M.R. Roboto:

Macro Record Robot Language
Language Reference Manual

Authors:

Adam Marczyk (alm2126@columbia.edu)
Hema Krishnan (hk2230@columbia.edu)
Jason Kopylec (jkk2106@columbia.edu)
Sanjit Tewari (sot5@columbia.edu)

1.0 Introduction

The Macro Record Robot Language (M.R. Roboto) places in the hands of the programmer a powerful tool for controlling GUI-based operating systems. By simulating the actions of a user, one can test interactive applications, automate repetitive tasks with ease, and set one's computer to perform tasks at scheduled times when no user can be at the keyboard. M.R. Roboto provides a simple script-style API to mimic keyboard and mouse input, and utilizes Java's rich language libraries without requiring specific skill in programming Java syntax.

This manual outlines the language syntax and semantics for creating M.R. Roboto programs and compilers. Wherever possible, context-free grammars have been included to precisely define the syntactic constructs. The overall flavor of the language is similar to scripting languages such as Unix shell scripts, with additional power and flexibility added through the use of Java-like control constructs.

2.0 The Basics

The following section defines and describes the basic components and fundamental syntactic rules of a program written in the M.R. Roboto language.

2.1 Character Set

M.R. Roboto program source files can be composed of any characters from the standard ASCII character set, i.e., characters from '\3' (CTRL+C) to '\377' (DEL).

2.2 File Name Conventions

M.R. Roboto programs should be named with the extension *.mmr*. The file name will become the name of the Java class that is created, so it should take the form of an *identifier* (see section 2.3). For example, the file *TestProgram.mmr* compiles into a Java class file named *TestProgram.class*.

File Name Grammar

```
filename
     : identifier ".mmr"
```

2.3 Identifiers

Identifiers are defined as the names of functions, variables, or class files. In this language, an identifier must start with an alphabetic character followed by any number (up to Java's maximum) of letters, digits or the underscore character, '_'. Reserved words are illegal as identifiers (see section 2.3.1).

Examples of Legal Identifiers: `abc123, x, i_count, tmp`
Examples of Illegal Identifiers: `abc&c, _ac2, 1stP, end`

Identifier Grammar

```
identifier
     : letter (alphanumeric | "_")*

alphanumeric
     : letter | digit

letter
     : [A-Za-z]

digit
     : [0-9]
```

2.3.1 Reserved Words

The following are reserved as keywords in the language. They may not be used as identifiers. Keywords in Java are also illegal identifiers, but they are not listed here.

```
moveMouse                      click
rightClick                     doubleClick
length                         substring
getX                           getY
type                           press
hold                           release
releaseAll                     wait
return                         for
to                             while
step                           end
if                             then
else                           function
int                            string
color                          and
or
```

These reserved words represent either built-in library functions, data types or control flow keywords; their exact functions will be discussed in detail below.

2.4 Variables

Variables are mutable, dynamically valued blocks of memory that store values used by the program during its execution. All variables must be declared and assigned a value before being used; variable names are identifiers and must be distinct. M.R. Roboto supports three built-in data types for variables: *string, integer,* and *color*. Each of these is described in detail below.

2.4.1 String Literals

A string is a sequence of printing and non-printing characters. Strings in the M.R. Roboto language are the same as those in Java, with the addition of escape codes for keys that do not produce printing characters. Escape sequences are indicated by the key name (see below chart), surrounded by backslashes. Strings are delimited with double quotes. To include literal double quotes or backslashes, pair them, as in Java.

Example Strings: `"abc", "How are you today?", "\TAB\", "\\"`

Key Escape Codes

`\F1\ - \F12\`	F1 – F12 keys
`\ALT\`	[ALT]
`\CTRL\`	[CTRL]
`\CAPS_LOCK\`	[CAPS LOCK]
`\DELETE\`	[DELETE]
`\BACKSPACE\`	[BACKSPACE]
`\TAB\`	[TAB]
`\ENTER\`	[RETURN]/[ENTER]
`\DOWN\`	Down Arrow Key
`\UP\`	Up Arrow Key
`\LEFT\`	Left Arrow Key
`\RIGHT\`	Right Arrow Key
`\ESCAPE\`	[ESC]
`\HOME\`	[HOME]
`\END\`	[END]
`\INSERT\`	[INSERT]
`\PRINTSCREEN\`	[PRINTSCREEN]

`\SHIFT\`	[SHIFT]
`\PGDOWN\`	[PAGE DOWN]
`\PGUP\`	[PAGE UP]
`\NUM_LOCK\`	[NUM LOCK]

String Grammar

```
string
     : """ (.)* """
```

2.4.2 Integers

An integer is either 0 or a positive whole number composed of a sequence of digits. The minimum and maximum values are machine-specific. M.R. Roboto does not distinguish between signed and unsigned integers and does not allow for negative values.

Example Integers: 0, 1, 2, 13, 725

Integer Grammar

```
integer
     : [0-9]+
```

2.4.3 Pixel Color

A pixel color variable stores the RGB color value of a given pixel, typically the one directly beneath the tip of the mouse pointer. Valid values are BLACK, BLUE, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE and YELLOW (reflecting the Java color constants).

Pixel Color Grammar

```
pixelcolor
     : ("BLACK"|"BLUE"|"DARK_GRAY"|"GRAY"|"GREEN"|
"LIGHT_GRAY"|"MAGENTA"|"ORANGE"|"PINK"|"RED"|
"WHITE"|"YELLOW")
```

2.5 Declaration

Variables must be declared before use in order to determine their type. A declaration statement consists of the type being declared, followed by the variable name identifier. Variables are declared one per line.

Declaration Grammar

```
declaration
     : var_type identifier

var_type
     : ("int" | "string" | "color")
```

2.6 Assignment

Once a variable has been declared, it must be assigned a value before it can be used. An assignment statement consists of the name of the variable, followed by an equals-sign character, followed by the value the programmer wishes to assign to it (which must be of the same type as the variable was initially declared to have). The value can be either an atomic value (e.g., 1, "a"), or an expression that evaluates to a value of the correct type (e.g., (5-3), ("a"++"b")). (See section 3.2 for grammar for assignment statements.)

2.7 Operators

There are five types of operators in the M.R. Roboto language: string, mathematical, equality, logical, and parenthetical. All operators in this language are infix.

There is only one string operator, the binary concatenation operator "++", which takes two strings as operands and combines them into one.

There are four mathematical operators: addition ("+"), multiplication ("*"), division ("/"),and subtraction ("-"), each of which takes two integers as operands. If the result of an integer operation would otherwise be a decimal number, the decimal part is ignored and a whole number is returned as if the result had been processed by the "floor" operation (e.g., 5/2 = 2, which is the same as floor(2.5)).

There are seven equality operators, "=", "==", "<", ">", "<>", "<=" and ">=". All of them take two operands, which can be of any type but must both be of the same type. The first operator assigns the value of the right operand (which may be either a variable or literal value) to the left operand (which must be a variable). The other equality operators return either 0 for true or 1 for false, depending on whether the first operand is exactly equal to, less than, greater than, not equal to, less than or equal to, or greater than or equal to the second operand, respectively.

There are two logical operators, "and" and "or", which take two operands of integer type. The "and" operator returns 1 if both its operands are non-zero; otherwise it returns 0. The "or" operator returns 1 if either of its operands are non-zero; otherwise it returns 0.

The parenthetical operators are "(" and ")", which must always be matched. These operators do nothing by themselves, but any statement inside them is "promoted" to the highest level of precedence and evaluated as an atomic unit.

2.7.1 Precedence

Precedence for the operators follows roughly the same rules as in the C and Java programming languages. The operators are listed below in order of increasing precedence (operators on the same line have equal precedence):

```
=
or
and
== < > <> <= >=
+ - ++
* /
( )
```

Expression and Precedence Grammar

```
expr
     : expr1 ("=" expr1)*

expr1
     : expr2 ("or" expr2)*

expr2
     : expr3 ("and" expr3)*

expr3
     : expr4 (equalityop expr4)*

expr4
     : expr5 (plusop expr5)*

expr5
     : expr6 (multop expr6)*
```

```
expr6
     : "(" expr ")" | value | string

value
     : integer | identifier

equalityop
     : "==" | "<" | ">" | "<>" | "<=" | ">="

plusop
     : "+" | "-" | "++"

multop
     : "*" | "/"
```

3.0 Program Structure

The following section builds on the previous section to define in detail the overall structure of a program in the M.R. Roboto language.

3.1 The Main Block

A program in the M.R. Roboto language has one main block of code, consisting of multiple statements (section 3.2) each on its own line, that is run each time that program runs and that is the first thing to be executed each time that program runs (analogous to the main() function in a program in C or Java). The main block is followed by one blank line and then one or more function definitions (section 3.5) that may be invoked by the main block during its execution.

Main Block Grammar

```
program
     : block newline (function)*

newline
     : "\n"

block
     : (stmt)+
```

3.2 Statements

A block of code in the M.R. Roboto language is made up of one or more statements, each followed by a newline character. A statement may be any of the following: a comment (section 3.3), a declaration (section 2.5), a conditional statement (section 3.4.1), a loop (section 3.4.2), or an expression incorporating one or more operators (section 2.7.1).

Expression Grammar

```
stmt
     :  (comment  |  declaration  |  conditional  |  loop  |
function_call | expr) newline
```

3.3 Comments

Comments are statements inserted by the programmer to more clearly explain the operation of the program. They are not considered part of the program source code and are discarded before the program is compiled. Comments are marked by a double asterisk ("**") at the beginning of a line and extend through the end of that line. Multiple-line comments may be created by starting multiple successive lines with this delimiter.

Comment Grammar

```
comment
     : "**" (.)*
```

3.4 Flow Control

M.R. Roboto allows the programmer to control the flow of program execution through two types of constructs: conditional statements, which allow a block of code to be skipped, and loop statements, which allow a block of code to be executed multiple times.

3.4.1 Conditional Statements

M.R. Roboto supports one type of conditional statement, namely the standard if-else type conditional common to many programming languages.

3.4.1.1 If-Else Statements

An if-else conditional statement consists of the keyword "if" followed by a boolean expression (i.e., one that evaluates either to 0 or to non-zero), followed by a block of code, followed by an optional "else" keyword followed by a block of code, followed by an "end" keyword. If the boolean expression evaluates to non-zero, the block of code following the "if" will be executed; otherwise, if there is an "else" statement, the block of code following it will be executed. In either case, program execution then continues starting at the next statement following the "end". Conditional statements can be nested; in such a case, the "end" aligns with the most recent "if".

Conditional Statement Grammar

```
conditional
     : "if " expr newline block ("else" newline block)?
"end"
```

3.4.2 Loop Statements

M.R. Roboto supports two types of loop statements, the for loop (section 3.4.2.1) which allows a block of code to be executed a certain number of times, and the while loop (section 3.4.2.2), which allows a block of code to be executed repeatedly until a certain condition is met.

Loop Statement Grammar

```
loop
     : while_loop | for_loop
```

3.4.2.1 For Loops

Similar to the "for" statement in common languages such as C++ and Java, the M.R. Roboto "for" loop sets an initial variable and repeatedly executes a block of code, altering the initial variable's value in a predetermined way at each execution until it reaches some predefined termination condition. Specifically, the M.R. Roboto "for" loop takes a variable (which must be of integer format), sets its value to the first value given, and by default, at each iteration increments its value by 1 until it reaches the second value given, which is the termination condition. If the optional "step" keyword is supplied, the variable's value will be incremented by the supplied value rather than by 1.

For Loop Statement Grammar

```
for_loop
     : "for "  identifier "," value " to "  value ("step"
value)? newline block "end"
```

3.4.2.2 While Loops

The M.R. Roboto "while" loop takes a boolean expression and repeatedly executes a block of code until that expression evaluates to 0. (Obviously, the code should alter the value of that expression in some way at each iteration, to prevent the loop from running indefinitely.) Like the for loop, a while loop is closed by the keyword "end" to indicate that the following code is no longer considered part of the loop.

While Loop Statement Grammar

```
while_loop
     : "while " expr newline block "end"
```

3.5 User-Defined Functions

In addition to the main block of code (section 3.1), the M.R. Roboto language allows programmers to avoid needless repetition by encapsulating code that may be called multiple times during the execution of a program into one or more user-defined functions, which when invoked will run the code they contain.

Functions are defined separately from the main block of code, separated from it and by each other by blank lines. They begin with the keyword "function" and end with the keyword "end". Each function has a name, which must be a valid identifier; a function may have the same name as a variable, but not the same name as another function. After its name, a function must have a set of parentheses which may contain one or more arguments (section 3.5.1). In the body of a function is a block of code ending with one "return" statement (section 3.5.2).

Function Grammar

```
function
     : "function " identifier "(" (var_type identifier (",
" var_type identifier)*)? ")" newline block "return" value
newline "end" newline newline
```

3.5.1 Arguments

A function is defined with one or more arguments, which are variables associated with that function whose value is set when the function is invoked (section 3.5.3). Arguments passed to a function must be of the same number and types as the function defines them. All arguments are passed by value, meaning that any change made to a variable passed as an argument to a function within that function does not affect the value of that variable in the invoking block of code.

3.5.2 Return Values

Each user-defined function must end with a line containing the keyword "return" and a value, which can be either a numeric literal or an integer variable. When a function executes, its value as a statement in the invoking block of code is considered to be the same as its return value.

3.5.3 Invocation

Functions can be called either from the main block of code or from within another function via a statement that consists of the function's name, an opening parenthesis, an optional list of arguments of the correct number and types, and a closing parenthesis.

Function Call Grammar

```
function_call
     : identifier "(" (value (", " value)*)?")"
```

3.6 Scope

M.R. Roboto incorporates a series of scope rules to determine the "visibility" of variables and user-defined functions. All scoping in this language is static, meaning that the scope of a variable or function can be completely determined at compile time.

3.6.1 Variable Scope

M.R. Roboto has no global variables. Each variable can be "seen" only by other statements in the same block of code; a variable declared in the main block of code can be affected only by other statements in that main block, while a variable declared in the body of a function can be affected only by other statements in that function. The one exception to this rule is function arguments (section 3.5.1), which allow the value of variables declared in one block of code to be visible to another block of code.

3.6.2 Function Scope

All functions in this language have global scope, meaning they can be invoked from anywhere: within the main block, within another function, or even within their own body (i.e., recursive function calls are allowed).

4.0 The M.R. Roboto Library

The M.R. Roboto language has a variety of built-in library functions which the programmer can always invoke from any point in a program to perform certain specialized tasks.

4.1 Library Functions

mouseMove(int x, int y) – Move the mouse pointer to the given screen coordinates [x,y], expressed in pixels. 0,0 is considered to be the lower left corner. The mouse pointer may not move beyond the dimensions of the actual screen.
click() – Click the left mouse button once.
doubleClick() – Click the left mouse button twice.
rightClick() – Click the right mouse button once.

length(string) – Return the number of characters in the given string.
substring(string, int x, int y) – Return a substring of the given string from position x to position y-1, inclusive. Strings are zero-indexed.

getX() – Return the width of the current screen resolution, in pixels.
getY() – Return the height of the current screen resolution, in pixels.

getColor() – Return the color of the pixel currently beneath the mouse pointer.

type(string) – Mimic a sequence of key presses. Capitalized letters, or other characters normally produced by holding the Shift key, will be automatically handled by the program. Non-printing characters can be typed by including key escape codes (section 2.4.1).
press(string) – Press and release one key. The string supplied as an argument must consist of a string of characters equating to only one key press (i.e., a non-capital letter, a digit, a punctuation mark that can be produced without pressing Shift, or a key escape code).
hold(string) – Press and hold one key.
release(string) – Release a single key that is currently being held.
releaseAll() – Release all keys currently being held.

wait(int) – Pause execution for the specified number of seconds.

5.0 Code Examples

```
** Assume a text file is currently open in Notepad
** Type Hello World and then CTRL-S to save, then exit
** the program by selecting Exit from the File menu.

string x
string y
string s
int i
x = "Hello Sir"
y = "World"
s = substring(x,0,5) + " " + y
type(s)
hold("\CTRL\")
press("s")
releaseAll()
hold("\ALT\")
press("f")
releaseAll()
for i, 0 to 6
     press("\DOWN")
end
press("\ENTER\")
```

```
** Move the mouse to the lower left corner of the screen.
** Click on the Windows Start button, move up and select
** an item from the menu.

int x
x = 10
moveMouse(x,10)
click()
x = x + 90
moveMouse(x,10)
click()
```

```
** Move the mouse until it's no longer over a red pixel,
** wait a bit, then inform the user of that fact.

int x
x = 10
while(getColor() == RED)
    x = x + 1
    moveMouse(x, 50)
end
if(getColor() == WHITE)
    displayWhite()
end

function displayWhite()
    wait(10)
    type("The mouse is over a white pixel.")
    return 1
end
```

6.0 Appendix A: The Complete M.R. Roboto Grammar

```
digit -> [0-9]
letter -> [A-za-z]
alphanumeric -> letter | digit
string -> """ (.)* """
integer -> (digit)+
pixelcolor -> ("BLACK"|"BLUE"|"DARK_GRAY"|"GRAY"|"GREEN"|
"LIGHT_GRAY"|"MAGENTA"| "ORANGE"|"PINK"|"RED"|"WHITE"|"YELLOW")
newline -> "\n"

identifier -> letter (alphanumeric | "_")*
filename -> identifier ".mmr"
var_type -> "int" | "string" | "color"
value -> integer | identifier

program -> block newline (function)*
block -> (stmt)+
stmt -> (comment | declaration | conditional | loop | function_call |
expr) newline

comment -> "**" (printing)*
declaration -> var_type identifier
conditional -> "if(" expr ")" newline block
      ("else" newline block)? "end"
loop -> while_loop | for_loop
function_call -> identifier "(" (identifier ("," identifier)*)? ")"
expr -> expr1 ("=" expr1)*
expr1 -> expr2 ("or" expr2)*
expr2 -> expr3 ("and" expr3)*
expr3 -> expr4 (equalityop expr4)*
expr4 -> expr5 (plusop expr5)*
expr5 -> expr6 (multop expr6)*
expr6 -> "(" expr ")" | value | string

equalityop -> "==" | "<" | ">" | "<>" | "<=" | ">="
plusop -> "+" | "-" | "++"
multop -> "*" | "/"

for_loop -> "for" identifier "," value "to" value ("step" value)?
newline block "end"
while_loop -> "while" expr newline block "end"

function -> "function" identifier "(" (var_type identifier
      ("," var_type identifier)*)? ")" newline block "end" newline
      newline
```