SCoLang - Language Reference Manual

Table of Contents:

1. Introduction

In this document we propose, SCoLang, a strongly-typed language that is made with the premise that everything is a contract waiting to execute. Each contract in our language has two key components: listeners and actions. In a contract, we bound each action to a set of listeners. A listener is set active once a set of predefined conditions are met. When all the listeners in a contract are marked active, the associated actions in the contract are executed.

The goal behind our language is to construct programs that help eliminate inefficiency in routine tasks. For instance, one could create a contract to turn on lights if a set of predefined criteria are satisfied. In short, we envision that our language could be used for anything from simple load balancing to complex IOT applications.

2. Types

Basic Data Types:

long, float, char	Regular primitive types
Array	An aggregate data type; [] Python style (slicing, referencing, etc)
bool	Ternary data type: true, false, null
String	A text value. Can be a single char or an arbitrary number of chars concatenated together.

Advanced Data Types:

Contract	A structure that stores associated listeners with actions. At a high level, this binds listeners to actions and then triggers actions if conditions for listeners are satisfied.
Listener	A data structure that stores a boolean condition. It also handles checking to see if the condition is met. (e.g. memory usage exceeds a certain amount)
Action	A data structure that stores an event. This event is meant to be triggered by a contract if the conditions for the associated listener(s) are meant. (e.g. turn lights on)

3. Lexical Convention

Identifiers:

Identifiers consist of one or more characters with a leading alphabetic character followed by any number of alphanumeric

characters or underscores.

Keywords:

Contract	A structure that stores associated listeners with actions. At a high level, this binds listeners to actions and then triggers actions if conditions for listeners are satisfied.
Listener	A data structure that stores a boolean condition. It also handles checking to see if the condition is met. (e.g. memory usage exceeds a certain amount)
Action	A data structure that stores an event. This event is meant to be triggered by a contract if the conditions for the associated listener(s) are meant. (e.g. turn lights on)
for	standard for loop (for control flow)
resolve	Resolve a listener (conditions satisfied)
reject	Stop listening before satisfied (conditions satisfied such that it will never resolve)
if	standard if (for control flow)
elif	standard elif (for control flow)
else	standard else (for control flow)
while	standard while (for control flow)
break	Breaks the current control flow

continue	Skips the current iteration within an iterator

Comments:

```
/* Comment */
/*
    Multi-line Comment
*/
```

Operators:

Unary Operators:

!	Not

Binary Operators:

==	Equal to comparative
!=	Is not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
&&	And operator
||	Or operator

Special Operators:

=	Assignment
.	Access elements underneath composite data types
->	Binds listeners to actions to create a contract

Mathematical Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
<<	Bitshift left
>>	Bitshift right
^	Binary XOR
&	Binary AND
|	Binary OR

Punctuators:

[]	Array assignment and referencing
()	Statement Precedence
{}	Code Blocks
;	Statement boundary. Must be at the end of each statement to signal such.

4. Syntax Notation and Program Structure

Operator Precedence

The order of operators is as follows:

-> . []
!
* / %
+ -

>> <<
> < >= <=
== !=
& ^ |
&& ||
=

Declaration:

```
<type> <name>
long a;
long[] a;
Action[] actions;
```

Assignment:

Basic:

```
<type> <varname> = <varvalue>;
long a = 34333;
long[] arr = {1,2,3,4};
```

Advanced:

```
<type> <varname> = <codeblock>
Action display = {
    print(a);
}
Listener listen = {
    while(a != b)
    {
        if(a > b)
        {
            a -= b;
        }
        elif( b > a)
        {
            b -= a;
        }
    }
}
```

Binding:

```
<listener> -> <action>
Listener a = {...}
Action b = {...}
```

Control Flow:

if/elif/else structure

```
if(condition){
}
elif(condition){
..}
else {
..
}
```

for structure

```
for(initialization; terminal condition; increment) {
}
```

while structure

```
while(condition) {
}
```

Program Structure:

- Variables Declarations
- Listener Declarations
- Action Declarations
- Bindings

5. Standard Library Functions

webhook(String endpoint, int port)	Initiates a connection to a specified endpoint and port. Returns a boolean (default 0, set to 1 for a tick if the

	endpoint is requested).
print(String str)	Print a string that is passed into the function into standard out