TiMRS Timers, Made Readable and Simple

Language Reference Manual

Jeff Kline	jk4209	-	Systems Architect
Faisal Rahman	fr2422	-	Language Guru
Daniel Rindone	dcr2165	-	Project Manager
Eric Webb	edw2139	-	Tester

COMS 4115 - Programming Languages and Translators
Prof. Stephen Edwards

Spring 2021

1:

Table of Contents

1.0 Introduction

TiMRS is a mixture of Python and C orientated syntax. The use of Python keeps the programming of timers simplistic, while incorporating C allows the compiler to parse easier and allow more direct program memory management, should it be needed. A major component of running a TiMRS script is providing the user with a graphical clock and some indication of progress throughout the timing routine.

This manual will go into depth to describe the TiMRS language. The goal of this language is to give users a new way to easily design and script complex timers for any task. This method allows one to code complex timing procedures using TiMRS' built-in timer types with customizable durations and functionalities. These include accounting for repetition, intervals, multiple processes, saving sessions, and other techniques to track and perform any task where it may be useful.

Our language aims to provide a simple solution to implement flexible and dynamic timing systems for multiple different uses. This paper is structured to be used as a guide to properly and effectively implementing this language.

This manual will emulate the C Language Reference Manual and the Python Language reference manuals found here, respectively:
https://www.bell-labs.com/usr/dmr/www/cman.pdf
https://docs.python.org/3/reference/

2.0 Lexical Conventions

In general, whitespace is ignored except in the case where it serves to separate tokens. Newlines are acknowledged as a separator of statements. The tab spacing follows Pythonic characteristics and belongs to the previous statement in a block of code as it will be covered in **3.0 Syntax Notation.**

3.0 Syntax Notation

Throughout the course of this manual, specific syntax as it is reflected in the TiMRS language will be designated by the `Courier New` font style.

The newline character, as it does in Python, acts as the statement terminator as mentioned previously. Code blocks are indicated by the use of a tab character. Please see below for a use case:

Example:

```
// 5 times it runs the following:
5 rounds:
     // 2 minutes followed by 10 seconds
     2 min then print(“Done”)
     10 sec then run(“./game/chess”)
     5 min then alert(“STOP”)
     // 30 seconds then 15 seconds 2 times
     2 rounds of:
          30 sec
          15 sec
     // 3 minutes
     3 min
     // 30 seconds
     30 sec
```

3.1 Comments

Comments will be introduced by using a double forward slash, an example is provided below

Example:

```
// this is a comment
```

Everything following the comment syntax and before a new line is to be considered a comment. Block comment capability has deliberately not been included in the TiMRS language and comments do not have the ability to nest.

3.2 Keywords

For this language, there are certain phrases that hold special value and are reserved as keywords that are not recommended to be used otherwise:

- in
- def
- is
- True
- False

3.3 Constants

There are three constants found in this language, further explanation of each constant is described in the following sections:

Numerical Constants:

- Integers
- Floating point numbers

String Constants:

- Strings

3.4 Numerical Constants

In the TiMRS language, a numerical constant refers to any collection of numbered digits. Only whole integers and decimal float numbers are recognized, and all other classifications are excluded from recognition. All numbers must be non-negative.

Example:

Whole numbers:

```
34 sec
4 min
1 hr
```

Float numbers:

```
55.0 sec
4.4 min
0.3 hr
```

3.5 String Constants

A string constant for the TiMRS language is indicated through any given sequence of numbers and characters that are surrounded by double quotation marks. These marks are represented as:

Example:

```
// start 5 min timer with label
5 min "heat oven"
```

A string literal classifies under the datatype '`string`' and is terminated by a null byte `\0` to indicate the end of the string. In TiMRS, the backslash character `\` is used for escaping characters in the string. `\n` is used for creating a newline character and `\t` is used for creating a tab.

3.6 Data Types

There are four primitive data types recognized in TiMRS:
`int`, `float`, `string`, and `bool`

Additionally, four complex data types exist as well:
`Timer`, `hr`, `min`, and `sec`

Please see examples of use below:

Primitive data types:

Type	Description
`int`	Integer Example: 10
`float`	Floating point number Example: 4.5
`string`	String: Example: “hello”
`bool`	Boolean Example: True

Complex data types:

Type	Description

`Timer`	A struct within TiMRS that is composed of:
	
	`string label`
	`hr`
	`min`
	`sec`

`hr`	int value, 3600 seconds Example: `2 hr`
`min`	int value, 60 seconds Example: `3 min`
`sec`	float value, 1 second Example: `1 sec`

4.0 Functions

The user is able to formulate more specific user-defined functions in addition to the standard TiMRS library which will be covered in the next section. These functions have arguments and return types that depend on the output of the given function.A function begins with the keyword `def` followed by the function's name and any number of parameters. When initially defining a function, the function's statement is preceded by finishing the function with a colon " : ". When calling a function, a colon is not needed.

Syntax:

```
// function declaration
def name(list of parameters):
     statement

// function call
name(list of parameters)
```

Example:

```
// user-defined function declaration
def cook_pizza(min x, string msg):
if (x <= 30):
          x min msg
      else:
          (x - 10) min msg
      alert(msg + "done")

// function call
cook_pizza(10, first_pizza)
```

4.1 The Standard TiMRS Library

A number of pre-constructed library functions assist with key tasks in TiMRS. These operate exactly how functions are described above, but in this case their behavior cannot be modified by the user.

Functions included in TiMRS cover a number of basic needs when constructing a timer, these include:

`alert(expression)`
Prints the expression to standard output. Accepts strings.

`run(expression)`
Accesses the path located in the expression and then proceeds to execute that task.

`pause(expression)`
Pauses the timer value included in the expression with the ability to start where it left off.

`start(expression)`
Starts a paused timer value in the expression at the time where it was paused.

`stop(expression)`
Stops the timer value included in the expression and exits that timer loop.

4.2 The `Timer` Function Call

A critical function included in the TiMRS language is the `Timer` function. This function assists the programmer in creating custom timers (and timer attributes) and identifying them for later use. This function takes the form:

Syntax:

```
Timer
    str label = "cook_pizza"
    int hr = 0
    int min = 1
    int sec = 10

// creates a timer structure labeled "cook_pizza"
Timer cook_pizza = 1 min 10 sec
```

Example:

```
cook_pizza(min x, string msg, string alt):
     x min
     print(msg)
     alert(alt)
```

5.0 Expressions and Operators

The TiMRS language follows relatively Pythonic conventions when it comes to grouping expressions. Expressions in this language include integer expressions, Boolean expressions, relational expressions, string expressions, function calls related to the `Timer` object. A more in-depth explanation of these expressions and their operators are listed in the following sections.

5.1 Integer Expressions

Expressions involving integers often include mathematical arithmetic expressions, these include use of the infix operators:

`+`, `-`, `*`, `/`, and `()`	Standard operations for mathematical arithmetic

Precedence rules follow standard conventions in TiMRS integer expressions. Operands may be used through one of the three `int` and `float` complex data types described in section **3.6 - Data Types**, or are defined as numbers expressed through the definition at **3.4 - Numerical Constants.**

Example:

```
1 hr + (2 sec * 3 min)
```

5.2 Boolean Expressions

Examples of Boolean expressions in the TiMRS language begin with either the keywords *true* or *false*.

Logical operations in TiMRS include the following:

Logical Operators:

||	Standard logical `OR` Returns true if either of the left or right boolean expressions evaluate to true. Example: `a || b`
&&	Standard logical `AND` Returns true if the left and right boolean expressions both evaluate to true, otherwise it returns false. Example: `b && a`
!	Standard logical `NOT` Applies as a unary operator to a boolean expression. Results in true if the value of the operand is false and vice-versa. Example: `!b && a`

The grouping of the above operators is from left-to-right.

Additional elements of Boolean expressions include the formation of relational expressions as defined below.

5.3 Relational Expressions

Operators to express relational expressions are of the following form:

!=	Inequality
==	Equality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Operands for these comparisons are integers. The result of the comparison is a bool whose value is true if the comparison evaluates to true. Otherwise, it returns false.

The grouping of the above operators is from left-to-right.

5.4 Operator Precedence

( )	Highest
!	
*, /, %	
+, -	
>, <, <=, >=	
==, !=	
&&	
||	
=	Lowest

6.0 Declarations

Declarations in TiMRS, as well as their initialization and assignment, take the following form and are explained in more detail in the subsequent sections:

Example:

```
// implicit declaration
first_pizza = "cheese"
second_pizza = "hawaiian"
extra_time = 30.0
```

6.1 Initialization

There are various ways to initialize variables, TiMRS-specific calls and functions. Specific examples are provided in the following section.

6.2 Variables and Assignment

Assigning values to variables in TiMRS is done using the following operator:

=	Assigns the variable on the right hand side to the variable on the left

The initialization of a variable or a timer can happen during or after declaration:

Example:

```
first_pizza
first_pizza = "cheese"
```

7.0 TiMRS-Specific Commands

TiMRS includes a number of built-in commands to handle common functions associated with timer management. Below, those commands are explained in more detail.

TiMRS-specific commands:

`# rounds`	Looping condition based on '#' value
`start`	Starts a timing event
`pause`	Pause timer
`stop`	Stop timer
`del`	Removes a timer object
`run`	Runs a file at a given location
`alert`	Print a user-defined notification to the screen

Example:

```
// simple timer that opens a file after a given period of time
loc = "./training/v1.mov"
// 5 times it runs the following:
5 rounds:
     // timer, no label
     2 min
     // timer with label
     10.0 sec "REST"
     5 min
     alert("RESET")
     // runs the file/program at loc
     run(loc)
     // 30 sec, 15 sec, 2 times
     2 rounds:
          30.0 sec "JUMP"
          15.0 sec "SIT"
     3 min
     30.0 sec
```

7.1 Control Flow and Conditional Statements

In TiMRS, `if`, `elif`, `else`, and `while` statements are used to assist in conditional statements and loops that the `rounds` command can not achieve. See below for their use cases.

7.2 If / Elif / Else Statements

In TiMRS, `if, elif, else` statements are used by implementing the following:

Syntax:

```
if (expression):
     statement
elif (expression):
     statement
else:
     statement
```

Example:

```
// if statement
if (first_pizza == “hawaiian”):
     alert(“add pineapple”)
     // timer called with an identifier
     extra_time sec
     cook_pizza(10, first_pizza)

// elif statement
elif (first_pizza == “cheese”):
     cook_pizza(8, first_pizza)

// else statement
else:
     cook_pizza(10, first_pizza)
```

7.3 While Statements

In TiMRS, `while` statements are used by implementing the following:

Syntax:

```
while (condition expression):
     statement
```

Example:

```
// while statement
int a = 0
while(a < 10):
     1 min
     a = a + 1
```

8.0 Lexical Scope

The scope of a variable in TiMRS is where the variable lives in the program and where that variable can be accessed within the code. Global variables can be accessed throughout the entirety of the code while local variables can only be accessed within their function.