SSOL Language Reference Manual

Madeleine Tipp	Jeevan Farias	Daniel Mesko
mrt2148	jtf2126	dpm2153
Manager	Language Guru	System Architect

October 15, 2018

Contents

1 Lexical Conventions **2**
- 1.1 Identifiers . . . 2
- 1.2 Keywords . . . 2
- 1.3 Literals . . . 3
- 1.4 Comment . . . 3
- 1.5 Punctuator . . . 3
- 1.6 Whitespace . . . 3

2 Types **3**
- 2.1 Primitives . . . 3
- 2.2 Complex Types . . . 3
  - 2.2.1 Point . . . 4
  - 2.2.2 Curve . . . 4
  - 2.2.3 Canvas . . . 4
- 2.3 Arrays . . . 4

3 Syntax **4**
- 3.1 Type Specifiers . . . 4
  - 3.1.1 Primitives . . . 4
  - 3.1.2 Complex Types . . . 4
- 3.2 Arrays . . . 5
  - 3.2.1 Declaration . . . 5
  - 3.2.2 Accessing . . . 5
- 3.3 Operators . . . 5
  - 3.3.1 Arithmetic . . . 5
  - 3.3.2 Comparison . . . 5
  - 3.3.3 Logical . . . 5
  - 3.3.4 Assignment . . . 5
  - 3.3.5 Canvas . . . 5
- 3.4 Statements . . . 6
  - 3.4.1 Sequencing . . . 6
  - 3.4.2 Control Flow . . . 6
  - 3.4.3 Loops . . . 6
- 3.5 Functions . . . 6
  - 3.5.1 Declaration . . . 6
  - 3.5.2 Function Calls . . . 7

4 Execution **7**
- 4.1 Scope . . . 7
- 4.2 main() . . . 7

5 Standard Library Functions **7**
5.1 draw() 7
5.2 printf() 7

6 Sample Code **7**

Introduction

SSOL is a programming language that allows users to create shapes algorithmically and render them in an SVG file. It features two built-in shape objects, Point and Curve, which can be used as building blocks to define more complex polygons or curved figures. The shapes are then added to a user-defined Canvas object, which abstractly represents the plane on which the shapes are to be drawn. The Canvas object can then be passed into the built-in *draw*() function to be rendered and stored as an SVG file. Without using *draw*(), SSOL functions as a minimal, general purpose programming language similar to C.

1 Lexical Conventions

1.1 Identifiers

Identifiers consist of one of more characters where the leading character is a uppercase or lowercase letter followed by a sequence uppercase/lowercase letters, digits and possibly underscores. Identifiers are primarily used in variable declaration.

1.2 Keywords

Keyword	Definition
if	initiates a typical if-else control flow statement
else	
while	initiates a while loop
for	initiates a for loop
break	ends a loop
continue	skips an interation of a loop
return	returns the accompanying value (must be of the appropriate return type)
void	used to identify a function that does not return a value
int	type identifier for int
float	type identifier for float
bool	type identifier for bool
char	type identifier for char
String	type identifier for String
Point	type identifier for Point
Curve	type identifier for Curve
Canvas	type identifier for Canvas
true	literal Boolean value
false	literal Boolean value

1.3 Literals

Type	Definition
Int	A sequence of one or more digits representing an un-named(not associated with any identifier) integer, with the leading digit being non-zero (i.e. [1-9][0-9]*)
Float	A sequence of digits seperated by a '.' representing an un-named float-point number (i.e. [0-9]*.[0- 9][0- 9]*)
Char	A single character enclosed by single quotation marks representing an un-names character. (i.e. '.')
String	A sequence of characters enclosed by a pair of double quotation marks representing an un-named string. (i.e. ^ ".*" $)
Bool	8-bit boolean variable, either $true$ or $false$

1.4 Comment

SSOL supports single line and multi-line comments. Single line comments are initiated by two "/" characters. (i.e. //), and are terminated by a newline character. Multi-line comments are initiated by the character sequence '/*' and terminated by the character sequence '*/*.

```
// This is a single line comment

/* This is a
     multi-line comment */
```

1.5 Punctuator

A punctuator is a symbol that has semantic significance but does not specify an operation to be performed. The punctuators [], (), and {} must occur in pairs, possibly separated by expressions, declarations, or statements. The semi-colon (;) is used to denote the end of every statement or expression. SSOL includes the following punctuators: [](),;

SSOL uses the semi-colon for sequencing and denoting the end of an operation. Terminate every statement with a semicolon (;).

1.6 Whitespace

Whitespace (space, tabs, and newlines) is ignored in SSOL.

2 Types

2.1 Primitives

Type	Definition
int	4 byte signed integer
float	8 byte floating-point decimal number
bool	1 byte Boolean value
char	1 byte ASCII character
String	array of ASCII characters

2.2 Complex Types

The following built-in complex data types are represented as objects with member fields and are instantiated using their associated constructors. The individual fields of the objects can be accessed and modified with . notation, ex: `object.field`

2.2.1 Point

A Point object contains two fields: an x and a y coordinate value, both of type `float`.
A Point object is instantiated using its sole constructor:

```
Point(double x, double y)
```

2.2.2 Curve

A curve object represents a Bezier curve, defined by two endpoints and two control points. Curves are instantiated using the following two constructors:

```
Curve(Point a, Point b)
Curve(Point a, Point b, Point c1, Point c2)
```

The first constructor creates a straight line defined by endpoints a and b. The second constructor creates a curve defined by endpoints a and b and control points $c1$ and $c2$.

2.2.3 Canvas

A canvas object represents a two-dimensional coordinate plane to which Point and Curve objects are added. These graphical elements are added using the | operator. Canvas objects are outputted to files via the *draw* library function.

A canvas object is instantiated using either of the following two constructors:

```
Canvas()
Canvas(int x, int y)
```

The first constructor creates a Canvas object with default dimensions of 1000×1000.
This second constructor creates a Canvas object with the dimensions specified by the values for x and y.

2.3 Arrays

Arrays are a built-in data structure consisting of sequential elements of a single type. See section 3.2 for usage.

3 Syntax

3.1 Type Specifiers

SSOL is a language with explicit typing. All variables and functions must be declared with a type specifier, which tells compiler which operations are valid for the former and what to expect the latter to return. In SSOL declaration and assignment must happen in separate statements.

3.1.1 Primitives

```
int x;
x = 3;
String myString;
myString = "hello";
bool b;
b = true;
```

3.1.2 Complex Types

```
//Canvas can be instantiated with default size or with a user specified size
Canvas can1; can1 = Canvas();
Canvas can2; can2 = Canvas(100,100);
```

```
Point pt; pt = Point(10,20);

//Create a straight line by declaring a Curve with only 2 arguments
Curve crv1; crv1 = Curve( (10,20),(100,200) );

//Create a bezier curve by declaring a Curve using 4 arguments
//Here we demonstrate that Curve will except
Curve crv2; crv2 = Curve( pt, (40,40), (100,100), (120,140) );
```

3.2 Arrays

Arrays in SSOL are instantiated with a fixed size and can only hold a single type, which can be either primitive or complex. `Array.length` returns the length of the array.

3.2.1 Declaration

```
int arr[5];
arr = [1,2,3,4,5];
```

3.2.2 Accessing

Use brackets and an index to retrieve a value from an array. The specified index must be within the bound of the array. The variable returned by the array access operation must match the variable that its value is assigned to.

```
int i; i = intArr[0];
Point p; p = Point(intArr[0], intArr[1]);
```

3.3 Operators

3.3.1 Arithmetic

Addition (+), subtraction (-), multiplication (*), division (/), and modulo (%) are standard arithmetic operators in SSOL which comply with order of operations. Increment (++) and decrement (–) are also valid operators. These are all valid operations for both *int* and *float*, but cannot be used on *int* and *float* together.

3.3.2 Comparison

Comparison in SSOL is done via ==, !=, <,>,<=, and >=. Only matching types can be compared. These operators return a Boolean value of *true* or *false*

3.3.3 Logical

SSOL can perform logical operations of Boolean values with `&&` (AND), || (OR), and ! (NOT).

```
bool b1; b1 = true && true;
bool b2; b2 = false || false;
```

3.3.4 Assignment

The assignment statement is of the form `<identifier> = expression`, where `<identifier>` has been previously declared.

3.3.5 Canvas

The Canvas object of SSOL has a set of unique operators for sequencing and addition to a the canvas.

|	Use pipe to sequence Point and Curve objects
|=	Use pipend to add an object or sequence of objects to the canvas

Ex.

```
canvas |= crv1 | crv2 | crv3 | crv4;
```

3.4 Statements

3.4.1 Sequencing

Consecutive statements are sequenced using the `;` operator.

3.4.2 Control Flow

SSOL supports the standard `if...else` format of conditional statements. `if` requires a Boolean statement to be evaluated.

```
int i = 3;
if(i>4){
    print("i > 4");
} else {
    if(i<3){
        print("i < 3");
    }
    else{
        print("i == 4");
    }
}
```

3.4.3 Loops

SSOL supports **for** loops and **while** loops. For loops are an iterative construct that requires a starting index variable, a bounding condition, and an operation to be performed at the end of each iteration.
A **while** loop requires a Boolean expression to be evaluated every time the loop is executed.

```
int i;
for(i = 0; i<arr.length; i++){
    <loop-body>
}

int j; j = 0;
while(j<10){
    j+=1;
}
```

3.5 Functions

3.5.1 Declaration

Functions are declared as follows:

```
<function-return-type> <function-name>([arg1],[arg2],...)
{
    <function-body>
    [return <some-value>]
}
```

If the function has non-void return type, then it must return some value of that type at the end of the function, or at the end of any potential path of execution within the function, if there are conditional statements/loops. This is achieved using the keyword `return`.

3.5.2 Function Calls

Functions are called as follows:

```
<function-name>([arg1],[arg2],...)
```

If a function returns a value, that value can be assigned to a variable, assuming the variable has been previously declared, as in

```
<identifier> = <function-name>([arg1],[arg2],...)
```

4 Execution

4.1 Scope

Variables persist only within the block of code in which they are declared. A block of code is enclosed by curly braces ({ ... }).

Variables that are declared outside of any code block are considered global and are visible to all functions within a program.

4.2 main()

Every valid SSOL program needs at least one function called *main*(). This is the routine that will be executed at runtime, so program trajectory must start from here. Within *main*(), other user-defined functions may be called. The return type of *main*() is *void*.

5 Standard Library Functions

5.1 draw()

draw() is the crux of the SSOL language. This method takes a single *canvas* object and a file name as a string as arguments. *draw*() can be called as many times as the programmer desires, but there will be a 1:1 correlation between function calls and .SVG files written (if *draw* is called with the same filename twice, the file will be overwritten).

5.2 printf()

printf() is a formatted string printing function. %s for string, %i for int, %f for float, %c for char, %b for bool.

6 Sample Code

```
void main(){

    int l; l = 1000;
    int w; w = 1000;

    Canvas can; can = Canvas(l,w);

    //Create 4 straight lines that form a square
   Curve top; top = Curve(Point(w*.1,l*.1),Point(w*.9,l*.1));
   Curve right; right = Curve(Point(w*.9,l*.1),Point(w*.9,l*.9));
   Curve bottom; bottom = Curve(Point(w*.9,l*.1),Point(w*.9,l*.1));
   Curve left; left = Curve(Point(w*.9,l*.1),Point(w*.1,l*.1));

   //Create a circle inside the square using 2 bezier curves
   Curve semiTop; semiTop = Curve(Point(w*.25,l*.5), Point(w*.25, l*.25),
```

```
        Point(w*.75,l*.5), Point(w*.75,l*.25);
    Curve semiBottom; semiBottom = Curve(Point(w*.25,l*.5), Point(w*.25, l*.75),
        Point(w*.75,l*.5), Point(w*.75,l*.75);
}
```