# Module Interface Specification for SE 4G06, TRON 4TB6

Team 26, STRONE
Jordan Bierbrier
Azriel Gingoyon
Taranjit Lotey
Udeep Shah
Abraham Taha

January 18, 2023

# 1 Revision History

| Date | Version | Notes |
|---|---|---|
| Date 1 | 1.0 | Notes |
| Date 2 | 1.1 | Notes |

# 2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

# Contents

# 3 Introduction

The following document details the Module Interface Specifications for Synesthesia Wear, a wearable product that assists users by using signal processing on gathered sounds to provide appropriate feedback (via vibrations) to the user according to inputted sound configuration settings. As a result, this gives the users peace of mind knowing that if their attention is needed (doorbell, ring, name call, etc.), Synesthesia Wear will be able to alert them.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at *Team 26 Capstone GitHub Repository*.

# 4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by SE 4G06, TRON 4TB6.

| Data Type | Notation | Description |
|---|---|---|
| character | char | a single symbol or digit |
| integer | $\mathbb{Z}$ | a number without a fractional component in (-$\infty$, $\infty$) |
| natural number | $\mathbb{N}$ | a number without a fractional component in [1, $\infty$) |
| real | $\mathbb{R}$ | any number in (-$\infty$, $\infty$) |

The specification of SE 4G06, TRON 4TB6 uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, SE 4G06, TRON 4TB6 uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

# 5 Module Decomposition

The following table is taken directly from the *Module Guide Document* for this project.

| **Level 1** | **Level 2** |
|---|---|
| Hardware-Hiding Module | |
| Behaviour-Hiding Module | Login Module<br>Bluetooth connection Module<br>Keyword Selection Module<br>Output Signal Module<br>Profile Module<br>Battery Status Module |
| Software Decision Module | Sound Classification Module<br>Bluetooth Communication Module<br>Microphone Module |

Table 1: Module Hierarchy

# 6 MIS of [Module Name —SS]

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

## 6.1 Module

[Short name for the module —SS]

## 6.2 Uses

## 6.3 Syntax

### 6.3.1 Exported Constants

### 6.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| [accessProg —SS] | - | - | - |

## 6.4 Semantics

### 6.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

### 6.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

### 6.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

### 6.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

#### 6.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

# 7 MIS of Bluetooth Communication Module

## 7.1 Module

BTComuModule

## 7.2 Uses

None

## 7.3 Syntax

### 7.3.1 Exported Constants

DataTx:{ BT.send()}

### 7.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|---|---|---|---|
| RecvNewClass | DataRx | - | Class_Full |
| RmClass | DataRx | - | Class_Empty |
| UpdatePref | DataRx | - | - |
| SendBattV | BattV | DataTx | - |

## 7.4 Semantics

### 7.4.1 State Variables

None

### 7.4.2 Environment Variables

BattV: {AnalogRead(BatteryVolt)}
DataRx: {BT.recieve(T1),BT.recieve(T2),BT.recieve(T3)}

### 7.4.3 Assumptions

Connection with the application is already establised. Bluetooth tries to automatically reconnect if the application is disconnected.

### 7.4.4 Access Routine Semantics

RecvNewClass():

- transition: if(DataRx == T1) then addclass(dataRx)
- output: None
- exception: Class_Full

RmClass():

- transition: if(DataRx == T2) then rmclass(dataRx)
- output: None
- exception: Class_Empty

UpdatePref():

- transition: if(DataRx == T3) then prefchange(DataRx.class,DataRx.pref)
- output: None
- exception: None

SendBattV(BattV):

- transition: None
- output: DataTx == BattV
- exception:

# 8 Appendix

[Extra information if required —SS]