

# The Chirp-Scribbler Lite Language Specification

Li Xu

Department of Computer Science  
University of Massachusetts Lowell  
Lowell, MA 01854

## 1 Introduction

Chirp-Scribbler Lite is a simplified version of the Chirp language. The Chirp language is designed as a simple C-like language to program embedded robot controllers. As the hardware features of each robot device are highly specific to the individual robot platform, to support multiple types of robots, Chirp provides a set of common language constructs along with specialized data types and functions to program each robot platform. Chirp-Scribbler Lite is the Chirp language variant for the Scribbler robots. This document describes the language constructs and usage of Chirp-Scribbler Lite.

## 2 Syntax

Chirp-Scribbler Lite has a syntax similar to C. A Chirp-Scribbler Lite program consists of the data section and code section. The data section has global variable definitions which define the type and name of global variables. Chirp-Scribbler Lite has only global variables, the type of the variables can be either int or byte type. The code section includes Chirp-Scribbler Lite statements, such as assignments, if-else conditionals, system function calls and loop statements.

Chirp-Scribbler Lite is case sensitive, eg., `Foo` and `foo` are distinct names. Chirp-Scribbler Lite keywords are reserved – the programmer cannot use a Chirp-Scribbler Lite keyword as the name of a variable. The valid keywords are: `int`, `byte`, `if`, `else`, `loop`, `while`, `until`, `for`, `break`.

Chirp-Scribbler Lite also uses the following special characters:

$$\{ \} < > = + - * / \% ! \&\& || ( ) . , ; "$$

The Chirp-Scribbler Lite grammar in Figure 1 specifies the usage of the special characters.

Chirp-Scribbler Lite accepts C/C++ style comments: comments are delimited by matching `/*` and `*/` for multi-line comments, or follow `//` for single line comments.

Chirp-Scribbler Lite identifiers start with a letter or the underscore character which is then followed by letter, underscore or digit characters. Chirp-Scribbler Lite identifiers are used for variable and system function names. Chirp-Scribbler Lite constants are integers and string constants. String constants are quoted by double quotes and consist of character sequence of non-quote characters.

<i>ChirpLetter</i>	→	a   b   c   ...   z   A   B   ...   Z   _
<i>Digit</i>	→	0   1   2   ...   9
<i>ID</i>	→	<i>ChirpLetter</i> ( <i>ChirpLetter</i>   <i>Digit</i> )*
<i>STRING</i>	→	" ( <i>Char</i> )* "

Chirp-Scribbler Lite supports the following boolean, relational and arithmetic operators:

- `&&` for boolean and, `||` for or, `!` for not;

<i>program</i>	→	<i>data_section code_section</i>
<i>data_section</i>	→	( <i>var_def</i> )*
<i>var_def</i>	→	<i>data_type</i> ID ;
<i>data_type</i>	→	int   byte
<i>code_section</i>	→	( <i>statement</i> )*
<i>statement</i>	→	<i>assignment_stmt</i> ;   <i>if_else_stmt</i>   <i>loop_stmt</i>   <i>break_stmt</i> ;   <i>function_call</i> ;   <i>block</i>
<i>assignment_stmt</i>	→	ID = <i>expression</i>
<i>if_else_stmt</i>	→	if ( <i>expression</i> ) <i>statement</i> ( <b>else</b> <i>statement</i> )?
<i>loop_stmt</i>	→	loop ( <b>while</b> ( <i>expression</i> ) )? <i>block</i> ( <b>until</b> ( <i>expression</i> ) )?   for ID ( <i>expression</i> : <i>expression</i> ( : <i>expression</i> )? ) <i>block</i>
<i>break_stmt</i>	→	<b>break</b>
<i>function_call</i>	→	ID ( . ID )* ( ( <i>expression</i> ( , <i>expression</i> )* )? )
<i>block</i>	→	{ ( <i>statement</i> )* }
<i>expression</i>	→	! <i>relational_expr</i>   <i>relational_expr</i> ( ( &&      ) <i>relational_expr</i> )*
<i>relational_expression</i>	→	<i>arithmetic_expression</i> ( ( ==   !=   <=   <   >=   > ) <i>arithmetic_expression</i> )?
<i>arithmetic_expression</i>	→	<i>term</i> ( ( +   - ) <i>term</i> )*
<i>term</i>	→	<i>factor</i> ( ( *   /   % ) <i>factor</i> )*
<i>factor</i>	→	ID   INT   STRING   ( <i>expression</i> )

Figure 1: Chirp-Scribbler Lite Grammar

- relational comparison ==, !=, <=, <, >=, >;
- arithmetic operation +, -, \*, /, %. The % is the modulus operator.

The full syntax of Chirp-Scribbler Lite is shown in Figure 1. The terminal tokens are in the typewriter font. The meta symbol \* indicates the unit can repeat zero or multiple times, and ? indicates the unit can repeat zero or one time.

## 3 Semantics

### 3.1 Data Type

Chirp-Scribbler Lite has int and byte data types, which correspond to the 16-bit WORD and 8-bit BYTE in Scribbler PBASIC. Similar to C, user can use integer types to represent boolean values: 0 represents boolean false and any non-zero value represents boolean true.

### 3.2 Variables

Chirp-Scribbler Lite allows only global variables.

### 3.3 Statements

Chirp-Scribbler Lite has the following statements:

- Assignment Statement:  
Assign the expression value to a variable.
- If-Else Statement:  
The if-else statement evaluates the expression to a boolean value (using the zero/non-zero

numeric value as boolean result) and executes the following “then” or “else” branch respectively. The `else` statement is always bound to the nearest unbound `if` statement to resolve ambiguity.

- **Loop and Break Statements:**  
Chirp-Scribbler Lite provides the unconditional and conditional loops with `while` and `until` condition constructs. The `loop` statements will either execute the loop body in repetition (infinite loop), or evaluate the `while` (loop-while-body) or `until` (loop-body-until) condition expression and iterate the loop body depending on the condition: for `while` loop, the loop body will be executed if the condition is true; for `until` loop, the loop body will terminate if the condition is true. Chirp-Scribbler Lite also has `for`-loop statements, in which the loop counter variable iterates from the loop-start value to the loop-end value with an optional step increment (`for-id-start-end-step`). The `break` statement is used in the loop body to exit loop iteration.
- **Call Statement:**  
Chirp-Scribbler Lite allows user to call `System.Scribbler` functions to control the robot device. Function arguments are either expressions or variables. If the function will store results in the argument, the argument need to be variable to store the value.
- **Block Statement:**  
Similar to C, Chirp-Scribbler Lite allows user to use curly braces to group statements into code blocks.

### 3.4 Expressions

Chirp-Scribbler Lite expressions compute values of the basic types. As boolean values are represented by integer values, Chirp-Scribbler Lite defines boolean *and*, *or* and *not* operator to evaluate boolean values. The relational operators are `==`, `!=`, `<=`, `<`, `>=`, `>`. Chirp-Scribbler Lite also supports arithmetic operations.

### 3.5 System Functions

To control Scribbler I/O hardware, Chirp-Scribbler Lite provides the following `System.Scribbler` built-in functions:

1. `System.Scribbler.wait(time)`  
Pause Scribbler for `time` millisecond.
2. `System.Scribbler.sound(freq, time)`  
Play the specified frequency sound for the duration of `time` millisecond.
3. `System.Scribbler.input(var1, var2, ...)`  
Read variable values from the serial port, this will change the variable value.
4. `System.Scribbler.print(exp1, exp2, ...)`  
Print out expression values (including string constants) through the serial port.
5. `System.Scribbler.senseStall(stall_var)`  
Read the stall sensor value and set `stall_var` to 0 or 1.
6. `System.Scribbler.setLED(left, center, right)`  
Set Scribbler’s three LEDs to on or off, the `left`, `center`, `right` arguments are the value of each LED, with 0 for off and 1 for on.

7. `System.Scribbler.senseLight(left_var, center_var, right_var)`  
Read the light sensors' light value and set the variable `left_var`, `center_var` and `right_var` for the three light sensors.
8. `System.Scribbler.senseObjLeft(obj_var)`  
Turn on the left object detection sensor, detect whether object is on the left and store result in `obj_var`.
9. `System.Scribbler.senseObjRight(obj_var)`  
Turn on the right object detection sensor, detect whether object is on the right and store result in `obj_var`.
10. `System.Scribbler.senseLine(left_var, right_var)`  
Read line sensor states to `left_var` and `right_var` for the left and right line sensors, 0 for no-line detected, 1 for line detected.
11. `System.Scribbler.moveForward(left_speed, right_speed)`  
Turn on Scribbler motors and run the motors forward at `left_speed` for the left motor, and `right_speed` for the right motor. Motor speed level is between 0 — 10.
12. `System.Scribbler.moveBackward(left_speed, right_speed)`  
Turn on Scribbler motors and run the motors backward at `left_speed`, `right_speed`. Motor speed level is between 0 — 10.
13. `System.Scribbler.turnFront(left_speed, right_speed, time)`  
Control Scribbler motors to rotate forward at `left_speed`, `right_speed` for the `time` duration. Motor speed is between 0 — 10, time is in millisecond.
14. `System.Scribbler.turnBack(left_speed, right_speed, time)`  
Control Scribbler motors to rotate backward at `left_speed`, `right_speed` for the `time` duration. Motor speed is between 0 — 10, time is in millisecond.
15. `System.Scribbler.stop()`  
Turn off the Scribbler motors.

## 4 Examples

This section shows several code examples of using Chirp-Scribbler Lite to program the Scribbler robot. A Chirp-Scribbler Lite reference compiler is implemented in Java. To run an example program, type `java csc.cslc prog.csl` to use the `cslc` reference compiler to compile `prog.csl` into `.bs2` code and use the Scribbler StampEditor to compile and upload the target binary code onto Scribbler.

- Control Scribbler LED:

```
int i;
int led1;
int led2;
int led3;

for i (1:100) {
    led1 = i%2;
    led2 = (i/2)%2;
    led3 = (i/4)%3;
}
```

```

        System.Scribbler.setLED(led3, led2, led1);
        System.Scribbler.print("i is: ", i);
        System.Scribbler.wait(1000);
    }

    System.Scribbler.print("now i am done");

```

- Read the light and stall sensors:

```

int a;
int l1;
int l2;
int l3;

loop {
    System.Scribbler.senseLight(l1, l2, l3);
    System.Scribbler.print("light: ", l1, ", ", l2, ", ", l3);
    System.Scribbler.wait(500);

    System.Scribbler.senseStall(a);
    if (a) {
        System.Scribbler.print("stalled: a=", a);
    } else {
        System.Scribbler.print("not stalled: a=", a);
    }
}

```

- Run Scribbler motors:

```

loop {
    System.Scribbler.moveForward(3, 3);
    System.Scribbler.wait(1000);
    System.Scribbler.stop();
    System.Scribbler.moveBackward(3, 3);
    System.Scribbler.wait(1000);
    System.Scribbler.stop();
    System.Scribbler.print("stopped");
    System.Scribbler.wait(3000);
}

```

## References

- [1] Andy Lindsay. *What's a Microcontroller*. Parallax, Inc.
- [2] Parallax, Inc. *BASIC Stamp Reference Manual, version 2.1*.
- [3] Parallax, Inc. Scribbler Robot web page. <http://www.scribblerrobot.com/>.
- [4] Li Xu and Fred Martin. The chirp language specification. Technical Report TR-2005-003, Dept. of Computer Science, UMass Lowell.