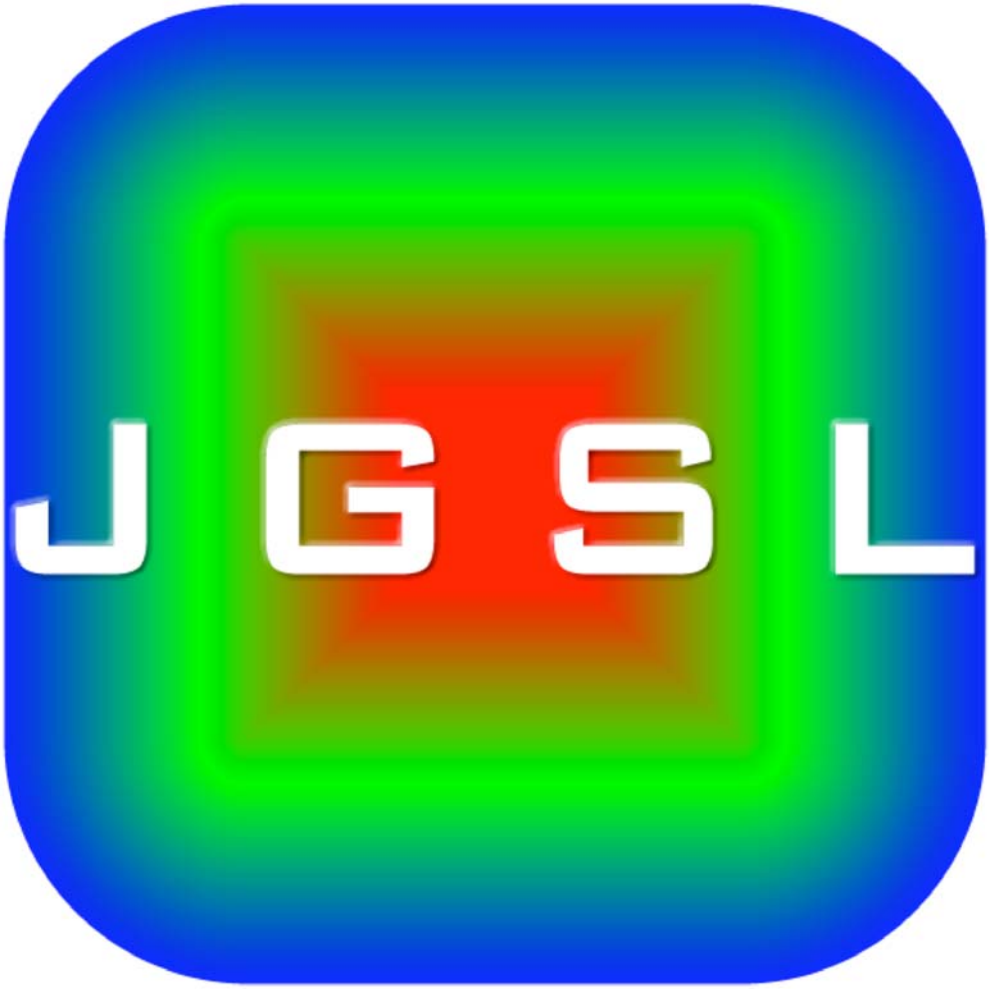
# JGSL

## Language Specification



# Table of Contents



# Index of Tables

# Document Version History

**Table 1 Document Version History**

| Date | Version | Modification | Author |
|---|---|---|---|
| 1/31/2005 | 0.1 | Document creation | J. Chavez |
| 2/1/2005 | 0.2 | Introduction | J. Chavez |
| 2/3/2005 | 0.3 | Conceptual design | J. Chavez |
| 2/4/2005 | 0.4 | Script file format | J. Chavez |
| 2/21/2005 | 0.5 | General cleanup. Feedback from grammar design | |

The JGSL project is maintained on the Internet at: http://jgsl.dev.java.net.

# 1 Introduction

This document contains the JGSL language specification and design and is intended to be a living specification document. The intention of this document is to describe the JGSL in terms of the core graphics concepts that will be made possible in the implementation. This not a formal specification in the same sense as say the Java language specification but is more focused on capturing the core constructs of the JGSL. The Conceptual Design section describes the overall structure of JGSL commands and syntax. However, it does not contain the complete list of commands and constructs that will be available in the JGSL. The Script File Format section describes the layout of a JGSL script file. This does not imply that disk based files will be the only method of input to the JGSL. It is more intended as a concrete representation of the Conceptual Design.

# 2 JGSL Language

## *2.1 Conceptual Design*

In the conceptual design of the JGSL the basic command structure is defined and the structure of the JGSL script file is detailed.

### 2.1.1 Commands

In the simplest form of the JGSL an expression will consist of a *command name*, zero or more comma delimited *command attributes* and zero or more parenthesized comma delimited *parameters*. A command has the following format:

command name: command attributes (parameters);

The purpose of the *command name* is to invoke the intended command action in the context of the script. A command may execute in several contexts: line scope or script scope or global scope. Line scope implies that the command will execute in a single script line and is completely self-contained. A command executed at script scope will be allowed to take information from the current script and leave information behind upon completion. Finally, commands with global scope have the ability to exist across multiple script invocations. The intention of command attributes to affect the state of the command

prior to execution. The command can contain attributes that can be retained from one execution to the next which are passed by value. This implies that a command is an instance of a command template (for lack of a better term). On the other hand parameters are intended for current execution of the command and are also passed by value. Thus a command is allowed to modify its internal state during execution and retain that state for subsequent invocations. A command cannot have a return value. A command will either succeed or fail. In the case of success the next script expression will be executed until the end of the script is encountered. In the case of failure the script will terminate and an error condition will be raised in the JGSL. For commands that do not have attributes or parameters the ":" and "()" are not required.

### 2.1.2 Attributes and Parameters

For all intents and purposes attributes and parameters are the same with exception to their usage in a command. For the remainder of this discussion I refer to both collectively as attributes. Attributes are instances of language constructs that can hold data. The intention of attributes is to supply data to commands for processing. At its core an attribute is an item of data that does not have a specific type. This will free the user from having to define types for the attributes that are used in the script. Some examples are attributes are:

GREEN
250
"A string of characters"
"500"
myAtttribute

In the examples above is a symbolic constant `GREEN`, numeric constant `250`, a string constant "`A string of characters`" and an attribute name `myAttribute`. All symbolic constants will be defined by the JGSL and cannot be extended by the user. The main justification for this is to keep the language simple. Numeric constants will have a defined range as limited by the underlying language Java. String constants will be delimited by double quotes (") and will be type converted when possible for the purposes of initialization, assignment and for logical and arithmetic evaluation (covered in Operators). An attribute name can be assigned any constant types as well as the value of another attribute. Type conversion will take place automatically.

### 2.1.3 Operators

#### 2.1.3.1 Initialization, Assignment & Equality

The equals "=" operator is used to initialize or assign a constant or the value of another attribute on the left hand side of the operator. Two attributes or an attribute and constant can be tested for equality using the "=" operator. The result of this test is TRU or FALSE.

#### 2.1.3.2 Logic

The logical operators for and, or and not are defined as: AND, OR and NOT. The AND and OR operators are binary and result in the symbolic constant TRUE or FALSE based on the result of the expression evaluation. The NOT operator is unary and inverts the value of an expression that results in TRUE or FALSE.

#### 2.1.3.3 Arithmetic

The following arithmetic operations are available:
Addition: +
This operator is overloaded to perform concatenation of two attribute values where possible.
Subtraction: –
Multiplication: *
Division: /
Modulus: %

### 2.1.4 Keywords

In Table 2 below is a preliminary list of reserved words in the JGSL:

**Table 2 JGSL Keywords (preliminary)**

| | | | |
|---|---|---|---|
| AND | OR | NOT | IF |
| THEN | ELSE | ELSEIF | BEGIN |
| END | CLEAR | CANVAS | DRAW |
| TEXT | RECTANGLE | SQUARE | CIRCLE |
| ELIPSE | POLYGON | LINE | GRADIENT |

| | | | |
|---|---|---|---|
| FILL | BORDER | COLOR | FORGROUND |
| BACKGROUND | REPEAT | LOOP | WAIT |
| LOG | DEBUG | ERROR | WARNING |
| WRITE | READ | JGSL | VERSION |
| DECLARE | DOC | TRUE | FALSE |

## *2.2 Script File Format*

In this section the overall structure of the script file format is provided. The use of text based files is one source of input to the JGSL. Other means of executing the JGSL are also planned.

### 2.2.1 Layout

**Table 3 Layout of JGSL Script File**

| |
|---|
| <comments \| documentation>* |
| <BEGIN> |
| <attributes \| commands \| comments \| documentation> |
| <END> |

### 2.2.2 Comments

Comments in JGSL are of two varieties: single or multi-line. For single line comments the following delimiters can be used: "//" or "#" with the end of line terminating the comment. Multi-line comments are delimited by the "/*" and "*/" pair.

### 2.2.3 Commands

The basic structure of a command is:

command name: command attributes (parameters);

This is also described in the Commands section.

### 2.2.4 Documentation

JGSL will provide a documentation facility that will allow for the embedding of documentation within a script.