**PROJECT AUTOMATIC DISPENSER SYSTEM**

Protocol Document

Version: 0.0.2

Revised on: 10/7/2019

REVISION HISTORY

|  |  |  |
| --- | --- | --- |
| **Version No** | **Release Date** | **Revision Details** |
| 0.0.1 |  | Initial revision with frame format and communication mechanism details.  Basic commands related to data transfer. |
| 0.0.2 | 10/7/2019 | Modify some of frame format |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

TABLE OF CONTENTS

1. Introduction 5

2. System architecture 5

3. Protocol 6

3.1 Communication 6

3.1.1 From dispenser to cabling gateway 6

3.1.2 From cabling gateway to server 6

3.2 Device identification 6

3.2.1 Dispenser 6

3.2.2 Cabling gateway 6

3.3 Frame Types 6

3.3.1 Request 6

3.3.2 Response 7

3.3.3 Periodic Ping 7

3.4 Exchange Mechanism 7

3.4.1 Acknowledged Emission 7

3.4.2 Acknowledged Emission with Results 7

4. Frame Format 7

4.5 Header 7

4.5.1 Length & Complemented Length Field 7

4.5.2 Frame Type Field 8

4.5.3 Command code Field 8

4.6 Frames 8

4.6.1 Request Frame Format 8

4.6.2 Response Frame Format 8

4.6.3 Periodic Ping Frame Format 8

5. Error Management 8

6. Command Repertoire 9

6.7 Error notification – 0x01 9

6.8 Event notification – 0x02 10

6.9 Card Detect Command – 0x03 11

6.10 Bottle Code Detect Command – 0x04 11

6.11 Dispensing spirit – 0x05 12

6.12 Synchronize with server– 0x06 13

6.13 Unknown Error Command – 0x07 13

7. Annexure 14

6.1. Frame Checksum 14

8. Timeout error 14

# Introduction

This document is intended to describe the communication among a Dispenser, two cabling gateway and a communication gateway.

Document describes:

* Protocol used to communicate between controller and other devices.
* Headers of all frames
* Data of each frame
* Common errors

# System architecture

Server

Cabling gateway L1

Ethernet Switch

Cabling gateway L1

# Protocol

## Communication

### From dispenser to cabling gateway

* Interface: RS232 full duplex
* Baud Rate: 115200
* Data bits: 8
* Parity: None
* Stop bit: 1
* Cable: CAT5/CAT5e/CAT6

### From cabling gateway to server

* Interface: Virtual COM, serial to Ethernet
* Baud Rate: 10/100M
* Protocol: TCP
* Cable: CAT5/CAT5e/CAT6

## Device identification

### Dispenser

The dispenser will be identified by the port of the cabling gateway it connected.

### Cabling gateway

The cabling gateway is identified by its IP address. Its IP address is configurable through an internal web server.

## Frame Types

Message frame will have different type according to exchange direction and data.

We will have three types of frames:

* Request
* Response
* Periodic Ping

### Request

A request is a frame which is established by sender (dispenser or server) to receiver (server or dispenser). This frame type is used to request a service of receiver.

### Response

A response corresponding to a frame is established by the slave (controller) after receiving a request from master (reader). Depend on the request type, this frame can contain data or just the status of the requested action.

### Periodic Ping

A periodic ping is a frame established by master (server) towarding to the slave (dispenser) at a predefined period to update it status and get display status respectively.

## Exchange Mechanism

The mechanism is based on the frame type.

### Acknowledged Emission

This is the normal case of data transaction. A request is emitted by the requester, and after treatment of this request the addressee emits the acknowledgement frame to the requester.

### Acknowledged Emission with Results

This mechanism uses acknowledged emission and just adds an answer frame. This answer frame can be a one time frame or a periodic frame with real time data. This operation type is used by master device to send periodic data to slave device.

# Frame Format

Each frame which will be getting exchanged between slave and master will have a header and message body.

## Frame format:

* #define START\_FLAG 0x12
* #define END\_FLAG 0x13
* #define ESCAPE\_FLAG 0x7D

|  |  |  |
| --- | --- | --- |
| START\_FLAG | PAYLOAD | END\_FLAG |

## Byte Stuffing

Whenever a flag or escape byte appears in the message, it is escaped by 0x7D and the byte itself is XOR-ed with 0x20. So, for example 0x7E becomes 0x7D 0x5E. Similarly 0x7D becomes 0x7D 0x5D. The receiver unsuffs the escape byte and XORs the next byte with 0x20 again to get the original

## PAYLOAD

All payload will use following format

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 5 | 6..7 | DATA | CKS |
| L | ID | CMD | FT | COMMANDNUMBER |  |  |

* ID: device ID (1byte)

When sending from the dispenser, the dispenser does not need to fill this field. This will be updated by the cabling gateway.

* FT: frame type (1byte)
* CMD: This is a hexadecimal command code attributed for individual functionality of master-slave interface.
* L: length of frame (1byte), the length exclude the start\_flag, ifself, checksum and end\_flag
* COMMANDNUMBER (2byte): the number of command being sent.
* DATA: all data in the payload
* CKS: 8 bit checksum, excluding start and end flag.

### Length Field

For the length the value will be depend on the number of bytes in the frame. These value is in range from 0 to 255, excluding the start\_flag and end\_flag and checksum

### Frame Type Field

Values for frame type filed are as given below:

|  |  |
| --- | --- |
| 0x00 | Invalid Frame Types (NOT supported) |
| 0x01 | Request Frame Type |
| 0x02 | Response Frame Type |
| 0x03 | Periodic Ping Frame Type |
| 0x04 – 0xFF | Reserved for future use |

### Command code Field

Command codes are different for functionalities of slave.

## Frames

### Request Frame Format

Bytes

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6..7 |  |  |  |  |  |  | N |  |
| Start | Frame Len | ID | CMD | FT | COMMANDNUMBER | Request Data | | | | | | Frame Cks | END\_FLAG |
|  | L |  | CMD | 0x01 |  | D0 | D1 | D2 | … | Dn-1 | Dn | CKS |  |

The ID field is the port number of the cabling gateway that the server is talking to. If the dispenser send data to server, the ID field will be filled on the cabling gateway.

### Response Frame Format

Bytes

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6..7 | 6 | 7 | 8 | … | N-2 | N-1 | N |  |
| Start | Frame Len | ID | CMD | FT | COMMANDNUMBER | Error Code | Response Data | | | | | Frame Cks | END\_FLAG |
|  | L |  | CMD | 0x02 |  | R | D0 | D1 | … | Dn-1 | Dn | CKS |  |

ID field is used for the cabling gateway to forward data to the right port.

### Periodic Ping Frame Format

Bytes

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6..7 | 6 |  |
| Start | Frame Len | ID | CMD | FT | COMMANDNUMBER | Frame Cks | END\_FLAG |
|  | L |  | CMD | 0x03 |  | CKS |  |

Frame Cks (checksum): sum of all bytes received (including Start byte)

# Error Management

If the dispenser receives a command with error, it does not response. The server need to resend after timeout.

The same with the server when it receive a command from dispenser.

~~The system need to know & aware of all error cases. This information shall appear in the ‘~~**~~R~~**~~’ field in the Response/Acknowledgement frame.~~

~~Timeout:~~

* ~~T1: b/n 10 bytes received (10 x baud rate?)~~

~~Generic error codes are as given below.~~

|  |  |
| --- | --- |
| **~~Error Code~~** | **~~Description~~** |
| ~~0x00~~ | ~~NO Error / ACK~~ |
| ~~0x01~~ | ~~Unrecognized Frame Type~~ |
| ~~0x02~~ | ~~Unrecognized Command~~ |
| ~~0x03~~ | ~~Length not matching with received frame (after T1 finish receiving)~~ |
| ~~0x04~~ | ~~Bad Checksum in received packet~~ |
| ~~0x05~~ | ~~Invalid Data~~ |
| ~~0x06~~ | ~~NAK~~ |

# Command Repertoire

Commands and responses between dispenser and server are defined under this section.

|  |  |
| --- | --- |
| **Command Code** | **Description** |
| 0x01 | Error notification |
| 0x02 | Event notification |
| 0x03 | Card detection |
| 0x04 | Bottle bar code detected |
| 0x05 | Dispensing spirit |
| 0x06 | Synchronize dispenser with server |
| 0x07 | Update status to server |
| 0x08 | Lock the bottle |
| 0x09 | Check connection |

## Error notification – 0x01

This command is used for the dispenser to report error to the server.

Request Frame: from dispenser to server

|  |  |  |
| --- | --- | --- |
| 1 | 0x12 | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | ID | Port number of the cabling gateway |
| 4 | CMD | Command byte: **0x01** |
| 5 | FT | Frame Type byte: **0x01** |
| 6..7 | COMMAND NUMBER | This number will increase for every command send to server |
| 8 | ERROR | Error code |
| 9 | CKS | Frame Checksum byte |
| 10 | END\_FLAG |  |

Response Frame: from server to dispenser

|  |  |  |
| --- | --- | --- |
| 1 | 0x12 | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | ID | This field is update by the cabling gateway |
| 4 | CMD | Command byte: **0x01** |
| 5 | FT | Frame Type byte: **0x02** |
| 6..7 | COMMAND NUMBER | This number is the command number the server received |
| 8 | ACK | Generic Ack / Error byte |
| 9 | CKS | Frame Checksum byte |
| 11 | END\_FLAG |  |

Error code byte:

* 0x00: Normal.
* 0x01: Error – Not sufficient liquor.
* 0x02: Error – Bottle is removed without RFID
* 0x03: Error-Bottle is installed without RFID
* 0x03: Eror-RFID reader not response
* 0x04: Eror-Bar code reader not response

## Event notification – 0x02

This command is used by Dispenser an event occurs.

Request Frame: from Dispenser to server

|  |  |  |
| --- | --- | --- |
| 1 | START\_FLAG | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | ID |  |
| 4 | CMD | Command byte: **0x02** |
| 5 | FT | Frame Type byte: **0x01** |
| 6-7 | COMMAND NUMBER | This number will increase for every command send to server |
| 8 | Event Code | event |
|  | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

List of event and data

|  |  |
| --- | --- |
| Event code | Event |
| 0x02 | Start refilling |
| 0x03 | Cup refilled |
| 0x04 | Bottle installed |
| 0x05 | Bottle Removed |

Response Frame: from server to dispenser

|  |  |  |
| --- | --- | --- |
|  | START\_FLAG | Start of Frame |
|  | L | Frame Length byte |
|  | ID |  |
|  | CMD | Command byte: **0x02** |
|  | FT | Frame Type byte: **0x02** |
|  | COMMAND NUMBER | This number is the command number the server received |
|  | DATA | REMAINING SHOT (if new bottle installed) |
|  | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

## Card Detect Command – 0x03

This command is used by dispenser when bartender swipe the card.

Request Frame: from dispenser to server

|  |  |  |
| --- | --- | --- |
| 1 | 0x12 | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | ID |  |
| 4 | CMD | Command byte: **0x03** |
| 5 | FT | Frame Type byte: **0x01** |
| 6-7 | COMMAND NUMBER | This number will increase for every command send to server |
| 8 | D0 | UID lengh |
| 9 | D1 to Dn | UIDbytes (MSB to LSB) |
|  | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

UID length (n):

* 4: 4bytes type
* 7: 7bytes type
* 10: 10bytes type

Response Frame: from server to dispenser

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0xAA | Start of Frame | |
|  | L | Frame Length byte | |
|  | ID |  | |
|  | CMD | Command byte: **0x03** | |
|  | FT | Frame Type byte: **0x02** | |
|  | COMMAND NUMBER | | This number is the command number the server received |
|  | ACK | Generic Ack / Error byte | |
|  | D0 | Authentication Status | |
|  | D1 | Card holder role | |
|  | D2..Dn | Bartender name | |
|  | CKS | Frame Checksum byte | |
|  | END\_FLAG |  | |

Authentication Status:

* 0x00: NFC tag authenticated
* 0x01: NFC tag NOT authenticated

Card holder role:

* 0x00: Unauthenticated
* 0x01: Normal bartender

0x02: Manager

Name:

D2..Dn: Bartender name or “Unrecognized”

## Bottle Code Detect Command – 0x04

This command is used by dispenser when a bottle is scanned.

Request Frame: from dispenser to server

|  |  |  |
| --- | --- | --- |
| 1 | 0xAA | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | ID |  |
| 4 | CMD | Command byte: **0x04** |
| 5 | FT | Frame Type byte: **0x01** |
| 6-7 | COMMAND NUMBER | This number will increase for every command send to server | |
| 8 | D0 | Barcode length (n) |
| 9 | D1 to Dn | Barcode (MSB to LSB) |
|  | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

UID length (n): depend on spirit

Response Frame: from server to dispenser

|  |  |  |
| --- | --- | --- |
|  | 0xAA | Start of Frame |
|  | L | Frame Length byte |
|  | ID |  |
|  | CMD | Command byte: **0x04** |
|  | FT | Frame Type byte: **0x02** |
|  | COMMAND NUMBER | This number is the command number the server received | |
|  | ACK | Generic Ack / Error byte |
|  | D0 | Authentication Status |
|  | D1 | Compare |
|  | D2 | Remaining shot |
|  | D3..Dn | Spirit name |
|  | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

Authentication Status:

* 0x00: Authorized to install the bottle
* 0x01: The bottle is not authorized to install

D1..Dn: Bottle name or “Unrecognized”

Compare:

* 0x00: The same spirit name as previous session one
* 0x01: Difference spirit name

## Dispensing spirit – 0x05

This command is used by dispenser when the POUR button is pressed.

Request Frame: from dispenser to server

|  |  |  |
| --- | --- | --- |
| 1 |  | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | ID |  |
| 4 | CMD | Command byte: **0x05** |
| 5 | FT | Frame Type byte: **0x01** |
| 6-7 | COMMAND NUMBER | This number will increase for every command send to server |
| 8 | D0 | RFID UID length |
| 9 | D1..Dn | RFID UID |
| 10 | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

UID length (n):

* 4: 4bytes type
* 7: 7bytes type
* 10: 10bytes type

Response Frame: from server to dispenser

|  |  |  |
| --- | --- | --- |
|  |  | Start of Frame |
|  | L | Frame Length byte |
|  | ID |  |
|  | CMD | Command byte: **0x05** |
|  | FT | Frame Type byte: **0x02** |
|  | COMMAND NUMBER | This number is the command number the server received |
|  | ACK | Generic Ack / Error byte |
|  | D0 | Total shot the bartender made on this section |
|  | D1 | Remaining shot of the bottle |
|  | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

## Synchronize with server– 0x06

This command is used by dispenser when it need to synchronize with server, e.g. replace the dispenser if the current one is broken.

Request Frame: from dispenser to server

|  |  |  |
| --- | --- | --- |
| 1 |  | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | ID |  |
| 4 | CMD | Command byte: **0x06** |
| 5 | FT | Frame Type byte: **0x01** |
| 6-7 | COMMAND NUMBER | This number will increase for every command send to server |
| 8 | CKS | Frame Checksum byte |
| 9 | END\_FLAG |  |

Response Frame: from server to dispenser

|  |  |  |
| --- | --- | --- |
|  |  | Start of Frame |
|  | L | Frame Length byte |
|  | ID |  |
|  | CMD | Command byte: **0x06** |
|  | FT | Frame Type byte: **0x02** |
|  | COMMAND NUMBER | This number is the command number the server received |
|  | ACK | Generic Ack / Error byte |
|  | D1 | Remaining shot |
| 7 | D2..Dn | Bottle Name |
| 9 | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

## Update status

This command is used by dispenser to update its status to server.

Request Frame: from dispenser to server

|  |  |  |
| --- | --- | --- |
| 1 |  | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | ID |  |
| 4 | CMD | Command byte: **0x07** |
| 5 | FT | Frame Type byte: **0x01** |
| 6-7 | COMMAND NUMBER | This number will increase for every command send to server |
| 8 | D0 | RFID reader status |
| 9 | D1 | Barcode reader status |
| 10 | D2 | Cup status |
| 11 | D3 | Bottle presence status |
| 12 | CKS | Frame Checksum byte |
| 13 | END\_FLAG |  |

Status:

* 0x00: OFF/NOTFILLED/NOT AVAILABLE
* 0x01: ON/FILLED/AVAILABLE

Response Frame: from server to dispenser

|  |  |  |
| --- | --- | --- |
|  |  | Start of Frame |
|  | L | Frame Length byte |
|  | ID |  |
|  | CMD | Frame Type byte: **0x07** |
|  | FT | Command byte: **0x05** |
|  | COMMAND NUMBER | This number is the command number the server received |
|  | ACK | Generic Ack / Error byte |
|  | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

## Display message (TODO, not this version)

Request Frame: from server to dispenser to display message on the screen.

|  |  |  |
| --- | --- | --- |
| 1 |  | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | CMD | Command byte: **0x07** |
| 4 | FT | Frame Type byte: **0x01** |
| 5..6 | COMMAND NUMBER | This number will increase for every command send to server |
| 5 | D0..Dn | Message |
| (6+n) +1 | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

Response Frame: from server to dispenser

|  |  |  |
| --- | --- | --- |
| 1 |  | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | CMD | Command byte: **0x06** |
| 4 | FT | Frame Type byte: **0x02** |
| 5..6 | COMMAND NUMBER | This number is the command number the server received |
| 5 | ACK | Generic Ack / Error byte |
| 9 | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

## Lock the bottle (TODO, not this version)

Request Frame: from server to dispenser to lock the bottle

|  |  |  |
| --- | --- | --- |
| 1 |  | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | CMD | Command byte: **0x08** |
| 4 | FT | Frame Type byte: **0x01** |
| 5..6 | COMMAND NUMBER | This number will increase for every command send to server |
| 5 | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

Response Frame: from server to dispenser

|  |  |  |
| --- | --- | --- |
| 1 |  | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | CMD | Command byte: **0x08** |
| 4 | FT | Frame Type byte: **0x02** |
| 5..6 | COMMAND NUMBER | This number is the command number the server received |
| 5 | ACK | Generic Ack / Error byte |
| 9 | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

## UnLock the bottle (TODO, not this version)

Request Frame: from server to dispenser to lock the bottle

|  |  |  |
| --- | --- | --- |
| 1 |  | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | CMD | Command byte: **0x09** |
| 4 | FT | Frame Type byte: **0x01** |
| 5..6 | COMMAND NUMBER | This number will increase for every command send to server |
| 5 | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

Response Frame: from server to dispenser

|  |  |  |
| --- | --- | --- |
| 1 | 0xAA | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | CMD | Command byte: **0x09** |
| 4 | FT | Frame Type byte: **0x02** |
| 5..6 | COMMAND NUMBER | This number is the command number the server received |
| 5 | ACK | Generic Ack / Error byte |
| 9 | CKS | Frame Checksum byte |
|  | END\_FLAG |  |

## Check connection

Request Frame: from dispenser to server to check for connection

|  |  |  |
| --- | --- | --- |
| 1 |  | Start of Frame |
| 2 | L | Frame Length byte |
| 3 | ID |  |
| 4 | CMD | Command byte: **0x08** |
| 5 | FT | Frame Type byte: **0x01** |
| 6-7 | COMMAND NUMBER | This number will increase for every command send to server |
| 8 | CKS | Frame Checksum byte |
| 9 | END\_FLAG |  |

Response Frame: from server to dispenser

|  |  |  |
| --- | --- | --- |
|  |  | Start of Frame |
|  | L | Frame Length byte |
|  | ID |  |
|  | CMD | Command byte: **0x0A** |
|  | FT | Frame Type byte: **0x02** |
|  | COMMAND NUMBER | This number is the command number the server received |
|  | ACK | Generic Ack / Error byte |
|  | CKS | Frame Checksum byte |

# Annexure

## Frame Checksum

Frame checksum is 16-bit value, which is calculated by summing all the bytes (except checksum bytes) in the frame.

# Timeout error

After timeout, if there are no response from server, the dispenser will resend the command. If after 5 times trying and no success, the dispenser shows the Waiting for server screen and continuously check for connection with server with check connection command.

If connection return to normal state, the dispenser returns to the previous state.

# Server software

## Developing environment

Windows server (testing on window 10)

Visual Studio

Database: MySQL

## Server software block diagram

## Database for server

The server has

### Device information

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Status | Spirit barcode | Spirit Name | Dispenser Volume(15/30) | EstimatedRemaining shot | Dispenser shot | Name |

### Spirit information (synchronize from POS)

|  |  |  |  |
| --- | --- | --- | --- |
| Barcode ID | Spririt name | Volume |  |
|  |  |  |  |

### User information (Synchronize from POS)

|  |  |  |  |
| --- | --- | --- | --- |
| RFID UID | Name | Role (Manager/bartender) |  |
|  |  |  |  |

### Operation log (update by the server software)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dispenser ID | RFID UID | Event type | Data | Timestamp | Result |

|  |  |
| --- | --- |
| Event type | Description |
| 0 | Initialization |
| 1 | RFID scanned |
| 2 | Barcode scanned |
| 3 | Dispensed |
| 4 | Bottle Removed |
| 5 | New Bottle installed |
| 6 | Timeout |

## Interface between dispenser and server:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dispenser ID | RFID UID | Event type | Data | Result |

|  |  |  |  |
| --- | --- | --- | --- |
| Event type | Description | Data from dispenser to server | Data from server to dispenser |
| 0 | Initialization | Cup status/Bottle Status/RFID status/Barcode status |  |
| Synchronize | SYNC request | ID/Spirit name/Remaining shot |
| 1 | RFID scanned | UID | Valid/Invalid |
| 2 | Barcode scanned | UID/Barcode | Valid/Invalid/name |
| 3 | New Bottle installed | UID | Remaining shot |
| 4 | Bottle Removed | UID |  |
| 5 | Dispensed | UID | Remaining shot/Total shot |
| 6 | Lock/UnLock the bottle |  | Command (LOCK/UNLOCK) |
| 7 | Periodic report | Cup status/Bottle Status |  |
| 8 | Report duration set |  | Duration to send the report (sec) |
| 9 | Timeout | Current RFID UID |  |

## Interface between server to POS:

### RFID table

|  |  |  |  |
| --- | --- | --- | --- |
| RFID UID | Name | Role (Manager/bartender) |  |
|  |  |  |  |

### Barcode table

|  |  |  |  |
| --- | --- | --- | --- |
| Barcode ID | Spririt name | Volume |  |
|  |  |  |  |

### Operation log (push from server to POS)

|  |  |  |  |
| --- | --- | --- | --- |
| RFID UID | Event type | Data | Timestamp |
| Bartender RFID | Dispense | Spirit type | Server timestamp of the dispensing |

# Dispenser attributes:

## Status:

ONLINE/OFFLINE

## State-substate:

Init, ready, bottle error, replace bottle, fatal error

## RFID UID:

Current UID, maximum 10 byte or WAITING

## Bartender name-role:

Name:

Current bartender name/UNAUTHENTICATED, maximum 10 byte

Role:

Bartender/Manager/NONE

## Spirit barcode:

Maximum 80 bytes or NONE

## Spirit name-initremainingshot:

Maximum 80 bytes or NONE

## Dispense action:

START/STOP

## Remaining shot:

2 bytes.

## Command from server:

100 bytes.

## Response from dispenser:

100 bytes.

# Cabling gateway:

Cabling gateway consists of serial switch (7 to 1 serial port switch, implemented on TIVA C) and a MQTT gateway (Raspberry PI in this case)

serial

Server

Raspberry PI

Device

Switch

Ethernet

serial

serial

Device

Data flow: (this version we don’t implement ACK)

Device send request to gateway

Gateway and forward this request to server

Server check database and send response

Gateway send response to device