.height;
136     int x = screenSize.width - size.width;
137     frame.setLocation(x, y);
138     sysLogger.debug("JGSLSwingFrame: frame setup completed...");
139
140     if (viewButton == null) {
141         sysLogger.debug("JGSLSwingFrame: goButton == null");
142     }
143
144     viewButton.addActionListener(this);
145
146     jarButton.addActionListener(this);
147     selectScriptOutputButton.addActionListener(this);
148     // quitButton.addActionListener(this);
149
150     saveScriptOutputCheckBox.addItemListener(this);
151

```



```
152     statusList.addListSelectionListener(this);
153     statusList.setModel(statusListModel);
154     sysLogger.debug("JGSLSwingFrame: status list setup completed...");
155
156
157     menuBar = new JMenuBar();
158     fileMenu = new JMenu("File");
159
160     newMenuItem = new JMenuItem("New");
161     newMenuItem.addActionListener(this);
162     openMenuItem = new JMenuItem("Open");
163     openMenuItem.addActionListener(this);
164     saveMenuItem = new JMenuItem("Save");
165     saveMenuItem.addActionListener(this);
166     saveAsMenuItem = new JMenuItem("Save As");
167     saveAsMenuItem.addActionListener(this);
168
169     exitMenuItem = new JMenuItem("Exit");
170     exitMenuItem.addActionListener(this);
171
172     fileMenu.add(newMenuItem);
173     fileMenu.addSeparator();
174     fileMenu.add(openMenuItem);
175     fileMenu.add(saveMenuItem);
176     fileMenu.add(saveAsMenuItem);
177     fileMenu.addSeparator();
178     fileMenu.add(exitMenuItem);
179
180
181     viewMenu = new JMenu();
182
183
184     helpMenu = new JMenu("Help");
185     helpMenuItem = new JMenuItem("Help Topics");
186     helpMenuItem.addActionListener(this);
187     aboutMenuItem = new JMenuItem("About");
188     aboutMenuItem.addActionListener(this);
189
190     helpMenu.add(helpMenuItem);
191     helpMenu.add(aboutMenuItem);
192
193     menuBar.add(fileMenu);
194     // menuBar.add(viewMenu);
195     menuBar.add(helpMenu);
196     sysLogger.debug("JGSLSwingFrame: menu bar setup completed...");
197
198
199     frame.setJMenuBar(menuBar);
200     sysLogger.debug("JGSLSwingFrame: menu bar added to main window...");
201
202
203     frameTitle = res.getString("jgsl.i-gui.title");
```

```

204     if (frameTitle != null) {
205         frame.setTitle(frameTitle);
206     } else {
207         frame.setTitle("JGSL 1.0 - Java Web Start");
208     }
209     sysLogger.debug("JGSLSwingFrame: window title set...");
210
211     addItem("JGSL 1.0 ready...");
212
213     sysLogger.debug("END: JGSLSwingFrame");
214
215 }
216
217 /**
218  * Return reference to the main panel.
219  */
220 public JPanel getMainPanel() {
221     return mainPanel;
222 }
223
224 /**
225  * Start the JGSL. This method is needed to work around the strange startup requirements by the IntelliJ IDEA GUI
226  * builder.
227  */
228 public static void startJGSL(String[] args) {
229     sysLogger.debug("BEGIN: startJGSL");
230     JFrame f = new JFrame();
231
232     f.setSize(640, 480);
233
234     JGSLSwingFrame mainFrame = new JGSLSwingFrame(f);
235
236     f.setContentPane(mainFrame.getMainPanel());
237
238     f.addWindowListener(new WindowAdapter() {
239         public void windowClosing(WindowEvent ev) {
240             System.exit(0);
241         }
242     });
243
244     f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
245
246
247 //     f.pack();
248     f.setVisible(true);
249     sysLogger.debug("END: startJGSL");
250
251 }
252
253 /**
254  * Program entry point.
255  */

```

```

256 public static void main(String[] args) {
257     startJGSL(args);
258 }
259
260 public void actionPerformed(ActionEvent actionEvent) {
261
262     if (actionEvent.getSource() == quitButton || actionEvent.getSource() == exitMenuItem) {
263         frame.setVisible(false);
264         System.exit(0);
265     } else if (actionEvent.getSource() == newItem) {
266         newScript();
267     } else if (actionEvent.getSource() == openMenuItem) {
268         openScript();
269     } else if (actionEvent.getSource() == saveMenuItem) {
270         storeScript();
271     } else if (actionEvent.getSource() == viewButton) {
272         viewScript();
273     } else if (actionEvent.getSource() == jarButton) {
274         jarScript();
275     } else if (actionEvent.getSource() == selectScriptOutputButton) {
276         selectScriptOutputFile();
277     } else if (actionEvent.getSource() == saveAsMenuItem) {
278         saveAsScript();
279     }
280 }
281
282 private void selectScriptOutputFile() {
283     JFileChooser chooser = new JFileChooser();
284     ImageFileFilter filter = new ImageFileFilter();
285     chooser.setFileFilter(filter);
286     int returnVal = chooser.showSaveDialog(getMainPanel());
287     if (returnVal == JFileChooser.APPROVE_OPTION) {
288         File outputFileName = chooser.getSelectedFile();
289         this.scriptOutputFileName.setText(outputFileName.getAbsolutePath());
290     }
291 }
292
293 private void jarScript() {
294     JFileChooser chooser = new JFileChooser();
295     JARFileFilter filter = new JARFileFilter();
296     chooser.setFileFilter(filter);
297     int returnVal = chooser.showSaveDialog(getMainPanel());
298     if (returnVal == JFileChooser.APPROVE_OPTION) {
299         File jarFileName = chooser.getSelectedFile();
300         if (!jarFileName.getName().endsWith(".jar")) {
301             jarFileName = new File(jarFileName.getAbsolutePath() + ".jar");
302         }
303         // parse and validate
304         // generate implementation class
305         // execute viewer
306         ScriptEngine se = new ScriptEngine();
307         try {

```

```

308     JGSLScript script = se.jarInteractive(currentFileName, jarFileName);
309     if (script.hasErrors()) {
310         // TODO - Get the errors as an object and add to status list
311         // TODO - Make status list context aware and select error to show user in script
312         addItem(script.getParseStatus());
313     }
314 } catch (ScriptEngineException e) {
315     e.printStackTrace();
316     addItem("An error was encountered creating a JAR for your script, details are provided below");
317     addItem(e.getMessage());
318 } catch (ScriptParserException e) {
319     e.printStackTrace();
320     addItem("An error was encountered parsing your script, details are provided below");
321     addItem(e.getMessage());
322 }
323 }
324 }
325
326 private void storeScript() {
327     if (isNew) {
328         saveAsScript();
329     } else {
330         newFileName = currentFileName;
331         saveScript();
332     }
333 }
334 }
335
336 private void viewScript() {
337     // save the script
338     storeScript();
339
340     // parse and validate
341     // generate implementation class
342     // execute viewer
343     ScriptEngine se = new ScriptEngine();
344     try {
345         String scriptOutputFileName = null;
346         if(saveScriptOutput) {
347             scriptOutputFileName = this.scriptOutputFileName.getText();
348         }
349         JGSLScript script = se.viewInteractive(currentFileName, scriptOutputFileName); // TODO add file name to GUI
350         if (script.hasErrors()) {
351             // TODO - Make status list context aware and select error to show user in script
352             addItem(script.getParseStatus());
353         }
354     } catch (ScriptParserException e) {
355         e.printStackTrace();
356         addItem("An error was encountered parsing your script, details are provided below");
357         addItem(e.getMessage());
358     }
359 }

```

```

360     }
361
362     public void itemStateChanged(ItemEvent itemEvent) {
363
364         if(itemEvent.getSource() == saveScriptOutputCheckBox) {
365             saveScriptOuput = !saveScriptOuput;
366         }
367     }
368
369     public void valueChanged(ListSelectionEvent listSelectionEvent) {
370         //TODO: implement method
371     }
372
373     private void newScript() {
374         String basicScript = "/*\n" +
375             "JGSL Script\n" +
376             "*/\n" +
377             "\n" +
378             "begin\n" +
379             "\n" +
380             "// Initialize the canvas\n" +
381             "canvas (640, 480, BLACK, WHITE, \"jgsl script\");\n" +
382             "\n" +
383             "end\n";
384         try {
385             InputStreamReader isr = new InputStreamReader(Thread.currentThread().getContextClassLoader().
getResourceAsStream("jgsl/resources/template.jgsl"));
386             BufferedReader br = new BufferedReader(isr);
387             String line = "";
388             StringBuffer sb = new StringBuffer(1024);
389             while ((line = br.readLine()) != null) {
390                 sb.append(line);
391                 sb.append("\n");
392             }
393             isr.close();
394             scriptTextArea.setText(sb.toString());
395             addItem("Template script loaded, use the script editor write your script.");
396             frame.setTitle(frameTitle + " - (new)");
397         } catch (IOException e1) {
398             e1.printStackTrace();
399             addItem("Unable to read file template file, creating basic script.");
400             scriptTextArea.setText(basicScript);
401         }
402         isNew = true;
403     }
404
405     private void saveScript() {
406         addItem("Saving " + newFileName);
407         try {
408             FileWriter fw = new FileWriter(newFileName);
409             fw.write(scriptTextArea.getText(), 0, scriptTextArea.getText().length());
410             fw.close();

```

```

411     currentFileName = newFileName;
412     addStatusItem("Saved file " + newFileName);
413     isNew = false;
414     frame.setTitle(frameTitle + " - " + currentFileName);
415 } catch (IOException e1) {
416     e1.printStackTrace();
417     addStatusItem("Unable to write file: " + newFileName);
418 }
419 }
420
421 private void saveAsScript() {
422     JFileChooser chooser = new JFileChooser();
423     JGSLFileFilter filter = new JGSLFileFilter();
424     chooser.setFileFilter(filter);
425     int returnVal = chooser.showSaveDialog(getMainPanel());
426     if (returnVal == JFileChooser.APPROVE_OPTION) {
427         newFileName = chooser.getSelectedFile();
428         if (!newFileName.getName().endsWith(".jgsl")) {
429             newFileName = new File(newFileName.getAbsolutePath() + ".jgsl");
430         }
431         saveScript();
432     }
433 }
434
435 private void openScript() {
436     JFileChooser chooser = new JFileChooser();
437     JGSLFileFilter filter = new JGSLFileFilter();
438     chooser.setFileFilter(filter);
439     int returnVal = chooser.showOpenDialog(getMainPanel());
440     if (returnVal == JFileChooser.APPROVE_OPTION) {
441         File newFileName = chooser.getSelectedFile();
442         addStatusItem("Opening " + newFileName);
443         try {
444             FileReader fr = new FileReader(newFileName);
445             BufferedReader br = new BufferedReader(fr);
446             String line = "";
447             StringBuffer sb = new StringBuffer(1024);
448             while ((line = br.readLine()) != null) {
449                 sb.append(line);
450                 sb.append("\n");
451             }
452             fr.close();
453             scriptTextArea.setText(sb.toString());
454             scriptTextArea.setCaretPosition(0);
455             currentFileName = newFileName;
456             addStatusItem("Opened " + newFileName);
457             isNew = false;
458             frame.setTitle(frameTitle + " - " + currentFileName);
459         } catch (IOException e1) {
460             e1.printStackTrace();
461             addStatusItem("Unable to read file: " + newFileName);
462         }

```

```

463     }
464 }
465
466 private void addStatusItem(String item) {
467     if (statusListModel.getSize() > MAX_STATUS_LIST_SIZE * 1.25) {
468         statusListModel.removeRange(MAX_STATUS_LIST_SIZE, (int) ((MAX_STATUS_LIST_SIZE * 1.25) - 1));
469     }
470     statusListModel.addElement(item);
471     statusList.ensureIndexIsVisible(statusListModel.size() - 1);
472
473 }
474
475
476 {
477 // GUI initializer generated by IntelliJ IDEA GUI Designer
478 // >>> IMPORTANT!! <<<
479 // DO NOT EDIT OR ADD ANY CODE HERE!
480     $$$setupUI$$$();
481 }
482
483 /**
484  * Method generated by IntelliJ IDEA GUI Designer >>> IMPORTANT!! <<< DO NOT edit this method OR call it in
485  * your
486  * code!
487  */
488 private void $$$setupUI$$$() {
489     mainPanel = new JPanel();
490     mainPanel.setLayout(new GridLayoutManager(3, 3, new Insets(0, 0, 0, 0), -1, -1));
491     scriptPanel = new JPanel();
492     scriptPanel.setLayout(new GridLayoutManager(1, 1, new Insets(0, 0, 0, 0), -1, -1));
493     mainPanel.add(scriptPanel, new GridConstraints(0, 0, 1, 3, GridConstraints.ANCHOR_CENTER, GridConstraints.
494     FILL_BOTH, GridConstraints.SIZEPOLICY_CAN_SHRINK | GridConstraints.SIZEPOLICY_CAN_GROW, GridConstraints.
495     SIZEPOLICY_CAN_SHRINK | GridConstraints.SIZEPOLICY_CAN_GROW, new Dimension(320, 200), new Dimension
496     (640, 480), null));
497     scriptPanel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLoweredBevelBorder(), "Script
498     Editor"));
499     scriptScrollPane = new JScrollPane();
500     scriptPanel.add(scriptScrollPane, new GridConstraints(0, 0, 1, 1, GridConstraints.ANCHOR_CENTER,
501     GridConstraints.FILL_BOTH, GridConstraints.SIZEPOLICY_CAN_SHRINK | GridConstraints.
502     SIZEPOLICY_WANT_GROW, GridConstraints.SIZEPOLICY_CAN_SHRINK | GridConstraints.
503     SIZEPOLICY_WANT_GROW, null, null, null));
504     scriptTextArea = new JTextArea();
505     scriptScrollPane.setViewportView(scriptTextArea);
506     actionPanel = new JPanel();
507     actionPanel.setLayout(new GridLayoutManager(1, 4, new Insets(0, 0, 0, 0), -1, -1));
508     mainPanel.add(actionPanel, new GridConstraints(2, 0, 1, 2, GridConstraints.ANCHOR_WEST, GridConstraints.
509     FILL_NONE, GridConstraints.SIZEPOLICY_FIXED, GridConstraints.SIZEPOLICY_FIXED, null, null, null));
510     actionPanel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(), "Script Actions"));
511     viewButton = new JButton();
512     viewButton.setText("View");
513     viewButton.setToolTipText("View your JGLS script.");
514     actionPanel.add(viewButton, new GridConstraints(0, 0, 1, 1, GridConstraints.ANCHOR_CENTER, GridConstraints.

```



```

FILL_HORIZONTAL, GridConstraints.SIZEPOLICY_FIXED, GridConstraints.SIZEPOLICY_FIXED, null, null, null));
506     jarButton = new JButton();
507     jarButton.setText("Create JAR");
508     jarButton.setToolTipText("Create an executable JAR file from your JGSL script.");
509     actionPanel.add(jarButton, new GridConstraints(0, 1, 1, 1, GridConstraints.ANCHOR_CENTER, GridConstraints.
FILL_HORIZONTAL, GridConstraints.SIZEPOLICY_FIXED, GridConstraints.SIZEPOLICY_FIXED, null, null, null));
510     final JPanel panel0 = new JPanel();
511     panel0.setLayout(new GridLayoutManager(2, 2, new Insets(0, 0, 0, 0), -1, -1));
512     actionPanel.add(panel0, new GridConstraints(0, 2, 1, 1, GridConstraints.ANCHOR_CENTER, GridConstraints.
FILL_BOTH, GridConstraints.SIZEPOLICY_CAN_SHRINK | GridConstraints.SIZEPOLICY_CAN_GROW, GridConstraints.
SIZEPOLICY_CAN_SHRINK | GridConstraints.SIZEPOLICY_CAN_GROW, null, null, null));
513     selectScriptOutputButton = new JButton();
514     selectScriptOutputButton.setText("Select");
515     selectScriptOutputButton.setToolTipText("Press this button to enter a file name.");
516     panel0.add(selectScriptOutputButton, new GridConstraints(1, 1, 1, 1, GridConstraints.ANCHOR_CENTER,
GridConstraints.FILL_HORIZONTAL, GridConstraints.SIZEPOLICY_CAN_SHRINK | GridConstraints.
SIZEPOLICY_CAN_GROW, GridConstraints.SIZEPOLICY_FIXED, null, null, null));
517     scriptOutputFileName = new JTextField();
518     scriptOutputFileName.setEditable(false);
519     scriptOutputFileName.setInheritsPopupMenu(false);
520     scriptOutputFileName.setText("jgsl_image.jpg");
521     scriptOutputFileName.setToolTipText("File name to which your JGSL script image will be saved to.");
522     panel0.add(scriptOutputFileName, new GridConstraints(1, 0, 1, 1, GridConstraints.ANCHOR_WEST,
GridConstraints.FILL_HORIZONTAL, GridConstraints.SIZEPOLICY_WANT_GROW, GridConstraints.
SIZEPOLICY_FIXED, null, new Dimension(150, -1), null));
523     saveScriptOutputCheckBox = new JCheckBox();
524     saveScriptOutputCheckBox.setText("Save script output");
525     saveScriptOutputCheckBox.setToolTipText("Check this to generate an image from your JGSL script.");
526     panel0.add(saveScriptOutputCheckBox, new GridConstraints(0, 0, 1, 1, GridConstraints.ANCHOR_WEST,
GridConstraints.FILL_NONE, GridConstraints.SIZEPOLICY_CAN_SHRINK | GridConstraints.
SIZEPOLICY_CAN_GROW, GridConstraints.SIZEPOLICY_FIXED, null, null, null));
527     statusPanel = new JPanel();
528     statusPanel.setLayout(new GridLayoutManager(1, 1, new Insets(0, 0, 0, 0), -1, -1));
529     mainPanel.add(statusPanel, new GridConstraints(1, 0, 1, 3, GridConstraints.ANCHOR_CENTER, GridConstraints.
FILL_BOTH, GridConstraints.SIZEPOLICY_CAN_SHRINK | GridConstraints.SIZEPOLICY_CAN_GROW, GridConstraints.
SIZEPOLICY_CAN_SHRINK | GridConstraints.SIZEPOLICY_CAN_GROW, null, null, null));
530     statusPanel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLoweredBevelBorder(), "Status"));
531     statusScrollPane = new JScrollPane();
532     statusPanel.add(statusScrollPane, new GridConstraints(0, 0, 1, 1, GridConstraints.ANCHOR_CENTER,
GridConstraints.FILL_BOTH, GridConstraints.SIZEPOLICY_CAN_SHRINK | GridConstraints.
SIZEPOLICY_WANT_GROW, GridConstraints.SIZEPOLICY_CAN_SHRINK | GridConstraints.
SIZEPOLICY_WANT_GROW, null, null, null));
533     statusList = new JList();
534     statusList.setToolTipText("Status of JGSL actions.");
535     statusScrollPane.setViewportView(statusList);
536 }
537 }
538

```

```

1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.view.swing;
5
6
7  import org.apache.log4j.Logger;
8
9  import java.io.BufferedReader;
10 import java.io.File;
11 import java.io.IOException;
12 import java.io.InputStream;
13 import java.io.InputStreamReader;
14 import java.util.Map;
15
16 import jgsl.view.ScriptViewer;
17
18
19 /**
20  * The SwingScriptViewer class creates an JVM that executes the JGSLViewer class with an argument of the compiled
21  * JGSL
22  * script Java class.
23  *
24  * @author zenarchitect
25  * @version $Id: SwingScriptViewer.java,v 1.6 2005/05/21 01:42:11 zenarchitect Exp $
26  */
27 public class SwingScriptViewer implements ScriptViewer {
28     static Logger jgslLogger = Logger.getLogger("jgsl_log");
29     static Logger sysLogger = Logger.getLogger("jgsl_sys_log");
30     static final boolean DEBUG = false;
31
32     /**
33      * Rerender the script by creating a Process object with the properly JGSL runtime class path. The runtime classpath
34      * includes the compile JGSL script in the form of a Java class. Also required on the classpath are the jgsl_rt.jar
35      * and log4j-1.2.9.jar files.
36      *
37      * @param fullClassName
38      */
39     public void renderScript(String fullClassName, String saveToFileName) {
40         sysLogger.debug("BEGIN - renderScript");
41         // The script is rendered in a swing gui by creating a new class from the script
42         // that is a subclass of a base JFrame that overrides the paint method
43         try {
44             sysLogger.debug("fullClassName = " + fullClassName);
45             ProcessBuilder pb;

```

```

46     if (saveToFileName != null) {
47         pb = new ProcessBuilder(System.getProperty("java.home") +
48             File.separator + "bin" +
49             File.separator + "java", "" +
50             "jgsl.view.swing.JGSLViewer",
51             fullClassName,
52             saveToFileName);
53     } else {
54         pb = new ProcessBuilder(System.getProperty("java.home") +
55             File.separator + "bin" +
56             File.separator + "java", "" +
57             "jgsl.view.swing.JGSLViewer",
58             fullClassName);
59     }
60     if (DEBUG) {
61         pb.redirectErrorStream(true);
62     }
63     String jgslCache = System.getProperty("user.home") + File.separator + ".jgsl" + File.separator + "cache";
64     String classPath = System.getProperty("java.class.path");
65     classPath += File.pathSeparator + jgslCache;
66     classPath += File.pathSeparator + jgslCache + File.separator + "jgsl_rt.jar";
67     classPath += File.pathSeparator + jgslCache + File.separator + "log4j-1.2.9.jar";
68     sysLogger.debug("Class path = " + classPath);
69     Map<String, String> env = pb.environment();
70     env.put("CLASSPATH", classPath);
71
72     Process p = pb.start();
73
74     if (DEBUG) {
75         InputStream is = p.getInputStream();
76         BufferedReader br = new BufferedReader(new InputStreamReader(is));
77         String line;
78         while ((line = br.readLine()) != null) {
79             System.out.println(line);
80         }
81     }
82     sysLogger.debug("END - renderScript");
83
84 } catch (IOException e) {
85     e.printStackTrace(); // TODO: Handle exception
86     sysLogger.debug(e.getMessage());
87 } catch (Exception e) {
88     e.printStackTrace(); // TODO: Handle exception
89     sysLogger.debug(e.getMessage());
90 }
91 }
92
93 /**
94  * @directed
95  */

```

```
96     /*# JGSLViewer lnkJGSLViewer; */  
97 }  
98
```

/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/controller/script/ScriptEngineException.java

```

1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.controller.script;
5
6  /**
7   * ScriptEngineException is thrown by the ScriptEngine class to report exception conditions.
8   *
9   * @author zenarchitect
10  * @version $Id: ScriptEngineException.java,v 1.3 2005/05/16 00:54:15 zenarchitect Exp $
11  */
12
13  public class ScriptEngineException extends Throwable {
14      /**
15       * Constructs a new throwable with null as its detail message. The cause is not initialized, and may
16       * subsequently be initialized by a call to {@link #initCause}.
17       * <p/>
18       * <p>The {@link #fillInStackTrace()} method is called to initialize the stack trace data in the newly created
19       * throwable.
20       */
21      public ScriptEngineException() {
22          super();
23      }
24
25      /**
26       * Constructs a new throwable with the specified detail message. The cause is not initialized, and may subsequently
27       * be initialized by a call to {@link #initCause}.
28       * <p/>
29       * <p>The {@link #fillInStackTrace()} method is called to initialize the stack trace data in the newly created
30       * throwable.
31       *
32       * @param message the detail message. The detail message is saved for later retrieval by the {@link #getMessage()}
33       * method.
34       */
35      public ScriptEngineException(String message) {
36          super(message);
37      }
38
39      /**
40       * Constructs a new throwable with the specified detail message and cause. <p>Note that the detail message
41       * associated with cause is not automatically incorporated in this throwable's detail
42       * message.
43       * <p/>
44       * <p>The {@link #fillInStackTrace()} method is called to initialize the stack trace data in the newly created
45       * throwable.
46       *
47       * @param message the detail message (which is saved for later retrieval by the {@link #getMessage()} method).
48       * @param cause the cause (which is saved for later retrieval by the {@link #getCause()} method). (A

```

```

48      *      <tt>null</tt> value is permitted, and indicates that the cause is nonexistent or unknown.)
49      * @since 1.4
50      */
51  public ScriptEngineException(String message, Throwable cause) {
52      super(message, cause);
53  }
54
55  /**
56   * Constructs a new throwable with the specified cause and a detail message of <tt>(cause==null ? null :
57   * cause.toString())</tt> (which typically contains the class and detail message of <tt>cause</tt>). This
58   * constructor is useful for throwables that are little more than wrappers for other throwables (for example, {@link
59   * java.security.PrivilegedActionException}).
60   * <p/>
61   * <p>The {@link #fillInStackTrace()} method is called to initialize the stack trace data in the newly created
62   * throwable.
63   *
64   * @param cause the cause (which is saved for later retrieval by the {@link #getCause()} method). (A <tt>null</tt>
65   *      value is permitted, and indicates that the cause is nonexistent or unknown.)
66   * @since 1.4
67   */
68  public ScriptEngineException(Throwable cause) {
69      super(cause);
70  }
71  }
72

```

```
1  /* Generated By:JavaCC; Do not edit this line. JGSL_Parser.java */
2  package jgsl.parser;
3
4
5  import java.awt.Color;
6  import java.util.ArrayList;
7
8  import jgsl.io.ScriptError;
9  import jgsl.io.ScriptParserException;
10 import jgsl.io.ScriptParserUtil;
11 import jgsl.io.ScriptWarning;
12 import jgsl.model.Assignment;
13 import jgsl.model.Command;
14 import jgsl.model.Declaration;
15 import jgsl.model.JGSLColor;
16 import jgsl.model.JGSLDoubble;
17 import jgsl.model.JGSLInteger;
18 import jgsl.model.JGSLScript;
19 import jgsl.model.JGSLString;
20
21 /**
22  *
23  * @author zenarchitect
24  * @version $Id: JGSL_Parser.java,v 1.10 2005/05/21 01:42:08 zenarchitect Exp $
25  */
26
27 public class JGSL_Parser implements JGSL_ParserConstants {
28
29     private JGSLScript script = new JGSLScript();
30
31     public JGSLScript getScript() {
32         return script;
33     }
34
35     public static void main(String args[]) throws ParseException {
36         JGSL_Parser parser;
37         String filename = null;
38         long initTime = 0;
39         long parseTime = 0;
40         long startTime = 0;
41         long stopTime = 0;
42         if (args.length == 0)
43         {
44             System.out.println("jgsl parser: Reading from standard input . . .");
45             parser = new JGSL_Parser(System.in);
46         } else if (args.length == 1)
47         {
48             filename = args[0];
49             System.out.println("jgsl parser: Reading from file " + filename + " . . .");
50             try
51             {
52                 startTime = System.currentTimeMillis();
53                 parser = new JGSL_Parser(new java.io.FileInputStream(filename));
54                 stopTime = System.currentTimeMillis();
55                 initTime = stopTime - startTime;
56             } catch (java.io.FileNotFoundException e)
57             {
58                 System.out.println("jgsl parser: File " + filename + " not found.");
59                 return;
60             }
61         } else
62         {
63             System.out.println("jgsl parser: Usage is one of:");
64             System.out.println("    java jsgLJSGl < <stdin>");
65             System.out.println("OR");
66             System.out.println("    java jsgLJSGl inputfile");
67             return;
68         }
69         try
70         {
71             startTime = System.currentTimeMillis();
72             parser.parseScript();
73             stopTime = System.currentTimeMillis();
74             parseTime = stopTime - startTime;
75             System.out.println("jgsl parser: ");
76             System.out.println("    JGSL file parsed " + filename + " successfully in " + (initTime + parseTime) + " ms.");
77             System.out.println("    initialization time = " + initTime + " ms.");
78             System.out.println("    parse time = " + parseTime + " ms.");
79         } catch (ParseException e)
80         {
81             System.out.println(e.getMessage());
82             System.out.println("jgsl parser: Encountered errors during parse.");
83         }
84     }
85
86     /*
87     <comments | documentation> *
88
89     <BEGIN>
90
91     <attributes | commands | comments | documentation>
92
93     <END>
94
95     */
96
97     /*****
98     * THE JGSL GRAMMAR STARTS HERE *
99     *****/
100
101     /*
102     * Program structuring syntax follows.
103     */
104     final public void parseScript() throws ParseException {
105         Script();
106     }
107 }
```

```
108 final public void Script() throws ParseException {
109     label_1:
110     while (true) {
111         switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
112             case DOC:
113                 ;
114                 break;
115             default:
116                 jj_la1[0] = jj_gen;
117                 break label_1;
118         }
119         Documentation();
120     }
121     switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
122     case BEGIN:
123         ScriptBody();
124         break;
125     default:
126         jj_la1[1] = jj_gen;
127         ;
128     }
129     jj_consume_token(0);
130 }
131
132 final public void Documentation() throws ParseException {
133     Token doc;
134     jj_consume_token(DOC);
135     doc = jj_consume_token(String.LITERAL);
136     script.addDocumentation(doc.image);
137 }
138
139 final public void ScriptBody() throws ParseException {
140     jj_consume_token(BEGIN);
141     label_2:
142     while (true) {
143         switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
144             case CLEAR:
145             case CANVAS:
146             case DRAW:
147             case TEXT:
148             case RECTANGLE:
149             case SQUARE:
150             case CIRCLE:
151             case ELIPSE:
152             case ARC:
153             case POLYGON:
154             case LINE:
155             case WAIT:
156             case LOG:
157             case DEBUG:
158             case ERROR:
159             case WARNING:
160             case DECLARE:
161             case DOC:
162             case IDENTIFIER:
163                 ;
164                 break;
165             default:
166                 jj_la1[2] = jj_gen;
167                 break label_2;
168         }
169         switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
170             case CLEAR:
171             case CANVAS:
172             case DRAW:
173             case TEXT:
174             case RECTANGLE:
175             case SQUARE:
176             case CIRCLE:
177             case ELIPSE:
178             case ARC:
179             case POLYGON:
180             case LINE:
181             case WAIT:
182             case LOG:
183             case DEBUG:
184             case ERROR:
185             case WARNING:
186                 Command();
187                 break;
188             case DECLARE:
189                 Declaration();
190                 break;
191             case IDENTIFIER:
192                 Assignment();
193                 break;
194             case DOC:
195                 Documentation();
196                 break;
197             default:
198                 jj_la1[3] = jj_gen;
199                 jj_consume_token(-1);
200                 throw new ParseException();
201         }
202     }
203     jj_consume_token(END);
204 }
205
206 final public void Command() throws ParseException {
207     try {
208         switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
209             case CANVAS:
210                 Canvas();
211                 break;
212             case CLEAR:
213                 Clear();
214                 break;
215             case WAIT:
216                 Wait();
217                 break;
218             case DRAW:
219                 Draw();
220                 break;
```



```
221 case RECTANGLE:
222 case SQUARE:
223 case CIRCLE:
224 case ELLIPSE:
225 case ARC:
226 case POLYGON:
227 case LINE:
228 DrawShape();
229 break;
230 case TEXT:
231 DrawText();
232 break;
233 case LOG:
234 case DEBUG:
235 case ERROR:
236 case WARNING:
237 Log();
238 break;
239 default:
240 jj_la1[4] = jj_gen;
241 jj_consume_token(-1);
242 throw new ParseException();
243 }
244 jj_consume_token(SEMICOLON);
245 } catch (ParseException e) {
246 error_skipto(SEMICOLON);
247 }
248 }
249
250 void error_skipto(int kind) throws ParseException {
251 ParseException e = generateParseException(); // generate the exception object.
252
253 ScriptError se = new ScriptError(e.getMessage());
254 script.addError(se);
255
256 Token t;
257 do {
258 t = getNextToken();
259 } while (t.kind != kind);
260 }
261
262 final public void Clear() throws ParseException {
263 Token name;
264 name = jj_consume_token(CLEAR);
265 Command c = new Command(name.image);
266 script.add(c);
267 }
268
269 final public Color GetStandardColor() throws ParseException {
270 Token value;
271 try {
272 switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
273 case BLACK:
274 value = jj_consume_token(BLACK);
275 break;
276 case BLUE:
277 value = jj_consume_token(BLUE);
278 break;
279 case DARK_GRAY:
280 value = jj_consume_token(DARK_GRAY);
281 break;
282 case GRAY:
283 value = jj_consume_token(GRAY);
284 break;
285 case GREEN:
286 value = jj_consume_token(GREEN);
287 break;
288 case LIGHT_GRAY:
289 value = jj_consume_token(LIGHT_GRAY);
290 break;
291 case MAGENTA:
292 value = jj_consume_token(MAGENTA);
293 break;
294 case ORANGE:
295 value = jj_consume_token(ORANGE);
296 break;
297 case PINK:
298 value = jj_consume_token(PINK);
299 break;
300 case RED:
301 value = jj_consume_token(RED);
302 break;
303 case WHITE:
304 value = jj_consume_token(WHITE);
305 break;
306 case YELLOW:
307 value = jj_consume_token(YELLOW);
308 break;
309 default:
310 jj_la1[5] = jj_gen;
311 jj_consume_token(-1);
312 throw new ParseException();
313 }
314 } catch (ParseException e) {
315 ScriptError se = new ScriptError(e.getMessage());
316 script.addError(se);
317 {if (true) return null;}
318 }
319 // Color color = Color.RED;
320 String colorName = value.image.toLowerCase();
321 if(colorName.equals("black")) {
322 {if (true) return Color.BLACK;}
323 }
324 else if(colorName.equals("blue")) {
325 {if (true) return Color.BLUE;}
326 }
327 else if(colorName.equals("dark_gray")) {
328 {if (true) return Color.DARK_GRAY;}
329 }
330 else if(colorName.equals("gray")) {
331 {if (true) return Color.GRAY;}
332 }
```

```
332     }
333     else if(colorName.equals("green")) {
334         if (true) return Color.GREEN;
335     }
336     else if(colorName.equals("light_gray")) {
337         if (true) return Color.LIGHT_GRAY;
338     }
339     else if(colorName.equals("magenta")) {
340         if (true) return Color.MAGENTA;
341     }
342     else if(colorName.equals("orange")) {
343         if (true) return Color.ORANGE;
344     }
345     else if(colorName.equals("pink")) {
346         if (true) return Color.PINK;
347     }
348     else if(colorName.equals("red")) {
349         if (true) return Color.RED;
350     }
351     else if(colorName.equals("white")) {
352         if (true) return Color.WHITE;
353     }
354     else if(colorName.equals("yellow")) {
355         if (true) return Color.YELLOW;
356     }
357     throw new Error("Missing return statement in function");
358 }
359
360 final public Color GetRGB() throws ParseException {
361     Token r;
362     Token g;
363     Token b;
364     switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
365     case INTEGER_LITERAL:
366         r = jj_consume_token(INTEGER_LITERAL);
367         jj_consume_token(COMMA);
368         g = jj_consume_token(INTEGER_LITERAL);
369         jj_consume_token(COMMA);
370         b = jj_consume_token(INTEGER_LITERAL);
371         break;
372     case STRING_LITERAL:
373         r = jj_consume_token(STRING_LITERAL);
374         jj_consume_token(COMMA);
375         g = jj_consume_token(STRING_LITERAL);
376         jj_consume_token(COMMA);
377         b = jj_consume_token(STRING_LITERAL);
378         break;
379     default:
380         jj_la1[6] = jj_gen;
381         jj_consume_token(-1);
382         throw new ParseException();
383     }
384     //
385     Color rgb = new Color(0,0,0);
386     int rInt = -1;
387     int gInt = -1;
388     int bInt = -1;
389
390     try {
391         rInt = ScriptParserUtil.parseInt(r.image);
392     }
393     catch(ScriptParserException e) {
394         ScriptError se = new ScriptError(e.getMessage(), r.beginLine, r.beginColumn);
395         script.addError(se);
396     }
397     try {
398         rInt = ScriptParserUtil.parseInt(g.image);
399     }
400     catch(ScriptParserException e) {
401         ScriptError se = new ScriptError(e.getMessage(), g.beginLine, g.beginColumn);
402         script.addError(se);
403     }
404     try {
405         rInt = ScriptParserUtil.parseInt(b.image);
406     }
407     catch(ScriptParserException e) {
408         ScriptError se = new ScriptError(e.getMessage(), b.beginLine, b.beginColumn);
409         script.addError(se);
410     }
411
412     Color c = null;
413
414     if(rInt != -1 && gInt != -1 && bInt != -1) {
415         c = new Color(rInt, gInt, bInt);
416     }
417
418     {if (true) return c;}
419     throw new Error("Missing return statement in function");
420 }
421
422 final public Color GetColor() throws ParseException {
423     Color c;
424     switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
425     case BLACK:
426     case BLUE:
427     case DARK_GRAY:
428     case GRAY:
429     case GREEN:
430     case LIGHT_GRAY:
431     case MAGENTA:
432     case ORANGE:
433     case PINK:
434     case RED:
435     case WHITE:
436     case YELLOW:
437         c = GetStandardColor();
438         break;
439     case INTEGER_LITERAL:
440     case STRING_LITERAL:
441         c = GetRGB();
442         break;
443     default:
444         jj_la1[6] = jj_gen;
445         jj_consume_token(-1);
446         throw new ParseException();
447     }
448
449     Color c = null;
450
451     if(rInt != -1 && gInt != -1 && bInt != -1) {
452         c = new Color(rInt, gInt, bInt);
453     }
454
455     {if (true) return c;}
456     throw new Error("Missing return statement in function");
457 }
```

```

443     jj._la1[7] = jj._gen;
444     jj._consume_token(-1);
445     throw new ParseException();
446 }
447 {if (true) return c;}
448 throw new Error("Missing return statement in function");
449 }
450
451 final public void Canvas() throws ParseException {
452     Token name = null;
453     Token width = null;
454     Token height = null;
455     Color bgcolor = null;
456     Color fgcolor = null;
457     Token title = null;
458     ArrayList attributes = new ArrayList();
459     name = jj._consume_token(CANVAS);
460     switch ((jj._ntk==1)?jj._ntk():jj._ntk) {
461     case COLON:
462         jj._consume_token(COLON);
463         attributes = CanvasAttributes();
464         break;
465     default:
466         jj._la1[8] = jj._gen;
467         ;
468     }
469     jj._consume_token(LPAREN);
470     switch ((jj._ntk==1)?jj._ntk():jj._ntk) {
471     case INTEGER_LITERAL:
472         width = jj._consume_token(INTEGER_LITERAL);
473         break;
474     case STRING_LITERAL:
475         width = jj._consume_token(STRING_LITERAL);
476         break;
477     default:
478         jj._la1[9] = jj._gen;
479         jj._consume_token(-1);
480         throw new ParseException();
481     }
482     jj._consume_token(COMMA);
483     switch ((jj._ntk==1)?jj._ntk():jj._ntk) {
484     case INTEGER_LITERAL:
485         height = jj._consume_token(INTEGER_LITERAL);
486         break;
487     case STRING_LITERAL:
488         height = jj._consume_token(STRING_LITERAL);
489         break;
490     default:
491         jj._la1[10] = jj._gen;
492         jj._consume_token(-1);
493         throw new ParseException();
494     }
495     jj._consume_token(COMMA);
496     bgcolor = GetColor();
497     jj._consume_token(COMMA);
498     fgcolor = GetColor();
499     switch ((jj._ntk==1)?jj._ntk():jj._ntk) {
500     case COMMA:
501         jj._consume_token(COMMA);
502         title = jj._consume_token(STRING_LITERAL);
503         break;
504     default:
505         jj._la1[11] = jj._gen;
506         ;
507     }
508     jj._consume_token(RPAREN);
509     ArrayList parameters = new ArrayList(4);
510
511     if(bgcolor != null) {
512         parameters.add(new JGSLColor("background", bgcolor));
513     }
514     else {
515         parameters.add(new JGSLColor("background", Color.WHITE));
516         ScriptWarning se = new ScriptWarning("Setting canvas background to WHITE.");
517         script.addWarning(se);
518     }
519
520     if(fgcolor != null) {
521         parameters.add(new JGSLColor("foreground", fgcolor));
522     }
523     else {
524         parameters.add(new JGSLColor("foreground", Color.BLACK));
525         ScriptWarning se = new ScriptWarning("Setting canvas foreground to BLACK.");
526         script.addWarning(se);
527     }
528
529     addIniParam(parameters, "width", width);
530     addIniParam(parameters, "height", height);
531
532     if(title != null) {
533         parameters.add(new JGSLString("title", title.image));
534     }
535     Command c = new Command(name.image, attributes, parameters);
536     script.add(c);
537 }
538
539 final public ArrayList CanvasAttributes() throws ParseException {
540     Token attrib;
541     attrib = jj._consume_token(RED);
542     ArrayList attributes = new ArrayList(1);
543     {if (true) return attrib;}
544     throw new Error("Missing return statement in function");
545 }
546
547 final public void Wait() throws ParseException {
548     Token name;
549     Token durationSeconds;
550     name = jj._consume_token(WAIT);
551     jj._consume_token(LPAREN);
552     switch ((jj._ntk==1)?jj._ntk():jj._ntk) {
553     case INTEGER_LITERAL:

```

```
554     durationSeconds = jj_consume_token(INTEGER_LITERAL);
555     break;
556 case STRING_LITERAL:
557     durationSeconds = jj_consume_token(STRING_LITERAL);
558     break;
559 default:
560     jj_la1[12] = jj_gen;
561     jj_consume_token(-1);
562     throw new ParseException();
563 }
564 jj_consume_token(RPAREN);
565 ArrayList parameters = new ArrayList(1);
566 addInParameter(parameters, "duration", durationSeconds);
567 Command c = new Command(name.image, parameters);
568 script.add(c);
569 }
570
571 final public ArrayList DrawAttributes() throws ParseException {
572     Token attrib;
573     attrib = jj_consume_token(RED);
574     ArrayList attributes = new ArrayList(1);
575     {if (true) return attributes;}
576     throw new Error("Missing return statement in function");
577 }
578
579 final public void Draw() throws ParseException {
580     Token name;
581     Token x;
582     Token y;
583     ArrayList attributes = new ArrayList();
584     name = jj_consume_token(DRAW);
585     switch ((jj_ntk== -1)?jj_ntk():jj_ntk) {
586     case COLON:
587         jj_consume_token(COLON);
588         attributes = DrawAttributes();
589         break;
590     default:
591         jj_la1[13] = jj_gen;
592         ;
593     }
594     jj_consume_token(LPAREN);
595     switch ((jj_ntk== -1)?jj_ntk():jj_ntk) {
596     case INTEGER_LITERAL:
597         x = jj_consume_token(INTEGER_LITERAL);
598         break;
599     case STRING_LITERAL:
600         x = jj_consume_token(STRING_LITERAL);
601         break;
602     default:
603         jj_la1[14] = jj_gen;
604         jj_consume_token(-1);
605         throw new ParseException();
606     }
607     jj_consume_token(COMMA);
608     switch ((jj_ntk== -1)?jj_ntk():jj_ntk) {
609     case INTEGER_LITERAL:
610         y = jj_consume_token(INTEGER_LITERAL);
611         break;
612     case STRING_LITERAL:
613         y = jj_consume_token(STRING_LITERAL);
614         break;
615     default:
616         jj_la1[15] = jj_gen;
617         jj_consume_token(-1);
618         throw new ParseException();
619     }
620     jj_consume_token(RPAREN);
621     ArrayList parameters = new ArrayList(2);
622     addInParameter(parameters, "x", x);
623     addInParameter(parameters, "y", y);
624     addInParameter(parameters, "x", x);
625     addInParameter(parameters, "y", y);
626     Command c = new Command(name.image, attributes, parameters);
627     script.add(c);
628 }
629
630 //<COLON> (Attributes())? (<LPAREN> Parameters() <RPAREN>)?
631 final public void Log() throws ParseException {
632     switch ((jj_ntk== -1)?jj_ntk():jj_ntk) {
633     case LOG:
634         Message();
635         break;
636     case WARNING:
637         Warning();
638         break;
639     case DEBUG:
640         Debug();
641         break;
642     case ERROR:
643         Error();
644         break;
645     default:
646         jj_la1[16] = jj_gen;
647         jj_consume_token(-1);
648         throw new ParseException();
649     }
650 }
651
652 final public void Message() throws ParseException {
653     Token name;
654     Token message;
655     name = jj_consume_token(LOG);
656     jj_consume_token(LPAREN);
657     message = jj_consume_token(STRING_LITERAL);
658     jj_consume_token(RPAREN);
659     ArrayList parameters = new ArrayList(1);
660     parameters.add(new JGSI_String("message", message.image));
661     Command c = new Command(name.image, parameters);
662     script.add(c);
663 }
664
```

```

665 final public void Warning() throws ParseException {
666     Token name;
667     Token message;
668     name = jj_consume_token(WARNING);
669     jj_consume_token(LPAREN);
670     message = jj_consume_token(StringLiteral);
671     jj_consume_token(RPAREN);
672     ArrayList parameters = new ArrayList(1);
673     parameters.add(new JGSLString("message", message.image));
674     Command c = new Command(name.image, parameters);
675     script.add(c);
676 }
677
678 final public void Error() throws ParseException {
679     Token name;
680     Token message;
681     name = jj_consume_token(ERROR);
682     jj_consume_token(LPAREN);
683     message = jj_consume_token(StringLiteral);
684     jj_consume_token(RPAREN);
685     ArrayList parameters = new ArrayList(1);
686     parameters.add(new JGSLString("message", message.image));
687     Command c = new Command(name.image, parameters);
688     script.add(c);
689 }
690
691 final public void Debug() throws ParseException {
692     Token name;
693     Token message;
694     name = jj_consume_token(DEBUG);
695     jj_consume_token(LPAREN);
696     message = jj_consume_token(StringLiteral);
697     jj_consume_token(RPAREN);
698     ArrayList parameters = new ArrayList(1);
699     parameters.add(new JGSLString("message", message.image));
700     Command c = new Command(name.image, parameters);
701     script.add(c);
702 }
703
704 final public void DrawShape() throws ParseException {
705     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
706     case LINE:
707         DrawLine();
708         break;
709     case RECTANGLE:
710         DrawRectangle();
711         break;
712     case SQUARE:
713         DrawSquare();
714         break;
715     case CIRCLE:
716         DrawCircle();
717         break;
718     case ELIPSE:
719         DrawEllipse();
720         break;
721     case ARC:
722         DrawArc();
723         break;
724     case POLYGON:
725         DrawPolygon();
726         break;
727     default:
728         jj_la1[17] = jj_gen;
729         jj_consume_token(-1);
730         throw new ParseException();
731     }
732 }
733
734 final public void DrawLine() throws ParseException {
735     Token name;
736     Token x1;
737     Token y1;
738     Token x2;
739     Token y2;
740     ArrayList attributes = new ArrayList();
741     name = jj_consume_token(LINE);
742     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
743     case COLON:
744         jj_consume_token(COLON);
745         attributes = LineAttributes();
746         break;
747     default:
748         jj_la1[18] = jj_gen;
749         ;
750     }
751     jj_consume_token(LPAREN);
752     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
753     case INTEGER_LITERAL:
754         x1 = jj_consume_token(IntegerLiteral);
755         break;
756     case STRING_LITERAL:
757         x1 = jj_consume_token(StringLiteral);
758         break;
759     default:
760         jj_la1[19] = jj_gen;
761         jj_consume_token(-1);
762         throw new ParseException();
763     }
764     jj_consume_token(COMMA);
765     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
766     case INTEGER_LITERAL:
767         y1 = jj_consume_token(IntegerLiteral);
768         break;
769     case STRING_LITERAL:
770         y1 = jj_consume_token(StringLiteral);
771         break;
772     default:
773         jj_la1[20] = jj_gen;
774         jj_consume_token(-1);

```

```
775     throw new ParseException();
776 }
777 jj_consume_token(COMMA);
778 switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
779 case INTEGER_LITERAL:
780     x2 = jj_consume_token(INTEGER_LITERAL);
781     break;
782 case STRING_LITERAL:
783     x2 = jj_consume_token(STRING_LITERAL);
784     break;
785 default:
786     jj_la1[21] = jj_gen;
787     jj_consume_token(-1);
788     throw new ParseException();
789 }
790 jj_consume_token(COMMA);
791 switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
792 case INTEGER_LITERAL:
793     y2 = jj_consume_token(INTEGER_LITERAL);
794     break;
795 case STRING_LITERAL:
796     y2 = jj_consume_token(STRING_LITERAL);
797     break;
798 default:
799     jj_la1[22] = jj_gen;
800     jj_consume_token(-1);
801     throw new ParseException();
802 }
803 jj_consume_token(RPAREN);
804     ArrayList parameters = new ArrayList(4);
805     addInParameter(parameters, "x1", x1);
806     addInParameter(parameters, "y1", y1);
807     addInParameter(parameters, "x2", x2);
808     addInParameter(parameters, "y2", y2);
809     Command c = new Command(name,image, attributes, parameters);
810     script.add(c);
811 }
812
813 final public ArrayList LineAttributes() throws ParseException {
814     Color c;
815     c = GetColor();
816     ArrayList attributes = new ArrayList(1);
817     attributes.add(new JGSLColor("c", c));
818     {if (true) return attributes;}
819     throw new Error("Missing return statement in function");
820 }
821
822 final public void DrawRectangle() throws ParseException {
823     Token name;
824     Token x1;
825     Token y1;
826     Token width;
827     Token height;
828     ArrayList attributes = new ArrayList();
829     name = jj_consume_token(RECTANGLE);
830     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
831     case COLON:
832         jj_consume_token(COLON);
833         attributes = RectangleAttributes();
834         break;
835     default:
836         jj_la1[23] = jj_gen;
837         ;
838     }
839     jj_consume_token(LPAREN);
840     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
841     case INTEGER_LITERAL:
842         x1 = jj_consume_token(INTEGER_LITERAL);
843         break;
844     case STRING_LITERAL:
845         x1 = jj_consume_token(STRING_LITERAL);
846         break;
847     default:
848         jj_la1[24] = jj_gen;
849         jj_consume_token(-1);
850         throw new ParseException();
851     }
852     jj_consume_token(COMMA);
853     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
854     case INTEGER_LITERAL:
855         y1 = jj_consume_token(INTEGER_LITERAL);
856         break;
857     case STRING_LITERAL:
858         y1 = jj_consume_token(STRING_LITERAL);
859         break;
860     default:
861         jj_la1[25] = jj_gen;
862         jj_consume_token(-1);
863         throw new ParseException();
864     }
865     jj_consume_token(COMMA);
866     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
867     case INTEGER_LITERAL:
868         width = jj_consume_token(INTEGER_LITERAL);
869         break;
870     case STRING_LITERAL:
871         width = jj_consume_token(STRING_LITERAL);
872         break;
873     default:
874         jj_la1[26] = jj_gen;
875         jj_consume_token(-1);
876         throw new ParseException();
877     }
878     jj_consume_token(COMMA);
879     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
880     case INTEGER_LITERAL:
881         height = jj_consume_token(INTEGER_LITERAL);
882         break;
883     case STRING_LITERAL:
884         height = jj_consume_token(STRING_LITERAL);
885         break;
```

```

886 default:
887     jj_la1[27] = jj_gen;
888     jj_consume_token(-1);
889     throw new ParseException();
890 }
891 jj_consume_token(RPAREN);
892 ArrayList parameters = new ArrayList(4);
893 addInParameter(parameters, "x1", x1);
894 addInParameter(parameters, "y1", y1);
895 addInParameter(parameters, "width", width);
896 addInParameter(parameters, "height", height);
897 Command c = new Command(name,image, attributes, parameters);
898 script.add(c);
899 }
900
901 final public ArrayList RectangleAttributes() throws ParseException {
902     Color c;
903     c = GetColor();
904     ArrayList attributes = new ArrayList(1);
905     attributes.add(new JGSLColor("e", c));
906     if (true) return attributes;
907     throw new Error("Missing return statement in function");
908 }
909
910 final public void DrawSquare() throws ParseException {
911     Token name;
912     Token x1;
913     Token y1;
914     Token width;
915 ArrayList attributes = new ArrayList();
916 name = jj_consume_token(SQUARE);
917 switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
918 case COLON:
919     jj_consume_token(COLON);
920     attributes = SquareAttributes();
921     break;
922 default:
923     jj_la1[28] = jj_gen;
924     ;
925 }
926 jj_consume_token(LPAREN);
927 switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
928 case INTEGER_LITERAL:
929     x1 = jj_consume_token(INTEGER_LITERAL);
930     break;
931 case STRING_LITERAL:
932     x1 = jj_consume_token(STRING_LITERAL);
933     break;
934 default:
935     jj_la1[29] = jj_gen;
936     jj_consume_token(-1);
937     throw new ParseException();
938 }
939 jj_consume_token(COMMA);
940 switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
941 case INTEGER_LITERAL:
942     y1 = jj_consume_token(INTEGER_LITERAL);
943     break;
944 case STRING_LITERAL:
945     y1 = jj_consume_token(STRING_LITERAL);
946     break;
947 default:
948     jj_la1[30] = jj_gen;
949     jj_consume_token(-1);
950     throw new ParseException();
951 }
952 jj_consume_token(COMMA);
953 switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
954 case INTEGER_LITERAL:
955     width = jj_consume_token(INTEGER_LITERAL);
956     break;
957 case STRING_LITERAL:
958     width = jj_consume_token(STRING_LITERAL);
959     break;
960 default:
961     jj_la1[31] = jj_gen;
962     jj_consume_token(-1);
963     throw new ParseException();
964 }
965 jj_consume_token(RPAREN);
966 ArrayList parameters = new ArrayList(4);
967 addInParameter(parameters, "x1", x1);
968 addInParameter(parameters, "y1", y1);
969 addInParameter(parameters, "width", width);
970 Command c = new Command(name,image, attributes, parameters);
971 script.add(c);
972 }
973
974 final public ArrayList SquareAttributes() throws ParseException {
975     Color c;
976     c = GetColor();
977     ArrayList attributes = new ArrayList(1);
978     attributes.add(new JGSLColor("e", c));
979     if (true) return attributes;
980     throw new Error("Missing return statement in function");
981 }
982
983 final public void DrawCircle() throws ParseException {
984     Token name;
985     Token x1;
986     Token y1;
987     Token radius;
988 ArrayList attributes = new ArrayList();
989 name = jj_consume_token(CIRCLE);
990 switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
991 case COLON:
992     jj_consume_token(COLON);
993     attributes = CircleAttributes();
994     break;
995 default:
996     jj_la1[32] = jj_gen;

```

```
997 ;
998 }
999   jj_consume_token(LPAREN);
1000   switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1001   case INTEGER_LITERAL:
1002     x1 = jj_consume_token(INTEGER_LITERAL);
1003     break;
1004   case STRING_LITERAL:
1005     x1 = jj_consume_token(STRING_LITERAL);
1006     break;
1007   default:
1008     jj_la1[33] = jj_gen;
1009     jj_consume_token(-1);
1010     throw new ParseException();
1011   }
1012   jj_consume_token(COMMA);
1013   switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1014   case INTEGER_LITERAL:
1015     y1 = jj_consume_token(INTEGER_LITERAL);
1016     break;
1017   case STRING_LITERAL:
1018     y1 = jj_consume_token(STRING_LITERAL);
1019     break;
1020   default:
1021     jj_la1[34] = jj_gen;
1022     jj_consume_token(-1);
1023     throw new ParseException();
1024   }
1025   jj_consume_token(COMMA);
1026   switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1027   case FLOATING_POINT_LITERAL:
1028     radius = jj_consume_token(FLOATING_POINT_LITERAL);
1029     break;
1030   case STRING_LITERAL:
1031     radius = jj_consume_token(STRING_LITERAL);
1032     break;
1033   default:
1034     jj_la1[35] = jj_gen;
1035     jj_consume_token(-1);
1036     throw new ParseException();
1037   }
1038   jj_consume_token(RPAREN);
1039   ArrayList parameters = new ArrayList(4);
1040   addInParameter(parameters, "x1", x1);
1041   addInParameter(parameters, "y1", y1);
1042   addDecimalParam(parameters, "radius", radius);
1043   Command c = new Command(name,image, attributes, parameters);
1044   script.add(c);
1045 }
1046
1047 final public ArrayList CircleAttributes() throws ParseException {
1048   Color c;
1049   c = GetColor();
1050   ArrayList attributes = new ArrayList(1);
1051   attributes.add(new JGSLColor("e", c));
1052   if (true) return attributes;
1053   throw new Error("Missing return statement in function");
1054 }
1055
1056 final public void DrawEllipse() throws ParseException {
1057   Token name;
1058   Token x1;
1059   Token y1;
1060   Token width;
1061   Token height;
1062   ArrayList attributes = new ArrayList();
1063   name = jj_consume_token(ELIPSE);
1064   switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1065   case COLON:
1066     jj_consume_token(COLON);
1067     attributes = ElipseAttributes();
1068     break;
1069   default:
1070     jj_la1[36] = jj_gen;
1071     ;
1072   }
1073   jj_consume_token(LPAREN);
1074   switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1075   case INTEGER_LITERAL:
1076     x1 = jj_consume_token(INTEGER_LITERAL);
1077     break;
1078   case STRING_LITERAL:
1079     x1 = jj_consume_token(STRING_LITERAL);
1080     break;
1081   default:
1082     jj_la1[37] = jj_gen;
1083     jj_consume_token(-1);
1084     throw new ParseException();
1085   }
1086   jj_consume_token(COMMA);
1087   switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1088   case INTEGER_LITERAL:
1089     y1 = jj_consume_token(INTEGER_LITERAL);
1090     break;
1091   case STRING_LITERAL:
1092     y1 = jj_consume_token(STRING_LITERAL);
1093     break;
1094   default:
1095     jj_la1[38] = jj_gen;
1096     jj_consume_token(-1);
1097     throw new ParseException();
1098   }
1099   jj_consume_token(COMMA);
1100   switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1101   case INTEGER_LITERAL:
1102     width = jj_consume_token(INTEGER_LITERAL);
1103     break;
1104   case STRING_LITERAL:
1105     width = jj_consume_token(STRING_LITERAL);
1106     break;
1107   default:
1108     jj_la1[39] = jj_gen;
```



```

1109     jj_consume_token(-1);
1110     throw new ParseException();
1111 }
1112 jj_consume_token(COMMA);
1113 switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1114     case INTEGER_LITERAL:
1115         height = jj_consume_token(INTEGER_LITERAL);
1116         break;
1117     case STRING_LITERAL:
1118         height = jj_consume_token(STRING_LITERAL);
1119         break;
1120     default:
1121         jj_la1[40] = jj_gen;
1122         jj_consume_token(-1);
1123         throw new ParseException();
1124 }
1125 jj_consume_token(RPAREN);
1126 ArrayList parameters = new ArrayList(4);
1127     addIntParam(parameters, "x1", x1);
1128     addIntParam(parameters, "y1", y1);
1129     addIntParam(parameters, "width", width);
1130     addIntParam(parameters, "height", height);
1131     Command c = new Command(name,image, attributes, parameters);
1132     script.add(c);
1133 }
1134
1135 final public ArrayList EllipseAttributes() throws ParseException {
1136     Color c;
1137     c = GetColor();
1138     ArrayList attributes = new ArrayList(1);
1139     attributes.add(new JGSLColor("c", c));
1140     if (true) return attributes;
1141     throw new Error("Missing return statement in function");
1142 }
1143
1144 final public void DrawArc() throws ParseException {
1145     Token name;
1146     Token x1;
1147     Token y1;
1148     Token width;
1149     Token height;
1150     Token startAngle;
1151     Token arcAngle;
1152     ArrayList attributes = new ArrayList();
1153     name = jj_consume_token(ARC);
1154     switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1155         case COLON:
1156             jj_consume_token(COLON);
1157             attributes = ArcAttributes();
1158             break;
1159         default:
1160             jj_la1[41] = jj_gen;
1161             ;
1162     }
1163     jj_consume_token(LPAREN);
1164     switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1165         case INTEGER_LITERAL:
1166             x1 = jj_consume_token(INTEGER_LITERAL);
1167             break;
1168         case STRING_LITERAL:
1169             x1 = jj_consume_token(STRING_LITERAL);
1170             break;
1171         default:
1172             jj_la1[42] = jj_gen;
1173             jj_consume_token(-1);
1174             throw new ParseException();
1175     }
1176     jj_consume_token(COMMA);
1177     switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1178         case INTEGER_LITERAL:
1179             y1 = jj_consume_token(INTEGER_LITERAL);
1180             break;
1181         case STRING_LITERAL:
1182             y1 = jj_consume_token(STRING_LITERAL);
1183             break;
1184         default:
1185             jj_la1[43] = jj_gen;
1186             jj_consume_token(-1);
1187             throw new ParseException();
1188     }
1189     jj_consume_token(COMMA);
1190     switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1191         case INTEGER_LITERAL:
1192             width = jj_consume_token(INTEGER_LITERAL);
1193             break;
1194         case STRING_LITERAL:
1195             width = jj_consume_token(STRING_LITERAL);
1196             break;
1197         default:
1198             jj_la1[44] = jj_gen;
1199             jj_consume_token(-1);
1200             throw new ParseException();
1201     }
1202     jj_consume_token(COMMA);
1203     switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1204         case INTEGER_LITERAL:
1205             height = jj_consume_token(INTEGER_LITERAL);
1206             break;
1207         case STRING_LITERAL:
1208             height = jj_consume_token(STRING_LITERAL);
1209             break;
1210         default:
1211             jj_la1[45] = jj_gen;
1212             jj_consume_token(-1);
1213             throw new ParseException();
1214     }
1215     jj_consume_token(COMMA);
1216     switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1217         case INTEGER_LITERAL:
1218             startAngle = jj_consume_token(INTEGER_LITERAL);
1219             break;

```

```

1220 case STRING_LITERAL:
1221     startAngle = jj_consume_token(STRING_LITERAL);
1222     break;
1223     default:
1224         jj_la1[46] = jj_gen;
1225         jj_consume_token(-1);
1226         throw new ParseException();
1227     }
1228     jj_consume_token(COMMA);
1229     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
1230     case INTEGER_LITERAL:
1231         arcAngle = jj_consume_token(INTEGER_LITERAL);
1232         break;
1233     case STRING_LITERAL:
1234         arcAngle = jj_consume_token(STRING_LITERAL);
1235         break;
1236     default:
1237         jj_la1[47] = jj_gen;
1238         jj_consume_token(-1);
1239         throw new ParseException();
1240     }
1241     jj_consume_token(RPAREN);
1242     ArrayList parameters = new ArrayList(4);
1243     addIntParam(parameters, "x1", x1);
1244     addIntParam(parameters, "y1", y1);
1245     addIntParam(parameters, "width", width);
1246     addIntParam(parameters, "height", height);
1247     addIntParam(parameters, "startAngle", startAngle);
1248     addIntParam(parameters, "arcAngle", arcAngle);
1249     Command c = new Command(name,image, attributes, parameters);
1250     script.add(c);
1251 }
1252
1253 final public ArrayList ArcAttributes() throws ParseException {
1254     Color c;
1255     c = GetColor();
1256     ArrayList attributes = new ArrayList(1);
1257     attributes.add(new JGSLColor("c", c));
1258     if (true) return attributes;
1259     throw new Error("Missing return statement in function");
1260 }
1261
1262 final public void DrawPolygon() throws ParseException {
1263     Token name;
1264     Token x1;
1265     Token y1;
1266     Token width;
1267     Token height;
1268     Token startAngle;
1269     Token arcAngle;
1270     ArrayList attributes = new ArrayList();
1271     ArrayList parameters = new ArrayList();
1272     name = jj_consume_token(POLYGON);
1273     switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
1274     case COLON:
1275         jj_consume_token(COLON);
1276         attributes = PolygonAttributes();
1277         break;
1278     default:
1279         jj_la1[48] = jj_gen;
1280         ;
1281     }
1282     jj_consume_token(LPAREN);
1283     PolygonParameters(parameters);
1284     jj_consume_token(RPAREN);
1285     Command c = new Command(name,image, attributes, parameters);
1286     script.add(c);
1287 }
1288
1289 final public ArrayList PolygonAttributes() throws ParseException {
1290     Color c;
1291     c = GetColor();
1292     ArrayList attributes = new ArrayList(1);
1293     attributes.add(new JGSLColor("c", c));
1294     if (true) return attributes;
1295     throw new Error("Missing return statement in function");
1296 }
1297
1298 final public void PolygonParameters(ArrayList parameters) throws ParseException {
1299     Token x1 = null;
1300     Token y1 = null;
1301     label_3:
1302     while (true) {
1303         switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
1304         case INTEGER_LITERAL:
1305             x1 = jj_consume_token(INTEGER_LITERAL);
1306             ;
1307         break;
1308         default:
1309             jj_la1[49] = jj_gen;
1310             break label_3;
1311         }
1312         switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
1313         case INTEGER_LITERAL:
1314             y1 = jj_consume_token(INTEGER_LITERAL);
1315             break;
1316         case STRING_LITERAL:
1317             x1 = jj_consume_token(STRING_LITERAL);
1318             break;
1319         default:
1320             jj_la1[50] = jj_gen;
1321             jj_consume_token(-1);
1322             throw new ParseException();
1323         }
1324         jj_consume_token(COMMA);
1325         switch ((jj_ntk==1)?jj_ntk():jj_ntk) {
1326         case INTEGER_LITERAL:
1327             y1 = jj_consume_token(INTEGER_LITERAL);
1328             break;
1329         case STRING_LITERAL:
1330             y1 = jj_consume_token(STRING_LITERAL);

```

```

1331     break;
1332 default:
1333     jj_la1[51] = jj_gen;
1334     jj_consume_token(-1);
1335     throw new ParseException();
1336 }
1337 switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
1338 case COMMA:
1339     jj_consume_token(COMMA);
1340     break;
1341 default:
1342     jj_la1[52] = jj_gen;
1343     ;
1344 }
1345 }
1346 addIntParam(parameters, "x", x1);
1347 addIntParam(parameters, "y", y1);
1348 }
1349
1350 final public void DrawText() throws ParseException {
1351     Token name;
1352     Token x1;
1353     Token y1;
1354     Token text;
1355     ArrayList attributes = new ArrayList();
1356     name = jj_consume_token(TEXT);
1357     switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
1358     case COLON:
1359         jj_consume_token(COLON);
1360         attributes = TextAttributes();
1361         break;
1362     default:
1363         jj_la1[53] = jj_gen;
1364         ;
1365     }
1366     jj_consume_token(LPAREN);
1367     switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
1368     case INTEGER_LITERAL:
1369         x1 = jj_consume_token(INTEGER_LITERAL);
1370         break;
1371     case STRING_LITERAL:
1372         x1 = jj_consume_token(STRING_LITERAL);
1373         break;
1374     default:
1375         jj_la1[54] = jj_gen;
1376         jj_consume_token(-1);
1377         throw new ParseException();
1378     }
1379     jj_consume_token(COMMA);
1380     switch ((jj_ntk===-1)?jj_ntk():jj_ntk) {
1381     case INTEGER_LITERAL:
1382         y1 = jj_consume_token(INTEGER_LITERAL);
1383         break;
1384     case STRING_LITERAL:
1385         y1 = jj_consume_token(STRING_LITERAL);
1386         break;
1387     default:
1388         jj_la1[55] = jj_gen;
1389         jj_consume_token(-1);
1390         throw new ParseException();
1391     }
1392     jj_consume_token(COMMA);
1393     text = jj_consume_token(STRING_LITERAL);
1394     jj_consume_token(RPAREN);
1395     ArrayList parameters = new ArrayList(3);
1396     addIntParam(parameters, "x1", x1);
1397     addIntParam(parameters, "y1", y1);
1398     parameters.add(new IGSLString("text", text.image));
1399
1400     Command c = new Command(name.image, attributes, parameters);
1401     script.add(c);
1402 }
1403
1404 void addIntParam(ArrayList parameters, String paramName, Token t) throws ParseException {
1405     try {
1406         if(t == null) {
1407             String msg = "Unable to convert value to a number.";
1408             ScriptError se = new ScriptError(msg, t.beginLine, t.beginColumn);
1409             script.addError(se);
1410         }
1411         parameters.add(new IGSLInteger(paramName, t.image));
1412     }
1413     catch(NumberFormatException e) {
1414         String msg = "Unable to convert value " + t.image + " to a number.";
1415         ScriptError se = new ScriptError(msg, t.beginLine, t.beginColumn);
1416         script.addError(se);
1417     }
1418 }
1419
1420 void addDecimalParam(ArrayList parameters, String paramName, Token t) throws ParseException {
1421     try {
1422         parameters.add(new IGSLDouble(paramName, t.image));
1423     }
1424     catch(NumberFormatException e) {
1425         String msg = "Unable to convert value " + t.image + " to a number.";
1426         ScriptError se = new ScriptError(msg, t.beginLine, t.beginColumn);
1427         script.addError(se);
1428     }
1429 }
1430
1431 final public ArrayList TextAttributes() throws ParseException {
1432     Color c;
1433     c = GetColor();
1434     ArrayList attributes = new ArrayList(1);
1435     attributes.add(new IGSLColor("c", c));
1436     if (true) return attributes;
1437     throw new Error("Missing return statement in function");
1438 }
1439
1440 final public void Declaration() throws ParseException {
1441     jj_consume_token(DECLARE);

```

```
1442 switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1443 case COLOR:
1444     DeclareColor();
1445     break;
1446 case IDENTIFIER:
1447     jj_consume_token(IDENTIFIER);
1448     switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1449     case ASSIGN:
1450         jj_consume_token(ASSIGN);
1451         switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1452         case IDENTIFIER:
1453             jj_consume_token(IDENTIFIER);
1454             break;
1455         case INTEGER_LITERAL:
1456             jj_consume_token(INTEGER_LITERAL);
1457             break;
1458         case STRING_LITERAL:
1459             jj_consume_token(STRING_LITERAL);
1460             break;
1461         case FLOATING_POINT_LITERAL:
1462             jj_consume_token(FLOATING_POINT_LITERAL);
1463             break;
1464         default:
1465             jj_la1[56] = jj_gen;
1466             jj_consume_token(-1);
1467             throw new ParseException();
1468         }
1469         break;
1470     default:
1471         jj_la1[57] = jj_gen;
1472         ;
1473     }
1474     break;
1475     default:
1476         jj_la1[58] = jj_gen;
1477         jj_consume_token(-1);
1478         throw new ParseException();
1479     }
1480     jj_consume_token(SEMICOLON);
1481 }
1482
1483 final public void DeclareCanvas() throws ParseException {
1484     Token type;
1485     Token id;
1486     Token tokenValue = null;
1487     type = jj_consume_token(CANVAS);
1488     id = jj_consume_token(IDENTIFIER);
1489     switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1490     case ASSIGN:
1491         jj_consume_token(ASSIGN);
1492         tokenValue = jj_consume_token(CANVAS);
1493         break;
1494     default:
1495         jj_la1[59] = jj_gen;
1496         ;
1497     }
1498     String value = null;
1499     if(tokenValue != null) {
1500         value = tokenValue.image;
1501     }
1502     Declaration d = new Declaration(type.image, id.image, value);
1503     script.add(d);
1504 }
1505
1506 final public void DeclareColor() throws ParseException {
1507     Token type;
1508     Token id;
1509     type = jj_consume_token(COLOR);
1510     id = jj_consume_token(IDENTIFIER);
1511     switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1512     case ASSIGN:
1513         jj_consume_token(ASSIGN);
1514         switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1515         case BLACK:
1516             case BLUE:
1517             case DARK_GRAY:
1518             case GRAY:
1519             case GREEN:
1520             case LIGHT_GRAY:
1521             case MAGENTA:
1522             case ORANGE:
1523             case PINK:
1524             case RED:
1525             case WHITE:
1526             case YELLOW:
1527                 DeclareStandardColor(type, id);
1528                 break;
1529             case INTEGER_LITERAL:
1530             case STRING_LITERAL:
1531                 DeclareRGB(type, id);
1532                 break;
1533             default:
1534                 jj_la1[60] = jj_gen;
1535                 jj_consume_token(-1);
1536                 throw new ParseException();
1537             }
1538             break;
1539         default:
1540             jj_la1[61] = jj_gen;
1541             ;
1542         }
1543     }
1544
1545     final public void DeclareStandardColor(Token type, Token id) throws ParseException {
1546         Token value;
1547         switch ((jj_ntk==--1)?jj_ntk():jj_ntk) {
1548         case BLACK:
1549             value = jj_consume_token(BLACK);
1550             break;
1551         case BLUE:
1552             value = jj_consume_token(BLUE);
1553             break;
```





```
1775     for (int i = 0; i < 65; i++) {
1776         if (ji_la1[i] == ji_gen) {
1777             for (int j = 0; j < 32; j++) {
1778                 if ((ji_la1_0[i] & (1<<j)) != 0) {
1779                     la1tokens[j] = true;
1780                 }
1781                 if ((ji_la1_1[i] & (1<<j)) != 0) {
1782                     la1tokens[32+j] = true;
1783                 }
1784                 if ((ji_la1_2[i] & (1<<j)) != 0) {
1785                     la1tokens[64+j] = true;
1786                 }
1787             }
1788         }
1789     }
1790     for (int i = 0; i < 96; i++) {
1791         if (la1tokens[i]) {
1792             ji_exptentry = new int[1];
1793             ji_exptentry[0] = i;
1794             ji_exptentries.addElement(ji_exptentry);
1795         }
1796     }
1797     int[][] exptokseq = new int[ji_exptentries.size()][2];
1798     for (int i = 0; i < ji_exptentries.size(); i++) {
1799         exptokseq[i] = (int[])ji_exptentries.elementAt(i);
1800     }
1801     return new ParseException(token, exptokseq, tokenImage);
1802 }
1803
1804 final public void enable_tracing() {
1805 }
1806
1807 final public void disable_tracing() {
1808 }
1809
1810 }
1811
```

```

1  /* Generated By:JavaCC: Do not edit this line. ParseException.java Version 3.0 */
2  package jgsl.parser;
3
4  /**
5   * This exception is thrown when parse errors are encountered. You can explicitly create objects of this exception type
6   * by calling the method generateParseException in the generated parser.
7   * <p/>
8   * You can modify this class to customize your error reporting mechanisms so long as you retain the public fields.
9   */
10 public class ParseException extends Exception {
11     /**
12      * This constructor is used by the method "generateParseException" in the generated parser. Calling this constructor
13      * generates a new object of this type with the fields "currentToken", "expectedTokenSequences", and "tokenImage" set.
14      * The boolean flag "specialConstructor" is also set to true to indicate that this constructor was used to create
15      * this object. This constructor calls its super class with the empty string to force the "toString" method of parent
16      * class "Throwable" to print the error message in the form: ParseException: <result of getMessage>
17      */
18     public ParseException(Token currentTokenVal,
19                          int[][] expectedTokenSequencesVal,
20                          String[] tokenImageVal
21     ) {
22         super("");
23         specialConstructor = true;
24         currentToken = currentTokenVal;
25         expectedTokenSequences = expectedTokenSequencesVal;
26         tokenImage = tokenImageVal;
27     }
28
29     /**
30      * The following constructors are for use by you for whatever purpose you can think of. Constructing the exception in
31      * this manner makes the exception behave in the normal way - i.e., as documented in the class "Throwable". The
32      * fields "errorToken", "expectedTokenSequences", and "tokenImage" do not contain relevant information. The
JavaCC
33      * generated code does not use these constructors.
34      */
35
36     public ParseException() {
37         super();
38         specialConstructor = false;
39     }
40
41     public ParseException(String message) {
42         super(message);
43         specialConstructor = false;
44     }
45
46     /**
47      * This variable determines which constructor was used to create this object and thereby affects the semantics of the

```



```

48     * "getMessage" method (see below).
49     */
50     protected boolean specialConstructor;
51
52     /**
53     * This is the last token that has been consumed successfully. If this object has been created due to a parse error,
54     * the token following this token will (therefore) be the first error token.
55     */
56     public Token currentToken;
57
58     /**
59     * Each entry in this array is an array of integers. Each array of integers represents a sequence of tokens (by their
60     * ordinal values) that is expected at this point of the parse.
61     */
62     public int[][] expectedTokenSequences;
63
64     /**
65     * This is a reference to the "tokenImage" array of the generated parser within which the parse error occurred. This
66     * array is defined in the generated ...Constants interface.
67     */
68     public String[] tokenImage;
69
70     /**
71     * This method has the standard behavior when this object has been created using the standard constructors.
72     * Otherwise, it uses "currentToken" and "expectedTokenSequences" to generate a parse error message and returns it.
73     * If this object has been created due to a parse error, and you do not catch it (it gets thrown from the parser),
74     * then this method is called during the printing of the final stack trace, and hence the correct error message gets
75     * displayed.
76     */
77     public String getMessage() {
78         if (!specialConstructor) {
79             return super.getMessage();
80         }
81         String expected = "";
82         int maxSize = 0;
83         for (int i = 0; i < expectedTokenSequences.length; i++) {
84             if (maxSize < expectedTokenSequences[i].length) {
85                 maxSize = expectedTokenSequences[i].length;
86             }
87             for (int j = 0; j < expectedTokenSequences[i].length; j++) {
88                 expected += tokenImage[expectedTokenSequences[i][j]] + " ";
89             }
90             if (expectedTokenSequences[i][expectedTokenSequences[i].length - 1] != 0) {
91                 expected += "...";
92             }
93             expected += eol + "    ";
94         }
95         String retval = "Encountered \";
96         Token tok = currentToken.next;
97         for (int i = 0; i < maxSize; i++) {
98             if (i != 0) retval += " ";
99             if (tok.kind == 0) {

```

```

100         retval += tokenImage[0];
101         break;
102     }
103     retval += add_escapes(tok.image);
104     tok = tok.next;
105 }
106 retval += "\" at line " + currentToken.next.beginLine + ", column " + currentToken.next.beginColumn;
107 retval += "." + eol;
108 if (expectedTokenSequences.length == 1) {
109     retval += "Was expecting:" + eol + " ";
110 } else {
111     retval += "Was expecting one of:" + eol + " ";
112 }
113 retval += expected;
114 return retval;
115 }
116
117 /**
118  * The end of line string for this machine.
119  */
120 protected String eol = System.getProperty("line.separator", "\n");
121
122 /**
123  * Used to convert raw characters to their escaped version when these raw version cannot be used as part of an ASCII
124  * string literal.
125  */
126 protected String add_escapes(String str) {
127     StringBuffer retval = new StringBuffer();
128     char ch;
129     for (int i = 0; i < str.length(); i++) {
130         switch (str.charAt(i)) {
131             case 0 :
132                 continue;
133             case '\b':
134                 retval.append("\\b");
135                 continue;
136             case '\t':
137                 retval.append("\\t");
138                 continue;
139             case '\n':
140                 retval.append("\\n");
141                 continue;
142             case '\f':
143                 retval.append("\\f");
144                 continue;
145             case '\r':
146                 retval.append("\\r");
147                 continue;
148             case '\"':
149                 retval.append("\\\"");
150                 continue;
151             case '\\':

```

```
152         retval.append("\\");
153         continue;
154     case '\\':
155         retval.append("\\");
156         continue;
157     default:
158         if ((ch = str.charAt(i)) < 0x20 || ch > 0x7e) {
159             String s = "0000" + Integer.toString(ch, 16);
160             retval.append("\\u" + s.substring(s.length() - 4, s.length()));
161         } else {
162             retval.append(ch);
163         }
164         continue;
165     }
166 }
167 return retval.toString();
168 }
169
170 }
171
```

```

1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.io;
5
6      /**
7       * A single script error with line and column info.
8       *
9       * @author zenarchitect
10      * @version $Id: ScriptError.java,v 1.2 2005/05/16 00:54:16 zenarchitect Exp $
11      */
12      public class ScriptError implements Message {
13          private String message;
14          private int lineNumber;
15          private int colNumber;
16
17          public ScriptError(String message) {
18              this.message = message;
19          }
20
21
22          public ScriptError(String message, int lineNumber, int colNumber) {
23              this.message = message;
24              this.lineNumber = lineNumber;
25              this.colNumber = colNumber;
26          }
27
28          /**
29           * Returns the type of message
30           *
31           * @return MessageType
32           */
33          public MessageType getType() {
34              return Message.MessageType.ERROR;
35          }
36
37          /**
38           * Retuns the message
39           *
40           * @return A string containing a simple message
41           */
42          public String getMessage() {
43              return "\n>>>>> " + getType() + ": at line " + lineNumber + ", column " + colNumber + "\n\t" + message +
44              "\n>>>>>\n";
45          }

```

```
46  /**
47   * Return a detailed message
48   *
49   * @return A string containing a detailed message
50   */
51  public String getDetailMessage() {
52      return null; //TODO: To change body of implemented methods use File | Settings | File Templates.
53  }
54  }
55
```

```

1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.io;
5
6      /**
7       * A general script message generated by the parser.
8       *
9       * @author zenarchitect
10      * @version $Id: ScriptMessage.java,v 1.2 2005/05/16 00:54:16 zenarchitect Exp $
11      */
12      public class ScriptMessage implements Message {
13          private String message;
14
15          public ScriptMessage(String message) {
16              this.message = message;
17          }
18
19          /**
20           * Returns the type of message
21           *
22           * @return MessageType
23           */
24          public MessageType getType() {
25              return Message.MessageType.MESSAGE;
26          }
27
28          /**
29           * Returns the message
30           *
31           * @return A string containing a simple message
32           */
33          public String getMessage() {
34              return ">>>>> " + getType() + ":\n" + message + ">>>>>\n\n";
35          }
36
37          /**
38           * Return a detailed message
39           *
40           * @return A string containing a detailed message
41           */

```

```
42     public String getDetailMessage() {  
43         return null; //TODO: To change body of implemented methods use File | Settings | File  
Templates.  
44     }  
45 }  
46
```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/io/Message.java
```

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.io;
5
6      /**
7       * Interface for message types
8       *
9       * @author ZenArchitect
10      * @version $Id: Message.java,v 1.2 2005/05/16 00:54:16 zenarchitect Exp $
11      */
12     public interface Message {
13         /**
14          * Message types
15          */
16         public enum MessageType { MESSAGE, WARNING, ERROR }
17
18         ;
19
20         /**
21          * Returns the type of message
22          *
23          * @return MessageType
24          */
25         public MessageType getType();
26
27         /**
28          * Returns the message
29          *
30          * @return A string containing a simple message
31          */
32         public String getMessage();
33
34         /**
35          * Return a detailed message
36          *
37          * @return A string containing a detailed message
38          */
39         public String getDetailMessage();
```



```
40  
41 }  
42
```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/io/ParseStatus.java
```

```
1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.io;
5
6  /**
7   * ParseStatus interface for collecting and reporting errors and error counts.
8   *
9   * @author zenarchitect
10  * @version $Id: ParseStatus.java,v 1.2 2005/05/16 00:54:16 zenarchitect Exp $
11  */
12  public interface ParseStatus {
13      /**
14       * Add a ScriptError to the parse status
15       *
16       * @param se
17       */
18      public void addError(ScriptError se);
19
20      /**
21       * Add a ScriptWarning to the parse status
22       *
23       * @param sw
24       */
25      public void addWarning(ScriptWarning sw);
26
27      /**
28       * Add a ScriptMessage to the parse status
29       *
30       * @param sm
31       */
32      public void addMessage(ScriptMessage sm);
33
34      /**
35       * Return the error state of the script
36       *
37       * @return true of the script contains errors or false otherwise
38       */
39      boolean hasErrors();
```

```
40
41  /**
42   * Return the warning state of the script
43   *
44   * @return true of the script contains warnings or false otherwise
45   */
46  boolean hasWarnings();
47
48  /**
49   * Return the message state of the script
50   *
51   * @return true of the script contains messages or false otherwise
52   */
53  boolean hasMessages();
54  }
55
```

/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/io/ScriptWarning.java

```

1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4  package jgsl.io;
5
6  /**
7   * Record a script warning message.
8   *
9   * @author zenarchitect
10  * @version $Id: ScriptWarning.java,v 1.2 2005/05/16 00:54:17 zenarchitect Exp $
11  */
12  public class ScriptWarning implements Message {
13      private String message;
14      private int lineNumber = -1;
15      private int colNumber = -1;
16
17      public ScriptWarning(String message) {
18          this.message = message;
19      }
20
21      public ScriptWarning(String message, int lineNumber, int colNumber) {
22          this.message = message;
23          this.lineNumber = lineNumber;
24          this.colNumber = colNumber;
25      }
26
27      /**
28       * Returns the type of message
29       *
30       * @return MessageType
31       */
32      public MessageType getType() {
33          return Message.MessageType.WARNING;
34      }
35
36      /**
37       * Returns the message
38       *
39       * @return A string containing a simple message
40       */
41      public String getMessage() {

```

```
42     return ">>>>> " + getType() + ":\n" + message + ">>>>>\n\n";
43 }
44
45 /**
46  * Return a detailed message
47  *
48  * @return A string containing a detailed message
49  */
50 public String getDetailMessage() {
51     return null; //TODO: To change body of implemented methods use File | Settings | File
Templates.
52 }
53 }
54
```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/model/Script.java
```

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4
5  package jgsl.model;
6
7
8  /**
9   * The script interface provide the set of operations for a script
10  *
11  * @author zenarchitect
12  * @version $Id: Script.java,v 1.2 2005/05/16 00:54:19 zenarchitect Exp $
13  */
14  public interface Script {
15      /**
16       * Get the script name
17       *
18       * @return String containing the script name
19       */
20      String getScriptName();
21
22      /**
23       * Set the script name
24       *
25       * @param scriptName name of the script file
26       */
27      void setScriptName(String scriptName);
28
29      /**
30       * Return the Java implementation of this script
31       *
32       * @return the Java language implementation of this script
33       */
34      String getJava();
35
36      /**
37       * Returns the JGSL script documentation as specified in the DOC keyword by the script author.
38       *
39       * @return The script documentation
```

```
40     */  
41     String getDocumentation();  
42  
43 }  
44
```

```

1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4
5  package jgsl.model;
6
7  import java.io.Serializable;
8  import java.util.ArrayList;
9
10 /**
11  * A documentation statement is one that contains documentation of the JGSL script as written by the script
12  * author.
13  *
14  * @author zenarchitect
15  * @version $Id: Documentation.java,v 1.3 2005/05/16 00:54:18 zenarchitect Exp $
16  */
17 public class Documentation implements Statement, Serializable {
18     private ArrayList<String> docs;
19
20     public Documentation() {
21         docs = new ArrayList<String>();
22     }
23
24     public void addDoc(String doc) {
25         docs.add(doc);
26     }
27
28     /**
29     * This method returns the Java language equivalent of the JGSL statement.
30     *
31     * @return Java language statement from the JGSL
32     */
33     public String getJava() {
34         StringBuilder docString = new StringBuilder();
35
36         for (String doc : docs) {
37             docString.append(doc);
38         }
39
40         return docString.toString();
41     }
42
43     /**
44     * Set the JGSL statement body
45     */

```



```
45  public void setJGSL(String jgsl) {
46      //TODO: To change body of implemented methods use File | Settings | File Templates.
47  }
48
49  /**
50   * Return the type of statement. The String form of the class name.
51   */
52  public String getType() {
53      return "Documentation";
54  }
55 }
56
```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/model/Statement.java
```

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4
5  package jgsl.model;
6
7  /**
8   * The Statement interface provides the set of operations common to all JGSL statements.
9   *
10  * @author zenarchitect
11  * @version $Id: Statement.java,v 1.2 2005/05/16 00:54:19 zenarchitect Exp $
12  */
13  public interface Statement {
14      /**
15       * This method returns the Java language equivalent of the JGSL statement.
16       *
17       * @return Java language statement from the JGSL
18       */
19      String getJava();
20
21      /**
22       * Set the JGSL statement body
23       */
24      void setJGSL(String jgsl);
25
26      /**
27       * Return the type of statement. The String form of the class name.
28       */
29      String getType();
30  }
31
```

```

1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.model;
5
6  import java.util.ArrayList;
7  import java.util.ResourceBundle;
8
9  /**
10 * Enum of all possible commands and their corresponding command template and java code representation.
11 *
12 * @author zenarchitect
13 * @version $Id: Commands.java,v 1.3 2005/05/21 01:42:07 zenarchitect Exp $
14 */
15
16 public enum Commands {
17
18     DRAW("draw"),
19     WAIT("wait"),
20     CLEAR("clear"),
21     LINE("line"),
22     SQUARE("square"),
23     RECTANGLE("rectangle"),
24     CIRCLE("circle"),
25     ELIPSE("ellipse"),
26     ARC("arc"),
27     POLYGON("polygon"),
28     CANVAS("canvas"),
29     TEXT("text"),
30     LOG("log"),
31     DEBUG("debug"),
32     ERROR("error"),
33     WARNING("warning");
34
35     private String name;
36
37     private Commands(String name) {
38         this.name = name;
39     }
40
41     public String getName() {
42         return this.name;
43     }
44
45     public String getCommandTemplate() {
46         switch (this) {
47             case CLEAR:
48                 return "scriptClearCanvas(g2);\n";
49             case WAIT:

```

```

50         return "scriptSleep(%s);\n";
51     case DRAW:
52     case LINE:
53         return "%s\n" +
54             "scriptDrawLine(g2, %s, %s, %s, %s, %s);\n";
55     case SQUARE:
56     case RECTANGLE:
57         return "%s\n" +
58             "scriptDrawRectangle(g2, %s, %s, %s, %s, %s);\n";
59     case CIRCLE:
60         return "%s\n" +
61             "scriptDrawCircle(g2, %s, %s, %s, %s);\n";
62     case ELIPSE:
63         return "%s\n" +
64             "scriptDrawEllipse(g2, %s, %s, %s, %s, %s);\n";
65     case ARC:
66         return "%s\n" +
67             "scriptDrawArc(g2, %s, %s, %s, %s, %s, %s, %s);\n";
68     case POLYGON:
69         return "%s\n" +
70             "int xp[] = { %s};\n" +
71             "int yp[] = { %s};\n" +
72             "scriptDrawPolygon(g2, xp, yp, %s);\n";
73     case CANVAS:
74         return "%s\n" +
75             "%s\n" +
76             "this.setSize(%s, %s);\n" +
77             "this.setBackground(%s);\n" +
78             "this.setForeground(%s);\n" +
79             "this.setTitle(%s);\n";
80     case TEXT:
81         return "%s\n" +
82             "scriptDrawString(g2, %s, %s, %s, %s);\n";
83     case LOG:
84         return "jgslLogger.info(%s);\n";
85     case WARNING:
86         return "jgslLogger.warn(%s);\n";
87     case ERROR:
88         return "jgslLogger.error(%s);\n";
89     case DEBUG:
90         return "jgslLogger.debug(%s);\n";
91     default:
92         return "jgslLogger.debug(\"Unknown command\")\n";
93     }
94 }
95
96 public String getFormattedCommand(ArrayList<Argument> attributes, ArrayList<Argument> parameters) {
97     String cmd = this.getCommandTemplate();
98     String colorVarName = "null";
99     String colorDecl = "";
100     switch (this) {
101         case CLEAR:
102             return cmd;

```

```

103     case WAIT:
104         JGSLInteger duration = (JGSLInteger) parameters.get(0);
105         return String.format(cmd, duration.getJavaValue());
106     case DRAW:
107     case LINE:
108         JGSLInteger x1 = (JGSLInteger) parameters.get(0);
109         JGSLInteger y1 = (JGSLInteger) parameters.get(1);
110         JGSLInteger x2 = (JGSLInteger) parameters.get(2);
111         JGSLInteger y2 = (JGSLInteger) parameters.get(3);
112         if (attributes.size() == 1) {
113             JGSLColor color = (JGSLColor) attributes.get(0);
114             colorVarName = color.getName();
115             colorDecl = color.getJavaValue();
116         }
117         return String.format(cmd, colorDecl, x1.getJavaValue(), y1.getJavaValue(), x2.getJavaValue(), y2.getJavaValue
118         (), colorVarName);
119     case SQUARE:
120         JGSLInteger xs = (JGSLInteger) parameters.get(0);
121         JGSLInteger ys = (JGSLInteger) parameters.get(1);
122         JGSLInteger ws = (JGSLInteger) parameters.get(2);
123         if (attributes.size() == 1) {
124             JGSLColor color = (JGSLColor) attributes.get(0);
125             colorVarName = color.getName();
126             colorDecl = color.getJavaValue();
127         }
128         return String.format(cmd, colorDecl, xs.getJavaValue(), ys.getJavaValue(), ws.getJavaValue(), ws.getJavaValue
129         (), colorVarName);
130     case RECTANGLE:
131         JGSLInteger xr = (JGSLInteger) parameters.get(0);
132         JGSLInteger yr = (JGSLInteger) parameters.get(1);
133         JGSLInteger wr = (JGSLInteger) parameters.get(2);
134         JGSLInteger hr = (JGSLInteger) parameters.get(2);
135         if (attributes.size() == 1) {
136             JGSLColor color = (JGSLColor) attributes.get(0);
137             colorVarName = color.getName();
138             colorDecl = color.getJavaValue();
139         }
140         return String.format(cmd, colorDecl, xr.getJavaValue(), yr.getJavaValue(), wr.getJavaValue(), hr.getJavaValue
141         (), colorVarName);
142     case CIRCLE:
143         JGSLInteger xc = (JGSLInteger) parameters.get(0);
144         JGSLInteger yc = (JGSLInteger) parameters.get(1);
145         JGSLDouble rc = (JGSLDouble) parameters.get(2);
146         if (attributes.size() == 1) {
147             JGSLColor color = (JGSLColor) attributes.get(0);
148             colorVarName = color.getName();
149             colorDecl = color.getJavaValue();
150         }
151         return String.format(cmd, colorDecl, xc.getJavaValue(), yc.getJavaValue(), rc.getJavaValue(), colorVarName);
152     case ELIPSE:

```

```

150     JGSLInteger xe = (JGSLInteger) parameters.get(0);
151     JGSLInteger ye = (JGSLInteger) parameters.get(1);
152     JGSLInteger we = (JGSLInteger) parameters.get(2);
153     JGSLInteger he = (JGSLInteger) parameters.get(2);
154     if (attributes.size() == 1) {
155         JGSLColor color = (JGSLColor) attributes.get(0);
156         colorVarName = color.getName();
157         colorDecl = color.getJavaValue();
158     }
159     return String.format(cmd, colorDecl, xe.getJavaValue(), ye.getJavaValue(), we.getJavaValue(), he.getJavaValue
(), colorVarName);
160     case ARC:
161         JGSLInteger xa = (JGSLInteger) parameters.get(0);
162         JGSLInteger ya = (JGSLInteger) parameters.get(1);
163         JGSLInteger wa = (JGSLInteger) parameters.get(2);
164         JGSLInteger ha = (JGSLInteger) parameters.get(2);
165         JGSLInteger sa = (JGSLInteger) parameters.get(2);
166         JGSLInteger aa = (JGSLInteger) parameters.get(2);
167         if (attributes.size() == 1) {
168             JGSLColor color = (JGSLColor) attributes.get(0);
169             colorVarName = color.getName();
170             colorDecl = color.getJavaValue();
171         }
172         return String.format(cmd, colorDecl, xa.getJavaValue(), ya.getJavaValue(), wa.getJavaValue(), ha.getJavaValue
(),
173             sa.getJavaValue(), aa.getJavaValue(), colorVarName);
174     case POLYGON:
175         String xp = "";
176         String yp = "";
177         for (int i = 0; i < parameters.size(); i += 2) {
178             xp += ((JGSLInteger) parameters.get(i)).getJavaValue();
179             yp += ((JGSLInteger) parameters.get(i + 1)).getJavaValue();
180             if (i < parameters.size() - 2) {
181                 xp += ",";
182                 yp += ",";
183             }
184         }
185         if (attributes.size() == 1) {
186             JGSLColor color = (JGSLColor) attributes.get(0);
187             colorVarName = color.getName();
188             colorDecl = color.getJavaValue();
189         }
190         return String.format(cmd, colorDecl, xp, yp, colorVarName);
191     case TEXT:
192         JGSLInteger x = (JGSLInteger) parameters.get(0);
193         JGSLInteger y = (JGSLInteger) parameters.get(1);
194         JGSLString text = (JGSLString) parameters.get(2);
195         if (attributes.size() == 1) {
196             JGSLColor color = (JGSLColor) attributes.get(0);
197             colorVarName = color.getName();

```

```

198         colorDecl = color.getJavaValue();
199     }
200     return String.format(cmd, colorDecl, text.getJavaValue(), x.getJavaValue(), y.getJavaValue(), colorVarName);
201 case CANVAS:
202     JGSLColor bg = (JGSLColor) parameters.get(0);
203     JGSLColor fg = (JGSLColor) parameters.get(1);
204     JGSLInteger w = (JGSLInteger) parameters.get(2);
205     JGSLInteger h = (JGSLInteger) parameters.get(3);
206     String title = "\"" + ResourceBundle.getBundle("jgsl.resources.JGSL").getString("app.viewer.default-
name") + "\"";
207     if (parameters.size() == 5) {
208         title = ((JGSLString) parameters.get(4)).getJavaValue();
209     }
210     return String.format(cmd, bg.getJavaValue(), fg.getJavaValue(), w.getJavaValue(), h.getJavaValue(), bg.
getName(), fg.getName(), title);
211 case LOG:
212     JGSLString logMsg = (JGSLString) parameters.get(0);
213     return String.format(cmd, logMsg.getJavaValue());
214 case WARNING:
215     JGSLString warningMsg = (JGSLString) parameters.get(0);
216     return String.format(cmd, warningMsg.getJavaValue());
217 case ERROR:
218     JGSLString errorMsg = (JGSLString) parameters.get(0);
219     return String.format(cmd, errorMsg.getJavaValue());
220 case DEBUG:
221     JGSLString debugMsg = (JGSLString) parameters.get(0);
222     return String.format(cmd, debugMsg.getJavaValue());
223 default:
224     return "System.out.println(\"Unknown command\")";
225 }
226 }
227 }
228

```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/io/ImageFileFilter.java
```

```
1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.io;
5
6
7  import javax.swing.filechooser.FileFilter;
8  import java.io.File;
9
10 /**
11  * FileFilter for Image files
12  *
13  * @author zenarchitect
14  * @version $Id: ImageFileFilter.java,v 1.1 2005/05/21 01:42:07 zenarchitect Exp $
15  */
16 public class ImageFileFilter extends FileFilter {
17     //Accept all directories and all JAR files.
18     public boolean accept(File f) {
19         if (f.isDirectory()) {
20             return true;
21         }
22
23         String extension = ImageUtils.getExtension(f);
24         if (extension != null) {
25             if (extension.equalsIgnoreCase(ImageUtils.jpg) ||
26                 extension.equalsIgnoreCase(ImageUtils.png) ||
27                 extension.equalsIgnoreCase(ImageUtils.gif) ||
28                 extension.equalsIgnoreCase(ImageUtils.bmp)) {
29                 return true;
30             } else {
31                 return false;
32             }
33         }
34
35         return false;
36     }
37
38     public String getDescription() {
39         return "Image Files (*.jpg, *.png, *.gif, *.bmp)";
40     }
41 }
```



```
40     }
41 }
42
43 class ImageUtils {
44     public final static String jpg = "jpg";
45     public final static String png = "png";
46     public final static String gif = "gif";
47     public final static String bmp = "bmp";
48
49     /*
50      * Get the extension of a file.
51      */
52     public static String getExtension(File f) {
53         String ext = null;
54         String s = f.getName();
55         int i = s.lastIndexOf('.');
56
57         if (i > 0 && i < s.length() - 1) {
58             ext = s.substring(i + 1).toLowerCase();
59         }
60         return ext;
61     }
62 }
```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/io/JARFileFilter.java
```

```
1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.io;
5
6
7  import javax.swing.filechooser.FileFilter;
8  import java.io.File;
9
10 /*
11  * FileFilter for .JAR files
12  *
13  * @author zenarchitect
14  * @version $Id: JARFileFilter.java,v 1.2 2005/05/16 00:54:16 zenarchitect Exp $
15  */
16 public class JARFileFilter extends FileFilter {
17     //Accept all directories and all JAR files.
18     public boolean accept(File f) {
19         if (f.isDirectory()) {
20             return true;
21         }
22
23         String extension = JGSLUtils.getExtension(f);
24         if (extension != null) {
25             if (extension.equals(JARUtils.jar)) {
26                 return true;
27             } else {
28                 return false;
29             }
30         }
31
32         return false;
33     }
34
35     //The description of this filter
36     public String getDescription() {
37         return "JAR Files (*.jar)";
38     }
39 }
```

```
40
41 class JARUtils {
42     public final static String jar = "jar";
43
44     /*
45      * Get the extension of a file.
46      */
47     public static String getExtension(File f) {
48         String ext = null;
49         String s = f.getName();
50         int i = s.lastIndexOf('.');
51
52         if (i > 0 && i < s.length() - 1) {
53             ext = s.substring(i + 1).toLowerCase();
54         }
55         return ext;
56     }
57 }
```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/io/JGSLFileFilter.java
```

```
1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.io;
5
6
7  import javax.swing.filechooser.FileFilter;
8  import java.io.File;
9
10 /**
11  * FileFilter for .jgsl files
12  *
13  * @author zenarchitect
14  * @version $Id: JGSLFileFilter.java,v 1.5 2005/05/16 00:54:16 zenarchitect Exp $
15  */
16 public class JGSLFileFilter extends FileFilter {
17     //Accept all directories and all jgsl files.
18     public boolean accept(File f) {
19         if (f.isDirectory()) {
20             return true;
21         }
22
23         String extension = JGSLUtils.getExtension(f);
24         if (extension != null) {
25             if (extension.equals(JGSLUtils.jgsl)) {
26                 return true;
27             } else {
28                 return false;
29             }
30         }
31
32         return false;
33     }
34
35     //The description of this filter
36     public String getDescription() {
37         return "JGSL Files (*.jgsl)";
38     }
39 }
```

```
40
41 class JGSLUtils {
42     public final static String jgsl = "jgsl";
43
44     /*
45      * Get the extension of a file.
46      */
47     public static String getExtension(File f) {
48         String ext = null;
49         String s = f.getName();
50         int i = s.lastIndexOf('.');
51
52         if (i > 0 && i < s.length() - 1) {
53             ext = s.substring(i + 1).toLowerCase();
54         }
55         return ext;
56     }
57 }
```

/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/view/ScriptViewer.java

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.view;
5
6      /**
7       * Interface for JGSL script viewer windows.
8       *
9       * @author zenarchitect
10      * @version $Id: ScriptViewer.java,v 1.3 2005/05/21 01:42:12 zenarchitect Exp $
11      */
12
13     public interface ScriptViewer {
14         /**
15          * Render the script code in fullClassName to a GUI window.
16          *
17          * @param fullClassName
18          */
19         void renderScript(String fullClassName, String saveToFileType);
20
21     }
22
```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/io/ScriptParserUtil.java
```

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.io;
5
6      /**
7       * Parse a Sting into and int
8       *
9       * @author zenarchitect
10      * @version $Id: ScriptParserUtil.java,v 1.2 2005/05/16 00:54:17 zenarchitect Exp $
11      */
12      public class ScriptParserUtil {
13          public static int parseInt(String val) throws ScriptParserException {
14              int retVal = 0;
15              try {
16                  Integer rInt = Integer.parseInt(val);
17                  retVal = rInt.intValue();
18              }
19              catch (NumberFormatException e) {
20                  throw new ScriptParserException("Unable to convert " + val + "to a number.");
21              }
22              return retVal;
23          }
24      }
25
```

```

1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4
5      package jgsl.model;
6
7      import java.io.Serializable;
8
9      /**
10     * An Assignment is a statement in which the value of one attribute is assigned to another via the "="
11     operator.
12     *
13     * @author zenarchitect
14     * @version $Id: Assignment.java,v 1.2 2005/05/16 00:54:17 zenarchitect Exp $
15     */
16     public class Assignment implements Statement, Serializable {
17         /**
18         * Left-hand side of assignment
19         */
20         private String lhs;
21
22         /**
23         * Right-hand side of the assignment
24         */
25         private String rhs;
26
27         /**
28         * Constructs an instance with the left-hand side and right-hand side arguments of the assignment
29         statement
30         *
31         * @param lhs Left-hand side of the assignment
32         * @param rhs Right-hand side of the assignment
33         */
34         public Assignment(String lhs, String rhs) {
35             this.lhs = lhs;
36             this.rhs = rhs;
37         }
38
39         /**
40         * This method returns the Java language equivalent of the JGSL statement.
41         *
42         * @return Java language statement from the JGSL

```



```

42     */
43     public String getJava() {
44         return null; //To change body of implemented methods use File | Settings | File Templates.
45     }
46
47     /**
48     * Set the JGSL statement body
49     */
50     public void setJGSL(String jgsl) {
51         //To change body of implemented methods use File | Settings | File Templates.
52     }
53
54     /**
55     * Return the type of statement. The String form of the class name.
56     */
57     public String getType() {
58         return null; //To change body of implemented methods use File | Settings | File Templates.
59     }
60
61     public String getLhs() {
62         return lhs;
63     }
64
65     public void setLhs(String lhs) {
66         this.lhs = lhs;
67     }
68
69     public String getRhs() {
70         return rhs;
71     }
72
73     public void setRhs(String rhs) {
74         this.rhs = rhs;
75     }
76 }
77

```

/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/model/Command.java

```

1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4
5  package jgsl.model;
6
7  import java.io.Serializable;
8  import java.util.ArrayList;
9
10     /**
11     * A command statement is a JGSL command that performs a graphics operation.
12     *
13     * @author zenarchitect
14     * @version $Id: Command.java,v 1.3 2005/05/16 00:54:17 zenarchitect Exp $
15     */
16  public class Command implements Statement, Serializable {
17      private Commands name;
18      private ArrayList<Argument> attributes;
19      private ArrayList<Argument> parameters;
20
21      public Command(String name) {
22          setName(name);
23      }
24
25      public Command(String name, ArrayList<Argument> parameters) {
26          setName(name);
27          this.parameters = parameters;
28      }
29
30      public Command(String name, ArrayList<Argument> attributes, ArrayList<Argument> parameters)
31      {
32          setName(name);
33          this.attributes = attributes;
34          this.parameters = parameters;
35      }
36
37      public void setName(String name) {
38          if (name.equals(Commands.LINE.getName())) {
39              this.name = Commands.LINE;

```

```

40     if (name.equals(Commands.RECTANGLE.getName())) {
41         this.name = Commands.RECTANGLE;
42     }
43     if (name.equals(Commands.SQUARE.getName())) {
44         this.name = Commands.SQUARE;
45     }
46     if (name.equals(Commands.CIRCLE.getName())) {
47         this.name = Commands.CIRCLE;
48     }
49     if (name.equals(Commands.ELIPSE.getName())) {
50         this.name = Commands.ELIPSE;
51     }
52     if (name.equals(Commands.ARC.getName())) {
53         this.name = Commands.ARC;
54     }
55     if (name.equals(Commands.POLYGON.getName())) {
56         this.name = Commands.POLYGON;
57     }
58     if (name.equals(Commands.DRAW.getName())) {
59         this.name = Commands.DRAW;
60     } else if (name.equals(Commands.CLEAR.getName())) {
61         this.name = Commands.CLEAR;
62     } else if (name.equals(Commands.WAIT.getName())) {
63         this.name = Commands.WAIT;
64     } else if (name.equals(Commands.CANVAS.getName())) {
65         this.name = Commands.CANVAS;
66     } else if (name.equals(Commands.TEXT.getName())) {
67         this.name = Commands.TEXT;
68     } else if (name.equals(Commands.LOG.getName())) {
69         this.name = Commands.LOG;
70     } else if (name.equals(Commands.WARNING.getName())) {
71         this.name = Commands.WARNING;
72     } else if (name.equals(Commands.ERROR.getName())) {
73         this.name = Commands.ERROR;
74     } else if (name.equals(Commands.DEBUG.getName())) {
75         this.name = Commands.DEBUG;
76     }
77 }
78
79 /**
80  * This method returns the Java language equivalent of the JGSL statement.

```

```
81      *
82      * @return Java language statement from the JGSL
83      */
84  public String getJava() {
85      assert(name != null);
86      return name.getFormattedCommand(attributes, parameters);
87  }
88
89  /**
90   * Set the JGSL statement body
91   */
92  public void setJGSL(String jgsl) {
93      //TODO: To change body of implemented methods use File | Settings | File Templates.
94  }
95
96  /**
97   * Return the type of statement. The String form of the class name.
98   */
99  public String getType() {
100     assert(name != null);
101     return name.getName();
102 }
103 }
104
```

/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/model/Declaration.java

```

1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4
5  package jgsl.model;
6
7  import java.io.Serializable;
8
9
10     /**
11     * A Declaration statement is one that contains the declaration of a script variable.
12     *
13     * @author zenarchitect
14     * @version $Id: Declaration.java,v 1.2 2005/05/16 00:54:18 zenarchitect Exp $
15     */
16  public class Declaration implements Statement, Serializable {
17      private String type;
18      private String identifier;
19      private String value;
20
21      /**
22       * Create a declaration with a given type, identifier and initial value
23       *
24       * @param type      type of the declaration
25       * @param identifier script identifier
26       * @param value      initial value
27       */
28      public Declaration(String type, String identifier, String value) {
29          this.type = type;
30          this.identifier = identifier;
31          this.value = value;
32      }
33
34      /**
35       * This method returns the Java language equivalent of the JGSL statement.
36       *
37       * @return Java language statement from the JGSL
38       */
39      public String getJava() {

```

```
40     return null; //To change body of implemented methods use File | Settings | File Templates.
41 }
42
43 /**
44  * Set the JGSL statement body
45  */
46 public void setJGSL(String jgsl) {
47     //To change body of implemented methods use File | Settings | File Templates.
48 }
49
50 /**
51  * Return the type of statement. The String form of the class name.
52  */
53 public String getType() {
54     return null; //To change body of implemented methods use File | Settings | File Templates.
55 }
56 }
57
```

```
1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.model;
5
6  import java.awt.Color;
7
8  /**
9   * Declare an instance of a color type.
10  *
11  * @author zenarchitect
12  * @version $Id: JGSLColor.java,v 1.3 2005/05/21 01:42:07 zenarchitect Exp $
13  */
14  public class JGSLColor implements Type, Value, Argument {
15      private String name;
16      private Color color = Color.BLACK;
17
18      public JGSLColor(String name, Color color) {
19          this.name = name + System.currentTimeMillis();
20          this.color = color;
21      }
22
23      public Color getColor() {
24          return color;
25      }
26
27      /**
28       * Get the java Class meta-data for this type
29       *
30       * @return The Class mete-data for this type
31       */
32      public Class getJavaClass() {
33          return color.getClass();
34      }
35
36      /**
37       * Get the Java type as a String
38       *
39       * @return a String containing the type
40       */
41      public String getJavaType() {
42          return color.getClass().getName();
43      }
44
45      /**
46       * Get the Java representation of this value
47       *
48       * @return A String containing the Java representation of this value
```

```
49     */
50     public String getJavaValue() {
51         String javaValue = String.format("java.awt.Color %s = new java.awt.Color(%d, %d, %d, %d);\n", name, color.
getRed(), color.getGreen(), color.getBlue(), color.getAlpha());
52         return javaValue;
53     }
54
55     /**
56      * Get the name of the argument
57      *
58      * @return String containing the name
59      */
60     public String getName() {
61         return name;
62     }
63
64 }
65
```



```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/model/JGSLDouble.java
```

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.model;
5
6      // TODO - write javadocs
7      /**
8      * @author zenarchitect
9      * @version $Id: JGSLDouble.java,v 1.2 2005/05/16 00:54:18 zenarchitect Exp $
10     */
11     public class JGSLDouble implements Type, Value, Argument {
12         private String name;
13         private Double value;
14
15         public JGSLDouble(String name, Double value) {
16             this.name = name;
17             this.value = value;
18         }
19
20         public JGSLDouble(String name, String value) {
21             this.name = name;
22             this.value = Double.valueOf(value);
23         }
24
25         public Double getValue() {
26             return value;
27         }
28
29         /**
30          * Get the java Class meta-data for this type
31          *
32          * @return The Class mete-data for this type
33          */
34         public Class getJavaClass() {
35             return value.getClass();
36         }
37
38         /**
39          * Get the Java type as a String
```

```
40      *
41      * @return a String containing the type
42      */
43      public String getJavaType() {
44          return value.getClass().getName();
45      }
46
47      /**
48       * Get the Java representation of this value
49       *
50       * @return A String containing the Java representation of this value
51       */
52      public String getJavaValue() {
53          return value.toString();
54      }
55
56      /**
57       * Get the name of the argument
58       *
59       * @return String containing the name
60       */
61      public String getName() {
62          return name;
63      }
64  }
65
```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/model/JGSLInteger.java
```

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.model;
5
6      // TODO - write javadocs
7      /**
8      * @author zenarchitect
9      * @version $Id: JGSLInteger.java,v 1.2 2005/05/16 00:54:18 zenarchitect Exp $
10     */
11     public class JGSLInteger implements Type, Value, Argument {
12         private String name;
13         private Integer value;
14
15         public JGSLInteger(String name, Integer value) {
16             this.name = name;
17             this.value = value;
18         }
19
20         public JGSLInteger(String name, String value) {
21             this.name = name;
22             this.value = Integer.valueOf(value);
23         }
24
25         public Integer getValue() {
26             return value;
27         }
28
29         /**
30          * Get the java Class meta-data for this type
31          *
32          * @return The Class mete-data for this type
33          */
34         public Class getJavaClass() {
35             return value.getClass();
36         }
37
38         /**
39          * Get the Java type as a String
```

```
40      *
41      * @return a String containing the type
42      */
43      public String getJavaType() {
44          return value.getClass().getName();
45      }
46
47      /**
48       * Get the Java representation of this value
49       *
50       * @return A String containing the Java representation of this value
51       */
52      public String getJavaValue() {
53          return value.toString();
54      }
55
56      /**
57       * Get the name of the argument
58       *
59       * @return String containing the name
60       */
61      public String getName() {
62          return name;
63      }
64  }
65
```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/model/JGSLString.java
```

```
1  /*
2  * Copyright (c) 2005 Perception Software. All Rights Reserved.
3  */
4  package jgsl.model;
5
6  // TODO - write javadocs
7  /**
8   * @author zenarchitect
9   * @version $Id: JGSLString.java,v 1.2 2005/05/16 00:54:18 zenarchitect Exp $
10  */
11  public class JGSLString implements Type, Value, Argument {
12      private String name;
13      private String value;
14
15      public JGSLString(String name, String value) {
16          this.name = name;
17          this.value = value;
18      }
19
20      public String getValue() {
21          return value;
22      }
23
24      /**
25       * Get the java Class meta-data for this type
26       *
27       * @return The Class mete-data for this type
28       */
29      public Class getJavaClass() {
30          return value.getClass();
31      }
32
33      /**
34       * Get the Java type as a String
35       *
36       * @return a String containing the type
37       */
38      public String getJavaType() {
39          return value.getClass().getName();
40      }
41  }
```

```
40     }
41
42     /**
43      * Get the Java representation of this value
44      *
45      * @return A String containing the Java representation of this value
46      */
47     public String getJavaValue() {
48         return value;
49     }
50
51     /**
52      * Get the name of the argument
53      *
54      * @return String containing the name
55      */
56     public String getName() {
57         return name;
58     }
59 }
60
```

# /Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/parser/JGSL\_ParserConstants. java

```
1      /* Generated By:JavaCC: Do not edit this line. JGSL_ParserConstants.java */
2  package jgsl.parser;
3
4  public interface JGSL_ParserConstants {
5      int EOF = 0;
6      int SINGLE_LINE_COMMENT = 10;
7      int FORMAL_COMMENT = 11;
8      int MULTI_LINE_COMMENT = 12;
9      int AND = 14;
10     int OR = 15;
11     int NOT = 16;
12     int IF = 17;
13     int THEN = 18;
14     int ELSE = 19;
15     int ELSEIF = 20;
16     int TRUE = 21;
17     int FALSE = 22;
18     int CLEAR = 23;
19     int CANVAS = 24;
20     int DRAW = 25;
21     int TEXT = 26;
22     int RECTANGLE = 27;
23     int SQUARE = 28;
24     int CIRCLE = 29;
25     int ELIPSE = 30;
26     int ARC = 31;
27     int POLYGON = 32;
28     int LINE = 33;
29     int WAIT = 34;
30     int LOG = 35;
31     int DEBUG = 36;
32     int ERROR = 37;
33     int WARNING = 38;
34     int WRITE = 39;
35     int READ = 40;
36     int JGSL = 41;
37     int VERSION = 42;
38     int BEGIN = 43;
```

```
39  int END = 44;
40  int REPEAT = 45;
41  int LOOP = 46;
42  int BLACK = 47;
43  int BLUE = 48;
44  int DARK_GRAY = 49;
45  int GRAY = 50;
46  int GREEN = 51;
47  int LIGHT_GRAY = 52;
48  int MAGENTA = 53;
49  int ORANGE = 54;
50  int PINK = 55;
51  int RED = 56;
52  int WHITE = 57;
53  int YELLOW = 58;
54  int GRADIENT = 59;
55  int FILL = 60;
56  int BORDER = 61;
57  int FOREGROUND = 62;
58  int BACKGROUND = 63;
59  int COLOR = 64;
60  int DECLARE = 65;
61  int DOC = 66;
62  int INTEGER_LITERAL = 67;
63  int DECIMAL_LITERAL = 68;
64  int HEX_LITERAL = 69;
65  int OCTAL_LITERAL = 70;
66  int FLOATING_POINT_LITERAL = 71;
67  int STRING_LITERAL = 72;
68  int IDENTIFIER = 73;
69  int LETTER = 74;
70  int DIGIT = 75;
71  int LPAREN = 76;
72  int RPAREN = 77;
73  int SEMICOLON = 78;
74  int COLON = 79;
75  int COMMA = 80;
76  int ASSIGN = 81;
77  int GT = 82;
78  int LT = 83;
79  int BANG = 84;
80  int EQ = 85;
81  int LE = 86;
```



```

82  int GE = 87;
83  int NE = 88;
84  int SC_OR = 89;
85  int SC_AND = 90;
86  int PLUS = 91;
87  int MINUS = 92;
88  int STAR = 93;
89  int SLASH = 94;
90  int MOD = 95;
91
92  int DEFAULT = 0;
93  int IN_SINGLE_LINE_COMMENT = 1;
94  int IN_FORMAL_COMMENT = 2;
95  int IN_MULTI_LINE_COMMENT = 3;
96
97  String[] tokenImage = {
98      "<EOF>",
99      "\" \",",
100     "\"\\t\"",
101     "\"\\n\"",
102     "\"\\r\"",
103     "\"\\f\"",
104     "\"//\"",
105     "\"#\"",
106     "<token of kind 8>",
107     "\"/*\"",
108     "<SINGLE_LINE_COMMENT>",
109     "\"*/\"",
110     "\"*/\"",
111     "<token of kind 13>",
112     "\"and\"",
113     "\"or\"",
114     "\"not\"",
115     "\"if\"",
116     "\"then\"",
117     "\"else\"",
118     "\"elseif\"",
119     "\"true\"",
120     "\"false\"",
121     "\"clear\"",
122     "\"canvas\"",
123     "\"draw\"",
124     "\"text\"",

```

```
125     "\"rectangle\"",
126     "\"square\"",
127     "\"circle\"",
128     "\"ellipse\"",
129     "\"arc\"",
130     "\"polygon\"",
131     "\"line\"",
132     "\"wait\"",
133     "\"log\"",
134     "\"debug\"",
135     "\"error\"",
136     "\"warning\"",
137     "\"write\"",
138     "\"read\"",
139     "\"jgsl\"",
140     "\"version\"",
141     "\"begin\"",
142     "\"end\"",
143     "\"repeat\"",
144     "\"loop\"",
145     "\"black\"",
146     "\"blue\"",
147     "\"dark_gray\"",
148     "\"gray\"",
149     "\"green\"",
150     "\"light_gray\"",
151     "\"magenta\"",
152     "\"orange\"",
153     "\"pink\"",
154     "\"red\"",
155     "\"white\"",
156     "\"yellow\"",
157     "\"gradient\"",
158     "\"fill\"",
159     "\"border\"",
160     "\"foreground\"",
161     "\"background\"",
162     "\"color\"",
163     "\"declare\"",
164     "\"doc\"",
165     "<INTEGER_LITERAL>",
166     "<DECIMAL_LITERAL>",
167     "<HEX_LITERAL>",
```

```
168     "<OCTAL_LITERAL>",
169     "<FLOATING_POINT_LITERAL>",
170     "<STRING_LITERAL>",
171     "<IDENTIFIER>",
172     "<LETTER>",
173     "<DIGIT>",
174     "\"(\",
175     "\")\",
176     "\";\",
177     "\":\",
178     "\",",
179     "\"=\",
180     "\">",
181     "\"<",
182     "\"!\",
183     "\"==",
184     "\"<=",
185     "\">=",
186     "\"!=",
187     "\"||",
188     "\"&&",
189     "\"+",
190     "\"-",
191     "\"*",
192     "\"/",
193     "\"%";
194 };
195
196 }
197
```

```

1  /* Generated By:JavaCC: Do not edit this line. Token.java Version 3.0 */
2  package jgsl.parser;
3
4  /**
5   * Describes the input token stream.
6   */
7
8  public class Token {
9      /**
10       * An integer that describes the kind of this token. This numbering system is determined by JavaCCParser, and a
11       * table
12       * of these numbers is stored in the file ...Constants.java.
13       */
14
15     public int kind;
16
17     /**
18      * beginLine and beginColumn describe the position of the first character of this token; endLine and endColumn
19      * describe the position of the last character of this token.
20      */
21     public int beginLine
22     ,
23     beginColumn
24     ,
25     endLine
26     ,
27     endColumn;
28
29     /**
30      * The string image of the token.
31      */
32     public String image;
33
34     /**
35      * A reference to the next regular (non-special) token from the input stream. If this is the last token from the
36      * input stream, or if the token manager has not read tokens beyond this one, this field is set to null. This is true
37      * only if this token is also a regular token. Otherwise, see below for a description of the contents of this field.
38      */
39     public Token next;
40
41     /**
42      * This field is used to access special tokens that occur prior to this token, but after the immediately preceding
43      * regular (non-special) token. If there are no such special tokens, this field is set to null. When there are more
44      * than one such special token, this field refers to the last of these special tokens, which in turn refers to the
45      * next previous special token through its specialToken field, and so on until the first special token (whose
46      * specialToken field is null). The next fields of special tokens refer to other special tokens that immediately
47      * follow it (without an intervening regular token). If there is no such token, this field is null.
48      */
49     public Token specialToken;

```

```

48
49  /**
50   * Returns the image.
51   */
52  public String toString() {
53      return image;
54  }
55
56  /**
57   * Returns a new Token object, by default. However, if you want, you can create and return subclass objects based on
58   * the value of ofKind. Simply add the cases to the switch for all those special cases. For example, if you have a
59   * subclass of Token called IDToken that you want to create if ofKind is ID, simply add something like :
60   * <p/>
61   * case MyParserConstants.ID : return new IDToken();
62   * <p/>
63   * to the following switch statement. Then you can cast matchedToken variable to the appropriate type and use it in
64   * your lexical actions.
65   */
66  public static final Token newToken(int ofKind) {
67      switch (ofKind) {
68          default :
69              return new Token();
70      }
71  }
72
73 }
74

```

```

1  /* Generated By:JavaCC: Do not edit this line. JGSL_ParseTokenManager.java */
2  package jgsl.parser;
3  public class JGSL_ParseTokenManager implements JGSL_ParseConstants
4  {
5      public java.io.PrintStream debugStream = System.out;
6      public void setDebugStream(java.io.PrintStream ds) { debugStream = ds; }
7      private final int jjStopStringLiteralDfa_0(int pos, long active0, long active1)
8      {
9          switch (pos)
10         {
11             case 0:
12                 if ((active0 & 0x240L) != 0L || (active1 & 0x40000000L) != 0L)
13                     return 2;
14                 if ((active0 & 0xffffffffffc000L) != 0L || (active1 & 0x7L) != 0L)
15                 {
16                     jjmatchedKind = 73;
17                     return 19;
18                 }
19                 return -1;
20             case 1:
21                 if ((active0 & 0x200L) != 0L)
22                     return 0;
23                 if ((active0 & 0xffbfffffff4000L) != 0L || (active1 & 0x7L) != 0L)
24                 {
25                     if (jjmatchedPos != 1)
26                     {
27                         jjmatchedKind = 73;
28                         jjmatchedPos = 1;
29                     }
30                     return 19;
31                 }
32                 if ((active0 & 0x40000000028000L) != 0L)
33                     return 19;
34                 return -1;
35             case 2:
36                 if ((active0 & 0xfeffeff77ffc0000L) != 0L || (active1 & 0x3L) != 0L)
37                 {
38                     jjmatchedKind = 73;
39                     jjmatchedPos = 2;
40                     return 19;
41                 }
42                 if ((active0 & 0x100100880014000L) != 0L || (active1 & 0x4L) != 0L)
43                     return 19;
44                 return -1;
45             case 3:
46                 if ((active0 & 0xee7aacf179c00000L) != 0L || (active1 & 0x3L) != 0L)
47                 {

```

```

48     if (jjmatchedPos != 3)
49     {
50         jjmatchedKind = 73;
51         jjmatchedPos = 3;
52     }
53     return 19;
54 }
55 if ((active0 & 0x10854306063c0000L) != 0L)
56     return 19;
57 return -1;
58 case 4:
59     if ((active0 & 0xec72244179100000L) != 0L || (active1 & 0x2L) != 0L)
60     {
61         jjmatchedKind = 73;
62         jjmatchedPos = 4;
63         return 19;
64     }
65     if ((active0 & 0x20888b000c00000L) != 0L || (active1 & 0x1L) != 0L)
66         return 19;
67     return -1;
68 case 5:
69     if ((active0 & 0xc832044108000000L) != 0L || (active1 & 0x2L) != 0L)
70     {
71         jjmatchedKind = 73;
72         jjmatchedPos = 5;
73         return 19;
74     }
75     if ((active0 & 0x2440200071100000L) != 0L)
76         return 19;
77     return -1;
78 case 6:
79     if ((active0 & 0xc812000008000000L) != 0L)
80     {
81         jjmatchedKind = 73;
82         jjmatchedPos = 6;
83         return 19;
84     }
85     if ((active0 & 0x200441000000000L) != 0L || (active1 & 0x2L) != 0L)
86         return 19;
87     return -1;
88 case 7:
89     if ((active0 & 0xc012000008000000L) != 0L)
90     {
91         jjmatchedKind = 73;
92         jjmatchedPos = 7;
93         return 19;
94     }
95     if ((active0 & 0x8000000000000000L) != 0L)
96         return 19;
97     return -1;
98 case 8:

```

```

99     if ((active0 & 0xc010000000000000L) != 0L)
100     {
101         jjmatchedKind = 73;
102         jjmatchedPos = 8;
103         return 19;
104     }
105     if ((active0 & 0x20000080000000L) != 0L)
106         return 19;
107     return -1;
108 default :
109     return -1;
110 }
111 }
112 private final int jjStartNfa_0(int pos, long active0, long active1)
113 {
114     return jjMoveNfa_0(jjStopStringLiteralDfa_0(pos, active0, active1), pos + 1);
115 }
116 private final int jjStopAtPos(int pos, int kind)
117 {
118     jjmatchedKind = kind;
119     jjmatchedPos = pos;
120     return pos + 1;
121 }
122 private final int jjStartNfaWithStates_0(int pos, int kind, int state)
123 {
124     jjmatchedKind = kind;
125     jjmatchedPos = pos;
126     try { curChar = input_stream.readChar(); }
127     catch(java.io.IOException e) { return pos + 1; }
128     return jjMoveNfa_0(state, pos + 1);
129 }
130 private final int jjMoveStringLiteralDfa0_0()
131 {
132     switch(curChar)
133     {
134         case 9:
135             return jjStopAtPos(0, 2);
136         case 10:
137             return jjStopAtPos(0, 3);
138         case 12:
139             return jjStopAtPos(0, 5);
140         case 13:
141             return jjStopAtPos(0, 4);
142         case 32:
143             return jjStopAtPos(0, 1);
144         case 33:
145             jjmatchedKind = 84;
146             return jjMoveStringLiteralDfa1_0(0x0L, 0x1000000L);
147         case 35:
148             return jjStopAtPos(0, 7);
149         case 37:

```



```

150     return jjStopAtPos(0, 95);
151 case 38:
152     return jjMoveStringLiteralDfa1_0(0x0L, 0x4000000L);
153 case 40:
154     return jjStopAtPos(0, 76);
155 case 41:
156     return jjStopAtPos(0, 77);
157 case 42:
158     return jjStopAtPos(0, 93);
159 case 43:
160     return jjStopAtPos(0, 91);
161 case 44:
162     return jjStopAtPos(0, 80);
163 case 45:
164     return jjStopAtPos(0, 92);
165 case 47:
166     jjmatchedKind = 94;
167     return jjMoveStringLiteralDfa1_0(0x240L, 0x0L);
168 case 58:
169     return jjStopAtPos(0, 79);
170 case 59:
171     return jjStopAtPos(0, 78);
172 case 60:
173     jjmatchedKind = 83;
174     return jjMoveStringLiteralDfa1_0(0x0L, 0x400000L);
175 case 61:
176     jjmatchedKind = 81;
177     return jjMoveStringLiteralDfa1_0(0x0L, 0x200000L);
178 case 62:
179     jjmatchedKind = 82;
180     return jjMoveStringLiteralDfa1_0(0x0L, 0x800000L);
181 case 65:
182 case 97:
183     return jjMoveStringLiteralDfa1_0(0x80004000L, 0x0L);
184 case 66:
185 case 98:
186     return jjMoveStringLiteralDfa1_0(0xa001880000000000L, 0x0L);
187 case 67:
188 case 99:
189     return jjMoveStringLiteralDfa1_0(0x21800000L, 0x1L);
190 case 68:
191 case 100:
192     return jjMoveStringLiteralDfa1_0(0x20010020000000L, 0x6L);
193 case 69:
194 case 101:
195     return jjMoveStringLiteralDfa1_0(0x102040180000L, 0x0L);
196 case 70:
197 case 102:
198     return jjMoveStringLiteralDfa1_0(0x5000000000400000L, 0x0L);
199 case 71:
200 case 103:

```

```

201     return jjMoveStringLiteralDfa1_0(0x80c000000000000L, 0x0L);
202 case 73:
203 case 105:
204     return jjMoveStringLiteralDfa1_0(0x20000L, 0x0L);
205 case 74:
206 case 106:
207     return jjMoveStringLiteralDfa1_0(0x200000000000L, 0x0L);
208 case 76:
209 case 108:
210     return jjMoveStringLiteralDfa1_0(0x10400a00000000L, 0x0L);
211 case 77:
212 case 109:
213     return jjMoveStringLiteralDfa1_0(0x200000000000000L, 0x0L);
214 case 78:
215 case 110:
216     return jjMoveStringLiteralDfa1_0(0x10000L, 0x0L);
217 case 79:
218 case 111:
219     return jjMoveStringLiteralDfa1_0(0x40000000008000L, 0x0L);
220 case 80:
221 case 112:
222     return jjMoveStringLiteralDfa1_0(0x80000100000000L, 0x0L);
223 case 82:
224 case 114:
225     return jjMoveStringLiteralDfa1_0(0x100210008000000L, 0x0L);
226 case 83:
227 case 115:
228     return jjMoveStringLiteralDfa1_0(0x10000000L, 0x0L);
229 case 84:
230 case 116:
231     return jjMoveStringLiteralDfa1_0(0x4240000L, 0x0L);
232 case 86:
233 case 118:
234     return jjMoveStringLiteralDfa1_0(0x400000000000L, 0x0L);
235 case 87:
236 case 119:
237     return jjMoveStringLiteralDfa1_0(0x20000c400000000L, 0x0L);
238 case 89:
239 case 121:
240     return jjMoveStringLiteralDfa1_0(0x400000000000000L, 0x0L);
241 case 124:
242     return jjMoveStringLiteralDfa1_0(0x0L, 0x2000000L);
243 default :
244     return jjMoveNfa_0(3, 0);
245 }
246 }
247 private final int jjMoveStringLiteralDfa1_0(long active0, long active1)
248 {
249     try { curChar = input_stream.readChar(); }
250     catch(java.io.IOException e) {
251         jjStopStringLiteralDfa_0(0, active0, active1);

```

```

252     return 1;
253 }
254 switch(curChar)
255 {
256     case 38:
257         if ((active1 & 0x4000000L) != 0L)
258             return jjStopAtPos(1, 90);
259         break;
260     case 42:
261         if ((active0 & 0x200L) != 0L)
262             return jjStartNfaWithStates_0(1, 9, 0);
263         break;
264     case 47:
265         if ((active0 & 0x40L) != 0L)
266             return jjStopAtPos(1, 6);
267         break;
268     case 61:
269         if ((active1 & 0x200000L) != 0L)
270             return jjStopAtPos(1, 85);
271         else if ((active1 & 0x400000L) != 0L)
272             return jjStopAtPos(1, 86);
273         else if ((active1 & 0x800000L) != 0L)
274             return jjStopAtPos(1, 87);
275         else if ((active1 & 0x1000000L) != 0L)
276             return jjStopAtPos(1, 88);
277         break;
278     case 65:
279     case 97:
280         return jjMoveStringLiteralDfa2_0(active0, 0x8022004401400000L, active1, 0L);
281     case 69:
282     case 101:
283         return jjMoveStringLiteralDfa2_0(active0, 0x5002d100c000000L, active1, 0x2L);
284     case 70:
285     case 102:
286         if ((active0 & 0x20000L) != 0L)
287             return jjStartNfaWithStates_0(1, 17, 19);
288         break;
289     case 71:
290     case 103:
291         return jjMoveStringLiteralDfa2_0(active0, 0x200000000000L, active1, 0L);
292     case 72:
293     case 104:
294         return jjMoveStringLiteralDfa2_0(active0, 0x200000000040000L, active1, 0L);
295     case 73:
296     case 105:
297         return jjMoveStringLiteralDfa2_0(active0, 0x1090000220000000L, active1, 0L);
298     case 76:
299     case 108:
300         return jjMoveStringLiteralDfa2_0(active0, 0x1800040980000L, active1, 0L);
301     case 78:
302     case 110:

```

```

303     return jjMoveStringLiteralDfa2_0(active0, 0x1000000004000L, active1, 0L);
304 case 79:
305 case 111:
306     return jjMoveStringLiteralDfa2_0(active0, 0x6000400900010000L, active1, 0x5L);
307 case 81:
308 case 113:
309     return jjMoveStringLiteralDfa2_0(active0, 0x100000000L, active1, 0L);
310 case 82:
311 case 114:
312     if ((active0 & 0x8000L) != 0L)
313     {
314         jjmatchedKind = 15;
315         jjmatchedPos = 1;
316     }
317     return jjMoveStringLiteralDfa2_0(active0, 0x84c00a082200000L, active1, 0L);
318 case 124:
319     if ((active1 & 0x2000000L) != 0L)
320         return jjStopAtPos(1, 89);
321     break;
322 default :
323     break;
324 }
325 return jjStartNfa_0(0, active0, active1);
326 }
327 private final int jjMoveStringLiteralDfa2_0(long old0, long active0, long old1, long active1)
328 {
329     if (((active0 &= old0) | (active1 &= old1)) == 0L)
330         return jjStartNfa_0(0, old0, old1);
331     try { curChar = input_stream.readChar(); }
332     catch(java.io.IOException e) {
333         jjStopStringLiteralDfa_0(1, active0, active1);
334         return 2;
335     }
336     switch(curChar)
337     {
338     case 65:
339     case 97:
340         return jjMoveStringLiteralDfa3_0(active0, 0x844810002000000L, active1, 0L);
341     case 66:
342     case 98:
343         return jjMoveStringLiteralDfa3_0(active0, 0x10000000000L, active1, 0L);
344     case 67:
345     case 99:
346         if ((active0 & 0x80000000L) != 0L)
347             return jjStartNfaWithStates_0(2, 31, 19);
348         else if ((active1 & 0x4L) != 0L)
349             return jjStartNfaWithStates_0(2, 66, 19);
350         return jjMoveStringLiteralDfa3_0(active0, 0x8000000008000000L, active1, 0x2L);
351     case 68:
352     case 100:
353         if ((active0 & 0x4000L) != 0L)

```

```

354     return jjStartNfaWithStates_0(2, 14, 19);
355     else if ((active0 & 0x1000000000000L) != 0L)
356         return jjStartNfaWithStates_0(2, 44, 19);
357     else if ((active0 & 0x100000000000000L) != 0L)
358         return jjStartNfaWithStates_0(2, 56, 19);
359     break;
360 case 69:
361 case 101:
362     return jjMoveStringLiteralDfa3_0(active0, 0x8000000840000L, active1, 0L);
363 case 71:
364 case 103:
365     if ((active0 & 0x800000000L) != 0L)
366         return jjStartNfaWithStates_0(2, 35, 19);
367     return jjMoveStringLiteralDfa3_0(active0, 0x30080000000000L, active1, 0L);
368 case 73:
369 case 105:
370     return jjMoveStringLiteralDfa3_0(active0, 0x200008440000000L, active1, 0L);
371 case 76:
372 case 108:
373     return jjMoveStringLiteralDfa3_0(active0, 0x1400000100400000L, active1, 0x1L);
374 case 78:
375 case 110:
376     return jjMoveStringLiteralDfa3_0(active0, 0x80000201000000L, active1, 0L);
377 case 79:
378 case 111:
379     return jjMoveStringLiteralDfa3_0(active0, 0x4000000000000L, active1, 0L);
380 case 80:
381 case 112:
382     return jjMoveStringLiteralDfa3_0(active0, 0x2000000000000L, active1, 0L);
383 case 82:
384 case 114:
385     return jjMoveStringLiteralDfa3_0(active0, 0x6002046020000000L, active1, 0L);
386 case 83:
387 case 115:
388     return jjMoveStringLiteralDfa3_0(active0, 0x20000180000L, active1, 0L);
389 case 84:
390 case 116:
391     if ((active0 & 0x10000L) != 0L)
392         return jjStartNfaWithStates_0(2, 16, 19);
393     break;
394 case 85:
395 case 117:
396     return jjMoveStringLiteralDfa3_0(active0, 0x1000010200000L, active1, 0L);
397 case 88:
398 case 120:
399     return jjMoveStringLiteralDfa3_0(active0, 0x4000000L, active1, 0L);
400 default :
401     break;
402 }
403 return jjStartNfa_0(1, active0, active1);
404 }

```

```

405 private final int jjMoveStringLiteralDfa3_0(long old0, long active0, long old1, long active1)
406 {
407     if (((active0 &= old0) | (active1 &= old1)) == 0L)
408         return jjStartNfa_0(1, old0, old1);
409     try { curChar = input_stream.readChar(); }
410     catch(java.io.IOException e) {
411         jjStopStringLiteralDfa_0(2, active0, active1);
412         return 3;
413     }
414     switch(curChar)
415     {
416         case 65:
417         case 97:
418             return jjMoveStringLiteralDfa4_0(active0, 0x10800000L, active1, 0L);
419         case 67:
420         case 99:
421             return jjMoveStringLiteralDfa4_0(active0, 0x800020000000L, active1, 0L);
422         case 68:
423         case 100:
424             if ((active0 & 0x10000000000L) != 0L)
425                 return jjStartNfaWithStates_0(3, 40, 19);
426             return jjMoveStringLiteralDfa4_0(active0, 0x2800000000000000L, active1, 0L);
427         case 69:
428         case 101:
429             if ((active0 & 0x80000L) != 0L)
430             {
431                 jjmatchedKind = 19;
432                 jjmatchedPos = 3;
433             }
434             else if ((active0 & 0x200000L) != 0L)
435                 return jjStartNfaWithStates_0(3, 21, 19);
436             else if ((active0 & 0x2000000000L) != 0L)
437                 return jjStartNfaWithStates_0(3, 33, 19);
438             else if ((active0 & 0x10000000000000L) != 0L)
439                 return jjStartNfaWithStates_0(3, 48, 19);
440             return jjMoveStringLiteralDfa4_0(active0, 0x4028200000100000L, active1, 0L);
441         case 72:
442         case 104:
443             return jjMoveStringLiteralDfa4_0(active0, 0x100000000000000L, active1, 0L);
444         case 73:
445         case 105:
446             return jjMoveStringLiteralDfa4_0(active0, 0x800000000000L, active1, 0L);
447         case 75:
448         case 107:
449             if ((active0 & 0x800000000000000L) != 0L)
450                 return jjStartNfaWithStates_0(3, 55, 19);
451             return jjMoveStringLiteralDfa4_0(active0, 0x8002000000000000L, active1, 0L);
452         case 76:
453         case 108:
454             if ((active0 & 0x200000000000L) != 0L)
455                 return jjStartNfaWithStates_0(3, 41, 19);

```

```

456     else if ((active0 & 0x1000000000000000L) != 0L)
457         return jjStartNfaWithStates_0(3, 60, 19);
458     return jjMoveStringLiteralDfa4_0(active0, 0x400000000000000L, active1, 0x2L);
459 case 78:
460 case 110:
461     if ((active0 & 0x40000L) != 0L)
462         return jjStartNfaWithStates_0(3, 18, 19);
463     return jjMoveStringLiteralDfa4_0(active0, 0x400040000000000L, active1, 0L);
464 case 79:
465 case 111:
466     return jjMoveStringLiteralDfa4_0(active0, 0x20000000000L, active1, 0x1L);
467 case 80:
468 case 112:
469     if ((active0 & 0x400000000000000L) != 0L)
470         return jjStartNfaWithStates_0(3, 46, 19);
471     return jjMoveStringLiteralDfa4_0(active0, 0x400000000L, active1, 0L);
472 case 83:
473 case 115:
474     return jjMoveStringLiteralDfa4_0(active0, 0x40000400000L, active1, 0L);
475 case 84:
476 case 116:
477     if ((active0 & 0x40000000L) != 0L)
478         return jjStartNfaWithStates_0(3, 26, 19);
479     else if ((active0 & 0x4000000000L) != 0L)
480         return jjStartNfaWithStates_0(3, 34, 19);
481     return jjMoveStringLiteralDfa4_0(active0, 0x200008008000000L, active1, 0L);
482 case 85:
483 case 117:
484     return jjMoveStringLiteralDfa4_0(active0, 0x1000000000L, active1, 0L);
485 case 86:
486 case 118:
487     return jjMoveStringLiteralDfa4_0(active0, 0x1000000L, active1, 0L);
488 case 87:
489 case 119:
490     if ((active0 & 0x2000000L) != 0L)
491         return jjStartNfaWithStates_0(3, 25, 19);
492     break;
493 case 89:
494 case 121:
495     if ((active0 & 0x400000000000000L) != 0L)
496         return jjStartNfaWithStates_0(3, 50, 19);
497     return jjMoveStringLiteralDfa4_0(active0, 0x100000000L, active1, 0L);
498 default :
499     break;
500 }
501 return jjStartNfa_0(2, active0, active1);
502 }
503 private final int jjMoveStringLiteralDfa4_0(long old0, long active0, long old1, long active1)
504 {
505     if (((active0 &= old0) | (active1 &= old1)) == 0L)
506         return jjStartNfa_0(2, old0, old1);

```



```

507 try { curChar = input_stream.readChar(); }
508 catch(java.io.IOException e) {
509     jjStopStringLiteralDfa_0(3, active0, active1);
510     return 4;
511 }
512 switch(curChar)
513 {
514     case 95:
515         return jjMoveStringLiteralDfa5_0(active0, 0x20000000000000L, active1, 0L);
516     case 65:
517     case 97:
518         return jjMoveStringLiteralDfa5_0(active0, 0x2000090000000L, active1, 0x2L);
519     case 69:
520     case 101:
521         if ((active0 & 0x400000L) != 0L)
522             return jjStartNfaWithStates_0(4, 22, 19);
523         else if ((active0 & 0x80000000000L) != 0L)
524             return jjStartNfaWithStates_0(4, 39, 19);
525         else if ((active0 & 0x200000000000000L) != 0L)
526             return jjStartNfaWithStates_0(4, 57, 19);
527         return jjMoveStringLiteralDfa5_0(active0, 0x200000000000000L, active1, 0L);
528     case 71:
529     case 103:
530         if ((active0 & 0x1000000000L) != 0L)
531             return jjStartNfaWithStates_0(4, 36, 19);
532         return jjMoveStringLiteralDfa5_0(active0, 0xc040000100000000L, active1, 0L);
533     case 73:
534     case 105:
535         return jjMoveStringLiteralDfa5_0(active0, 0x800044000100000L, active1, 0L);
536     case 75:
537     case 107:
538         if ((active0 & 0x800000000000L) != 0L)
539             return jjStartNfaWithStates_0(4, 47, 19);
540         break;
541     case 76:
542     case 108:
543         return jjMoveStringLiteralDfa5_0(active0, 0x20000000L, active1, 0L);
544     case 78:
545     case 110:
546         if ((active0 & 0x800000000000L) != 0L)
547             return jjStartNfaWithStates_0(4, 43, 19);
548         else if ((active0 & 0x80000000000000L) != 0L)
549             return jjStartNfaWithStates_0(4, 51, 19);
550         return jjMoveStringLiteralDfa5_0(active0, 0x200000000000000L, active1, 0L);
551     case 79:
552     case 111:
553         return jjMoveStringLiteralDfa5_0(active0, 0x400000000000000L, active1, 0L);
554     case 82:
555     case 114:
556         if ((active0 & 0x800000L) != 0L)
557             return jjStartNfaWithStates_0(4, 23, 19);

```



```

558     else if ((active0 & 0x20000000000L) != 0L)
559         return jjStartNfaWithStates_0(4, 37, 19);
560     else if ((active1 & 0x1L) != 0L)
561         return jjStartNfaWithStates_0(4, 64, 19);
562     return jjMoveStringLiteralDfa5_0(active0, 0x100000000L, active1, 0L);
563 case 83:
564 case 115:
565     return jjMoveStringLiteralDfa5_0(active0, 0x400000000L, active1, 0L);
566 case 84:
567 case 116:
568     return jjMoveStringLiteralDfa5_0(active0, 0x1000000000000000L, active1, 0L);
569 default :
570     break;
571 }
572 return jjStartNfa_0(3, active0, active1);
573 }
574 private final int jjMoveStringLiteralDfa5_0(long old0, long active0, long old1, long active1)
575 {
576     if (((active0 &= old0) | (active1 &= old1)) == 0L)
577         return jjStartNfa_0(3, old0, old1);
578     try { curChar = input_stream.readChar(); }
579     catch (java.io.IOException e) {
580         jjStopStringLiteralDfa_0(4, active0, active1);
581         return 5;
582     }
583     switch (curChar)
584     {
585     case 95:
586         return jjMoveStringLiteralDfa6_0(active0, 0x1000000000000000L, active1, 0L);
587     case 69:
588     case 101:
589         if ((active0 & 0x100000000L) != 0L)
590             return jjStartNfaWithStates_0(5, 28, 19);
591         else if ((active0 & 0x200000000L) != 0L)
592             return jjStartNfaWithStates_0(5, 29, 19);
593         else if ((active0 & 0x400000000L) != 0L)
594             return jjStartNfaWithStates_0(5, 30, 19);
595         else if ((active0 & 0x4000000000000000L) != 0L)
596             return jjStartNfaWithStates_0(5, 54, 19);
597         return jjMoveStringLiteralDfa6_0(active0, 0x8000000000000000L, active1, 0L);
598     case 70:
599     case 102:
600         if ((active0 & 0x100000L) != 0L)
601             return jjStartNfaWithStates_0(5, 20, 19);
602         break;
603     case 71:
604     case 103:
605         return jjMoveStringLiteralDfa6_0(active0, 0x200000000000000L, active1, 0L);
606     case 78:
607     case 110:
608         return jjMoveStringLiteralDfa6_0(active0, 0x40080000000L, active1, 0L);

```

```

609     case 79:
610     case 111:
611         return jjMoveStringLiteralDfa6_0(active0, 0x401000000000L, active1, 0L);
612     case 82:
613     case 114:
614         if ((active0 & 0x2000000000000000L) != 0L)
615             return jjStartNfaWithStates_0(5, 61, 19);
616         return jjMoveStringLiteralDfa6_0(active0, 0xc000000000000000L, active1, 0x2L);
617     case 83:
618     case 115:
619         if ((active0 & 0x1000000L) != 0L)
620             return jjStartNfaWithStates_0(5, 24, 19);
621         break;
622     case 84:
623     case 116:
624         if ((active0 & 0x2000000000000000L) != 0L)
625             return jjStartNfaWithStates_0(5, 45, 19);
626         return jjMoveStringLiteralDfa6_0(active0, 0x2000000000000000L, active1, 0L);
627     case 87:
628     case 119:
629         if ((active0 & 0x4000000000000000L) != 0L)
630             return jjStartNfaWithStates_0(5, 58, 19);
631         break;
632     default :
633         break;
634 }
635 return jjStartNfa_0(4, active0, active1);
636 }
637 private final int jjMoveStringLiteralDfa6_0(long old0, long active0, long old1, long active1)
638 {
639     if (((active0 &= old0) | (active1 &= old1)) == 0L)
640         return jjStartNfa_0(4, old0, old1);
641     try { curChar = input_stream.readChar(); }
642     catch(java.io.IOException e) {
643         jjStopStringLiteralDfa_0(5, active0, active1);
644         return 6;
645     }
646     switch(curChar)
647     {
648     case 65:
649     case 97:
650         if ((active0 & 0x2000000000000000L) != 0L)
651             return jjStartNfaWithStates_0(6, 53, 19);
652         break;
653     case 69:
654     case 101:
655         if ((active1 & 0x2L) != 0L)
656             return jjStartNfaWithStates_0(6, 65, 19);
657         break;
658     case 71:
659     case 103:

```

```

660     if ((active0 & 0x4000000000L) != 0L)
661         return jjStartNfaWithStates_0(6, 38, 19);
662     return jjMoveStringLiteralDfa7_0(active0, 0x10000008000000L, active1, 0L);
663 case 78:
664 case 110:
665     if ((active0 & 0x100000000L) != 0L)
666         return jjStartNfaWithStates_0(6, 32, 19);
667     else if ((active0 & 0x400000000000L) != 0L)
668         return jjStartNfaWithStates_0(6, 42, 19);
669     return jjMoveStringLiteralDfa7_0(active0, 0x800000000000000L, active1, 0L);
670 case 79:
671 case 111:
672     return jjMoveStringLiteralDfa7_0(active0, 0xc00000000000000L, active1, 0L);
673 case 82:
674 case 114:
675     return jjMoveStringLiteralDfa7_0(active0, 0x20000000000000L, active1, 0L);
676 default :
677     break;
678 }
679 return jjStartNfa_0(5, active0, active1);
680 }
681 private final int jjMoveStringLiteralDfa7_0(long old0, long active0, long old1, long active1)
682 {
683     if (((active0 &= old0) | (active1 &= old1)) == 0L)
684         return jjStartNfa_0(5, old0, old1);
685     try { curChar = input_stream.readChar(); }
686     catch(java.io.IOException e) {
687         jjStopStringLiteralDfa_0(6, active0, 0L);
688         return 7;
689     }
690     switch(curChar)
691     {
692     case 65:
693     case 97:
694         return jjMoveStringLiteralDfa8_0(active0, 0x20000000000000L);
695     case 76:
696     case 108:
697         return jjMoveStringLiteralDfa8_0(active0, 0x8000000L);
698     case 82:
699     case 114:
700         return jjMoveStringLiteralDfa8_0(active0, 0x10000000000000L);
701     case 84:
702     case 116:
703         if ((active0 & 0x800000000000000L) != 0L)
704             return jjStartNfaWithStates_0(7, 59, 19);
705         break;
706     case 85:
707     case 117:
708         return jjMoveStringLiteralDfa8_0(active0, 0xc00000000000000L);
709     default :
710         break;

```

```

711     }
712     return jjStartNfa_0(6, active0, 0L);
713 }
714 private final int jjMoveStringLiteralDfa8_0(long old0, long active0)
715 {
716     if (((active0 &= old0)) == 0L)
717         return jjStartNfa_0(6, old0, 0L);
718     try { curChar = input_stream.readChar(); }
719     catch(java.io.IOException e) {
720         jjStopStringLiteralDfa_0(7, active0, 0L);
721         return 8;
722     }
723     switch(curChar)
724     {
725         case 65:
726         case 97:
727             return jjMoveStringLiteralDfa9_0(active0, 0x100000000000000L);
728         case 69:
729         case 101:
730             if ((active0 & 0x8000000L) != 0L)
731                 return jjStartNfaWithStates_0(8, 27, 19);
732             break;
733         case 78:
734         case 110:
735             return jjMoveStringLiteralDfa9_0(active0, 0xc000000000000000L);
736         case 89:
737         case 121:
738             if ((active0 & 0x200000000000000L) != 0L)
739                 return jjStartNfaWithStates_0(8, 49, 19);
740             break;
741         default :
742             break;
743     }
744     return jjStartNfa_0(7, active0, 0L);
745 }
746 private final int jjMoveStringLiteralDfa9_0(long old0, long active0)
747 {
748     if (((active0 &= old0)) == 0L)
749         return jjStartNfa_0(7, old0, 0L);
750     try { curChar = input_stream.readChar(); }
751     catch(java.io.IOException e) {
752         jjStopStringLiteralDfa_0(8, active0, 0L);
753         return 9;
754     }
755     switch(curChar)
756     {
757         case 68:
758         case 100:
759             if ((active0 & 0x4000000000000000L) != 0L)
760                 return jjStartNfaWithStates_0(9, 62, 19);
761             else if ((active0 & 0x8000000000000000L) != 0L)

```

```

762     return jjStartNfaWithStates_0(9, 63, 19);
763     break;
764     case 89:
765     case 121:
766         if ((active0 & 0x100000000000000L) != 0L)
767             return jjStartNfaWithStates_0(9, 52, 19);
768         break;
769     default :
770         break;
771 }
772 return jjStartNfa_0(8, active0, 0L);
773 }
774 private final void jjCheckNAdd(int state)
775 {
776     if (jjrounds[state] != jjround)
777     {
778         jjstateSet[jjnewStateCnt++] = state;
779         jjrounds[state] = jjround;
780     }
781 }
782 private final void jjAddStates(int start, int end)
783 {
784     do {
785         jjstateSet[jjnewStateCnt++] = jjnextStates[start];
786     } while (start++ != end);
787 }
788 private final void jjCheckNAddTwoStates(int state1, int state2)
789 {
790     jjCheckNAdd(state1);
791     jjCheckNAdd(state2);
792 }
793 private final void jjCheckNAddStates(int start, int end)
794 {
795     do {
796         jjCheckNAdd(jjnextStates[start]);
797     } while (start++ != end);
798 }
799 private final void jjCheckNAddStates(int start)
800 {
801     jjCheckNAdd(jjnextStates[start]);
802     jjCheckNAdd(jjnextStates[start + 1]);
803 }
804 static final long[] jjbitVec0 = {
805     0xfffffffffffffeL, 0xfffffffffffffL, 0xfffffffffffffL, 0xfffffffffffffL
806 };
807 static final long[] jjbitVec2 = {
808     0x0L, 0x0L, 0xfffffffffffffL, 0xfffffffffffffL
809 };
810 static final long[] jjbitVec3 = {
811     0x1ff00000fffffeL, 0xffffffffffc000L, 0xffffffffL, 0x600000000000000L
812 };

```

```

813 static final long[] jjbitVec4 = {
814     0x0L, 0x0L, 0x0L, 0xff7fffffff7ffffL
815 };
816 static final long[] jjbitVec5 = {
817     0x0L, 0xfffffffffffffL, 0xfffffffffffffL, 0xfffffffffffffL
818 };
819 static final long[] jjbitVec6 = {
820     0xfffffffffffffL, 0xfffffffffffffL, 0xffffL, 0x0L
821 };
822 static final long[] jjbitVec7 = {
823     0xfffffffffffffL, 0xfffffffffffffL, 0x0L, 0x0L
824 };
825 static final long[] jjbitVec8 = {
826     0x3fffffffffL, 0x0L, 0x0L, 0x0L
827 };
828 private final int jjMoveNfa_0(int startState, int curPos)
829 {
830     int[] nextStates;
831     int startsAt = 0;
832     jjnewStateCnt = 24;
833     int i = 1;
834     jjstateSet[0] = startState;
835     int j, kind = 0x7ffffff;
836     for (;;)
837     {
838         if (++jjround == 0x7ffffff)
839             ReInitRounds();
840         if (curChar < 64)
841         {
842             long l = 1L << curChar;
843             MatchLoop: do
844             {
845                 switch(jjstateSet[--i])
846                 {
847                     case 3:
848                         if ((0x3ff000000000000L & l) != 0L)
849                             jjCheckNAddTwoStates(6, 7);
850                         else if (curChar == 36)
851                         {
852                             if (kind > 73)
853                                 kind = 73;
854                             jjCheckNAdd(19);
855                         }
856                         else if (curChar == 34)
857                             jjCheckNAddStates(0, 2);
858                         else if (curChar == 47)
859                             jjstateSet[jjnewStateCnt++] = 2;
860                         if ((0x3fe000000000000L & l) != 0L)
861                         {
862                             if (kind > 67)
863                                 kind = 67;

```

```

864         jjCheckNAdd(5);
865     }
866     else if (curChar == 48)
867     {
868         if (kind > 67)
869             kind = 67;
870         jjCheckNAddTwoStates(21, 23);
871     }
872     break;
873 case 0:
874     if (curChar == 42)
875         jjstateSet[jjnewStateCnt++] = 1;
876     break;
877 case 1:
878     if ((0xffff7fffffffffffL & 1) != 0L && kind > 8)
879         kind = 8;
880     break;
881 case 2:
882     if (curChar == 42)
883         jjstateSet[jjnewStateCnt++] = 0;
884     break;
885 case 4:
886     if ((0x3fe0000000000000L & 1) == 0L)
887         break;
888     if (kind > 67)
889         kind = 67;
890     jjCheckNAdd(5);
891     break;
892 case 5:
893     if ((0x3ff0000000000000L & 1) == 0L)
894         break;
895     if (kind > 67)
896         kind = 67;
897     jjCheckNAdd(5);
898     break;
899 case 6:
900     if ((0x3ff0000000000000L & 1) != 0L)
901         jjCheckNAddTwoStates(6, 7);
902     break;
903 case 7:
904     if (curChar != 46)
905         break;
906     if (kind > 71)
907         kind = 71;
908     jjCheckNAdd(8);
909     break;
910 case 8:
911     if ((0x3ff0000000000000L & 1) == 0L)
912         break;
913     if (kind > 71)
914         kind = 71;

```

```
915         jjCheckNAdd(8);
916         break;
917     case 9:
918         if (curChar == 34)
919             jjCheckNAddStates(0, 2);
920         break;
921     case 10:
922         if ((0xffffffffbfffdbffL & 1) != 0L)
923             jjCheckNAddStates(0, 2);
924         break;
925     case 12:
926         if ((0x8400000000L & 1) != 0L)
927             jjCheckNAddStates(0, 2);
928         break;
929     case 13:
930         if (curChar == 34 && kind > 72)
931             kind = 72;
932         break;
933     case 14:
934         if ((0xff0000000000000L & 1) != 0L)
935             jjCheckNAddStates(3, 6);
936         break;
937     case 15:
938         if ((0xff0000000000000L & 1) != 0L)
939             jjCheckNAddStates(0, 2);
940         break;
941     case 16:
942         if ((0xf0000000000000L & 1) != 0L)
943             jjstateSet[jjnewStateCnt++] = 17;
944         break;
945     case 17:
946         if ((0xff0000000000000L & 1) != 0L)
947             jjCheckNAdd(15);
948         break;
949     case 18:
950         if (curChar != 36)
951             break;
952         if (kind > 73)
953             kind = 73;
954         jjCheckNAdd(19);
955         break;
956     case 19:
957         if ((0x3ff001000000000L & 1) == 0L)
958             break;
959         if (kind > 73)
960             kind = 73;
961         jjCheckNAdd(19);
962         break;
963     case 20:
964         if (curChar != 48)
965             break;
```



```

966         if (kind > 67)
967             kind = 67;
968         jjCheckNAddTwoStates(21, 23);
969         break;
970     case 22:
971         if ((0x3ff0000000000000L & l) == 0L)
972             break;
973         if (kind > 67)
974             kind = 67;
975         jjstateSet[jjnewStateCnt++] = 22;
976         break;
977     case 23:
978         if ((0xff00000000000000L & l) == 0L)
979             break;
980         if (kind > 67)
981             kind = 67;
982         jjCheckNAdd(23);
983         break;
984     default : break;
985 }
986 } while(i != startsAt);
987 }
988 else if (curChar < 128)
989 {
990     long l = 1L << (curChar & 077);
991     MatchLoop: do
992     {
993         switch(jjstateSet[--i])
994         {
995             case 3:
996             case 19:
997                 if ((0x7fffffe87fffffeL & l) == 0L)
998                     break;
999                 if (kind > 73)
1000                     kind = 73;
1001                 jjCheckNAdd(19);
1002                 break;
1003             case 1:
1004                 if (kind > 8)
1005                     kind = 8;
1006                 break;
1007             case 10:
1008                 if ((0xffffffffffffffL & l) != 0L)
1009                     jjCheckNAddStates(0, 2);
1010                 break;
1011             case 11:
1012                 if (curChar == 92)
1013                     jjAddStates(7, 9);
1014                 break;
1015             case 12:
1016                 if ((0x14404410144044L & l) != 0L)

```

```

1017         jjCheckNAddStates(0, 2);
1018         break;
1019     case 21:
1020         if ((0x1000000001000000L & 1) != 0L)
1021             jjCheckNAdd(22);
1022         break;
1023     case 22:
1024         if ((0x7e0000007eL & 1) == 0L)
1025             break;
1026         if (kind > 67)
1027             kind = 67;
1028         jjCheckNAdd(22);
1029         break;
1030     default : break;
1031 }
1032 } while(i != startsAt);
1033 }
1034 else
1035 {
1036     int hiByte = (int)(curChar >> 8);
1037     int i1 = hiByte >> 6;
1038     long l1 = 1L << (hiByte & 077);
1039     int i2 = (curChar & 0xff) >> 6;
1040     long l2 = 1L << (curChar & 077);
1041     MatchLoop: do
1042     {
1043         switch(jjstateSet[--i])
1044         {
1045             case 3:
1046             case 19:
1047                 if (!jjCanMove_1(hiByte, i1, i2, l1, l2))
1048                     break;
1049                 if (kind > 73)
1050                     kind = 73;
1051                 jjCheckNAdd(19);
1052                 break;
1053             case 1:
1054                 if (jjCanMove_0(hiByte, i1, i2, l1, l2) && kind > 8)
1055                     kind = 8;
1056                 break;
1057             case 10:
1058                 if (jjCanMove_0(hiByte, i1, i2, l1, l2))
1059                     jjAddStates(0, 2);
1060                 break;
1061             default : break;
1062         }
1063     } while(i != startsAt);
1064 }
1065 if (kind != 0x7fffffff)
1066 {
1067     jjmatchedKind = kind;

```

```

1068     jjmatchedPos = curPos;
1069     kind = 0x7fffffff;
1070 }
1071 ++curPos;
1072 if ((i = jjnewStateCnt) == (startsAt = 24 - (jjnewStateCnt = startsAt)))
1073     return curPos;
1074 try { curChar = input_stream.readChar(); }
1075 catch(java.io.IOException e) { return curPos; }
1076 }
1077 }
1078 private final int jjMoveStringLiteralDfa0_3()
1079 {
1080     switch(curChar)
1081     {
1082     case 42:
1083         return jjMoveStringLiteralDfa1_3(0x1000L);
1084     default :
1085         return 1;
1086     }
1087 }
1088 private final int jjMoveStringLiteralDfa1_3(long active0)
1089 {
1090     try { curChar = input_stream.readChar(); }
1091     catch(java.io.IOException e) {
1092         return 1;
1093     }
1094     switch(curChar)
1095     {
1096     case 47:
1097         if ((active0 & 0x1000L) != 0L)
1098             return jjStopAtPos(1, 12);
1099         break;
1100     default :
1101         return 2;
1102     }
1103     return 2;
1104 }
1105 private final int jjMoveStringLiteralDfa0_1()
1106 {
1107     return jjMoveNfa_1(0, 0);
1108 }
1109 private final int jjMoveNfa_1(int startState, int curPos)
1110 {
1111     int[] nextStates;
1112     int startsAt = 0;
1113     jjnewStateCnt = 3;
1114     int i = 1;
1115     jjstateSet[0] = startState;
1116     int j, kind = 0x7fffffff;
1117     for (;;)
1118     {

```

```

1119     if (++jjround == 0x7fffffff)
1120         ReInitRounds();
1121     if (curChar < 64)
1122     {
1123         long l = 1L << curChar;
1124         MatchLoop: do
1125         {
1126             switch(jjstateSet[--i])
1127             {
1128                 case 0:
1129                     if ((0x2400L & l) != 0L)
1130                     {
1131                         if (kind > 10)
1132                             kind = 10;
1133                     }
1134                     if (curChar == 13)
1135                         jjstateSet[jjnewStateCnt++] = 1;
1136                     break;
1137                 case 1:
1138                     if (curChar == 10 && kind > 10)
1139                         kind = 10;
1140                     break;
1141                 case 2:
1142                     if (curChar == 13)
1143                         jjstateSet[jjnewStateCnt++] = 1;
1144                     break;
1145                 default : break;
1146             }
1147         } while(i != startsAt);
1148     }
1149     else if (curChar < 128)
1150     {
1151         long l = 1L << (curChar & 077);
1152         MatchLoop: do
1153         {
1154             switch(jjstateSet[--i])
1155             {
1156                 default : break;
1157             }
1158         } while(i != startsAt);
1159     }
1160     else
1161     {
1162         int hiByte = (int)(curChar >> 8);
1163         int i1 = hiByte >> 6;
1164         long l1 = 1L << (hiByte & 077);
1165         int i2 = (curChar & 0xff) >> 6;
1166         long l2 = 1L << (curChar & 077);
1167         MatchLoop: do
1168         {
1169             switch(jjstateSet[--i])

```

```

1170     {
1171         default : break;
1172     }
1173 } while(i != startsAt);
1174 }
1175 if (kind != 0x7ffffff)
1176 {
1177     jjmatchedKind = kind;
1178     jjmatchedPos = curPos;
1179     kind = 0x7ffffff;
1180 }
1181 ++curPos;
1182 if ((i = jjnewStateCnt) == (startsAt = 3 - (jjnewStateCnt = startsAt)))
1183     return curPos;
1184 try { curChar = input_stream.readChar(); }
1185 catch(java.io.IOException e) { return curPos; }
1186 }
1187 }
1188 private final int jjMoveStringLiteralDfa0_2()
1189 {
1190     switch(curChar)
1191     {
1192         case 42:
1193             return jjMoveStringLiteralDfa1_2(0x800L);
1194         default :
1195             return 1;
1196     }
1197 }
1198 private final int jjMoveStringLiteralDfa1_2(long active0)
1199 {
1200     try { curChar = input_stream.readChar(); }
1201     catch(java.io.IOException e) {
1202         return 1;
1203     }
1204     switch(curChar)
1205     {
1206         case 47:
1207             if ((active0 & 0x800L) != 0L)
1208                 return jjStopAtPos(1, 11);
1209             break;
1210         default :
1211             return 2;
1212     }
1213     return 2;
1214 }
1215 static final int[] jjnextStates = {
1216     10, 11, 13, 10, 11, 15, 13, 12, 14, 16,
1217 };
1218 private static final boolean jjCanMove_0(int hiByte, int i1, int i2, long l1, long l2)
1219 {
1220     switch(hiByte)

```

```

1221 {
1222     case 0:
1223         return ((jjbitVec2[i2] & l2) != 0L);
1224     default :
1225         if ((jjbitVec0[i1] & l1) != 0L)
1226             return true;
1227         return false;
1228 }
1229 }
1230 private static final boolean jjCanMove_1(int hiByte, int i1, int i2, long l1, long l2)
1231 {
1232     switch(hiByte)
1233     {
1234         case 0:
1235             return ((jjbitVec4[i2] & l2) != 0L);
1236         case 48:
1237             return ((jjbitVec5[i2] & l2) != 0L);
1238         case 49:
1239             return ((jjbitVec6[i2] & l2) != 0L);
1240         case 51:
1241             return ((jjbitVec7[i2] & l2) != 0L);
1242         case 61:
1243             return ((jjbitVec8[i2] & l2) != 0L);
1244         default :
1245             if ((jjbitVec3[i1] & l1) != 0L)
1246                 return true;
1247             return false;
1248     }
1249 }
1250 public static final String[] jjstrLiteralImages = {
1251     "", null, null, null, null, null, null, null, null, null, null, null, null, null,
1252     null, null, null, null, null, null, null, null, null, null, null, null, null, null,
1253     null, null, null, null, null, null, null, null, null, null, null, null, null, null,
1254     null, null, null, null, null, null, null, null, null, null, null, null, null, null,
1255     null, null, null, null, null, null, null, null, null, null, null, null, null, null,
1256     null, null, null, null, null, null, null, "\50", "\51", "\73", "\72", "\54", "\75",
1257     "\76", "\74", "\41", "\75\75", "\74\75", "\76\75", "\41\75", "\174\174", "\46\46",
1258     "\53", "\55", "\52", "\57", "\45", };
1259 public static final String[] lexStateNames = {
1260     "DEFAULT",
1261     "IN_SINGLE_LINE_COMMENT",
1262     "IN_FORMAL_COMMENT",
1263     "IN_MULTI_LINE_COMMENT",
1264 };
1265 public static final int[] jjnewLexState = {
1266     -1, -1, -1, -1, -1, -1, 1, 1, 2, 3, 0, 0, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
1267     -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
1268     -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
1269     -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
1270 };
1271 static final long[] jjtoToken = {

```

```

1272 0xffffffffffc001L, 0xffff38fL,
1273 };
1274 static final long[] jjtoSkip = {
1275 0x1c3eL, 0x0L,
1276 };
1277 static final long[] jjtoSpecial = {
1278 0x1c3eL, 0x0L,
1279 };
1280 static final long[] jjtoMore = {
1281 0x23c0L, 0x0L,
1282 };
1283 protected JavaCharStream input_stream;
1284 private final int[] jjrounds = new int[24];
1285 private final int[] jjstateSet = new int[48];
1286 StringBuffer image;
1287 int jjimageLen;
1288 int lengthOfMatch;
1289 protected char curChar;
1290 public JGSL_ParserTokenManager(JavaCharStream stream)
1291 {
1292  if (JavaCharStream.staticFlag)
1293     throw new Error("ERROR: Cannot use a static CharStream class with a non-static lexical analyzer.");
1294  input_stream = stream;
1295 }
1296 public JGSL_ParserTokenManager(JavaCharStream stream, int lexState)
1297 {
1298  this(stream);
1299  SwitchTo(lexState);
1300 }
1301 public void ReInit(JavaCharStream stream)
1302 {
1303  jjmatchedPos = jjnewStateCnt = 0;
1304  curLexState = defaultLexState;
1305  input_stream = stream;
1306  ReInitRounds();
1307 }
1308 private final void ReInitRounds()
1309 {
1310  int i;
1311  jjround = 0x80000001;
1312  for (i = 24; i-- > 0;)
1313     jjrounds[i] = 0x80000000;
1314 }
1315 public void ReInit(JavaCharStream stream, int lexState)
1316 {
1317  ReInit(stream);
1318  SwitchTo(lexState);
1319 }
1320 public void SwitchTo(int lexState)
1321 {
1322  if (lexState >= 4 || lexState < 0)

```

```

1323     throw new TokenMgrError("Error: Ignoring invalid lexical state : " + lexState + ". State unchanged.",
TokenMgrError.INVALID_LEXICAL_STATE);
1324 else
1325     curLexState = lexState;
1326 }
1327
1328 protected Token jjFillToken()
1329 {
1330     Token t = Token.newToken(jjmatchedKind);
1331     t.kind = jjmatchedKind;
1332     String im = jjstrLiteralImages[jjmatchedKind];
1333     t.image = (im == null) ? input_stream.GetImage() : im;
1334     t.beginLine = input_stream.getBeginLine();
1335     t.beginColumn = input_stream.getBeginColumn();
1336     t.endLine = input_stream.getEndLine();
1337     t.endColumn = input_stream.getEndColumn();
1338     return t;
1339 }
1340
1341 int curLexState = 0;
1342 int defaultLexState = 0;
1343 int jjnewStateCnt;
1344 int jjround;
1345 int jjmatchedPos;
1346 int jjmatchedKind;
1347
1348 public Token getNextToken()
1349 {
1350     int kind;
1351     Token specialToken = null;
1352     Token matchedToken;
1353     int curPos = 0;
1354
1355     EOFLoop :
1356     for (;;)
1357     {
1358         try
1359         {
1360             curChar = input_stream.BeginToken();
1361         }
1362         catch(java.io.IOException e)
1363         {
1364             jjmatchedKind = 0;
1365             matchedToken = jjFillToken();
1366             matchedToken.specialToken = specialToken;
1367             return matchedToken;
1368         }
1369         image = null;
1370         jjimageLen = 0;
1371

```



```

1372 for (;;)
1373 {
1374     switch(curLexState)
1375     {
1376         case 0:
1377             jjmatchedKind = 0x7fffffff;
1378             jjmatchedPos = 0;
1379             curPos = jjMoveStringLiteralDfa0_0();
1380             break;
1381         case 1:
1382             jjmatchedKind = 0x7fffffff;
1383             jjmatchedPos = 0;
1384             curPos = jjMoveStringLiteralDfa0_1();
1385             if (jjmatchedPos == 0 && jjmatchedKind > 13)
1386             {
1387                 jjmatchedKind = 13;
1388             }
1389             break;
1390         case 2:
1391             jjmatchedKind = 0x7fffffff;
1392             jjmatchedPos = 0;
1393             curPos = jjMoveStringLiteralDfa0_2();
1394             if (jjmatchedPos == 0 && jjmatchedKind > 13)
1395             {
1396                 jjmatchedKind = 13;
1397             }
1398             break;
1399         case 3:
1400             jjmatchedKind = 0x7fffffff;
1401             jjmatchedPos = 0;
1402             curPos = jjMoveStringLiteralDfa0_3();
1403             if (jjmatchedPos == 0 && jjmatchedKind > 13)
1404             {
1405                 jjmatchedKind = 13;
1406             }
1407             break;
1408     }
1409     if (jjmatchedKind != 0x7fffffff)
1410     {
1411         if (jjmatchedPos + 1 < curPos)
1412             input_stream.backup(curPos - jjmatchedPos - 1);
1413         if ((jjtoToken[jjmatchedKind >> 6] & (1L << (jjmatchedKind & 077))) != 0L)
1414         {
1415             matchedToken = jjFillToken();
1416             matchedToken.specialToken = specialToken;
1417             if (jjnewLexState[jjmatchedKind] != -1)
1418                 curLexState = jjnewLexState[jjmatchedKind];
1419             return matchedToken;
1420         }
1421         else if ((jjtoSkip[jjmatchedKind >> 6] & (1L << (jjmatchedKind & 077))) != 0L)
1422         {

```

```

1423     if ((jjtoSpecial[jjmatchedKind >> 6] & (1L << (jjmatchedKind & 077))) != 0L)
1424     {
1425         matchedToken = jjFillToken();
1426         if (specialToken == null)
1427             specialToken = matchedToken;
1428         else
1429         {
1430             matchedToken.specialToken = specialToken;
1431             specialToken = (specialToken.next = matchedToken);
1432         }
1433         SkipLexicalActions(matchedToken);
1434     }
1435     else
1436         SkipLexicalActions(null);
1437     if (jjnewLexState[jjmatchedKind] != -1)
1438         curLexState = jjnewLexState[jjmatchedKind];
1439     continue EOFLoop;
1440 }
1441 MoreLexicalActions();
1442 if (jjnewLexState[jjmatchedKind] != -1)
1443     curLexState = jjnewLexState[jjmatchedKind];
1444     curPos = 0;
1445     jjmatchedKind = 0x7fffffff;
1446     try {
1447         curChar = input_stream.readChar();
1448         continue;
1449     }
1450     catch (java.io.IOException e1) { }
1451 }
1452 int error_line = input_stream.getEndLine();
1453 int error_column = input_stream.getEndColumn();
1454 String error_after = null;
1455 boolean EOFSeen = false;
1456 try { input_stream.readChar(); input_stream.backup(1); }
1457 catch (java.io.IOException e1) {
1458     EOFSeen = true;
1459     error_after = curPos <= 1 ? "" : input_stream.GetImage();
1460     if (curChar == '\n' || curChar == '\r') {
1461         error_line++;
1462         error_column = 0;
1463     }
1464     else
1465         error_column++;
1466 }
1467 if (!EOFSeen) {
1468     input_stream.backup(1);
1469     error_after = curPos <= 1 ? "" : input_stream.GetImage();
1470 }
1471 throw new TokenMgrError(EOFSeen, curLexState, error_line, error_column, error_after, curChar, TokenMgrError.
LEXICAL_ERROR);
1472 }

```

```
1473 }
1474 }
1475
1476 void SkipLexicalActions(Token matchedToken)
1477 {
1478     switch(jjmatchedKind)
1479     {
1480         default :
1481             break;
1482     }
1483 }
1484 void MoreLexicalActions()
1485 {
1486     jjimageLen += (lengthOfMatch = jjmatchedPos + 1);
1487     switch(jjmatchedKind)
1488     {
1489         case 8 :
1490             if (image == null)
1491                 image = new StringBuffer(new String(input_stream.GetSuffix(jjimageLen)));
1492             else
1493                 image.append(input_stream.GetSuffix(jjimageLen));
1494             jjimageLen = 0;
1495             input_stream.backup(1);
1496             break;
1497         default :
1498             break;
1499     }
1500 }
1501 }
1502
```

```
/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/parser/JavaCharStream.java
```

```
1      /* Generated By:JavaCC: Do not edit this line. JavaCharStream.java Version 3.0 */
2  package jgsl.parser;
3
4  /**
5   * An implementation of interface CharStream, where the stream is assumed to contain only ASCII characters
6   (with
7   * java-like unicode escape processing).
8   */
9  public class JavaCharStream {
10     public static final boolean staticFlag = false;
11
12     static final int hexval(char c) throws java.io.IOException {
13         switch (c) {
14             case '0' :
15                 return 0;
16             case '1' :
17                 return 1;
18             case '2' :
19                 return 2;
20             case '3' :
21                 return 3;
22             case '4' :
23                 return 4;
24             case '5' :
25                 return 5;
26             case '6' :
27                 return 6;
28             case '7' :
29                 return 7;
30             case '8' :
31                 return 8;
32             case '9' :
33                 return 9;
34
35             case 'a' :
36             case 'A' :
37                 return 10;
38             case 'b' :
39             case 'B' :
40                 return 11;
41             case 'c' :
42             case 'C' :
43                 return 12;
44             case 'd' :
```

```

45     case 'D' :
46         return 13;
47     case 'e' :
48     case 'E' :
49         return 14;
50     case 'f' :
51     case 'F' :
52         return 15;
53     }
54
55     throw new java.io.IOException(); // Should never come here
56 }
57
58 public int bufpos = -1;
59 int bufsize;
60 int available;
61 int tokenBegin;
62 protected int bufline[];
63 protected int bufcolumn[];
64
65 protected int column = 0;
66 protected int line = 1;
67
68 protected boolean prevCharIsCR = false;
69 protected boolean prevCharIsLF = false;
70
71 protected java.io.Reader inputStream;
72
73 protected char[] nextCharBuf;
74 protected char[] buffer;
75 protected int maxNextCharInd = 0;
76 protected int nextCharInd = -1;
77 protected int inBuf = 0;
78
79 protected void ExpandBuff(boolean wrapAround) {
80     char[] newbuffer = new char[bufsize + 2048];
81     int newbufline[] = new int[bufsize + 2048];
82     int newbufcolumn[] = new int[bufsize + 2048];
83
84     try {
85         if (wrapAround) {
86             System.arraycopy(buffer, tokenBegin, newbuffer, 0, bufsize - tokenBegin);
87             System.arraycopy(buffer, 0, newbuffer,
88                 bufsize - tokenBegin, bufpos);
89             buffer = newbuffer;
90
91             System.arraycopy(bufline, tokenBegin, newbufline, 0, bufsize - tokenBegin);
92             System.arraycopy(bufline, 0, newbufline, bufsize - tokenBegin, bufpos);
93             bufline = newbufline;

```

```

94
95     System.arraycopy(bufcolumn, tokenBegin, newbufcolumn, 0, bufsize - tokenBegin);
96     System.arraycopy(bufcolumn, 0, newbufcolumn, bufsize - tokenBegin, bufpos);
97     bufcolumn = newbufcolumn;
98
99     bufpos += (bufsize - tokenBegin);
100 } else {
101     System.arraycopy(buffer, tokenBegin, newbuffer, 0, bufsize - tokenBegin);
102     buffer = newbuffer;
103
104     System.arraycopy(bufline, tokenBegin, newbufline, 0, bufsize - tokenBegin);
105     bufline = newbufline;
106
107     System.arraycopy(bufcolumn, tokenBegin, newbufcolumn, 0, bufsize - tokenBegin);
108     bufcolumn = newbufcolumn;
109
110     bufpos -= tokenBegin;
111 }
112 }
113 catch (Throwable t) {
114     throw new Error(t.getMessage());
115 }
116
117 available = (bufsize += 2048);
118 tokenBegin = 0;
119 }
120
121 protected void FillBuff() throws java.io.IOException {
122     int i;
123     if (maxNextCharInd == 4096)
124         maxNextCharInd = nextCharInd = 0;
125
126     try {
127         if ((i = inputStream.read(nextCharBuf, maxNextCharInd,
128             4096 - maxNextCharInd)) == -1) {
129             inputStream.close();
130             throw new java.io.IOException();
131         } else
132             maxNextCharInd += i;
133         return;
134     }
135     catch (java.io.IOException e) {
136         if (bufpos != 0) {
137             --bufpos;
138             backup(0);
139         } else {
140             bufline[bufpos] = line;
141             bufcolumn[bufpos] = column;
142         }

```

```
143         throw e;
144     }
145 }
146
147 protected char ReadByte() throws java.io.IOException {
148     if (++nextCharInd >= maxNextCharInd)
149         FillBuff();
150
151     return nextCharBuf[nextCharInd];
152 }
153
154 public char BeginToken() throws java.io.IOException {
155     if (inBuf > 0) {
156         --inBuf;
157
158         if (++bufpos == bufsize)
159             bufpos = 0;
160
161         tokenBegin = bufpos;
162         return buffer[bufpos];
163     }
164
165     tokenBegin = 0;
166     bufpos = -1;
167
168     return readChar();
169 }
170
171 protected void AdjustBuffSize() {
172     if (available == bufsize) {
173         if (tokenBegin > 2048) {
174             bufpos = 0;
175             available = tokenBegin;
176         } else
177             ExpandBuff(false);
178     } else if (available > tokenBegin)
179         available = bufsize;
180     else if ((tokenBegin - available) < 2048)
181         ExpandBuff(true);
182     else
183         available = tokenBegin;
184 }
185
186 protected void UpdateLineColumn(char c) {
187     column++;
188
189     if (prevCharIsLF) {
190         prevCharIsLF = false;
191         line += (column = 1);
```

```

192     } else if (prevCharIsCR) {
193         prevCharIsCR = false;
194         if (c == '\n') {
195             prevCharIsLF = true;
196         } else
197             line += (column = 1);
198     }
199
200     switch (c) {
201         case '\r' :
202             prevCharIsCR = true;
203             break;
204         case '\n' :
205             prevCharIsLF = true;
206             break;
207         case '\t' :
208             column--;
209             column += (8 - (column & 07));
210             break;
211         default :
212             break;
213     }
214
215     bufline[bufpos] = line;
216     bufcolumn[bufpos] = column;
217 }
218
219 public char readChar() throws java.io.IOException {
220     if (inBuf > 0) {
221         --inBuf;
222
223         if (++bufpos == bufsize)
224             bufpos = 0;
225
226         return buffer[bufpos];
227     }
228
229     char c;
230
231     if (++bufpos == available)
232         AdjustBuffSize();
233
234     if ((buffer[bufpos] = c = ReadByte()) == '\\') {
235         UpdateLineColumn(c);
236
237         int backSlashCnt = 1;
238
239         for (; ;) // Read all the backslashes
240     {

```



```

241         if (++bufpos == available)
242             AdjustBuffSize();
243
244         try {
245             if ((buffer[bufpos] = c = ReadByte()) != '\\') {
246                 UpdateLineColumn(c);
247                 // found a non-backslash char.
248                 if ((c == 'u') && ((backSlashCnt & 1) == 1)) {
249                     if (--bufpos < 0)
250                         bufpos = bufsize - 1;
251
252                     break;
253                 }
254
255                 backup(backSlashCnt);
256                 return "\\";
257             }
258         }
259         catch (java.io.IOException e) {
260             if (backSlashCnt > 1)
261                 backup(backSlashCnt);
262
263             return "\\";
264         }
265
266         UpdateLineColumn(c);
267         backSlashCnt++;
268     }
269
270     // Here, we have seen an odd number of backslash's followed by a 'u'
271     try {
272         while ((c = ReadByte()) == 'u') ++column;
273
274         buffer[bufpos] = c = (char) (hexval(c) << 12 |
275             hexval(ReadByte()) << 8 |
276             hexval(ReadByte()) << 4 |
277             hexval(ReadByte()));
278
279         column += 4;
280     }
281     catch (java.io.IOException e) {
282         throw new Error("Invalid escape character at line " + line +
283             " column " + column + ".");
284     }
285
286     if (backSlashCnt == 1)
287         return c;
288     else {
289         backup(backSlashCnt - 1);

```

```
290         return '\\';
291     }
292     } else {
293         UpdateLineColumn(c);
294         return (c);
295     }
296 }
297
298 /**
299  * @see #getEndColumn
300  * @deprecated
301  */
302
303 public int getColumn() {
304     return bufcolumn[bufpos];
305 }
306
307 /**
308  * @see #getEndLine
309  * @deprecated
310  */
311
312 public int getLine() {
313     return bufline[bufpos];
314 }
315
316 public int getEndColumn() {
317     return bufcolumn[bufpos];
318 }
319
320 public int getEndLine() {
321     return bufline[bufpos];
322 }
323
324 public int getBeginColumn() {
325     return bufcolumn[tokenBegin];
326 }
327
328 public int getBeginLine() {
329     return bufline[tokenBegin];
330 }
331
332 public void backup(int amount) {
333
334     inBuf += amount;
335     if ((bufpos -= amount) < 0)
336         bufpos += bufsize;
337 }
338
```

```

339 public JavaCharStream(java.io.Reader dstream,
340                        int startline, int startcolumn, int buffersize) {
341     inputStream = dstream;
342     line = startline;
343     column = startcolumn - 1;
344
345     available = bufsize = buffersize;
346     buffer = new char[buffersize];
347     bufline = new int[buffersize];
348     bufcolumn = new int[buffersize];
349     nextCharBuf = new char[4096];
350 }
351
352 public JavaCharStream(java.io.Reader dstream,
353                        int startline, int startcolumn) {
354     this(dstream, startline, startcolumn, 4096);
355 }
356
357 public JavaCharStream(java.io.Reader dstream) {
358     this(dstream, 1, 1, 4096);
359 }
360
361 public void ReInit(java.io.Reader dstream,
362                   int startline, int startcolumn, int buffersize) {
363     inputStream = dstream;
364     line = startline;
365     column = startcolumn - 1;
366
367     if (buffer == null || buffersize != buffer.length) {
368         available = bufsize = buffersize;
369         buffer = new char[buffersize];
370         bufline = new int[buffersize];
371         bufcolumn = new int[buffersize];
372         nextCharBuf = new char[4096];
373     }
374     prevCharIsLF = prevCharIsCR = false;
375     tokenBegin = inBuf = maxNextCharInd = 0;
376     nextCharInd = bufpos = -1;
377 }
378
379 public void ReInit(java.io.Reader dstream,
380                   int startline, int startcolumn) {
381     ReInit(dstream, startline, startcolumn, 4096);
382 }
383
384 public void ReInit(java.io.Reader dstream) {
385     ReInit(dstream, 1, 1, 4096);
386 }
387

```

```

388 public JavaCharStream(java.io.InputStream dstream, int startline,
389                        int startcolumn, int buffersize) {
390     this(new java.io.InputStreamReader(dstream), startline, startcolumn, 4096);
391 }
392
393 public JavaCharStream(java.io.InputStream dstream, int startline,
394                        int startcolumn) {
395     this(dstream, startline, startcolumn, 4096);
396 }
397
398 public JavaCharStream(java.io.InputStream dstream) {
399     this(dstream, 1, 1, 4096);
400 }
401
402 public void ReInit(java.io.InputStream dstream, int startline,
403                    int startcolumn, int buffersize) {
404     ReInit(new java.io.InputStreamReader(dstream), startline, startcolumn, 4096);
405 }
406
407 public void ReInit(java.io.InputStream dstream, int startline,
408                    int startcolumn) {
409     ReInit(dstream, startline, startcolumn, 4096);
410 }
411
412 public void ReInit(java.io.InputStream dstream) {
413     ReInit(dstream, 1, 1, 4096);
414 }
415
416 public String GetImage() {
417     if (bufpos >= tokenBegin)
418         return new String(buffer, tokenBegin, bufpos - tokenBegin + 1);
419     else
420         return new String(buffer, tokenBegin, bufsize - tokenBegin) +
421             new String(buffer, 0, bufpos + 1);
422 }
423
424 public char[] GetSuffix(int len) {
425     char[] ret = new char[len];
426
427     if ((bufpos + 1) >= len)
428         System.arraycopy(buffer, bufpos - len + 1, ret, 0, len);
429     else {
430         System.arraycopy(buffer, bufsize - (len - bufpos - 1), ret, 0,
431             len - bufpos - 1);
432         System.arraycopy(buffer, 0, ret, len - bufpos - 1, bufpos + 1);
433     }
434
435     return ret;
436 }

```

```

437
438 public void Done() {
439     nextCharBuf = null;
440     buffer = null;
441     bufline = null;
442     bufcolumn = null;
443 }
444
445 /**
446  * Method to adjust line and column numbers for the start of a token.
447  */
448 public void adjustBeginLineColumn(int newLine, int newCol) {
449     int start = tokenBegin;
450     int len;
451
452     if (bufpos >= tokenBegin) {
453         len = bufpos - tokenBegin + inBuf + 1;
454     } else {
455         len = bufsize - tokenBegin + bufpos + 1 + inBuf;
456     }
457
458     int i = 0, j = 0, k = 0;
459     int nextColDiff = 0, columnDiff = 0;
460
461     while (i < len &&
462         bufline[j = start % bufsize] == bufline[k = ++start % bufsize]) {
463         bufline[j] = newLine;
464         nextColDiff = columnDiff + bufcolumn[k] - bufcolumn[j];
465         bufcolumn[j] = newCol + columnDiff;
466         columnDiff = nextColDiff;
467         i++;
468     }
469
470     if (i < len) {
471         bufline[j] = newLine++;
472         bufcolumn[j] = newCol + columnDiff;
473
474         while (i++ < len) {
475             if (bufline[j = start % bufsize] != bufline[++start % bufsize])
476                 bufline[j] = newLine++;
477             else
478                 bufline[j] = newLine;
479         }
480     }
481
482     line = bufline[j];
483     column = bufcolumn[j];
484 }
485

```

```
486 }  
487
```

/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/model/Type.java

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.model;
5
6      // TODO - write javadocs
7      /**
8      * @author zenarchitect
9      * @version $Id: Type.java,v 1.2 2005/05/16 00:54:19 zenarchitect Exp $
10     */
11     public interface Type {
12         /**
13          * Get the java Class meta-data for this type
14          *
15          * @return The Class mete-data for this type
16          */
17         public Class getJavaClass();
18
19         /**
20          * Get the Java type as a String
21          *
22          * @return a String containing the type
23          */
24         public String getJavaType();
25     }
26 }
27
```

/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/model/Value.java

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.model;
5
6      // TODO - write javadocs
7      /**
8      * @author zenarchitect
9      * @version $Id: Value.java,v 1.2 2005/05/16 00:54:19 zenarchitect Exp $
10     */
11     public interface Value {
12         /**
13          * Get the Java representation of this value
14          *
15          * @return A String containing the Java representation of this value
16          */
17         public String getJavaValue();
18     }
19
```



/Users/jchavez/dev/java.net/jgsl.dev.java.net/jgsl/dev/src/jgsl/model/Argument.java

```
1      /*
2      * Copyright (c) 2005 Perception Software. All Rights Reserved.
3      */
4      package jgsl.model;
5
6
7      /**
8       * The name of a script argument.
9       *
10      * @author zenarchitect
11      * @version $Id: Argument.java,v 1.2 2005/05/16 00:54:17 zenarchitect Exp $
12      */
13      public interface Argument {
14          /**
15           * Get the name of the argument
16           *
17           * @return String containing the name
18           */
19          public String getName();
20
21      }
22
```

```

1      /* Generated By:JavaCC: Do not edit this line. TokenMgrError.java Version 3.0 */
2  package jgsl.parser;
3
4  public class TokenMgrError extends Error {
5      /*
6       * Ordinals for various reasons why an Error of this type can be thrown.
7       */
8
9      /**
10      * Lexical error occurred.
11      */
12     static final int LEXICAL_ERROR = 0;
13
14     /**
15      * An attempt was made to create a second instance of a static token manager.
16      */
17     static final int STATIC_LEXER_ERROR = 1;
18
19     /**
20      * Tried to change to an invalid lexical state.
21      */
22     static final int INVALID_LEXICAL_STATE = 2;
23
24     /**
25      * Detected (and bailed out of) an infinite loop in the token manager.
26      */
27     static final int LOOP_DETECTED = 3;
28
29     /**
30      * Indicates the reason why the exception is thrown. It will have one of the above 4 values.
31      */
32     int errorCode;
33
34     /**
35      * Replaces unprintable characters by their escaped (or unicode escaped) equivalents in the given string
36      */
37     protected static final String addEscapes(String str) {
38         StringBuffer retval = new StringBuffer();
39         char ch;
40         for (int i = 0; i < str.length(); i++) {
41             switch (str.charAt(i)) {
42                 case 0 :
43                     continue;
44                 case '\b':
45                     retval.append("\\b");
46                     continue;
47                 case '\t':
48                     retval.append("\\t");

```

```

49         continue;
50     case '\n':
51         retval.append("\\n");
52         continue;
53     case '\f':
54         retval.append("\\f");
55         continue;
56     case '\r':
57         retval.append("\\r");
58         continue;
59     case '\"':
60         retval.append("\\\"");
61         continue;
62     case '\\':
63         retval.append("\\\\");
64         continue;
65     case '\':
66         retval.append("\\'");
67         continue;
68     default:
69         if ((ch = str.charAt(i)) < 0x20 || ch > 0x7e) {
70             String s = "0000" + Integer.toString(ch, 16);
71             retval.append("\\u" + s.substring(s.length() - 4, s.length()));
72         } else {
73             retval.append(ch);
74         }
75         continue;
76     }
77 }
78 return retval.toString();
79 }
80
81 /**
82  * Returns a detailed message for the Error when it is thrown by the token manager to indicate a lexical error.
83  * Parameters : EOFSeen    : indicates if EOF caused the lexicl error curLexState : lexical state in which this
84  * error occured errorLine  : line number when the error occurred errorColumn : column number when the error
occured
85  * errorAfter  : prefix that was seen before this error occurred curchar    : the offending character Note: You can
86  * customize the lexical error message by modifying this method.
87  */
88 protected static String LexicalError(boolean EOFSeen, int lexState, int errorLine, int errorColumn, String errorAfter,
char curChar) {
89     return("Lexical error at line " +
90         errorLine + ", column " +
91         errorColumn + ". Encountered: " +
92         (EOFSeen ? "<EOF> " : ("\"" + addEscapes(String.valueOf(curChar)) + "\"") + " (" + (int) curChar + "), ")
+
93         "after : \"" + addEscapes(errorAfter) + "\"");
94 }
95
96 /**
97  * You can also modify the body of this method to customize your error messages. For example, cases like

```

```

98      * LOOP_DETECTED and INVALID_LEXICAL_STATE are not of end-users concern, so you can return something
like :
99      * <p/>
100     * "Internal Error : Please file a bug report .... "
101     * <p/>
102     * from this method for such cases in the release version of your parser.
103     */
104     public String getMessage() {
105         return super.getMessage();
106     }
107
108     /*
109     * Constructors of various flavors follow.
110     */
111
112     public TokenMgrError() {
113     }
114
115     public TokenMgrError(String message, int reason) {
116         super(message);
117         errorCode = reason;
118     }
119
120     public TokenMgrError(boolean EOFSeen, int lexState, int errorLine, int errorColumn, String errorAfter, char
curChar, int reason) {
121         this(LexicalError(EOFSeen, lexState, errorLine, errorColumn, errorAfter, curChar), reason);
122     }
123 }
124

```