CPSC-597
Project Report

Java Graphics Scripting Language

http://jgsl.dev.java.net/

Joe Chavez

Abstract

The Java Graphics Scripting Language (JGSL) is designed make the Java graphics API easily accessible. By providing a simple language it will be possible for non-programmers to use the rich graphics capabilities of the Java platform. Another goal is to reduce the overhead of drawing a Java 2D graphics object, like a circle, to one line of JGSL. By combining the JGSL drawing capabilities, the user can compose lines, circles, rectangles and many other shapes in a way not previously possible with the Java platform. This version of JGSL represents a vertical slice into the capabilities of the system. The overall goal is to deliver a framework upon which future enhancements can be added with little disruption to the overall application architecture. Upon completion of this project the JGSL will be released as open source software.

Keywords

Java, Graphics, Scripting, Language, Programming

Table of Contents

Abstract i
Keywords i
1 Introduction 6
  1.1 Objectives 7
  1.2 System Environment 8
    1.2.1 Development Environment 8
    1.2.2 Operational Environment 9
2 Requirements 10
  2.1 High-Level Requirements 10
  2.2 Detailed Requirements 10
  2.3 Use Case Model 11
3 Design 16
  3.1 Logical Model 16
  3.2 Model-View-Controller 17
  3.3 Class Collaboration 20
4 Implementation 22
  4.1 Project Build System 22
  4.2 Source Files 23
  4.3 Test Scripts 24
  4.4 Build Products 24
5 Test and Integration 26
  5.1 Test Model 26
  5.2 Test Cases 28
    5.2.1 All Commands Test Case 28
    5.2.2 Script Parser Test Case 28
    5.2.3 Script Engine Test Case 29
    5.2.4 Interactive GUI Test Case 29
    5.2.5 Command Console Test Case 30
    5.2.6 JGSL Viewer Test Case 30
    5.2.7 JGSL Installation Test Case 30
  5.3 Requirements Traceability Matrix 31
  5.4 Deployment Model 31
    5.4.1 Java Web Start (JNLP) Deployment 32
    5.4.2 Windows Deployment 33
    5.4.3 OS X Deployment 33
    5.4.4 Linux/Unix Deployment 34
6 Operating Instructions 35
  6.1 Interactive GUI 35
  6.2 Command Console 38
7 Installation Instructions 40
  7.1 Online Installation 40
  7.2 Offline Installation 43
    7.2.1 Windows 43
    7.2.2 Apple OS X 48
    7.2.3 Linux/Unix 49
  7.3 Source Distribution 49
    7.3.1 Prerequisites 49
    7.3.2 CVS Repository 49

7.3.3 Building JGSL 50
8 Recommendations for Enhancements 51
9 Bibliography 52
10 Appendices 53
A. Project Schedule 54
B. JGSL Language Specification 55
C. JGSL Grammar 56
D. JGSL Build File 57
E. Source Code 58
F. JGSL Java API Documentation 59
G. JGSL Tutorial 60

Index of Tables

Table 1 Document Version History v
Table 2 Ant Build Targets 23
Table 3 Requirements Traceability Matrix 31
Table 4 JGSL Command Console Platform Executable 38
Table 5 Command Console Option List 39
Table 6 Offline Installers 40

Index of Figures

Figure 1 Drawing a line in JGSL 6
Figure 2 Drawing a line in Java 6
Figure 3 Use Case Actors Diagram 11
Figure 4 Interactive GUI Use Case Diagram 12
Figure 5 Command Console Use Case Diagram 14
Figure 6 Script Engine Use Case Diagram 15
Figure 7 JGSL Logical Component Architecture 17
Figure 8 JGSL MVC Pattern Class Diagram 18
Figure 9 JGSL Main Class Diagram 19
Figure 10 Parser Class Diagram 20
Figure 11 Interactive GUI Collaboration Diagram 21
Figure 12 Generate Implementation Collaboration Diagram 21
Figure 13 File Organization (legend below) 22
Figure 14 SRC Directory Structure 23
Figure 15 TEST Directory Structure 24
Figure 16 BUILD Directory Structure 24
Figure 17 DIST Directory Structure 24
Figure 18 Test Model Diagram 26
Figure 19 Test Model Class Diagram 27
Figure 20 Test Model Component Diagram 28
Figure 21 Deployment Model Diagram 32
Figure 22 Java Web Start (JNLP) Deployment Diagram 32
Figure 23 Windows Deployment Diagram 33
Figure 24 Mac OS X Deployment Diagram 34
Figure 25 Linux/Unix Deployment Diagram 34
Figure 26 JGSL Interactive Main Window 35
Figure 27 JGSL Interactive File Menu 36

Figure 28 JGSL Interactive Script Editor........36
Figure 29 JGSL Interactive Status Window........36
Figure 30 JGSL Interactive Script Actions Toolbar........37
Figure 31 JGSL Viewer........37
Figure 32 JGSL Interactive Help Menu........38
Figure 33 Online Installer Opening Dialog........40
Figure 34 Online Installer Splash Screen........41
Figure 35 Online Installer Verify Certificate Dialog........41
Figure 36 Online Installer Create Shortcut Dialog........42
Figure 37 JGSL Main Window........42
Figure 38 Windows Installer Welcome Dialog........44
Figure 39 Windows Installer Installation Folder Dialog........45
Figure 40 Windows Installer Ready Dialog........46
Figure 41 Windows Installer Progress Dialog........47
Figure 42 Windows Installer Completion Dialog........48
Figure 43 JGSL Windows Shortcut........48
Figure 44 JGSL Main Window........50

Document Version History

Table 1 Document Version History

Date	Version	Modification	Author
May 1, 2005	0.1	Document outline, tables and indexes.	J. Chavez
May 3, 2005	0.2	Introduction, objectives, and high level requirements.	J. Chavez
May 5, 2005	0.3	Detailed requirements, design diagrams, implementation diagrams, requirements traceability matrix, use case model.	J. Chavez
May 7, 2005	0.4	Design, implementation, test, references.	J. Chavez
May 9, 2005	0.5	Design, implementation, test & integration, operating instruction and installation.	J. Chavez
May 14, 2005	0.6	Use cases descriptions.	J. Chavez
May 15, 2005	0.61	Installation: Online and source code access, initial test case descriptions.	J. Chavez
May 20, 2005	0.7	Installation: Offline for Windows, Linux and Mac.	J. Chavez
May 21, 2005	0.75	Operating instructions for Windows, Linux and Mac.	J. Chavez
May 22, 2005	0.8	Test & Integration: Test model diagrams, deployment model diagrams.	J. Chavez
May 24, 2005	0.9	Corrections and formatting.	J. Chavez
May 25, 2005	1.0d	Final draft.	J. Chavez
May 26, 2005	1.0	Final.	J. Chavez

1 Introduction

The primary goal of the Java Graphics Scripting Language (JGSL) is to create a simple, yet powerful, language that will allow non-programmers to access the graphics capabilities of Java. There are many scripting languages available in the Java open source arena but the JGSL will distinguish itself by providing a programming environment focused on computer graphics [Barron 2000]. This language can be a useful tool for teaching non-technical users the concepts and uses of computer graphics [Warren 2001]. With the JGSL it will be possible to reduce approximately 20 lines of Java code to a single line of script to generate a simple 2D graphics object such as a line or shape. Shown in Figure 1 is a JGSL script that draws a line shape using 4 lines of script. In comparison, the Java code shown in Figure 2 requires over 20 lines of code. JGSL scripting simplifies graphics programming on the Java platform by a factor of 5.

```
begin

canvas (640, 480, BLACK, WHITE, "line");

line (25, 25, 100, 75);

end
```

Figure 1 Drawing a line in JGSL

```
package jgsl.view.swing;

import javax.swing.JFrame;
import java.awt.*;
import java.awt.event.*;

public class ScriptComparison extends JFrame {

    public void paint(Graphics graphics) {
        super.paint(graphics);
        Graphics2D g2 = (Graphics2D) graphics;
        g2.drawLine(100, 100, 125, 125);
    }

    public static void main(String[] args) {
        ScriptComparison f = new ScriptComparison();
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent ev) {
                System.exit(0);
            }
        });

        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```

Figure 2 Drawing a line in Java

The selection of the Java platform for the basis of this scripting language is a natural one. The Java Virtual Machine (JVM) offers feature rich object-oriented language that provides type safety, automatic memory management, multi-threading and platform neutral code generation. The Java 5 Standard Edition (version 1.5) platform contains the Java Application Programming Interface (API), which provides a rich set of 2D and 3D graphics capabilities. The JGSL will use and extend these capabilities to implement its scripting language environment. The "java.net" online community "`javadesktop`" project has accepted the JGSL project and it is currently in "incubator" status. The JGSL is available as open source for any interested to party to make contributions to the project. The project website is located on the Internet at `https://jgsl.dev.java.net/`. This is where the project source code repository is located. Instructions on how to obtain the source code are contained in section 7.3 and are also available on the project website. Also contained on the project website are discussion forums, mailing lists, file downloads, and the issue tracker.

The development process for this project is iterative and incremental in nature and is loosely based on Extreme Programming (XP) [Beck and Andres 2000] and Agile software development processes [Ambler 2004]. Both of these processes are geared toward multi-developer projects that allow the customization of the processes for the needs of this project. Each increment represents a milestone in the functionality of the project. During the iterations, the products of the previous increment are used to implement the goals of the next milestone. Among these products are unit tests that validate each milestone and also ensure that new development does not break the system. Thus, the system is continuously integrated as part of the development process. If problems are found with products from a previous iteration, refactoring techniques [FOLWER 1999] are applied to ensure the cohesiveness of the system.

1.1 Objectives

Contained in the list below is the set of objectives for the JGSL project. The objectives are numbered with O.n sequencing where "n" represents an objective number. Following the list is a discussion of each objective.

O.1. Design of the JGSL programming language.
O.2. Design and development of a lexical analyzer and parser for the JGSL.
O.3. Design and development of the JGSL runtime framework.
O.4. Design and development of the JGSL user interface for batch and interactive processing.
O.5. Development of test cases.
O.6. Development of API documentation.
O.7. Development of user documentation.
O.8. Release JGSL.

Note: O = Objective

In objective O.1 I will design a programming language that will form the basis of the JGSL. This language will be human and machine-readable. It must be expressive enough to provide simple constructs for complex computer graphics processing concepts. Following this in objective O.2 I will design and develop a lexical analyzer and parser for the JGSL. This will be based upon the core concepts in computer science relating to this area of interest. Objective O.3 will form the basis for the application architecture of the JGSL. The runtime framework will encapsulate the lexical analyzer and parser, the processing of scripting commands, the graphics-rendering engine and script persistence. With the first three objectives the foundation is available upon which the JGSL user interface can be built in objective O.4. The user interface will consist of a command-line batch execution as well as interactive execution within a Graphic User Interface. Shown in Figure 7 is the logical component architecture of the JGSL system. In this diagram the objectives O.2, O.3 and O.4 are mapped into components over three layers. The Persistence Layer contains the Parser and Persistence components from objective O.2. The Graphics Processing Layer contains the Rendering and Processor component from objective O.3. Finally, the Interface Layer contains the Batch and Interactive components from objective O.4. Objective O.5, Development of Test Cases, will be performed in parallel with objectives O.1 through O.4. It is essential to the success of the project that test cases be designed and developed as the project progresses. The documentation set for objectives O.6 and O.7 will give the application developer and end user the information needed to effectively use the JGSL. Finally, in object O.8 the JGSL will be released to the community and be made freely available via an open source license.

1.2 System Environment

The System Environment consists of the Development Environment and the Operational Environment. The Development Environment consists of the set of software, tools and hardware needed to design, develop and implement the system. The Operational Environment consists of any combination of hardware and operating system that supports the Java 5 Runtime Environment. The JGSL operational environment requires a subset of the software components used in development at runtime. Also contained in the Operational Environment are online and offline installers for the JGSL distribution.

1.2.1 Development Environment

1.2.1.1 Software

- Java 5 Standard Edition (version 1.5)
- JavaCC – Java Compiler Compiler
- Jakarta Commons
    - Command Line Parameter Library
- Javassist
- Log4j
- JUnit

1.2.1.2 Tools

- IDE: IntelliJ IDEA from JetBrains
- Build Tool: Apache Ant
- Testing Tool: JUnit

1.2.1.3 Hardware

Apple Mac PowerBook G4

- CPU: 1.5GHz
- Video: 128MB Radeon Video Adapter
- RAM: 2GB

- Hard Drive: 80GB
- Operating System: Apple Mac OS X 10.4

IBM T30

- CPU: 2.0GHz
- Video: 32MB RAM
- RAM: 1GB
- Hard Drive: 60GB
- Operating System: Windows XP

Dell OptiPlex GX500

- CPU: 500MHz
- Video: 32MB RAM
- RAM: 512 MB
- Hard Drive: 80GB
- Operating System: Linux Enterprise Workstation 3.0

1.2.2 Operational Environment

1.2.2.1 Software

- Any operating system that provides the Java 5 Runtime Environment (version 1.5)
- JGSL
- Jakarta Commons
  - Command Line Interface Library
- Javassist
- Log4j

1.2.2.2 Tools

- Web Based Software Installer using Java Web Start
- Web Browser
  - Netscape 7
  - Firefox 1
  - IE 6
  - Safari 2

1.2.2.3 Hardware

- Any hardware platform that supports Java 5 Runtime Environment (version 1.5)

2 Requirements

Requirements for the JGSL are derived primarily from the JGSL Language Specification (see Appendix A). The high-level and detailed requirements are presented in this section. In the Test and Integration section the high-level requirements are mapped to the system functionality using a requirements traceability matrix.

2.1 High-Level Requirements

R.1. The JGSL shall provide a text-based scripting language for the purpose of creating and manipulating graphics objects.
R.2. The JGSL shall provide a framework for executing JGSL scripts.
R.3. The JGSL shall provide a visual user interface.
R.4. The JGSL shall provide a console user interface.
R.5. The JGSL shall provide a means to execute a script outside of the JGSL scripting environment.
R.6. The JGSL shall be made available on multiple platforms.
R.7. The JGSL shall be made available for download via the Internet.
R.8. The JGSL shall be made available as an open source project.

2.2 Detailed Requirements

R.1. The JGSL shall provide a textual scripting language for the purpose of creating and manipulating graphics objects.
- D1.1. The JGSL script shall be editable by any common text editor.
- D1.2. The JGSL script shall be designed as a BNF grammar.
- D1.3. The JGSL script shall be parsed using a lexical scanner.
- D1.4. The JGSL lexical scanner shall provide detailed information to the user when errors are found during script parsing.
- D1.5. The JGSL lexical scanner shall be capable of execution independently of the JGSL as a whole.

R.2. The JGSL shall provide an execution engine for JGSL scripts.
- D2.1. The JGSL execution engine shall utilize the JGSL lexical scanner to determine script validity.
- D2.2. The JGSL execution engine shall create an implementation independent form of the JGSL script.
- D2.3. The JGSL execution engine shall generate executable Java bytecode from the implementation independent form.

R.3. The JGSL shall provide a graphic user interface (GUI).
- D3.1. The JGSL GUI shall support the entry of JGSL scripts.
- D3.2. The JGSL GUI shall support the load and store of JGSL scripts to disk based files.
- D3.3. The JGSL GUI shall support the parsing of JGSL scripts with reporting.
- D3.4. The JGSL GUI shall support the executing of JGSL scripts.
- D3.5. The JGSL GUI shall support the viewing of JGSL scripts.
- D3.6. The JGSL GUI shall provide online help.

R.4. The JGSL shall provide a command console user interface.
- D4.1. The JGSL command console shall support the parsing of JGSL scripts with reporting.
- D4.2. The JGSL command console shall support the execution of JGSL scripts.
- D4.3. The JGSL command console shall support the viewing of JGSL scripts.

D4.4. The JGSL command console shall provide console based help.

R.5. The JGSL shall provide a means to execute a script outside of the JGSL scripting environment.

D5.1. The JGSL shall provide a means of packaging an executable script so that it can be executed as a standalone program (provided a Java 5 virtual machine is available).

R.6. The JGSL shall be made available on multiple platforms.

D6.1. The JGSL shall be made available as Java compiled bytecode.

D6.2. The JGSL shall be certified on the following platforms: Windows XP, Mac OS X 10.4 and Linux Enterprise Workstation 3.0.

R.7. The JGSL shall be made available for download via the Internet.

D7.1. The JGSL shall be located on a publicly available Web server using Java Web Start technology.

D7.2. The JGSL shall be located on a public available Web server as a binary package.

D7.3. The JGSL shall be made available in source code form via a Concurrent Versioning System (CVS).

R.8. The JGSL shall be made available as an open source project.

D8.1. The JGSL shall be maintained and distributed under an open source license.

2.3 Use Case Model

A Use Case Model is the next step in elaborating the project requirements. The set of actors for the system are the JGSL User and the JGSL Developer shown in Figure 3 Use Case Actors Diagram below. The JGSL User is an actor who wishes to use the JGSL as a scripting language to create graphics objects, taking advantage of the interactive and command console interfaces. The JGSL Developer is an actor who wished to use the JGSL as a graphics generation engine and, thus, would use the API scripting interface.

JGSL User
JGSL Developer

Figure 3 Use Case Actors Diagram

The set of Use Cases for the JGSL User actor are shown in Figure 4 and Figure 5 and consists of:

- Open JGSL GUI
- Write JGSL Script
- Save JGSL Script
- Load JGSL Script

- Execute JGSL Script
- View JGSL Help
- Run JGSL Command Console
- View JGSL Command Console Help

These use cases capture the overall flow of data from the user's perspective.

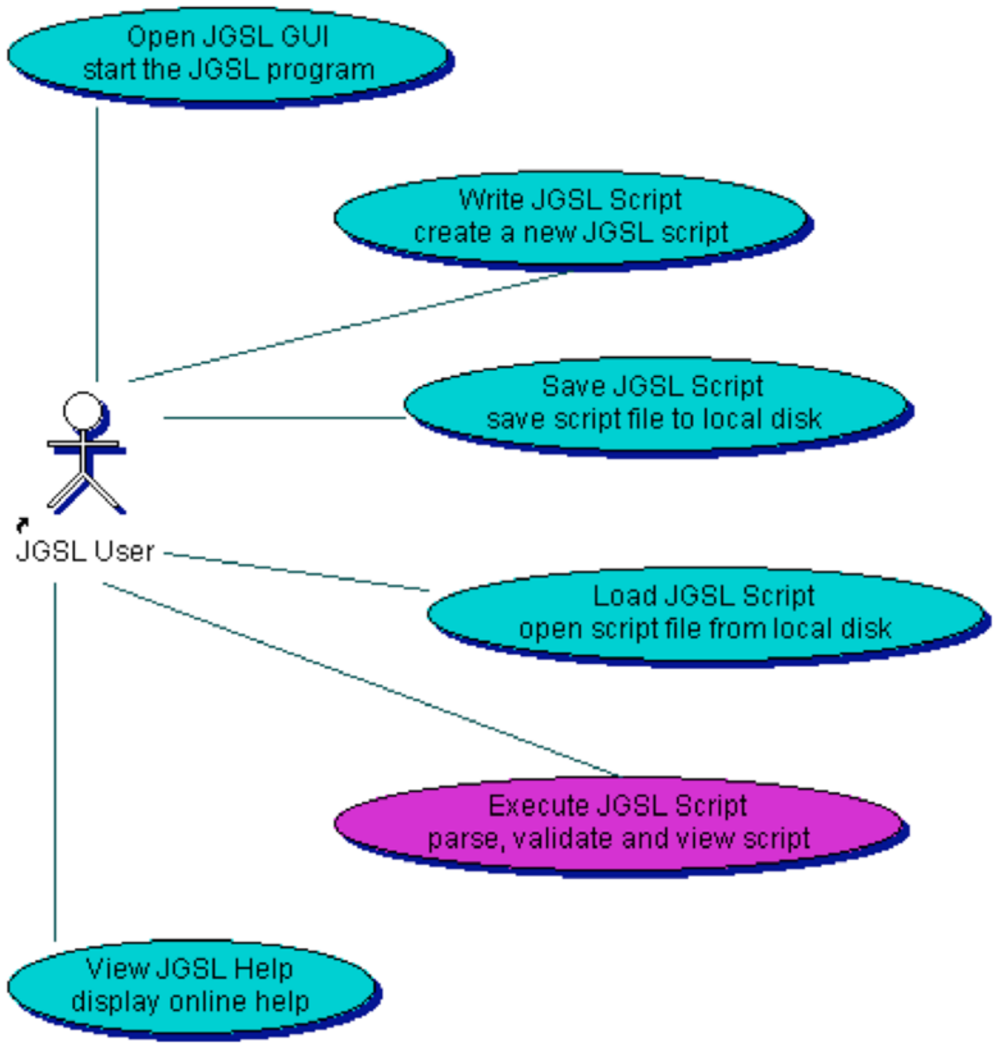


Figure 4 Interactive GUI Use Case Diagram

Interactive GUI Use Case	
Use Case	Interactive GUI
Version	1.0
Purpose	Presents the JGSL User with a graphic user interface that provides the ability to create, save, load and execute JGSL scripts.

Interactive GUI Use Case	
Requirements	R.3. The JGSL shall provide a visual user interface.
Derived Requirements	D1.1. The JGSL GUI shall support the entry of JGSL scripts. D1.2. The JGSL GUI shall support the load and store of JGSL scripts to disk based files. D1.3. The JGSL GUI shall support the parsing of JGSL scripts with reporting. D1.4. The JGSL GUI shall support the executing of JGSL scripts. D1.5. The JGSL GUI shall support the viewing of JGSL scripts. D1.6. The JGSL GUI shall provide online help.
States	• Open – open the JGSL GUI • Write – write a script • Save – save a script • Load – load a script • Execute – execute and view a script • Help – view help • Close – close the JGSL GUI
Scenarios	Scenario 1: New JGSL Script JGSL User: 1. Opens the JGSL GUI 2. Writes the JGSL script 3. Saves the JGSL script 4. Executes the JGSL script 5. Closes the JGSL GUI Scenario 2: Existing JGSL Script JGSL User: 1. Opens the JGSL GUI 2. Loads the JGSL script 3. Saves the JGSL script 4. Executes the JGSL script 5. Closes the JGSL GUI Scenario 3: View JGSL Help JGSL User: 1. Opens the JGSL GUI 2. Opens the JGSL Help
Notes & Assumptions	Script execution will fail if the JGSL parser detects any problems. The user will be informed of the exact problem.
Pre-conditions	The JGSL software must be installed on user workstation.
Post-conditions	The JGSL script is stored on local workstation.

Use Case Description 1 - Interactive GUI

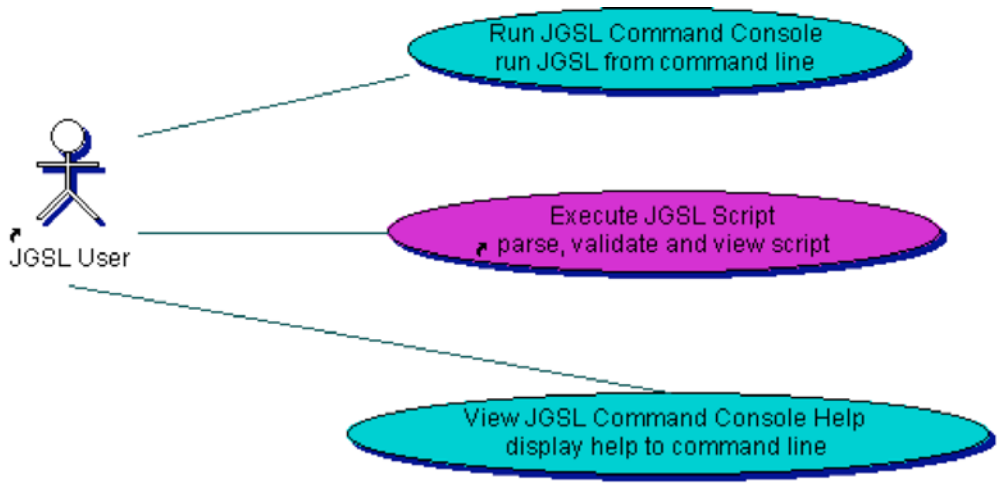

Run JGSL Command Console
run JGSL from command line
JGSL User
Execute JGSL Script
parse, validate and view script
View JGSL Command Console Help
display help to command line

Figure 5 Command Console Use Case Diagram

Command Console Use Case	
Use Case	Command Console
Version	1.0
Purpose	The JGSL User authors a script using a text editor and then executes the script using the JGSL command console.
Requirements	R.4. The JGSL shall provide a command console user interface.
Derived Requirements	D4.1. The JGSL command console shall support the parsing of JGSL scripts with reporting. D4.2. The JGSL command console shall support the execution of JGSL scripts. D4.3. The JGSL command console shall support the viewing of JGSL scripts. D4.4. The JGSL command console shall provide console based help.
States	• Run – load a script. • Execute – execute and view a script. • Help – view help.
Scenarios	Scenario 1: Existing JGSL Script JGSL User: 1. Run JGSL from the command line. 2. Execute the JGSL script. 3. View JGSL script.
Notes & Assumptions	Script execution will fail if the JGSL parser detects any problems. The user will be informed of the exact problem.
Pre-conditions	The JGSL software must be installed on user workstation. A JGSL script must exist prior to starting this use case.
Post-conditions	The JGSL script is displayed on the users workstation.

Use Case Description 2 Command Console

The JGSL Developer actor interacts with the Execute JGSL Script use case to parse, validate and view scripts.

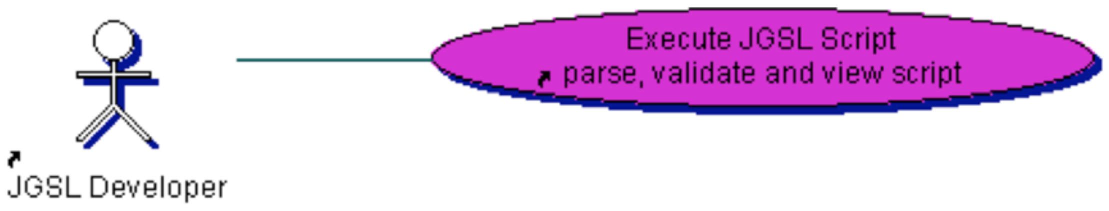


Figure 6 Script Engine Use Case Diagram

Script Engine Use Case	
Use Case	Script Engine
Version	1.0
Purpose	The JGSL Developer obtains the JGSL runtime framework and embeds it within their application.
Requirements	R.2. The JGSL shall provide an execution engine JGSL scripts. R.5. The JGSL shall provide a means to execute a script outside of the JGSL scripting environment.
Derived Requirements	D2.1. The JGSL execution engine shall utilize the JGSL lexical scanner to determine script validity. D2.2. The JGSL execution engine shall create an implementation independent form of the JGSL script. D2.3. The JGSL execution engine shall generate executable Java bytecode from the implementation independent form. D5.1. The JGSL shall provide a means of packaging an executable script so that it can be executed as a standalone program (provided a Java 5 virtual machine is available).
States	• Execute – execute and view a script.
Scenarios	Scenario 1: Execute JGSL Script JGSL User: 1. Execute the JGSL script.
Notes & Assumptions	Script execution will fail if the JGSL parser detects any problems. The JGSL Developer will receive API level error reporting as an exception or parse result data. Java Standard Edition 5 is available on the user workstation.
Pre-conditions	The JGSL engine must be installed on user workstation. A JGSL script must exist prior to starting this use case.
Post-conditions	The JGSL script is stored in compiled form on the workstation.

Use Case Description 3 Script Engine

3 Design

3.1 Logical Model

The requirements and Use Case Model are used to initiate the design of the JGSL software architecture. From this examination a high-level component model is created. The set of components were derived from an analysis of the high level and detailed requirements taking into account the scenarios described in the use cases. These components represent a logical view of the system in terms of the high level relationships between the components and the layers. The logical component layering is shown in Figure 7 as a UML component diagram. The directed arrows show the dependencies between the components in the stack. Each colored package represents a layer in the architecture. The interface layer contains the components for batch and interactive user interface. The batch component represents the non-GUI or textual interface to the JGSL. The interactive component is the GUI based interface to the JGSL. In the graphics processing, layer rendering and processor components are present. The rendering component represents the translation of the JGSL script into executable Java bytecode. The processor component handles requests from the interface layer and coordinates activities between the rendering component and the persistence layer. The parser and persistence components are in the persistence layer. The parser component provides lexical analysis and validation of the JGSL script prior to rendering. The persistence component handles the persistence of JGSL content for both scripts and bytecode. The diagram also indicates which objective the layers are associated with.

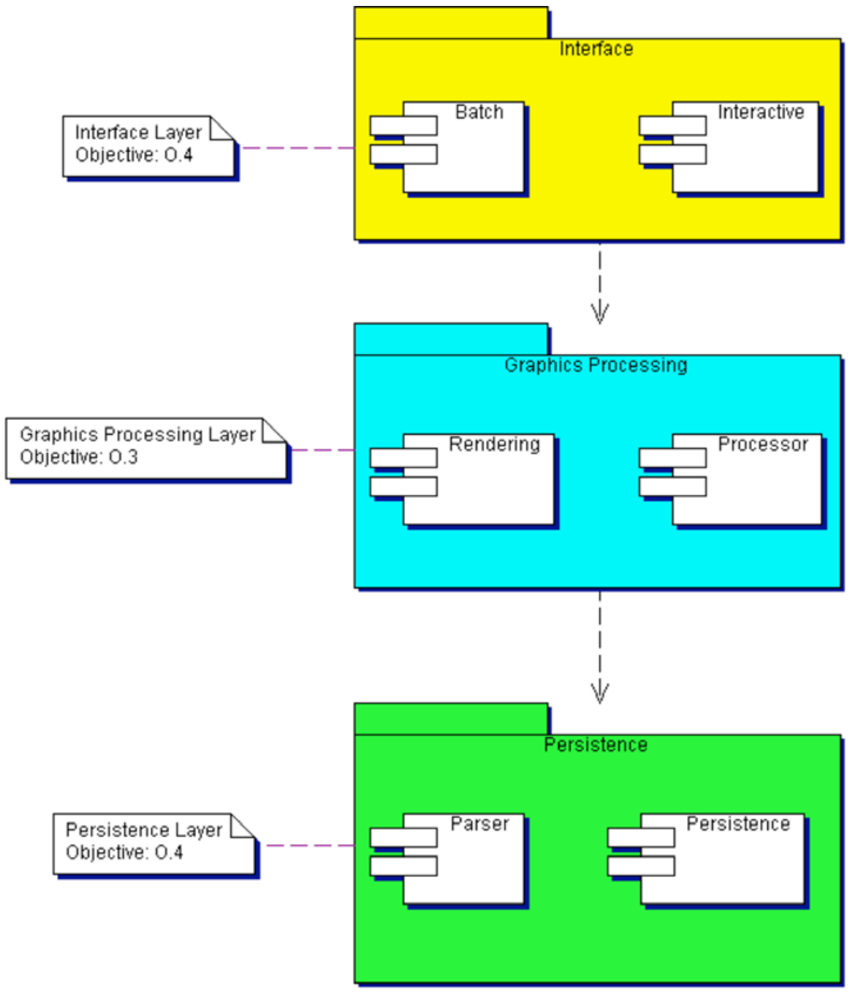


Figure 7 JGSL Logical Component Architecture

3.2 Model-View-Controller

The design of the application architecture is based on the logical component architecture. The implementation of the JGSL system is based on the Model-View-Controller (MVC) design pattern [FOWLER 2002]. In Figure 8 is a UML package diagram showing the Java packages for the model, view and controller. The view package consists of the classes needed to render a JGSL script and the script editor GUI class. The controller package consists of classes needed to implement the script engine. The model package consists of the language independent object model for the JGSL script format. The view package has dependencies on the controller and model packages. The controller depends on the model as the source of JGSL script meta-data and the language independent object model. Additional details regarding the class level relationships and interactions are in the remainder of the section.

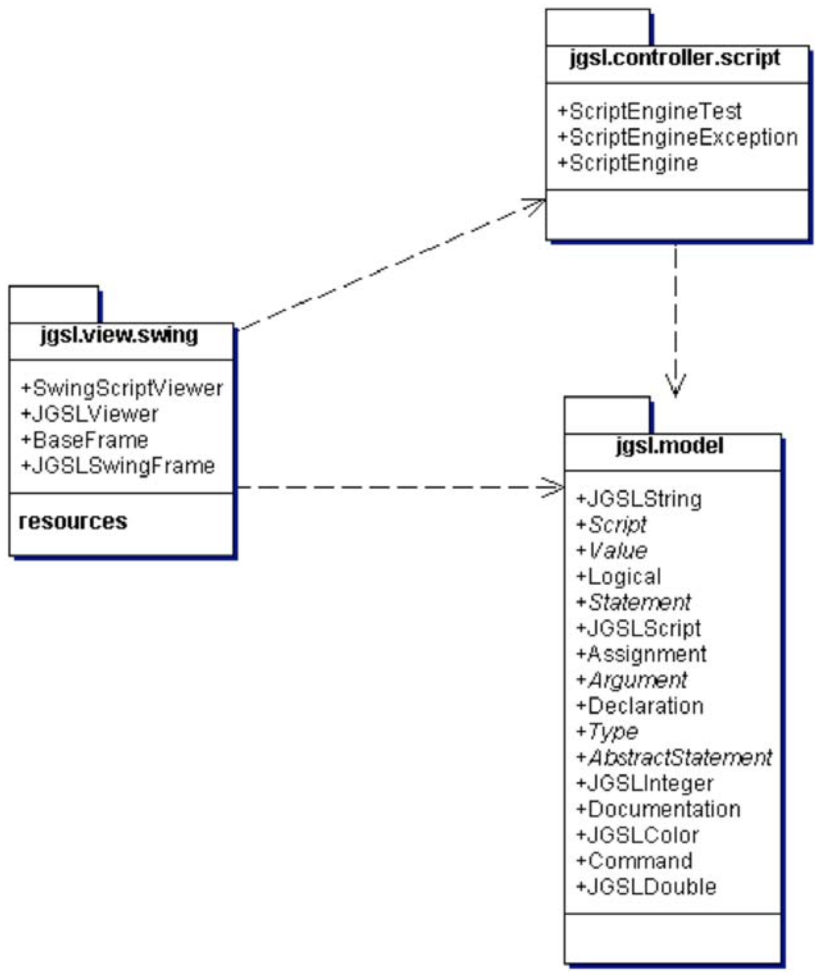


Figure 8 JGSL MVC Pattern Class Diagram

The next 2 diagrams show the static relationships between the classes in the application architecture. In Figure 9 is a UML class diagram showing association relationships between the main classes of the JGSL application architecture. The class JGSL contains the main public interface to the system. It contains a composition relationship with the `ScriptEngine` class. The `ScriptEngine` class provides the interface to parse and view scripts as well as the processing of command console mode execution parameters. The `ScriptEngine` class contains an aggregation relationship with the JGSL script class and composition relationships with the `ScriptParser` and `SwingScriptViewer` classes. The `JGSLScript` class represents the object model of a parsed and validated JGSL script and is aggregated by the `ScriptParser` and `JGSL_Parser` classes. The `ScriptParser` class encapsulates via composition relationship the `JGSL_Parser` class. The `JGSL_Parser` is generated from the `JGSL_Parser.jj` file that is a JAVACC grammar file. By encapsulating the `JGSL_Parser` class within the `ScriptParser` it is possible to replace a JAVACC grammar with another if the need arises. Both the `ScriptParser` and `JGSL_Parser` class use a single instance of the `JGSLScript` class to generate the language independent object model from a JGSL script file.

This same instance of the `JGSLScript` class is used by the `ScriptEngine` to generate the Java class from the JGSL object model. The `SwingSriptViewer` is encapsulated via a composition relationship by the `ScriptEngine` class. This class associates the `JGSLViewer` class that is used to view the generated JGSL object model.

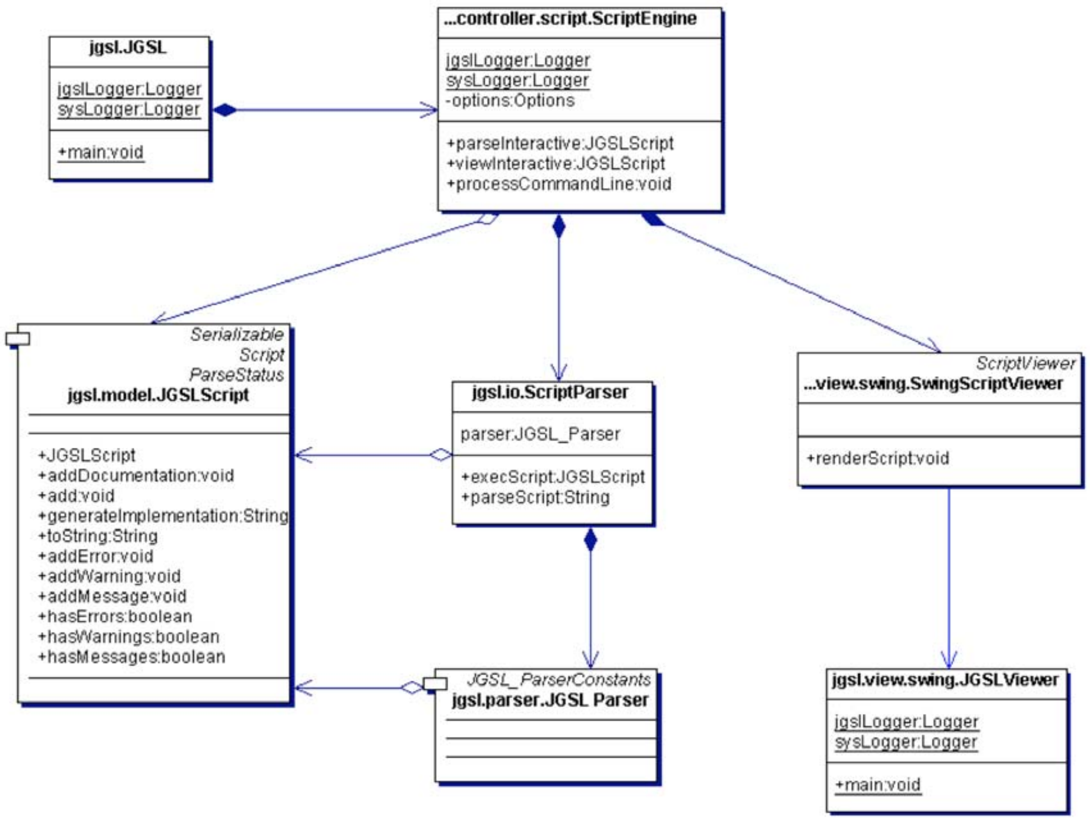


Figure 9 JGSL Main Class Diagram

The `ScriptParser` and its related classes are shown in Figure 10. As mentioned previously, the `ScriptParser` encapsulates the `JGSL_Parser` class and also provides a facility for reporting status and exceptions encountered during the parsing process. The `ScriptParser` provides the `JGSL_Parser` class an instance of the `JGSLScript` class. The `JGSL_Parser` uses the `JGSLScript` class to create the JGSL script object model from the textual JGSL script file. Any errors, warnings or messages generated by the `JGSL_Parser` are added to the `JGSLScript` using objects based on the `Message` interface. Any fatal errors encountered cause a `ScriptParserException` to be generated to the caller of the `ScriptParser` class.

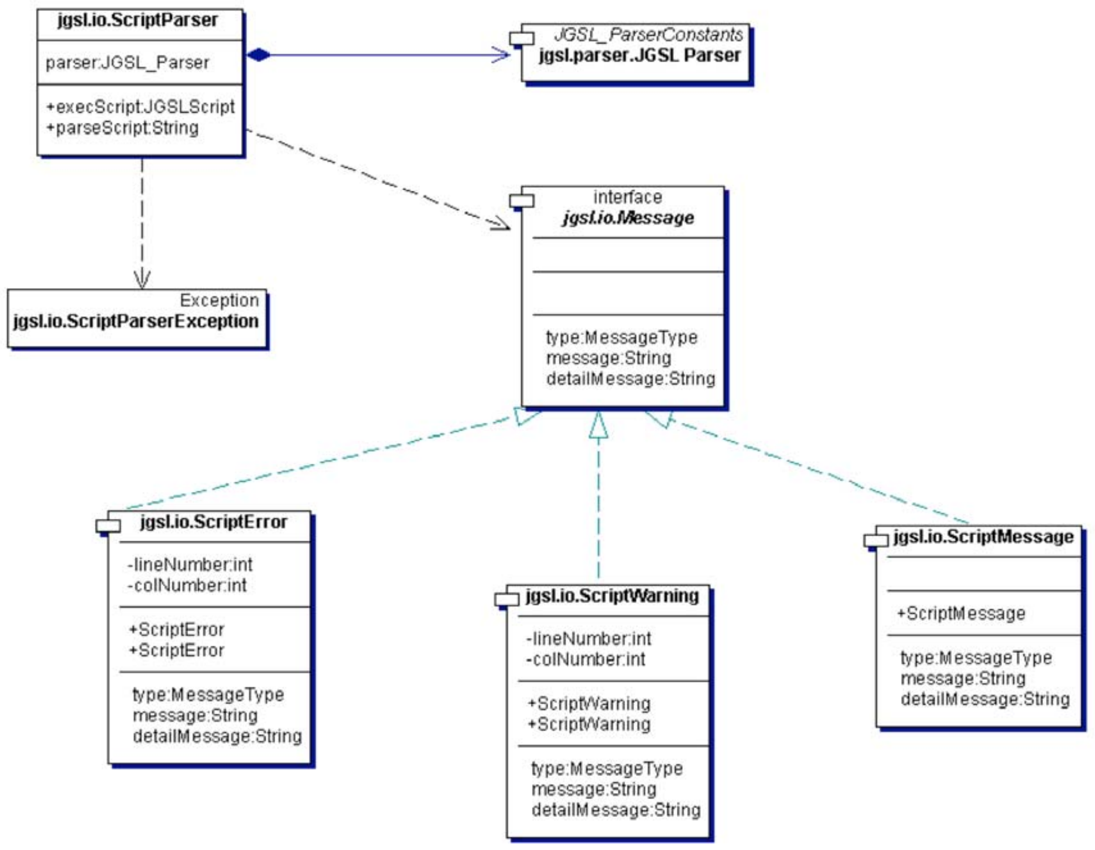


Figure 10 Parser Class Diagram

3.3 Class Collaboration

The next two UML collaboration diagrams describe the dynamic behavior of the classes described in the previous UML class diagrams. In Figure 11 is the collaboration diagram for the interactive GUI. In this diagram the JGSL User is the primary actor who executes a script via the `JGSLSwingFrame` class. The `JGSLSwingFrame` class is an application level window with which the user interacts. The `JGSLSwingFrame` class invokes a method on the `ScriptEngine` class to view the script. The `ScriptEngine` executes the script by invoking the `ScriptParser` which in turn invokes the `JGSL_Parser`. The result of the parse is an instance of the `JGSLScript` class that is used by the `ScriptParser` to generate a subclass of the `BaseFrame` class. This subclass will contain the Java bytecode that represents the graphics commands in the original JGSL script as written by the JGSL User actor. Once the Java class has been generated, the `ScriptParser` then creates a new Java Virtual Machine that invokes the `SwingScriptViewer` class with the name of the newly created JGSL java class. The `SwingScriptView` then dynamically instantiates the JGSL class (the `BaseFrame` subclass) via Java reflection and the script is displayed to the JGSL User in new Java application window. Figure 12 shows a collaboration diagram with details of the generate implementation interaction.

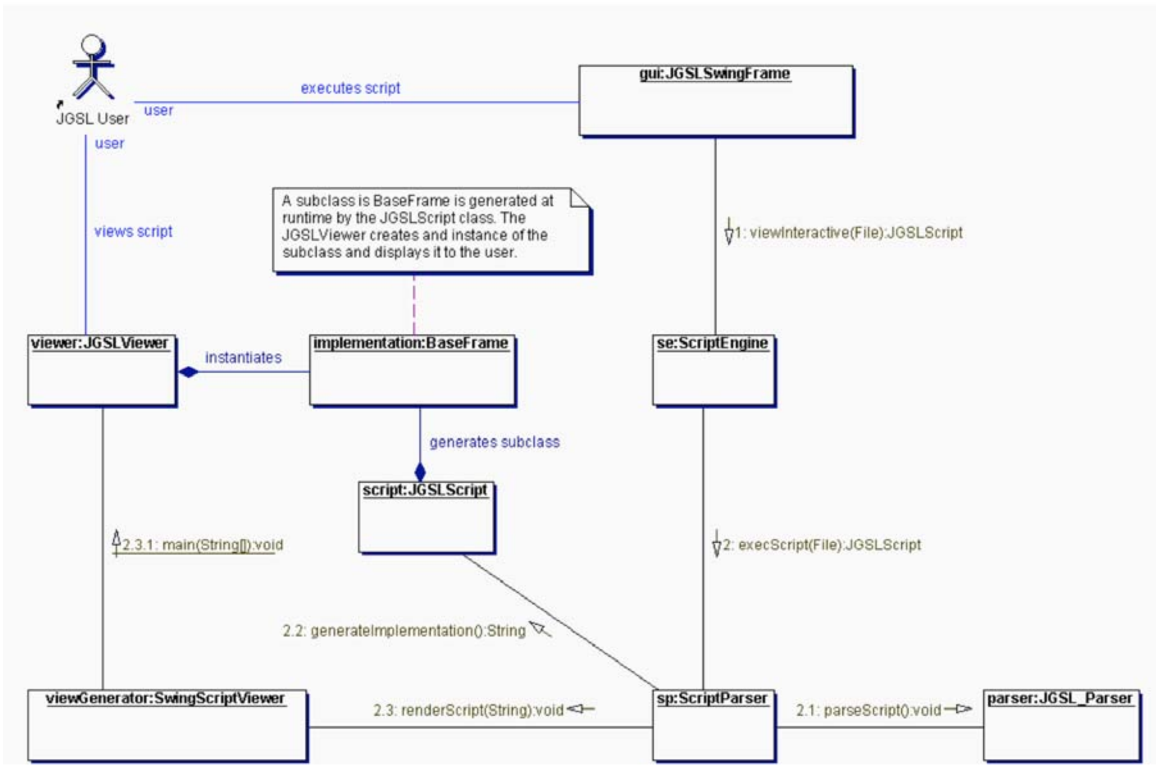

JGSL User
user
executes script
user
gui:JGSLSwingFrame
views script
A subclass is BaseFrame is generated at runtime by the JGSLScript class. The JGSLViewer creates and instance of the subclass and displays it to the user.
1: viewInteractive(File):JGSLScript
viewer:JGSLViewer
instantiates
implementation:BaseFrame
se:ScriptEngine
generates subclass
script:JGSLScript
2.3.1: main(String[]):void
2: execScript(File):JGSLScript
2.2: generateImplementation():String
viewGenerator:SwingScriptViewer
2.3: renderScript(String):void
sp:ScriptParser
2.1: parseScript():void
parser:JGSL_Parser

Figure 11 Interactive GUI Collaboration Diagram

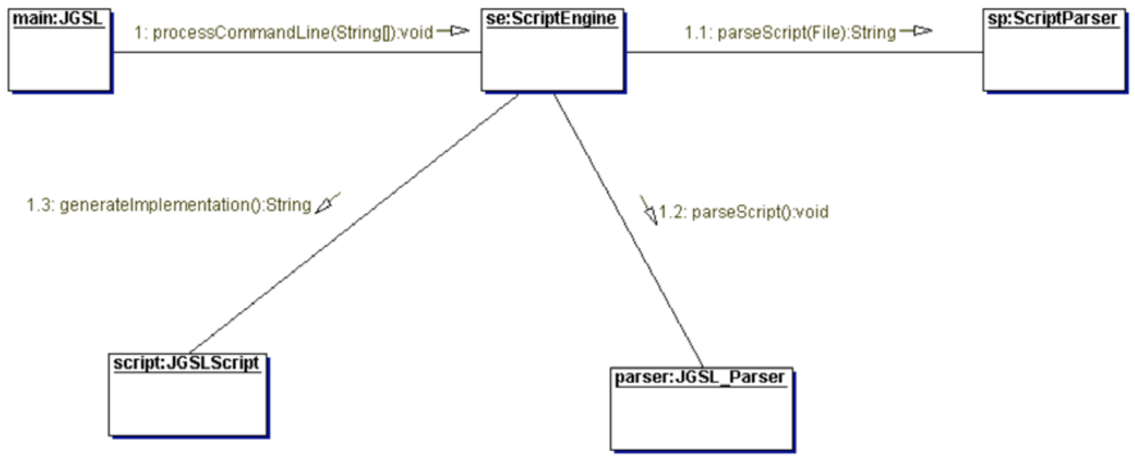
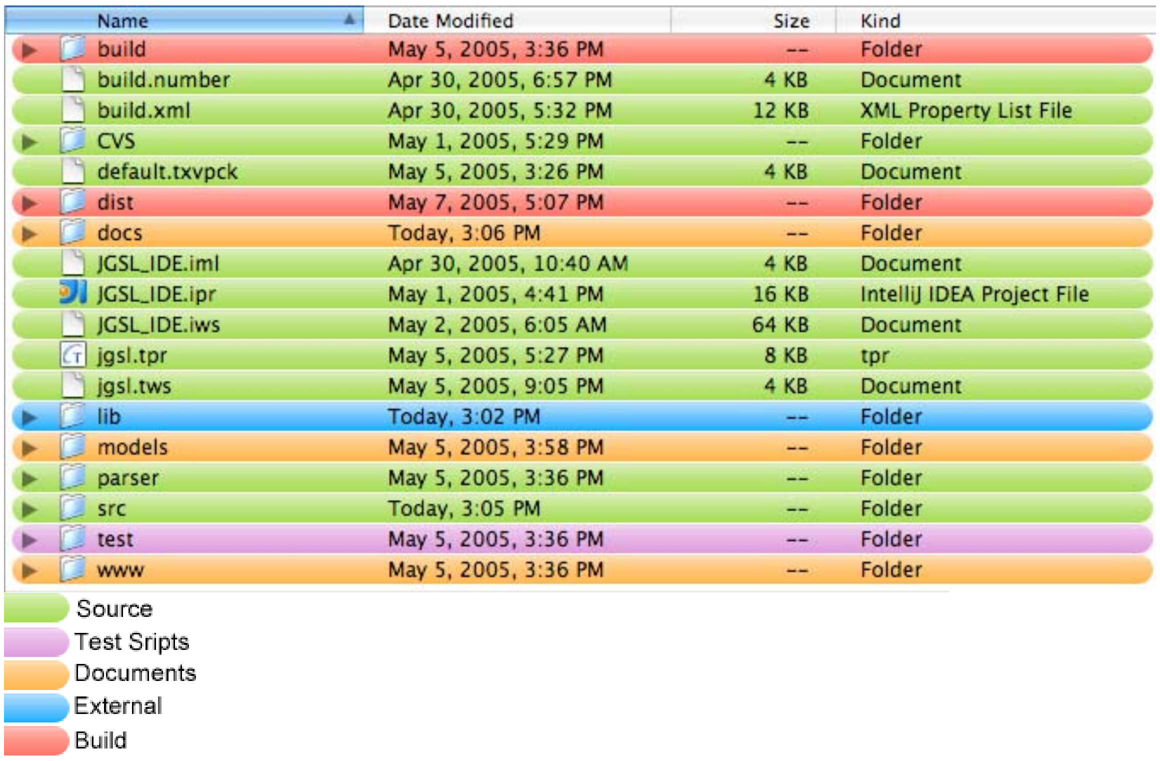

main:JGSL
1: processCommandLine(String[]):void
se:ScriptEngine
1.1: parseScript(File):String
sp:ScriptParser
1.3: generateImplementation():String
1.2: parseScript():void
script:JGSLScript
parser:JGSL_Parser

Figure 12 Generate Implementation Collaboration Diagram

4 Implementation

The project is implemented using the Java Programming language. The overall structure of the project is shown in Figure 13. The top-level project directory consists of source files, test scripts, documents, external libraries and build products. The set of source files consists of Java source code, IDE project files, version control files, and the Ant build script (`build.xml`). The test scripts are JGSL files used to test the JGSL application. The documents consist of specification documents, UML models and website documentation. The external files consist of the set of libraries needed for build time and or run time execution of the JGSL application. The build files consist of the products of building the system that include the compile Java classes and the Java Archive (JAR) files for the binary distribution. The contents of selected directories are discussed in further detail below. The directory CVS is present in each of the directories and is maintained by the Concurrent Versioning System (CVS). CVS is the source code control system used for current and future development of the JGSL.



Name	Date Modified	Size	Kind
build	May 5, 2005, 3:36 PM	--	Folder
build.number	Apr 30, 2005, 6:57 PM	4 KB	Document
build.xml	Apr 30, 2005, 5:32 PM	12 KB	XML Property List File
CVS	May 1, 2005, 5:29 PM	--	Folder
default.txvpck	May 5, 2005, 3:26 PM	4 KB	Document
dist	May 7, 2005, 5:07 PM	--	Folder
docs	Today, 3:06 PM	--	Folder
JGSL_IDE.iml	Apr 30, 2005, 10:40 AM	4 KB	Document
JGSL_IDE.ipr	May 1, 2005, 4:41 PM	16 KB	IntelliJ IDEA Project File
JGSL_IDE.iws	May 2, 2005, 6:05 AM	64 KB	Document
jgsl.tpr	May 5, 2005, 5:27 PM	8 KB	tpr
jgsl.tws	May 5, 2005, 9:05 PM	4 KB	Document
lib	Today, 3:02 PM	--	Folder
models	May 5, 2005, 3:58 PM	--	Folder
parser	May 5, 2005, 3:36 PM	--	Folder
src	Today, 3:05 PM	--	Folder
test	May 5, 2005, 3:36 PM	--	Folder
www	May 5, 2005, 3:36 PM	--	Folder

🟩	Source
🟪	Test Sripts
🟧	Documents
🟦	External
🟥	Build

Figure 13 File Organization (legend below)

4.1 Project Build System

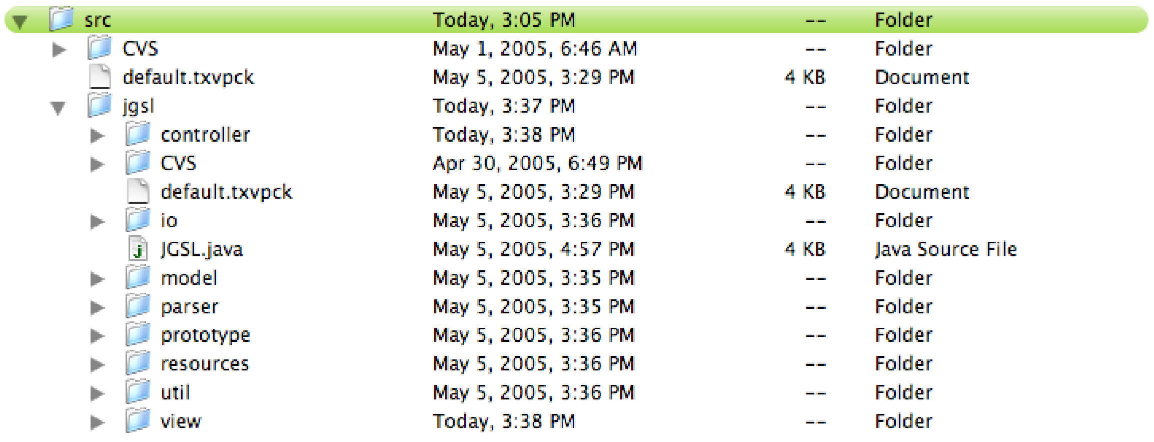
The file `build.xml` is an Ant project file for building the entire JGSL system. The file is in XML format and contains a project definition and a set of targets. Each target is in turn composed of a set of tasks. Each task performs a part of the build. Tasks can be combined to perform complex activities such as packaging a set of compiled classed into a Java Archive (JAR) file. The set of targets for the JGSL Ant build file are shown in Table 2. The complete build file is in Appendix D.

Table 2 Ant Build Targets

Target	Description
`clean`	`clean project build product from build and dist`
`compile`	`compiling the source`
`dist`	`generate the distribution`
`jgsl.javacc`	`build JGSL the parser`
`run`	`Run JGSL`
`run-parser`	`Run the JGSL parser`

4.2 Source Files

The SRC directory contains the Java source code for the JGSL application. The Java programming language uses a package name hierarchy to organize classes into a logical structure. This lends itself well to the structuring of source code by directory using the Java package naming convention for class hierarchies. For example, the java package "`jgsl.io`" contains the Java source code for all the classes in the `jgsl.io` package. The SRC directory also contains the JUnit java classes written to unit test the JGSL during the project.

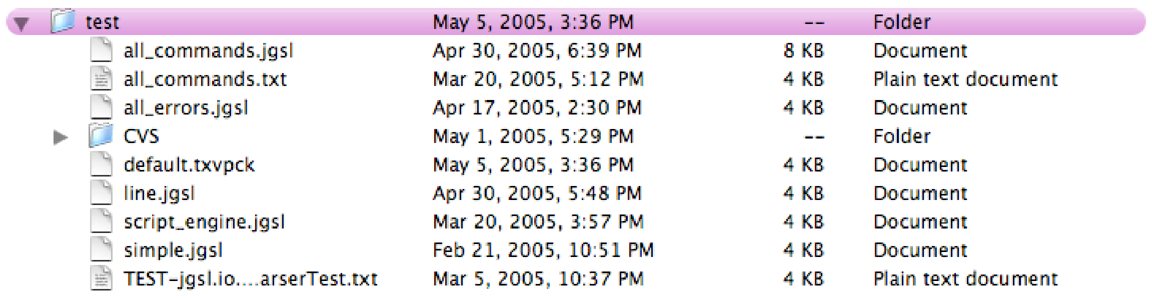


```
src                Today, 3:05 PM          --     Folder
  CVS              May 1, 2005, 6:46 AM    --     Folder
  default.txvpck   May 5, 2005, 3:29 PM    4 KB   Document
  jgsl             Today, 3:37 PM          --     Folder
    controller     Today, 3:38 PM          --     Folder
    CVS            Apr 30, 2005, 6:49 PM   --     Folder
    default.txvpck May 5, 2005, 3:29 PM    4 KB   Document
    io             May 5, 2005, 3:36 PM    --     Folder
    JGSL.java      May 5, 2005, 4:57 PM    4 KB   Java Source File
    model          May 5, 2005, 3:35 PM    --     Folder
    parser         May 5, 2005, 3:35 PM    --     Folder
    prototype      May 5, 2005, 3:36 PM    --     Folder
    resources      May 5, 2005, 3:36 PM    --     Folder
    util           May 5, 2005, 3:36 PM    --     Folder
    view           Today, 3:38 PM          --     Folder
```

Figure 14 SRC Directory Structure

4.3 Test Scripts

The test scripts in the `test` directory are the JGSL scripts written to test the JGSL application from the top down in black box fashion. The `all_commands.jgsl` script contains the set of commands that the JGSL is capable of executing.

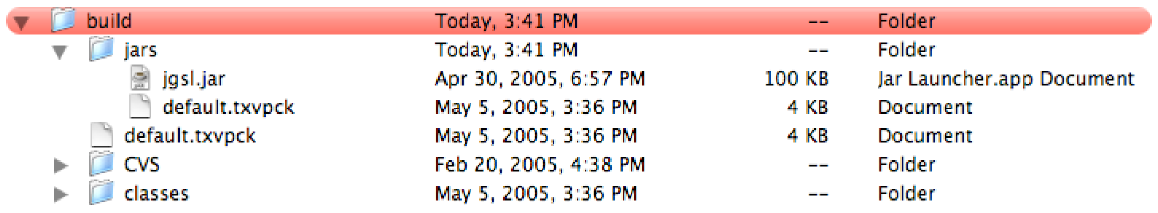


▼ test	May 5, 2005, 3:36 PM	--	Folder
all_commands.jgsl	Apr 30, 2005, 6:39 PM	8 KB	Document
all_commands.txt	Mar 20, 2005, 5:12 PM	4 KB	Plain text document
all_errors.jgsl	Apr 17, 2005, 2:30 PM	4 KB	Document
▶ CVS	May 1, 2005, 5:29 PM	--	Folder
default.txvpck	May 5, 2005, 3:36 PM	4 KB	Document
line.jgsl	Apr 30, 2005, 5:48 PM	4 KB	Document
script_engine.jgsl	Mar 20, 2005, 3:57 PM	4 KB	Document
simple.jgsl	Feb 21, 2005, 10:51 PM	4 KB	Document
TEST-jgsl.io....arserTest.txt	Mar 5, 2005, 10:37 PM	4 KB	Plain text document

Figure 15 TEST Directory Structure

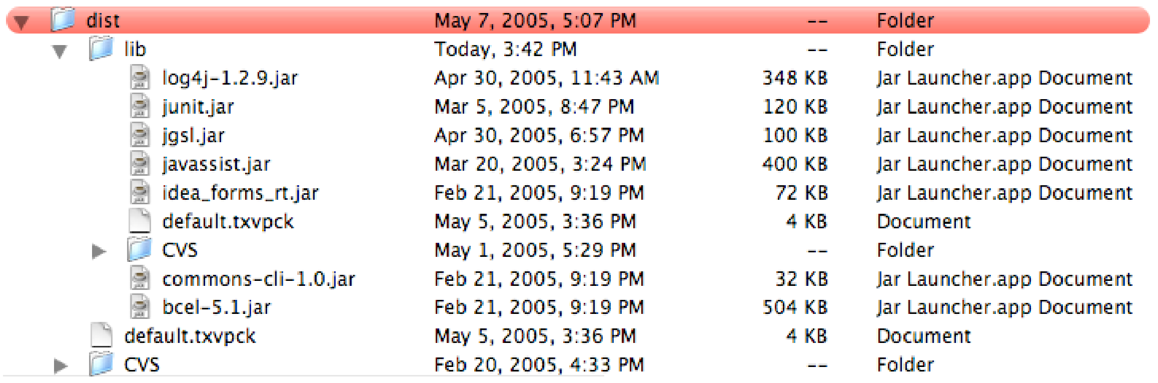
4.4 Build Products

There are two levels of build products for the JGSL. The first is generated during day-to-day development activities such as compiling source code and executing unit tests. This is the "`build`" directory shown in Figure 16. The second directory `dist` shown in Figure 17 is generated when the JGSL is ready for distribution and it contains the JGSL binary distribution along with the required runtime libraries.



▼ build	Today, 3:41 PM	--	Folder
▼ jars	Today, 3:41 PM	--	Folder
jgsl.jar	Apr 30, 2005, 6:57 PM	100 KB	Jar Launcher.app Document
default.txvpck	May 5, 2005, 3:36 PM	4 KB	Document
default.txvpck	May 5, 2005, 3:36 PM	4 KB	Document
▶ CVS	Feb 20, 2005, 4:38 PM	--	Folder
▶ classes	May 5, 2005, 3:36 PM	--	Folder

Figure 16 BUILD Directory Structure



▼ dist	May 7, 2005, 5:07 PM	--	Folder
▼ lib	Today, 3:42 PM	--	Folder
log4j-1.2.9.jar	Apr 30, 2005, 11:43 AM	348 KB	Jar Launcher.app Document
junit.jar	Mar 5, 2005, 8:47 PM	120 KB	Jar Launcher.app Document
jgsl.jar	Apr 30, 2005, 6:57 PM	100 KB	Jar Launcher.app Document
javassist.jar	Mar 20, 2005, 3:24 PM	400 KB	Jar Launcher.app Document
idea_forms_rt.jar	Feb 21, 2005, 9:19 PM	72 KB	Jar Launcher.app Document
default.txvpck	May 5, 2005, 3:36 PM	4 KB	Document
▶ CVS	May 1, 2005, 5:29 PM	--	Folder
commons-cli-1.0.jar	Feb 21, 2005, 9:19 PM	32 KB	Jar Launcher.app Document
bcel-5.1.jar	Feb 21, 2005, 9:19 PM	504 KB	Jar Launcher.app Document
default.txvpck	May 5, 2005, 3:36 PM	4 KB	Document
▶ CVS	Feb 20, 2005, 4:33 PM	--	Folder

Figure 17 DIST Directory Structure

Contained in Appendix E is the complete source code listing for the project. Following that, in Appendix F, is the API level documentation generated from the source code.

5 Test and Integration

Test and integration consists of a test model, test cases and the deployment model. The Test model is a high-level view of the JUnit test classes that implement the white box test cases for the JGSL application. The test cases describe the black box test cases for the JGSL application. The deployment model describes the integration of the JGSL components into the binary distribution and runtime environment of the JGSL software release.

5.1 Test Model

The test model consists of automated and user testing. The automated testing is built from several layers of white box unit tests coded with the JUnit [JUNIT 2004] testing framework combined with black box testing using JGSL scripts. Human driven testing is done with the JGSL Interactive GUI and Command Console to ensure proper operation of GUI components and interfaces. Shown in Figure 18 below is the Test Model Diagram for the JGSL. The Test Model is dependent on the Use Case Model to determine what tests need to be developed and executed. Also shown in the Test Model Diagram are the TextModelClassDiagram and the TestModelComponentDiagram.

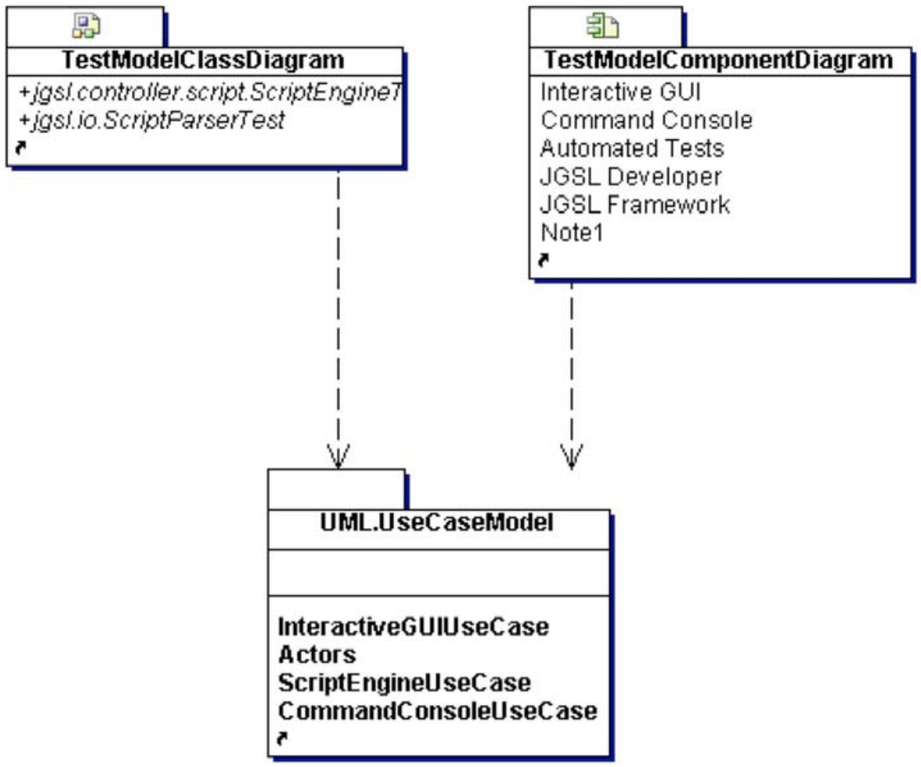


Figure 18 Test Model Diagram

The JUnit white box test classes are shown in Figure 19. The `ScriptParserTest` class contains the method `testParseScript` that takes as input a JGSL script to parse, and based on the success criteria, the JUnit framework will report success or failure of the test. This is a white box test. The criteria are coded into each test method. The `ScriptEngineTest` class contains methods to test the following script engine functions: The script engine proper, the script viewer, creating an image from a script, generating documentation from a script and generating a JAR from a script. The source code for these test classes is included in Appendix E. In Figure 20 the JUnit test classes are shown in relation to the rest of the Test Model (shaded in green).

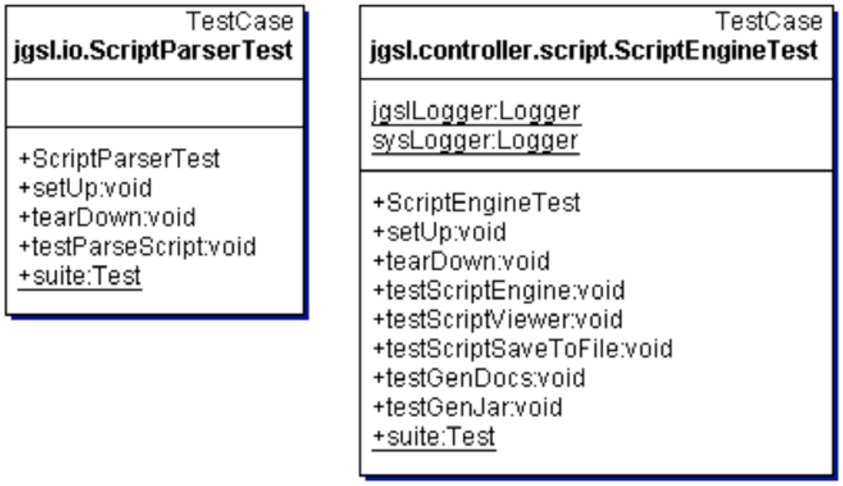


Figure 19 Test Model Class Diagram

The user driven tests for the Interactive GUI and Command Console are shown in Figure 20 (yellow shaded). The user driven tests are described in section 5.2. Also shown in Figure 20 is the component model for the Test Model. The JGSL Framework subsystem consists of the Script Engine, Script Parser, JGSL Viewer and JGSL Script components. This subsystem is the core of the JGSL runtime and contains the model and controller for the JGSL Framework. The Automated Test subsystem contains the JUnit white box test components. The Interactive GUI subsystem contains the view (`JGSLSwingView`) and the JGSL loader component (`JGSL`). The loader component contains the required Java Virtual Machine program entry point "`main`". The Command Console subsystem contains the JGSL loader component and uses console standard input and output streams to interact with the user. For the Command Console, the Script Engine component contains the controller logic for parsing and validating command line arguments and to execute the requested JGSL functions. The design of the Script Engine Test component in the Automated Tests subsystem is that it executes tests on the Script Engine component using command line arguments as input. Thus, the Script Engine Test component covers the Command Console subsystem test cases. Given that the JGSL loader component is also tested in the Interactive GUI subsystem, all tests for the Command Console subsystem are covered. A test case for the command console at the user level is also specified in section 5.2.5.

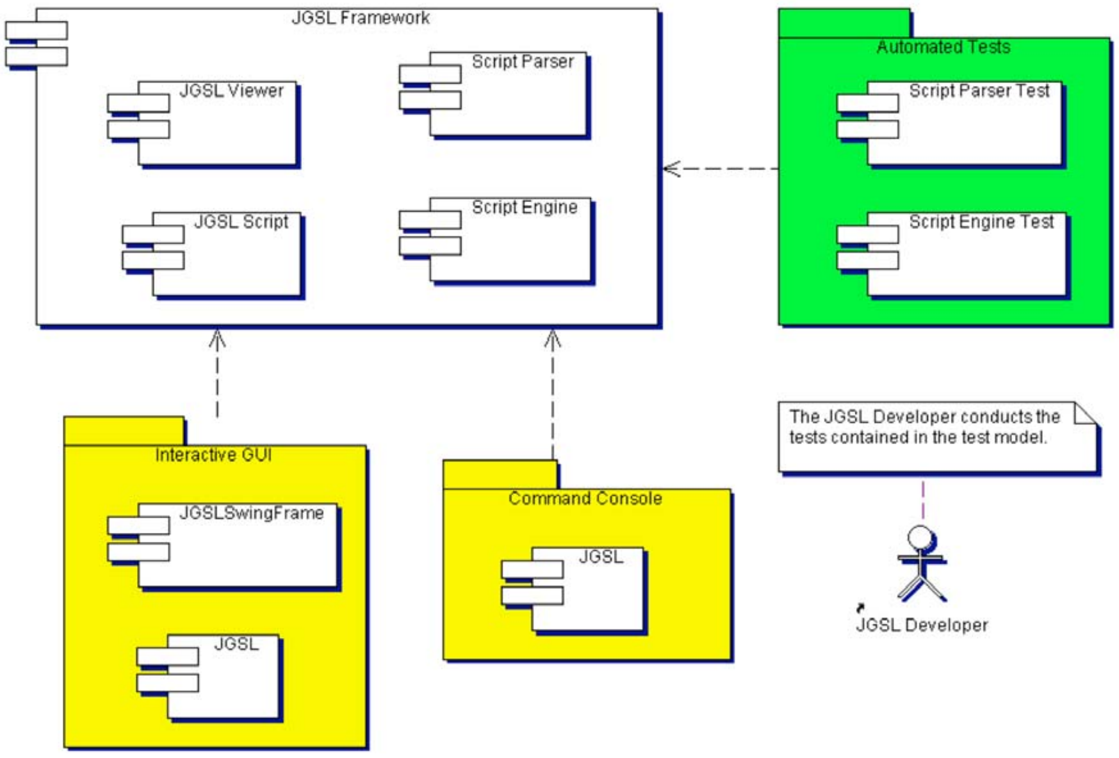


Figure 20 Test Model Component Diagram

5.2 Test Cases

Each of the User driven test cases are described in the sections that follow.

5.2.1 All Commands Test Case

Title	All Commands
Description	Tests all supported JGSL script commands.
Procedure	1. Open the JGSL Interactive GUI. 2. Load the “all_commands.jgsl” script file. 3. Press the “View” button. 4. Verify the output of the script is correct by visual inspection.
Success Criteria	Script output passes visual inspection.

5.2.2 Script Parser Test Case

Title	Script Parser
Description	Test the script parser data flow and processing.
Procedure	1. Open the JGSL Interactive GUI. 2. Load the “all_commands.jgsl” script file. 3. Press the “View” button. 4. Verify the status window does not display any errors or warnings.
Success Criteria	Status window displays script parse success.

5.2.3 Script Engine Test Case

Title	Script Engine
Description	Test the script engine controller data flow and processing.
Procedure	1. Open the JGSL Interactive GUI. 2. Load the "all_commands.jgsl" script file. 3. Press the "View" button. 4. Verify the status window does not display any errors or warnings and the JGSL viewer displays script output.
Success Criteria	Status window displays script parse success and viewer displays script output.

5.2.4 Interactive GUI Test Case

Title	Script Parser
Description	Non-automated test of the GUI by a human.
Procedure	1. Open the JGSL Interactive GUI. 2. Load the "all_commands.jgsl" script file. 3. Press the "View" button. 4. Verify the status window does not display any errors or warnings.
Success Criteria	Status window displays script parse success.
Continuation	5. Press the "Create JAR" button and select a location and file name. 6. Verify the status window does not display any errors or warnings. 7. Locate the JAR file and double-click on it. 8. Verify the proper JAR file execution by visual inspection.
Success Criteria	Visual inspection shows the proper output of the "all_commands.jgsl" script.
Continuation	9. Check the "Save script output" box. 10. Verify the status window does not display any errors or warnings. 11. Locate the JAR file and double-click on it. 12. Verify the proper JAR file execution by visual inspection.
Success Criteria	Visual inspection shows the proper output of the "all_commands.jgsl" script.
Continuation	13. Select the "Save" menu item. 14. Verify the status window contains file save messages.
Success Criteria	Locate the script file in the file system and verify contents with the content of the script editor.
Continuation	15. Select the "Save As" menu item. 16. Enter a new location and file name. 17. Verify the status window contains file save messages.
Success Criteria	Locate the script file in the file system and verify contents with the content of the script editor.
Continuation	18. Select the "New" menu item. 19. Verify the script editor contains the JGSL template script.
Success Criteria	Compare contents of script editor with the "template.jgsl" file.
Continuation	20. Select the "Help Topics" menu item. 21. Verify the script editor contains the JGSL online help script.
Success Criteria	Compare contents of script editor with the "online_help.jgsl" file.
Continuation	22. Select the "JGSL Tutorial" menu item. 23. Verify the script editor contains the JGSL tutorial script.
Success Criteria	Compare contents of script editor with the "jgsl_tutorial.jgsl" file.

Continuation	24. Select the "About" menu item. 25. Verify JGSL About dialog is displayed. 26. Click Okay to close.
Success Criteria	JGSL is displayed and closed.

5.2.5 Command Console Test Case

Title	Script Parser
Description	Non-automated test of the Command Console by a human.
Procedure	1. Open a command line terminal window. 2. Verify JAVA_HOME environment variable is set to the location of a Java 1.5 runtime. 3. Follow the operating instructions in section 6.2.
Success Criteria	All command line options function as expected.

5.2.6 JGSL Viewer Test Case

Title	Script Parser
Description	Verify the viewer works on the supported platforms.
Procedure	1. Open the JGSL Interactive GUI. 2. Load the "all_commands.jgsl" script file. 3. Press the "View" button. 4. Verify the JGSL Viewer is shown with the correct output for the script.
Success Criteria	Visual inspection of the JGSL View output is correct.

5.2.7 JGSL Installation Test Case

Title	Script Parser
Description	Verify the installer works on the supported platforms.
Procedure	For each of the supported platforms (Windows, OS X and Linux) perform the following actions: 1. Start installation a. Online – Open a web browser and click on the install link. b. Offline – Download the installer for the platform from the website. 2. Execute installation a. Online – Follow the installation instructions in section 7.1. b. Offline – Run the installer program following the platform specific instructions in section 7.2. 3. Verify successful installation by starting the JGSL Interactive GUI.
Success Criteria	JGSL Interactive GUI is displayed after installation.

5.3 Requirements Traceability Matrix

In Table 3 below is the mapping between the test cases for the JGSL and the JGSL high-level requirements. Requirement R.8 is not present in the matrix, as it is not verified in this set of test cases. R.8 is verified in section 7.3

Table 3 Requirements Traceability Matrix

Test Case	R.1	R.2	R.3	R.4	R.5	R.6	R.7
All Commands	✗	✗	✗	✗	✗		
Script Parser	✗	✗					
Script Engine	✗	✗			✗		
Interactive GUI			✗				
Command Console				✗			
JGSL Viewer					✗		
JGSL Installation						✗	✗

5.4 Deployment Model

The deployment model integrates the JGSL Java Archive (JAR) file with the external libraries into a single distributable package. There are four deployment packages in the deployment model (see Figure 21), one for the online installer and for each of the target platforms. Each of the distribution packages contains the following JAR files: `jgsl.jar`, `commons-cli-1.0.jar`, `idea_forms_rt.jar`, `javassist.jar` and `log4j-1.2.9.jar`. Additional files are included based on the platform specific requirements. Each of the deployment packages is described in the following sections. Installation instructions for the JGSL are in section 7.

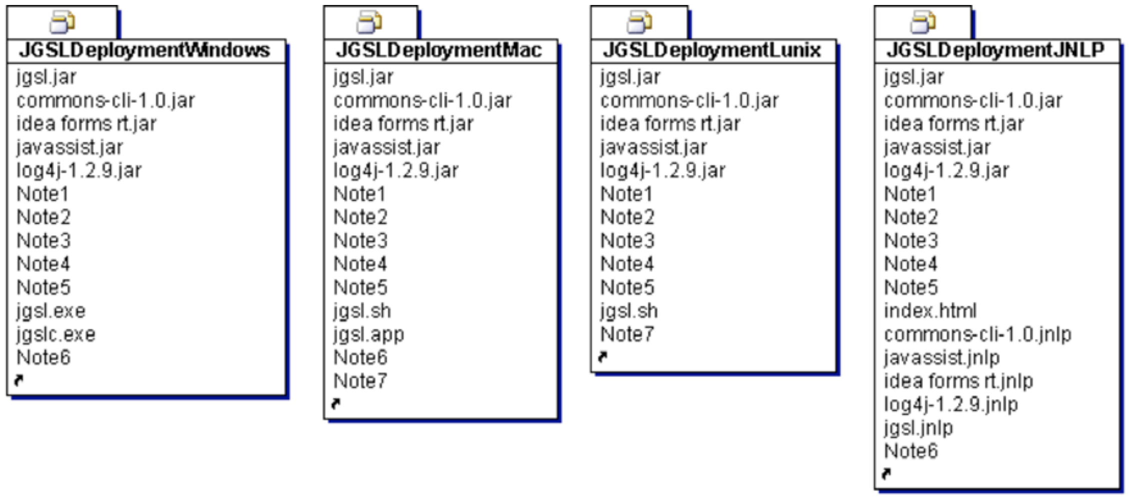


Figure 21 Deployment Model Diagram

5.4.1 Java Web Start (JNLP) Deployment

The Java Web Start (JWS) deployment uses the JNLP file format to describe the components for the Online installer. Each of the files shown in Figure 22 must be deployed on a Web server and be accessed via a standard Web browser. Installation is initiated by clicking on a link in the `index.html` file to the "`jgsl.jnlp`" file. This causes the JWS component of the Java runtime environment installed on the local workstation to begin execution of the directives in the "`jgsl.jnlp`". This in turn loads the associated `.jnlp` files that contain specifications for the included JAR files.

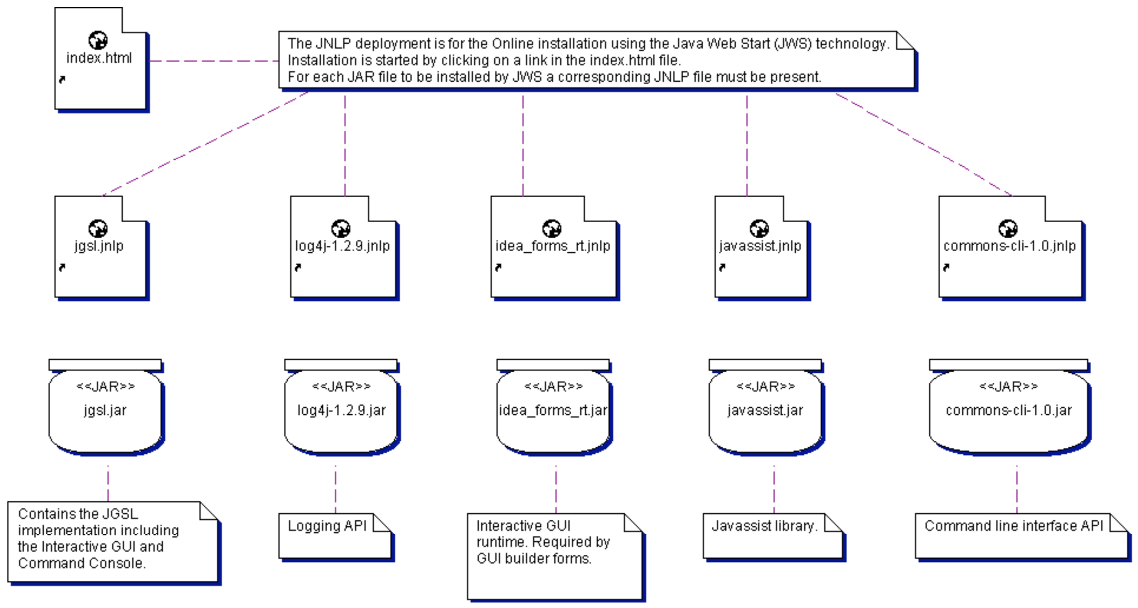

index.html
The JNLP deployment is for the Online installation using the Java Web Start (JWS) technology.
Installation is started by clicking on a link in the index.html file.
For each JAR file to be installed by JWS a corresponding JNLP file must be present.
jgsl.jnlp
log4j-1.2.9.jnlp
idea_forms_rt.jnlp
javassist.jnlp
commons-cli-1.0.jnlp
<<JAR>>
jgsl.jar
<<JAR>>
log4j-1.2.9.jar
<<JAR>>
idea_forms_rt.jar
<<JAR>>
javassist.jar
<<JAR>>
commons-cli-1.0.jar
Contains the JGSL implementation including the Interactive GUI and Command Console.
Logging API
Interactive GUI runtime. Required by GUI builder forms.
Javassist library.
Command line interface API

Figure 22 Java Web Start (JNLP) Deployment Diagram

5.4.2 Windows Deployment

Deployment on Windows adds two components: `jgsl.exe` and `jgslc.exe`. The "`jgsl.exe`" component is a native executable file for launching the JGSL Interactive GUI. The "`jgslc.exe`" component open the Command Console interface for the JGSL and must be run from a command prompt.

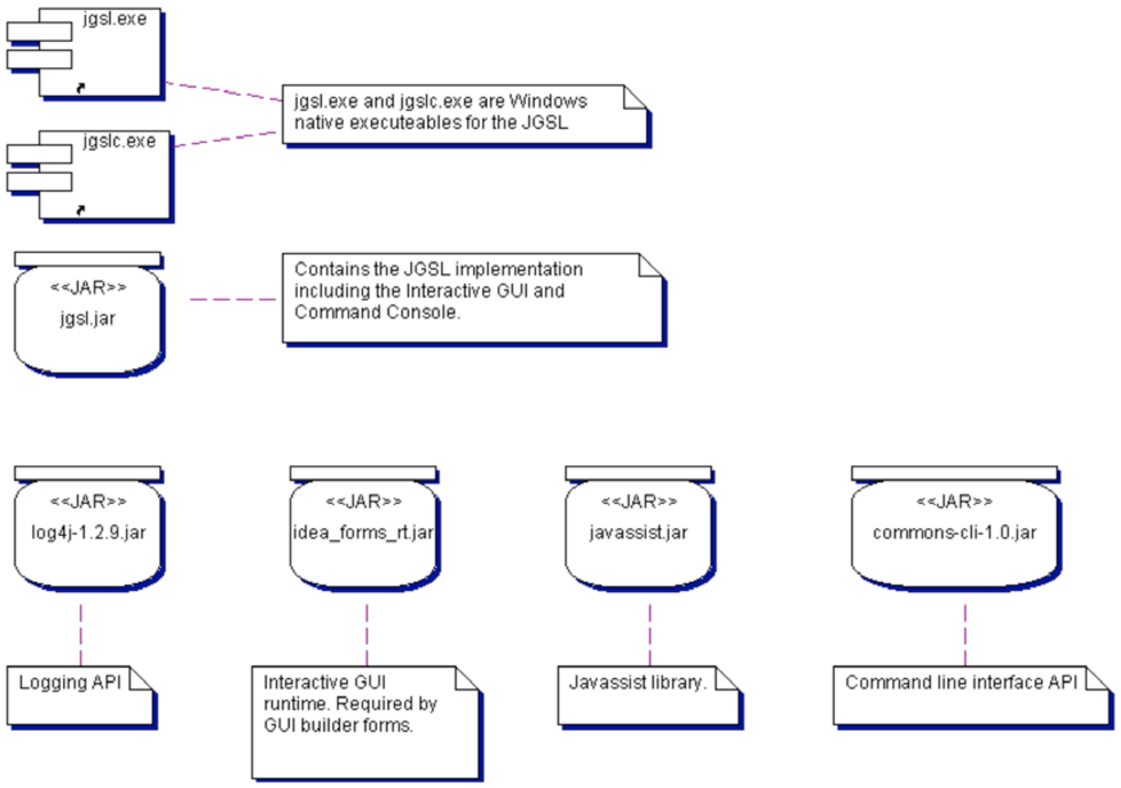

jgsl.exe
jgslc.exe
jgsl.exe and jgslc.exe are Windows native executeables for the JGSL
<<JAR>> jgsl.jar
Contains the JGSL implementation including the Interactive GUI and Command Console.
<<JAR>> log4j-1.2.9.jar
<<JAR>> idea_forms_rt.jar
<<JAR>> javassist.jar
<<JAR>> commons-cli-1.0.jar
Logging API
Interactive GUI runtime. Required by GUI builder forms.
Javassist library.
Command line interface API

Figure 23 Windows Deployment Diagram

5.4.3 OS X Deployment

OS X provides an package structure for applications that uses the "`.app`" extension. The "`jgsl.app`" is a directory that contains the required JGSL JAR files and a shell script for running the JGSL via an OS X Terminal.

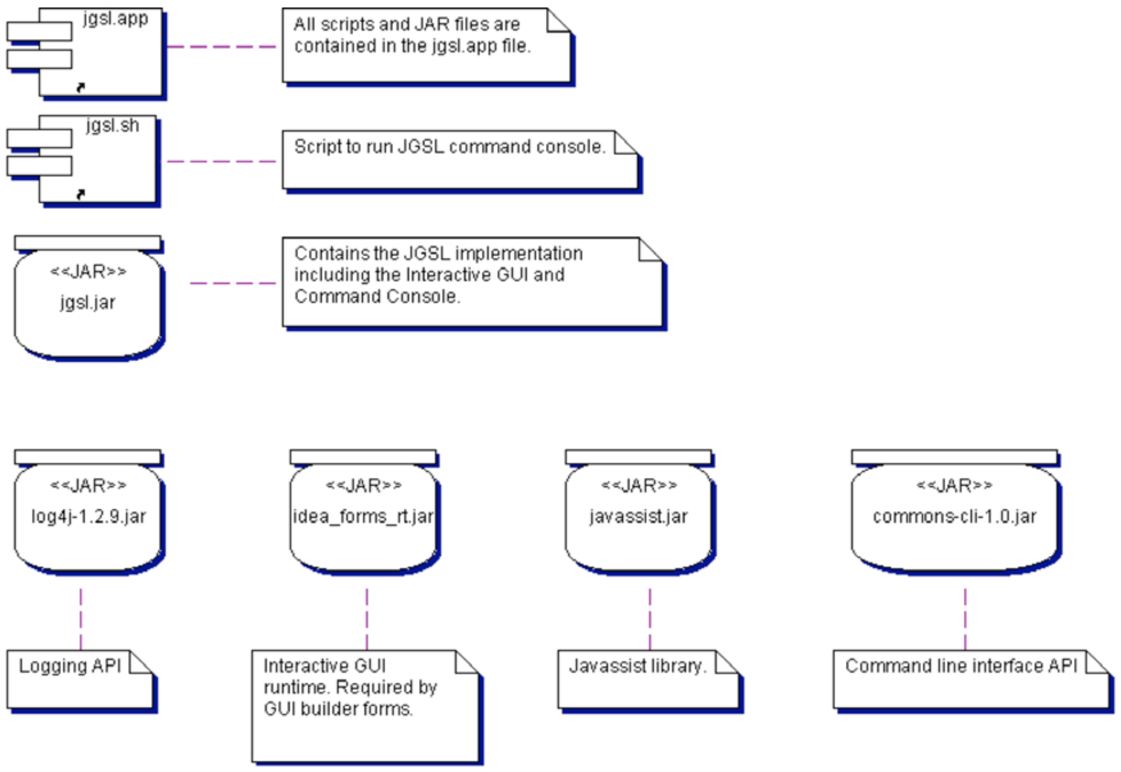

jgsl.app
All scripts and JAR files are contained in the jgsl.app file.
jgsl.sh
Script to run JGSL command console.
<<JAR>> jgsl.jar
Contains the JGSL implementation including the Interactive GUI and Command Console.
<<JAR>> log4j-1.2.9.jar
<<JAR>> idea_forms_rt.jar
<<JAR>> javassist.jar
<<JAR>> commons-cli-1.0.jar
Logging API
Interactive GUI runtime. Required by GUI builder forms.
Javassist library.
Command line interface API

Figure 24 Mac OS X Deployment Diagram

5.4.4 Linux/Unix Deployment

Linux/Deployment provides a shell script for launching the JGSL along with the JGSL JAR files.

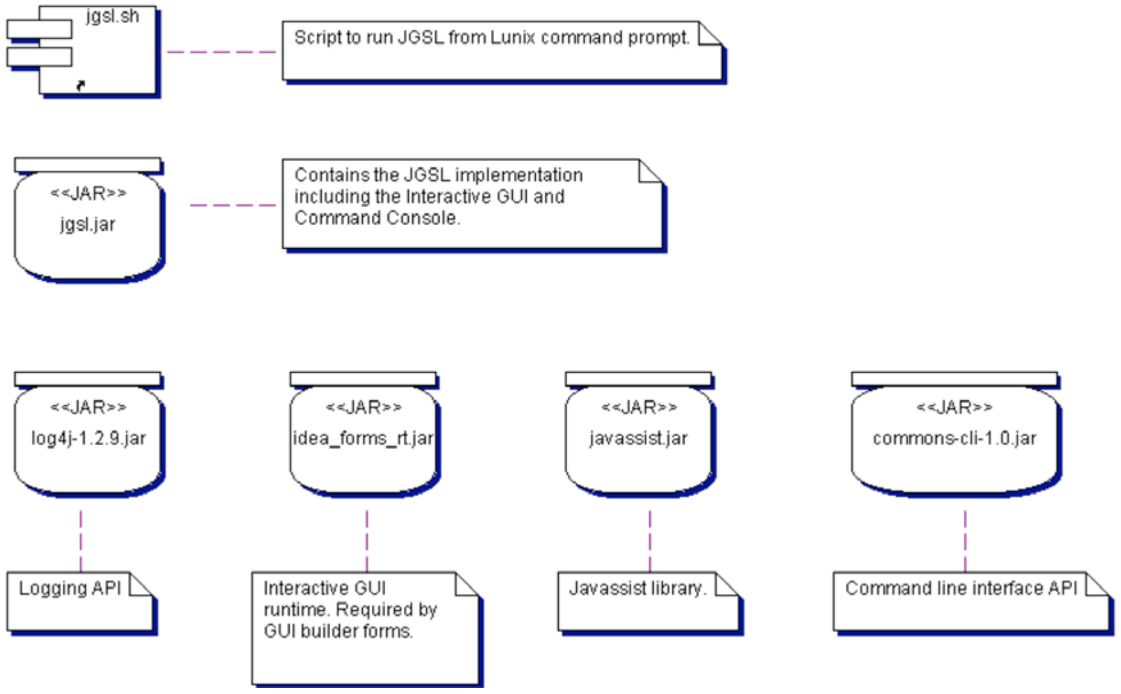

jgsl.sh
Script to run JGSL from Lunix command prompt.
<<JAR>> jgsl.jar
Contains the JGSL implementation including the Interactive GUI and Command Console.
<<JAR>> log4j-1.2.9.jar
<<JAR>> idea_forms_rt.jar
<<JAR>> javassist.jar
<<JAR>> commons-cli-1.0.jar
Logging API
Interactive GUI runtime. Required by GUI builder forms.
Javassist library.
Command line interface API

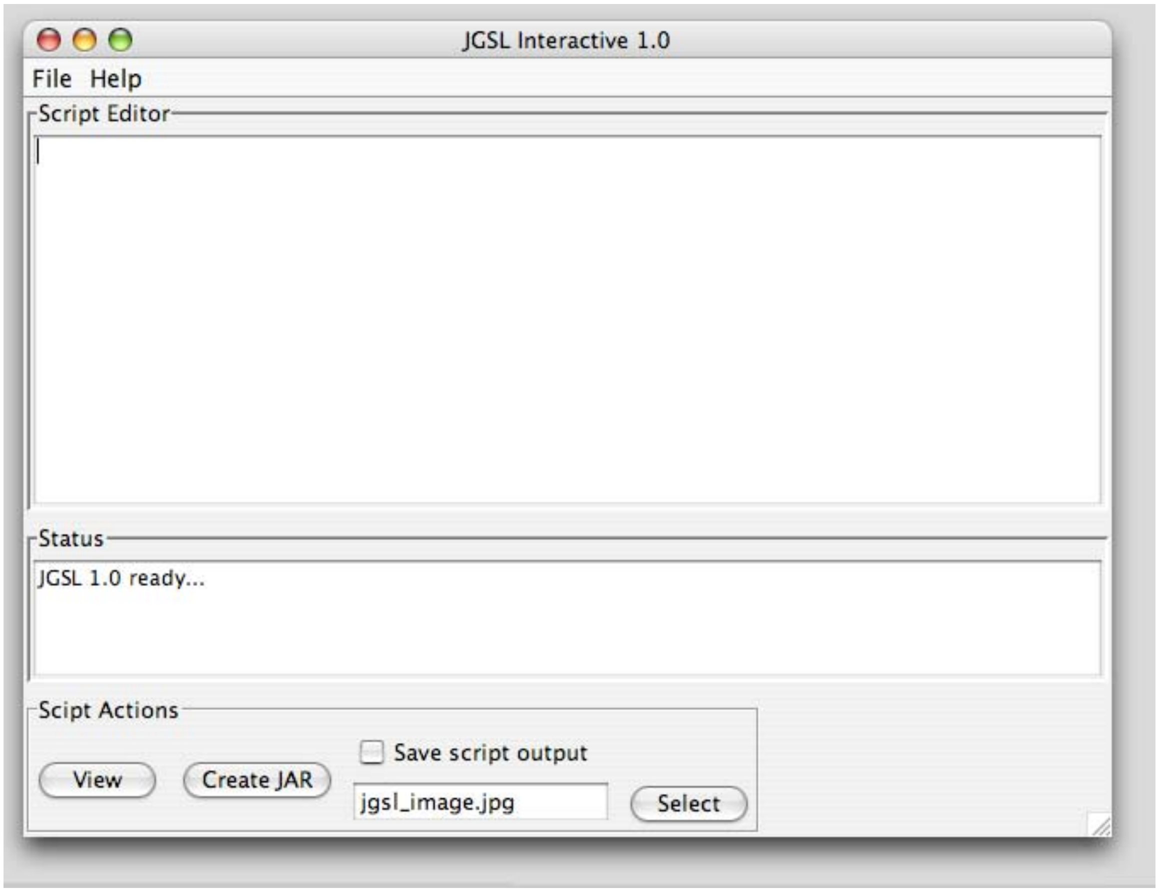
Figure 25 Linux/Unix Deployment Diagram

6 Operating Instructions

There are two modes of operating the JGSL: Interactive GUI and Command Console. The Interactive GUI provides an interface with which the user can edit and view JGSL scripts in a single environment. The Command Console interface is intended for advanced users who wish to execute the JGSL as command line process. Common to both operating modes is the creation of a JGSL working directory in the user's home directory for the platform. The directory is named "`.jgsl`" and contains the JGSL script compilation cache and a log file directory. The script compilation cache directory is where the JGSL writes the compiled form of the JGSL script. The `Logs` directory contains the user and system log files. The main benefit gained in creating a per user working directory is to allow the JGSL to operate properly on systems with multiple users. Under normal JGSL operations these directories should never need to be viewed or modified by the user. Details for operating in both modes are discussed in the following sections.

6.1 Interactive GUI

The JGSL GUI consists of a main application window with a menu system and toolbar. The main window is shown in Figure 26 below. The JGSL GUI can be opened via several different methods depending on the type of installation and the operating system. Information on starting the JGSL can be found in the section 7.



JGSL 1.0 ready...

jgsl_image.jpg

Figure 26 JGSL Interactive Main Window

To begin authoring a JGSL script, select the "New" item from the "File" menu.

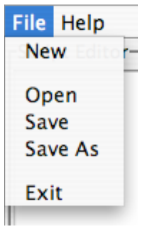
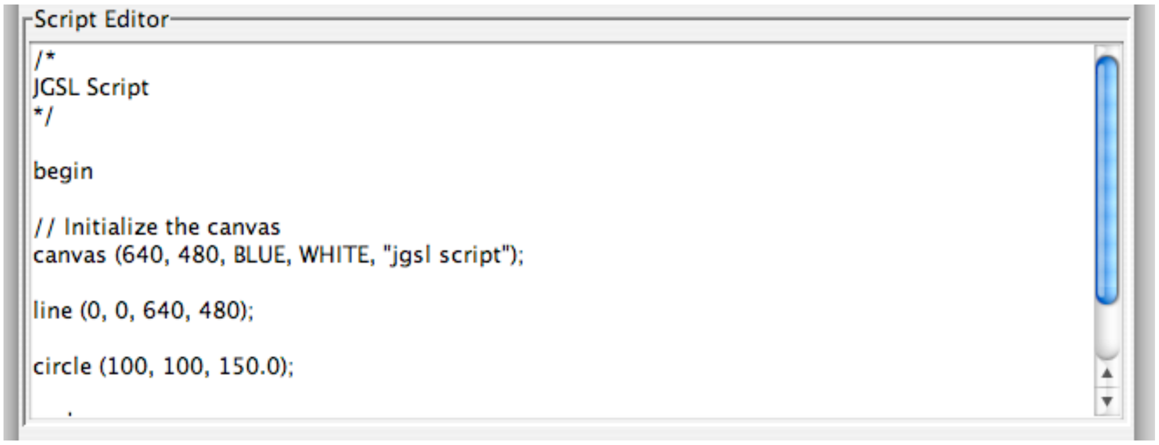


Figure 27 JGSL Interactive File Menu

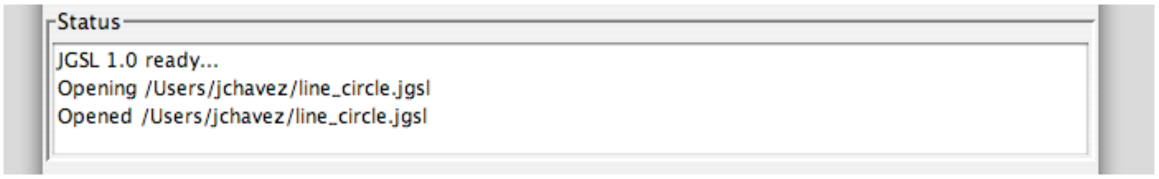
This will open the new JGSL script template in the "Script Editor". Here you can begin typing JGSL script commands. For example, type "`line (0, 0, 640, 480);`" after the line containing "canvas…", this will draw a line. Add another command "`circle (100, 100, 150.0);`" after the line you just inserted. Your script should look similar to the script shown below.



```
/*
JGSL Script
*/

begin

// Initialize the canvas
canvas (640, 480, BLUE, WHITE, "jgsl script");

line (0, 0, 640, 480);

circle (100, 100, 150.0);
```

Figure 28 JGSL Interactive Script Editor

Click on "Save" from the "File" menu and select a location and enter a file name for your new JGSL script. As you perform actions with the GUI, the "Status" window will be updated with processing messages.



```
JGSL 1.0 ready...
Opening /Users/jchavez/line_circle.jgsl
Opened /Users/jchavez/line_circle.jgsl
```

Figure 29 JGSL Interactive Status Window

The "Script Action" toolbar contains the actions that can be performed with your JGSL script. The "View" button will parse the script and, if there are no problems, the JGSL Viewer will display the script output. If there are problems the "Status" window will contain information regarding the location and type of problem.

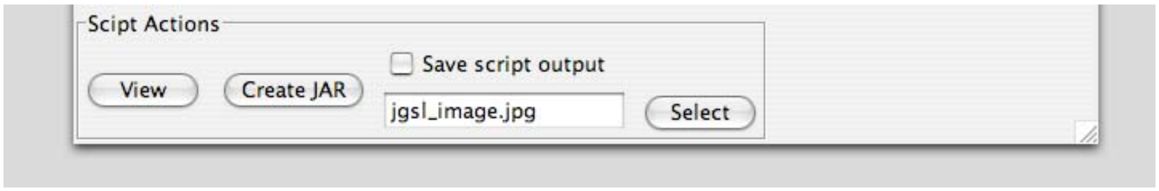


Figure 30 JGSL Interactive Script Actions Toolbar

The JGSL Viewer is a separate window that displays the results of executing a JGSL script. An example is shown in Figure 31.

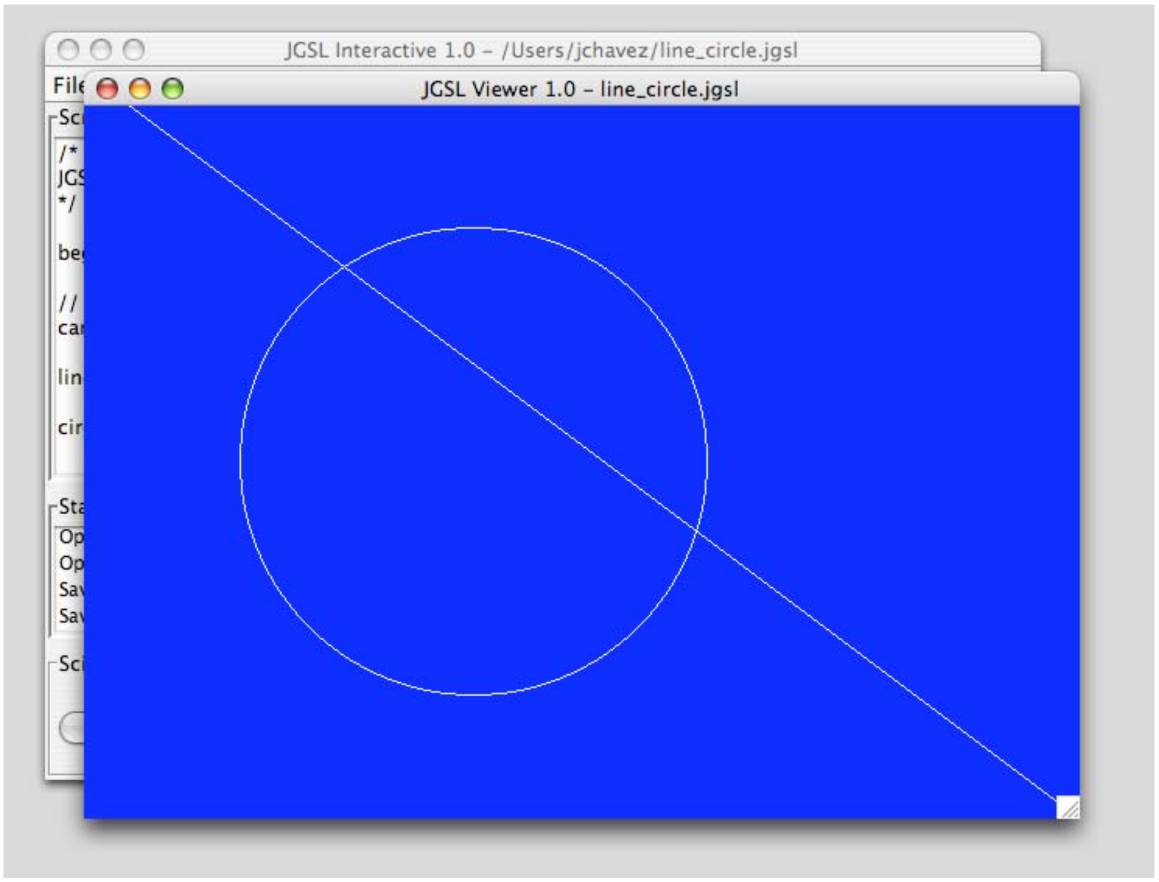

JGSL Interactive 1.0 – /Users/jchavez/line_circle.jgsl
JGSL Viewer 1.0 – line_circle.jgsl

Figure 31 JGSL Viewer

The "Create JAR" button will create an executable JAR file from your script which can be taken to any system with a Java 1.5 runtime environment and executed to view your script. By clicking this button you will be prompted for the location and name of the JAR file you would like to generate. Checking the "Save script output" box will cause the JGSL to generate an image file from your script when you click on "View". The supported image types are: JPG, PNG, GIF and BMP. Click on the "Select" button to specify the location and file name of the image file you would like to save.

Other actions you can perform on the "File" menu are "Open", "Save As" and "Exit". Clicking on "Open" will display a file open dialog from which you can select an existing JGSL script file to load into the "Script Editor". Clicking on "Save As" will allow you to save the current script under a different file name. Finally, the "Exit" menu will close the JGSL GUI.

The last item in the JGSL GUI is the help menu. The "Help Topics" item will load a JGSL script containing all of the supported JGSL commands and features. The "JGSL Tutorial" item will load a script containing instructions on how to write JGSL scripts. Finally, the "About" item will display information about the JGSL.

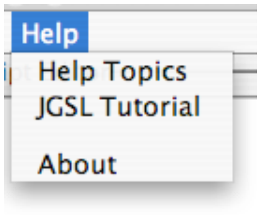


Figure 32 JGSL Interactive Help Menu

6.2 Command Console

The command console mode of the JGSL requires the use of a command line environment in order to invoke the JGSL. In order to use the command console, the offline installer for the target platform must be used. For each platform the command line executable is slightly different.

Table 4 JGSL Command Console Platform Executable

Platform	Executable	Environment
Windows XP	`jgslc.exe` located in `\Program Files\Perception Sofware\JGSL`	Running jgslc.exe with arguments will start the interactive GUI.
Linux	`jgsl.sh` located in the "`jgsl`" directory.	JAVA_HOME must be defined to the location of the Java 1.5 JRE.
Max OS X	`jgsl.sh` located in `JGSL.app/Contents/Resouces/Java`	JAVA_HOME must be defined to the location of the Java 1.5 JRE.

The command console supports a number of options that are shown in Table 5 below. Running the command console without any arguments will open the JGSL Interactive GUI. The "-d" option is used to process the input JGSL script for "DOC" style comments and produce an output file containing the script documentation. The "-p" option is used to parse a JGSL script and report the status. The "-e" option is used to parse and execute the input JGSL script and produce Java bytecode output. By adding the "-v" option to the "-e" option the input JGSL script can also be viewed using the JGSL viewer. Adding the "-f" option to the "-e" option will create an image file with the supplied file name including extension of the JGSL script output. The supported output extensions are: JPG, PNG, GIF and BMP. The "-j" option will create an executable JAR file that contains the JGSL script in compiled form. This JAR file can be run on any system that has Java 1.5 installed. For operating systems that support executable JAR files, double-clicking on it will run the JGSL JAR file. From the command line type "`java –jar <JGSL jar file>`" to execute the JGSL JAR file. The JGSL supports the generation of logging message from a JGSL script. The "-l" and "-s" options are used to control the user and JGSL system log levels respectively. The output of the log files is written to the directory "`.jgsl`" in the user's home directory. For Windows this directory is typically "`C:\Documents and`

`Settings\<user name>\.jgsl`” and for Unix based platforms this directory is “`~/.jgsl`”. Finally, the “-h” option displays the usage information shown in the table below.

Table 5 Command Console Option List

```
usage: jgsl.JGSL
 -d,--doc <jgsl script file> <script doc file>
      Generate script documentation
 -e,--exec <script file>
      Execute the script file
 -f,--filetype <Name of file>
      Save JGSL script output to file. File types are: jpg, png,
      gif, bmp
 -h,--help
      print this message
 -j,--jar <jgsl script file> <JAR file>
      Generate JAR file for script
 -l,--logLevel <user log level>
      Set user logging level to one of: LOG, DEBUG, ERROR,
      WARNING
 -p,--parse <script file>
      Parse the script file and print the results
 -s,--sysLogLevel <system log level>
      Set system logging level to one of: LOG, DEBUG, ERROR,
      WARNING
 -v,--view <Type of viewer, supports: swing>
      Parse, execute and view the script
```

7 Installation Instructions

Installation of the JGSL can be accomplished in several ways: Online installation using a standard Web browser, Offline installation using a standalone installer for the platforms the JGSL support and by building the JGSL from the source code. Online installation is provided via Java Web Start (JWS) through a Web browser running at the user's computer. JWS technology is the simplest method by which a user can install the software because it will provide instructions to install the required Java Runtime Environment if not present on the user's workstation. Offline installation is provided via standard installation method for the target platform.

Table 6 Offline Installers

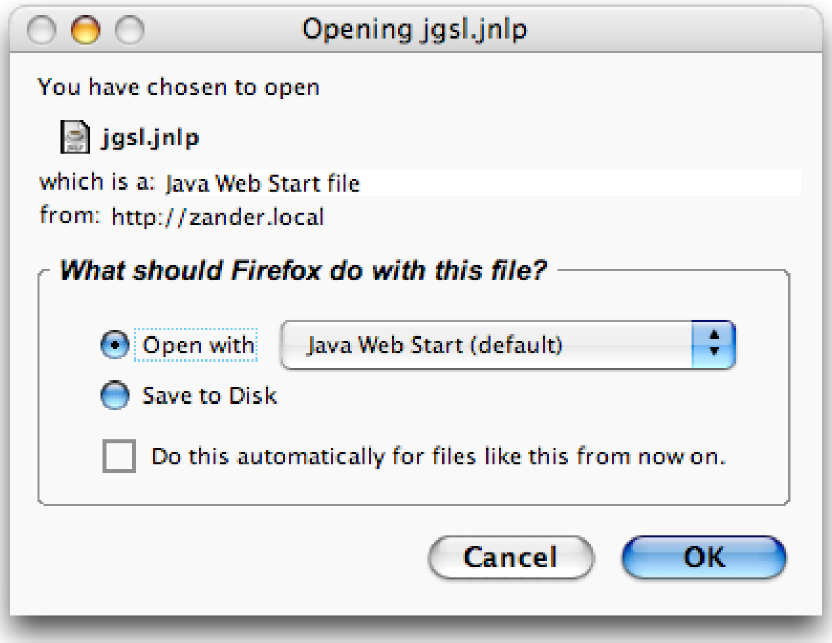
Platform	Method	Notes
Windows XP	Microsoft Windows Installer	Standard installation.
Apple OS X	Application Package	JGSL packaged in DMG file.
Unix	Shell Script	Shell script installer for Linux. All other Unix variants can use Generic script.

The final method is to use CVS to check out the source distribution from the JGSL CVS server and use Ant to build the JGSL from the source code. The details for each installation type are discussed in the remainder of this section.

7.1 Online Installation

Online installation instructions:

1. Open Web browser and navigate to `http://download.perceptionsoft.com.`
2. Click on the "Launch the JGSL" link. The opening or save download dialog for your browser will appear. Select "Open with Java Web Start" and click on OK to continue.


Opening jgsl.jnlp

You have chosen to open

jgsl.jnlp

which is a: Java Web Start file

from: http://zander.local

What should Firefox do with this file?

[✓ Open with] Java Web Start (default)

Save to Disk

Do this automatically for files like this from now on.

Figure 33 Online Installer Opening Dialog

3. Java Web Start will start the JGSL and display the window below while the software is downloaded and installed on the system.

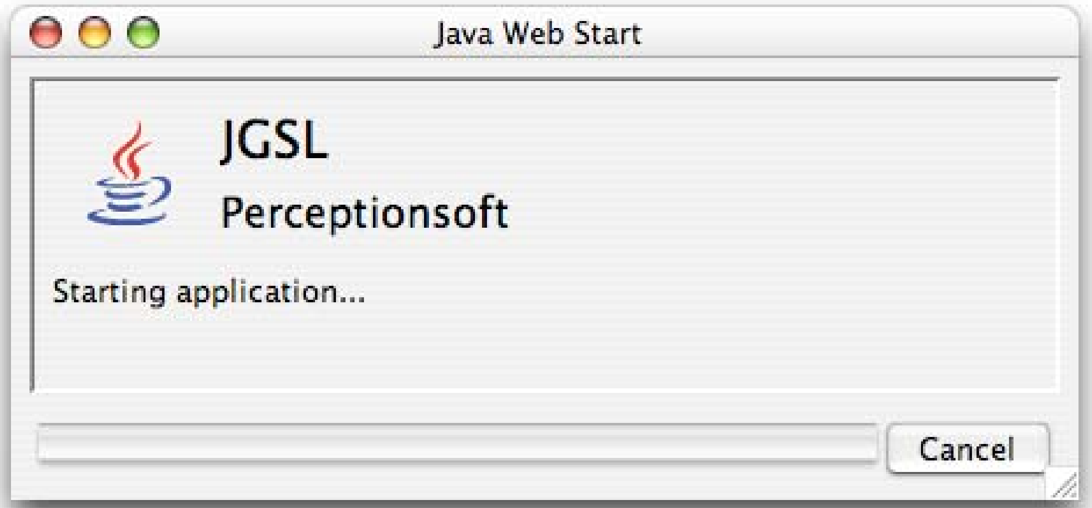
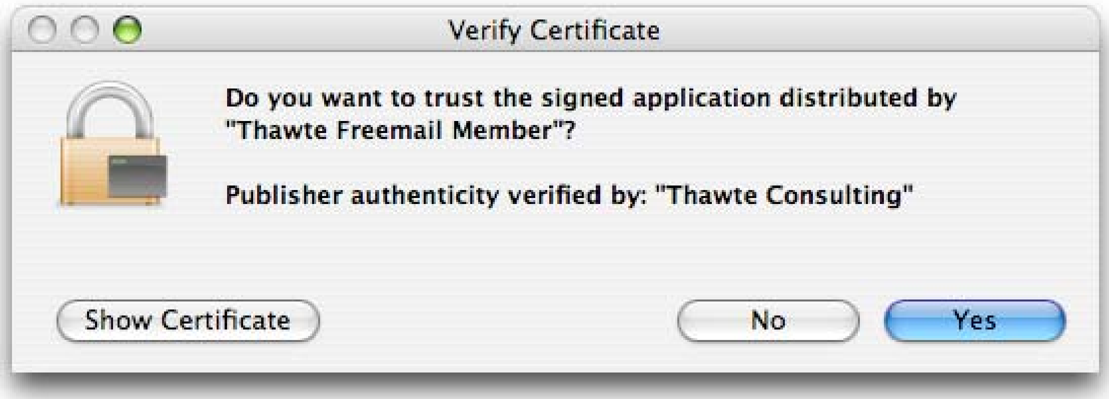


Figure 34 Online Installer Splash Screen

4. JGSL requires access to the workstations local file system for the following features: Script authoring, script rendering and script execution. One security restriction placed on JWS applications is that special permission is required by the user to access the file system. The dialog below asks for this permission; click “Yes” to continue. NOTE: This is a free certificate provided by Thawte Consulting that has been signed for use by the JGSL for JWS installations.



Verify Certificate

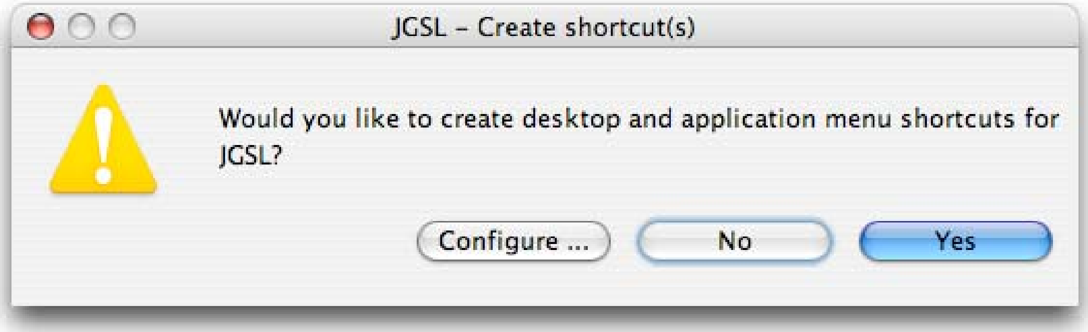
Do you want to trust the signed application distributed by "Thawte Freemail Member"?

Publisher authenticity verified by: "Thawte Consulting"

Figure 35 Online Installer Verify Certificate Dialog

5. An optional step is to have JWS create desktop and application menu shortcuts to the JGSL. This is optional; click “Yes” to create the shortcuts (NOTE: Shortcut creation is

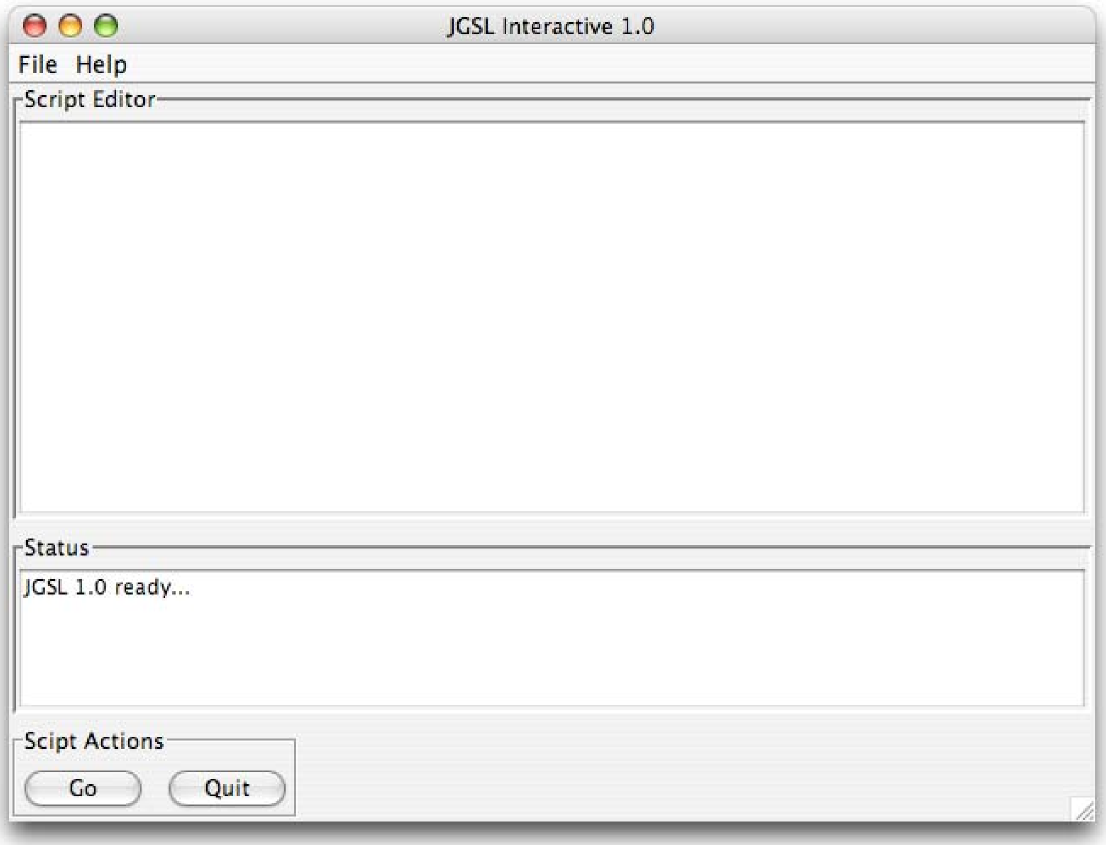
currently not working on OS X due to a bug in the JWS installer on that platform).


JGSL – Create shortcut(s)

Would you like to create desktop and application menu shortcuts for JGSL?

Figure 36 Online Installer Create Shortcut Dialog

6. Once the JWS installation is finished the "JGSL Interactive 1.0" window will be displayed and you can begin using the JGSL.



JGSL 1.0 ready...

Figure 37 JGSL Main Window

To run the JGSL again, open a Web browser and navigate to `http://download.perceptionsoft.com` and click on the link. Since the JGSL has already been downloaded once, JWS will run it immediately unless there is a new version. If there is a new version, JWS will download and install it automatically.

7.2 Offline Installation

The offline installation of the JGSL is tailored to each platform the JGSL is distributed on. For Windows an MSI package is supplied that provides the user with the familiar "wizard" installation process. An MSI package is in the format of the Microsoft Windows Installer and provides install, repair and uninstall capabilities. The MSI package will also add shortcuts to the JGSL executable and uninstall feature to the user's Programs menu. The JGSL executable is packaged as a native .EXE file that is standard on the Windows platform. There are two forms of the .EXE installed. The first is the "jgsl.exe" which is the JGSL Interactive GUI and the second is "jgslc.exe" which is the Command Console UI for the JGSL. For Apple OS X a disk image (DMG) file is provided that contains the JGSL packaged as an OS X ".app" file. The ".app" file contains the complete JGSL runtime including the Interactive GUI and a shell script to run the JGSL from a OS X terminal (or xterm). For Linux the distribution is provided as a ".zip" file that can be unpacked by the user via the command line utility "unzip". A shell script is provided to run the JGSL in the Interactive GUI mode or Command Console mode. The remainder of this section is devoted to the installation procedure for each platform.

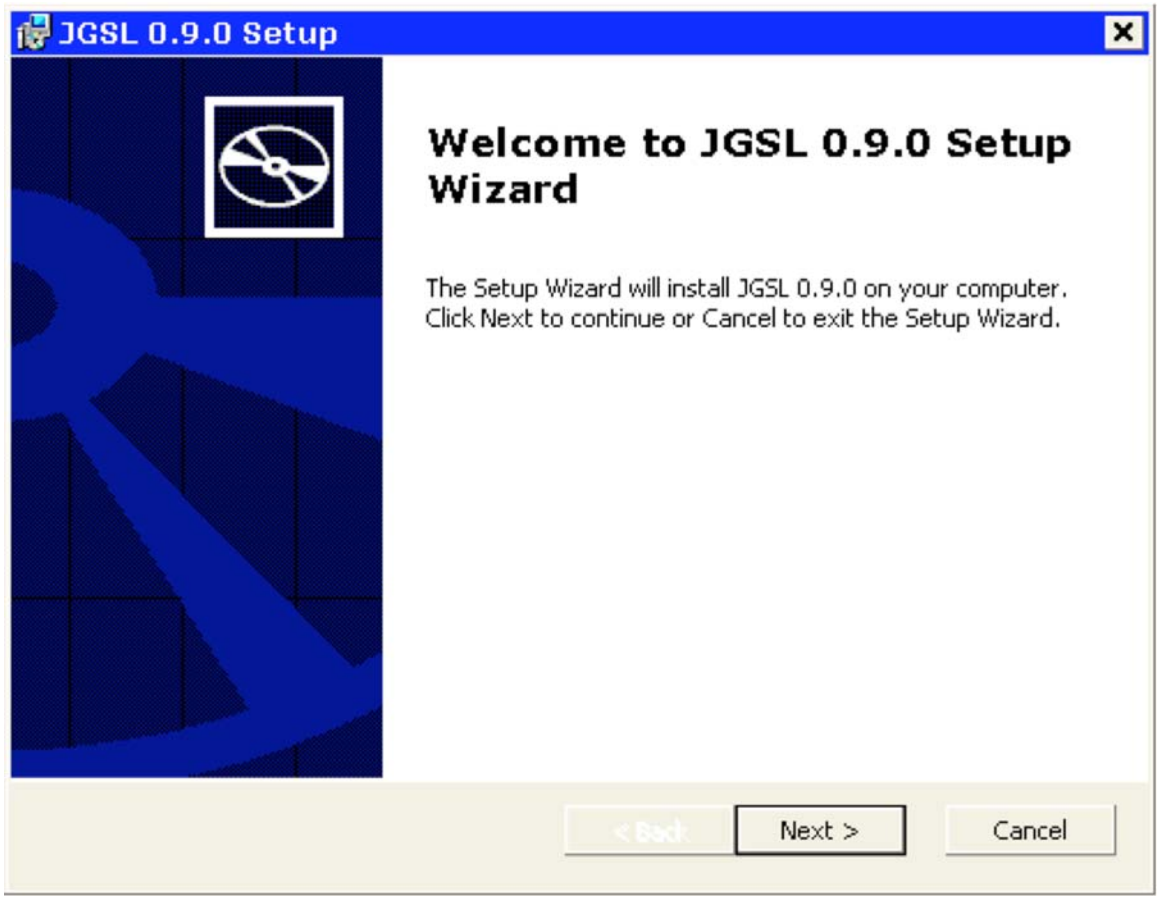
7.2.1 Windows

System Requirements:

- Windows XP
- Java Runtime Environment 1.5 - http://www.java.com/en/

To install the JGSL on Windows:

1. Download the JGSL Windows Installer file from `https://jgsl.dev.java.net/files/documents/2295/15141/jgsl-1.0.msi`
2. Run the JGSL Windows Installer and follow the instructions provided by the installer. This will display the "Welcome to …" screen click next to continue.

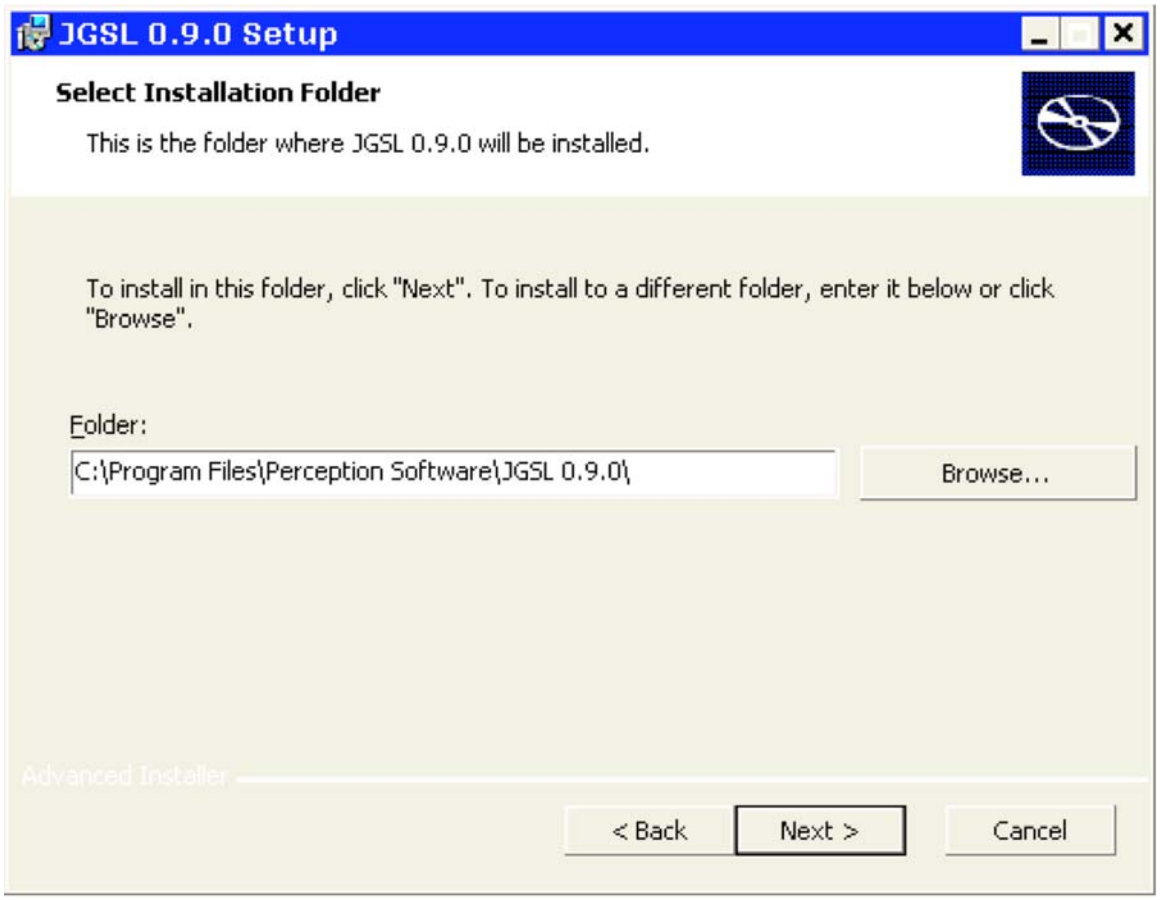


Welcome to JGSL 0.9.0 Setup Wizard

The Setup Wizard will install JGSL 0.9.0 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 38 Windows Installer Welcome Dialog

3. Select the destination directory for the JGSL installation or click “Next” to accept the default location.


JGSL 0.9.0 Setup

Select Installation Folder

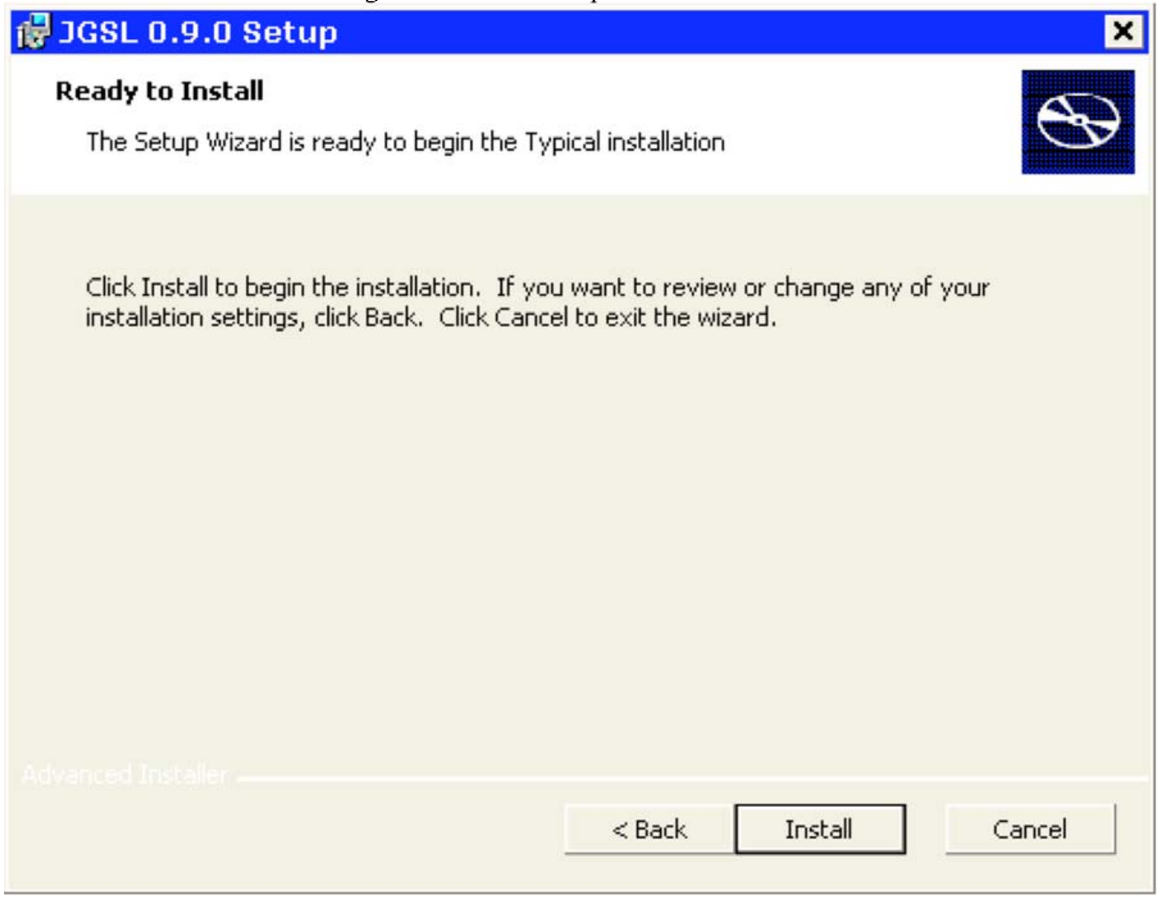
This is the folder where JGSL 0.9.0 will be installed.

To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

Folder: C:\Program Files\Perception Software\JGSL 0.9.0\

Figure 39 Windows Installer Installation Folder Dialog

4. Click the "Install" button to begin the installation process.



JGSL 0.9.0 Setup

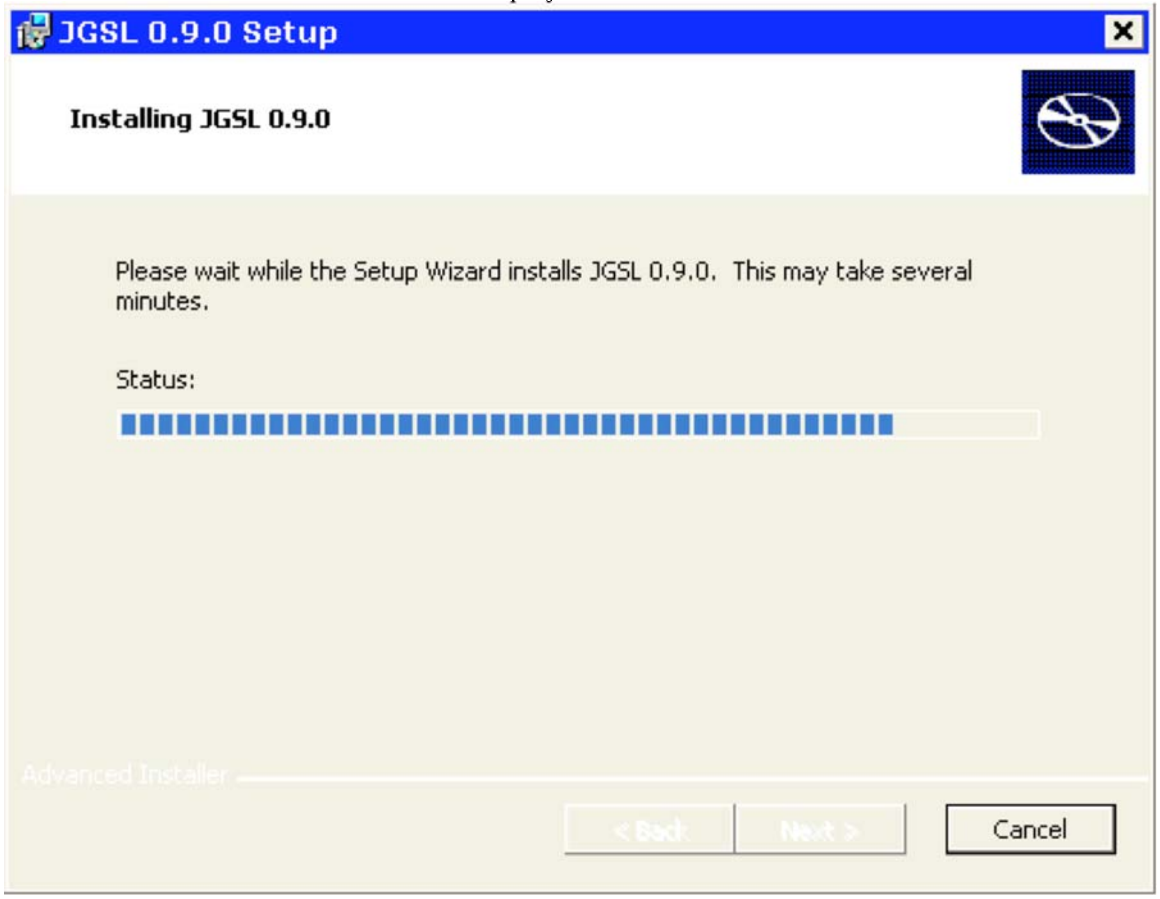
Ready to Install

The Setup Wizard is ready to begin the Typical installation

Click Install to begin the installation. If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Figure 40 Windows Installer Ready Dialog

5. Current status of the installation will be displayed.


JGSL 0.9.0 Setup

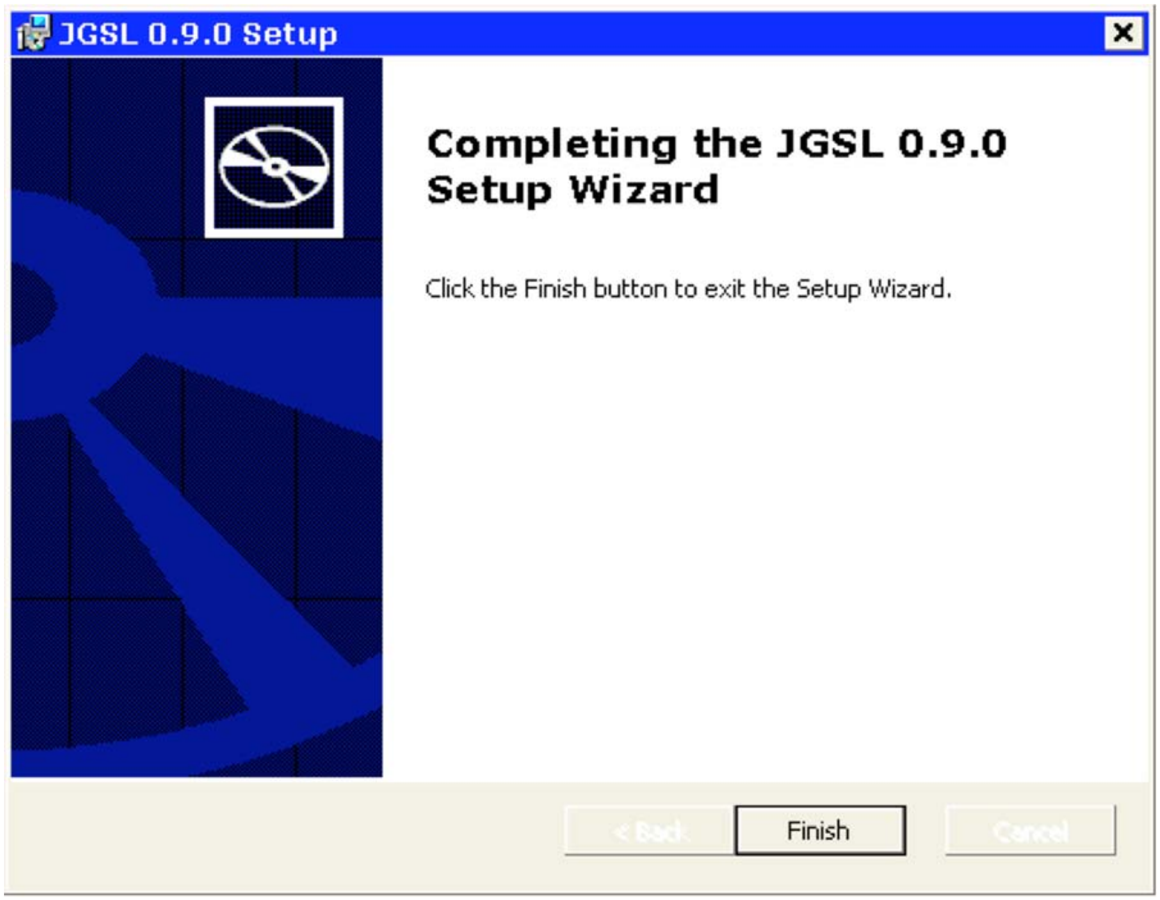
Installing JGSL 0.9.0

Please wait while the Setup Wizard installs JGSL 0.9.0. This may take several minutes.

Status:

Figure 41 Windows Installer Progress Dialog

6. When the installation process has been completed click “Finish” to close the JGSL installer.



Completing the JGSL 0.9.0 Setup Wizard

Click the Finish button to exit the Setup Wizard.

Figure 42 Windows Installer Completion Dialog

7. When installation is complete open the JGSL shortcut from the Programs menu to begin using the JGSL.

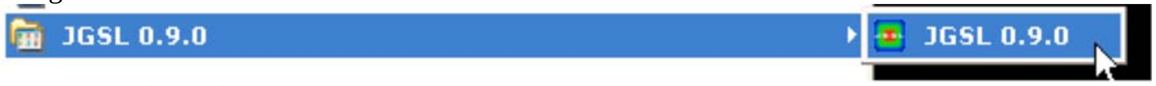


Figure 43 JGSL Windows Shortcut

7.2.2 Apple OS X

System Requirements:

- OS X 10.4
- Java 1.5 - http://www.apple.com/support/downloads/java2se50release1.html

To install the JGSL on Apple OS X:

1. Download the JGSL DMG file from `https://jgsl.dev.java.net/files/documents/2295/15113/jgsl-1.0.0.dmg.zip`.
2. Open the JGSL DMG file by double-clicking on it.
3. Drag the JGSL icon to your Applications folder.
4. Double-click the JGSL icon to begin.

7.2.3 Linux/Unix

System Requirements:

- Linux or Unix
- Java 1.5 - http://www.apple.com/support/downloads/java2se50release1.html

To install the JGSL on Linux/Unix:

1. Download the JGSL zip package file from `https://jgsl.dev.java.net/files/documents/2295/15112/jgsl-1.0.0.zip.`
2. Use the command line utility "unzip" to extract the contents of JGSL zip package to a directory of your choice.
3. Change into the "`jgsl`" directory and type "`jgsl.sh`" at the command prompt.

7.3 Source Distribution

To use JGSL from the source distribution there are several software prerequisites and a simple build process. The assumption for using the source distribution is that of a software developer level user. It is not necessary to use the source distribution to write JGSL scripts. Both the online and offline installers contain the complete JGSL script authoring tools.

7.3.1 Prerequisites

There are several software prerequisites listed below along with download URL:

- J2SE 5.0 SDK
  Windows, Linux, etc.: `http://java.sun.com/j2se/1.5.0/download.jsp`
  Apple OS X: `http://developer.apple.com/java/download/`
- Apache ANT 1.6+: `http://ant.apache.org/bindownload.cgi`
- CVS 1.12: `https://ccvs.cvshome.org/servlets/ProjectDocumentList?folderID=80&expandFolder=80&folderID=0`

7.3.2 CVS Repository

You must use CVS to checkout the source code from the repository. Before you can do a `cvs checkout`, you must set the `cvsroot` to indicate where the CVS repository for this project is located.

The command to set `cvsroot` and login is:

```
cvs -d :pserver:guest@cvs.dev.java.net:/cvs login
```

Then you can check out a working copy with:

```
cvs -d :pserver: guest @cvs.dev.java.net:/cvs checkout jgsl
```

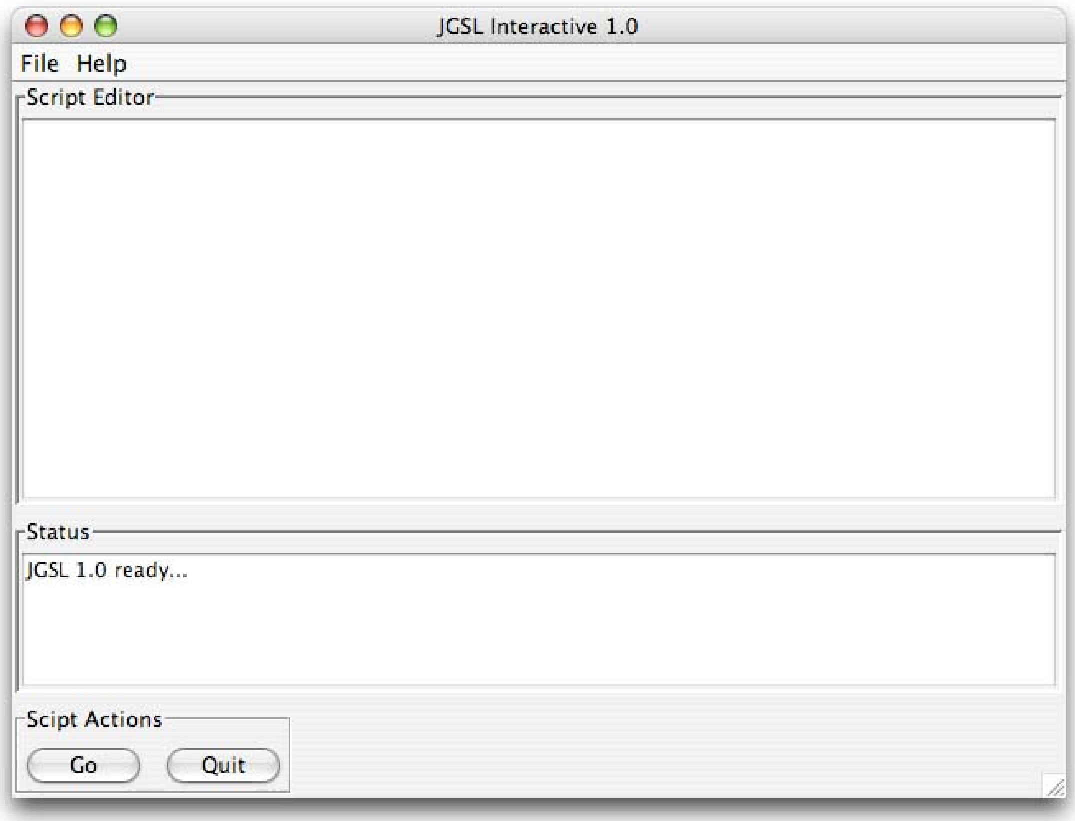
If this is the only project you are working on, you only need to set the `cvsroot` once. Thereafter, when you log in to this domain, the `cvs` repository for this project is assumed. If you are working on multiple projects, however, you must specify the `cvsroot` each time you log in to ensure that the `cvs` repository you are accessing is the right one.

7.3.3 Building JGSL

Building JGSL is done by running Ant from the command line. First, set your environment variables for JAVA_HOME and ANT_HOME. Second, in the JGSL directory containing the file "build.xml" type the following on the command line:

```
ant run
```

This will initialize the build environment, generate the parser, compile the Java source, create the JGSL JAR package, build the distribution and run the JGSL Interactive GUI. If all works as it should, the following window will appear:



JGSL 1.0 ready...

Figure 44 JGSL Main Window

If you see this window then the JGSL source distribution environment is working properly.

8 Recommendations for Enhancements

In version 1.0 the JGSL represents a vertical slice into the capabilities of the system. The overall goal is to deliver a framework upon which future enhancements can be added with little disruption to the overall application architecture. During the design and development of the JGSL several new capabilities were identified that would provide value to the language. Among those capabilities identified are:

- Visual script editor – draw the graphics objects and generate the JGSL script
- 3D support – scripting support for 3D objects
- Animation – add motion to JGSL scripting
- GUI Toolkits – support other GUI toolkits (SWT, etc.)

It is hoped that by releasing the JGSL to the open source community that other developers will contribute new capabilities to the JGSL.

9 Bibliography

[WARREN 2001] Warren, Peter. 2001. Teaching Programming Using Scripting Languages, The Journal of Computing in Small Colleges, v.17 n.2, p.205-216.

[BARRON 2000] Barron, D. W. 2000. The World of Scripting Languages. John Wiley & Sons, Inc., New York, NY.

[ASF 2004] The Apache Software Foundation. 2004. Apache Ant Manual, http://ant.apache.org/manual/index.html.

[JAVASSIST 2005] Javassist Project. 2005. Javaassist (Java Programming Assistant), http://www.jboss.org/products/javassist.html

[JAVACC 2004] JavaCC Project. 2004. Java Compiler Compiler, https://javacc.dev.java.net/

[JUNIT 2004] JUnit Project. 2004. JUnit Testing, http://www.junit.org/index.htm.

[COMMONS 2004] Apache Jakarta Project. 2004. Commons Component Library, http://jakarta.apache.org/commons/index.html.

[GEARY 1999] Geary, David. 1999. Graphic Java 2, Volume 2: Swing (3rd Edition). Prentice Hall PTR; 3rd edition.

[SELMAN 2002] Selman, Daniel. Java 3D Programming. Manning Publications Company.

[HARDY 1999] Hardy, Vincent J. Java 2D API Graphics. Prentice Hall.

[BECK 2000] Beck, Kent, Andres, Cynthia. 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley.

[AMBLER 2004] Ambler, Scott. 2004. AM Throughout the XP Lifecycle, http://www.agilemodeling.com/essays/agileModelingXPLifecycle.htm.

[FOWLER 1999] Fowler, Martin. 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.

[FOWLER 2002] Fowler, Martin. 2004. Patterns of Enterprise Application Architecture. Addison-Wesley Professional.

10 Appendices

A. Project Schedule

B. JGSL Language Specification

C. JGSL Grammar

D. JGSL Build File

E. Source Code

F. JGSL Java API Documentation

G. JGSL Tutorial