Table of Contents

1 INTRODUCTION ............................................................................................................................... 1

2 LEXICAL CONVENTIONS .............................................................................................................. 1
   2.1 COMMENTS ............................................................................................................................ 1
   2.2 IDENTIFIERS ......................................................................................................................... 1
   2.3 KEYWORDS ............................................................................................................................. 1
   2.4 CONSTANTS ........................................................................................................................... 2

3 OPERATORS ...................................................................................................................................... 2
   3.1 DOT OPERATOR .................................................................................................................... 2
   3.2 LOGICAL AND OPERATOR .................................................................................................... 2
   3.3 LOGICAL OR OPERATOR ....................................................................................................... 2
   3.4 EQUALITY OPERATORS ......................................................................................................... 3
   3.5 RELATIONAL OPERATORS .................................................................................................. 3
   3.6 NEGATION OPERATOR .......................................................................................................... 4
   3.7 ADDITIVE OPERATORS ......................................................................................................... 4
   3.8 MULTIPLICATIVE OPERATORS ............................................................................................. 5
   3.9 OPERATOR PRECEDENCE ..................................................................................................... 5

4 DATA TYPES ..................................................................................................................................... 5
   4.1 TYPE SPECIFIERS ........................................................................................................................ 5
   4.2 ARRAY TYPES .......................................................................................................................... 5
   4.3 SHAPE ...................................................................................................................................... 6
   4.4 CIRCLE ........................................................................................................................................ 6
   4.5 RECTANGLE ................................................................................................................................... 6
   4.6 TEXT FIELD .................................................................................................................................. 6

5 STATEMENTS ................................................................................................................................... 7
   5.1 FUNCTION DECLARATIONS .................................................................................................... 7
   5.2 VARIABLE DECLARATIONS ..................................................................................................... 7
   5.3 BLOCK STATEMENTS .............................................................................................................. 7
   5.4 CONDITIONAL STATEMENTS ................................................................................................ 8
   5.5 ITERATIVE STATEMENTS ......................................................................................................... 8
   5.6 ASSIGNMENT STATEMENTS ................................................................................................. 8

6 SCOPING CONVENTIONS .............................................................................................................. 9

7 EXAMPLE PROGRAM .................................................................................................................... 10
1 Introduction
This document outlines a reference manual for the Simple Flash Animation Language (SFAL) 2.0, which is a programming language designed for simple vector-graphic animation using Adobe Flash.

2 Lexical conventions
A program consists of at least one statement and other optional tokens described in this section.

2.1 Comments
Single-line comments begin with // and end with a new line. Multi-line comments begin with /* and end with the first encountered */.

```
//single-line comment
/*
 * multi-line comment
*/
```

2.2 Identifiers
An identifier begins with a letter or underscore and ends with any combination of letters, underscores, or integers. The longest identifier length is 128 characters.

```
fool
bar_2
__abc__3__def
```

2.3 Keywords
The following words are reserved for use as keywords and may not be used otherwise:

<table>
<thead>
<tr>
<th>BLACK</th>
<th>GRAY</th>
<th>RED</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLUE</td>
<td>GREEN</td>
<td>shape</td>
</tr>
<tr>
<td>BROWN</td>
<td>if</td>
<td>string</td>
</tr>
<tr>
<td>break</td>
<td>int</td>
<td>textfield</td>
</tr>
<tr>
<td>circle</td>
<td>ORANGE</td>
<td>YELLOW</td>
</tr>
<tr>
<td>continue</td>
<td>PURPLE</td>
<td>void</td>
</tr>
<tr>
<td>else</td>
<td>rect</td>
<td>WHITE</td>
</tr>
<tr>
<td>for</td>
<td>return</td>
<td></td>
</tr>
</tbody>
</table>
2.4 Constants

String literals are any sequence of printable characters in between two quotation marks:

“this is a string”

Integer literals are any sequence of digits, where non-zero numbers begin with a non-zero digit and negative numbers begin with a hyphen:

0
-123
9876

3 Operators

3.1 Dot Operator

The dot operator accesses a member of a structured data type. Currently, only shape data types have members, and those members are built-in functions.

```java
// tweens a square from its current position to (10, 20)
rect(BROWN, 4, 4).tween(10, 20);
```

3.2 Logical AND Operator

The binary operator `&&` returns 1 if both its operands, which themselves can be expressions, evaluate to a non-zero value:

```java
// returns 0
1 && 0
// returns 1 if both functions return non-zero value
isfunny() && istired()
// returns 1
(1+2 != SEVEN) && (3+4 != FOUR)
```

3.3 Logical OR Operator

The binary operator `||` returns 1 if either of its operands, which themselves can be expressions, evaluate to a non-zero value:

```java
// returns 1
1 || 0
// returns 1 if either function returns non-zero value
isfunny() || istired()
// returns 0
(1+2 == SEVEN) || (3+4 == FOUR)
```
3.4 Equality Operators
The binary operator \(==\) returns 1 if both its operands, which themselves can be expressions, evaluate to an equal value. Similarly, the binary operator \(!=\) returns 1 if both its operands, which themselves can be expressions, evaluate to different values.

```
// returns 0
1 == 0
// returns 1 if both functions return different values
isfunny() != istired()
// returns 0
1+2 == SEVEN
```

3.5 Relational Operators
The binary operator \(<\) returns 1 if its left-value operand is less than its right-value operand. Similarly, the binary operator \(<=\) returns 1 if its left-value operand is less than or equal to its right-value operand.

```
// returns 0
55 < 20
// returns 1
1+2 <= SEVEN
// returns 1
4 <= FOUR
```

The binary operator \(>\) returns 1 if its left-value operand is greater than its right-value operand. Similarly, the binary operator \(>=\) returns 1 if its left-value operand is greater than or equal to its right-value operand.

```
// returns 1
55 > 20
// returns 0
1+2 >= SEVEN
// returns 1
4 >= FOUR
```
3.6 Negation Operator
The unary operator ! returns 1 if its operand, which itself can be an expression, evaluates to a zero value:

```plaintext
// returns 1
!0
// returns 1 if function returns 0
!upisdown()
// returns 0
!(4 == FOUR)
```

3.7 Additive Operators
The binary operator + adds its two integer operands, or concatenates its right-value string operand to its left-value string operand, or adds a shape operand to an array (or “group”) of shapes.

```plaintext
// returns 15
9 + 6
// returns “hello world”
"hello" + " world"
// puts a square and circle into a shape array
sqr + circ
```

The binary operator – subtracts its right-value operand from its left-value operand:

```plaintext
// returns 3
9 - 6
// returns 5
25 - 20
```
3.8 Multiplicative Operators
The binary operator \( \ast \) multiplies its two integer operands:

```
// returns 14
2 \ast 7
// returns 120
THREE \ast 40
```

The binary operator \( / \) divides its right-value operand from its left-value operand. Division of an even number by an odd number (or vice versa) results in the rounding to the next lowest integer. Division by zero is illegal.

```
// returns 2
24 / 12
// returns 3
TEN / 3
```

3.9 Operator Precedence
For the arithmetic operators, the operator precedence is conventional: parenthesized expression, multiply, divide, addition, subtraction.

```
// (10 * 2) - 2 + ((51 - 3) / 12) + 3 = 25; returns 25
10 * 2 - 2 + (51 - 3) / 12 + 3
```

4 Data Types
4.1 Type specifiers
Any of the following words are type specifiers:

```
circle  shape  void
int     string
rect    textfield
```

4.2 Array Types
Arrays are created by appending to the type specifier the element count with an integer surrounded by square brackets:

```
// array of 4 integers
int[4] numbers;
// array of 20 shapes
shape[20] shapegroup;
```
4.3 Shape
The `shape` data type is any polygon, circle, rectangle, or text field that can be represented in Adobe Flash. This type has two built-in functions: `move(x, y)` and `tween(x, y)`.

4.4 Circle
The `circle` data type extends the `shape` type. This type only creates circles of a specified color and radius.

```javascript
// creates a red circle with a radius of 5 pixels
c = circle(RED, 5);
```

4.5 Rectangle
The `rect` data type extends the `shape` type. This type only creates rectangles of a specified color, height, and width.

```javascript
/* creates a blue rectangle with a radius of height of 3 pixels and a width of 4 pixels */
r = rect(BLUE, 3, 4);
```

4.6 Text field
The `textfield` data type extends the `shape` type and is used for text animation.

```javascript
/* creates a white text field with font size 24 and “hello world!” as text */
txt = textfield(WHITE, 24, “hello world!”);
```
5 Statements

All statements (except for function declarations) are terminated with a semicolon.

5.1 Function Declarations

Functions are declared by preceding a non-reserved string (the identifier) with a type specifier, appending to that identifier a comma-separated parameter list that is enclosed in parentheses (parameters are optional), and finally appending a block statement:

```c
int foo(int a, string b)
{
    ...
}
void bar()
{
    ...
}
```

5.2 Variable Declarations

Variables are declared by preceding a non-reserved string (the identifier) with a type specifier. Variable declaration is not required. If a variable is not already declared, the variable is created on first assignment and its type is determined automatically.

```c
// declared variable j
int j;
j = 0;

/* a new integer i is created and the result of j + 2 is assigned to it */
i = j + 2;

/* a new shape group s is created and assigned a gray circle and black rectangle */
s = circle(GRAY, i) + rect(BLACK, 4, 5);
```

5.3 Block Statements

Surrounding any statement(s) by curly braces creates a block statement.

```c
{
    /* begin block statement */
    int j;
    for (j = 0; j < 5; j = j + 1)
    {
        /* another block statement */
    }
}
```
5.4 Conditional Statements

The conditional statement takes the form: if (expression) statement else statement. The if clause is executed if its predicate evaluates to a non-zero value. The optional else portion is executed only if the if clause predicate evaluates to zero. The else is linked to the last encountered if.

```c
if (k == 0) {
    /* do something for k == 0 */
} else {
    /* do something for k != 0 */
}
```

5.5 Iterative Statements

One or more statements can be executed iteratively using a for loop:

```
for (expression-1; expression-2; expression-3) statement
```

// fill array of 5 red circles each w/ diff diameters
for (j = 0; j < 5; j = j + 1)
{
    shapegroup[j] = circle(RED, 10+j);
}

The continue keyword can be used to skip the remaining statements in the for loop, or the break keyword can be used to exit the for loop.

5.6 Assignment Statements

Assignment statements take the form of: left-value = right-value

```
foo = 1;
bar = func();
foo = FIVE;
bar = 2 + 4;
```
6  Scoping Conventions
Each block contains its own scope. Elements outside of the block are visible within the block but not vice versa.

```c
{  
    int j;  
    j = 3;  
    
    {  
        int i;  
        // i is 4  
        i = j + 1;  
    }  
    
    // error: i out of scope  
    j = i + 1;  
}
```
7 Example Program
The following program animates a group of circles and rectangles diagonally up the screen:

```c
void move(shape s)
{
    // move shape from bottom-left corner of stage
    // to top-right corner
    for (i = 0; i < 100; i = i + 1)
    {
        s[i].move(i, i);
    }
}

// create a shape group of 2 circles
// and 2 rectangles
s = circle(RED, 5) + rect(GREEN, 5, 2)
    + circle(ORANGE, 3) + rect(BLUE, 6, 3);

// move the shapes diagonally across stage
for (i = 0; i < 4; i = i + 1)
{
    move(s[i], 100, i);
}

/* tween a text field from its starting position (0,0) to (20,30) */
textfield(PURPLE, 32, "Hi!!").tween(20, 30);
```