Version <1.0>

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <10/08/16> | <1.0> | Initial specification | Akshay Raj Dayal |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

2. Overall Description 4

3. Specific Requirements 5

3.1 Functionality 5

3.1.1 API for high level communication logic 5

3.1.2 Hierarchal State Machine Design 5

3.1.3 Modularity with protocol specification 5

3.2 Implementation 6

3.2.1 State specification 6

3.2.2 State Descriptions : 6

3.3 Reliability 7

3.3.1 Testing 7

3.4 Supportability 7

3.4.1 Use standard software development conventions 7

3.4.2 Modular code development to facilitate maintenance 7

3.5 Licensing Requirements 7

# Introduction

Described in the document is the requirement analysis and initial implementation details for a high level hierarchal state machine based communication API.

## Purpose

Provide a high level communication protocol with error handling capabilities. The aim is to provide a state machine design which can be common for a variety of communication interfaces . The low level protocol details would be part of the specific implementation.

## Scope

Communication API v1

# Overall Description

1. Product function

This state machine based design provides a generic communication protocol for any communication channel. A file based data transfer method is used to test the protocol implementation.

1. Motivation

The motivation to write such a system is derived from the fact that low level communication protocols like serial or SPI usually have no flow control or error checking capabilities and are adapted for running over the physical layer. A robust and generic protocol stack built on top of the low level protocol will add robustness to the communication method will not compromising on the modularity of the protocol implementation itself.

# Specific Requirements

## Functionality

### API for high level communication logic

#### High level communication API for low level communication protocols

#### Error validation and handling schemes to be implemented

#### Packet framing and flexibility with respect to frame size, maximum retransmission retries, etc

#### Support for Server Client communication implemented over file (read/write) channel

#### Support for sending/receiving multiple data-types with configuration options on client/server end

#### Support for exception handling in disconnect/timeout events

### Hierarchal State Machine Design

#### State Design Pattern utilized to implement

#### Support for video stream compression

#### Monochrome streams in configuration

#### Resize stream in configuration

* + - 1. Stream locally saved videos/snapshots

### Modularity with protocol specification

#### Vendor specific protocols to be adopted data communication over desired channel

#### Configuration options in high level API for framing, power settings, timeout duration, etc.

## Implementation

### State specification

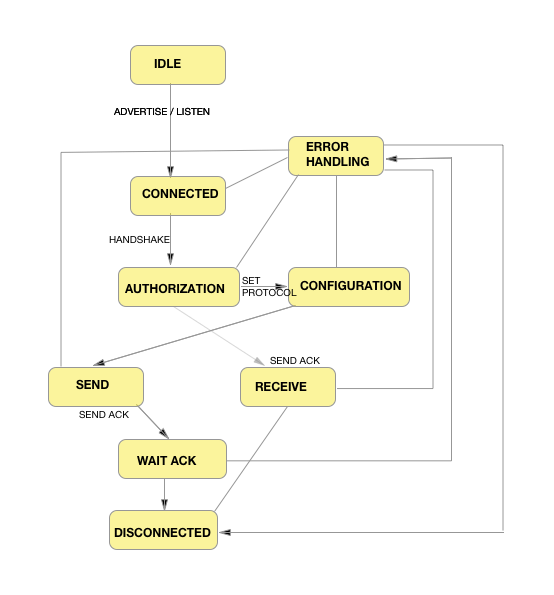


Figure 1: Server/Client States

* State machine has following states:
  + Idle
  + Connected
  + Authorization
  + Send
  + Receive
  + Disconnected
  + Configuration
  + Error Handling
* Server State Context provides state instances and a state machine interface is exposed to the user in the from of an API

### State Descriptions :

* Idle: Server/Client in idle do may be connected/disconnected but do not execute active send or receive
* Connected: The connected state is a result of successful handshake between server and client (3-way) .
* Authorization : before proceeding to configuration state, the client must authorize with server (passphrase)
* Configuration : The configuration state is responsible for specifying High level data transfer protocol parameters like Packet Length, maximum retransmission retires, encryption standard, timeout duration between sends, to the server(client). This configuration is then transmitted to the client(server) using a single packet exchange with acknowledgement.
* Send : Data is sent in this state to the client (server) . The api provides abstraction of low level protocol details. Vendor specific protocol may be used to send the data. First the data is received from the user and pushed to a local byte buffer. The send state remains active until the byte buffer is empty or an error is encountered. This state executes methods to package user data from the byte buffer into ‘dataframes’ which are configured according to the configuration settings. An acknowledgement counter is maintained in this state corresponding to packet headers. In this case, a file write is used as a low level protocol to test the implementation.
* Receive: data is received by the client(server) and stored in a local byte buffer after packages are reassembled by the Packet class methods.
* Disconnected : May occur voluntarily or as a by-product of error-handling
* Error Handling: The error handling state is used to handle exceptions in the data transfer between server/client. A state may throw exceptions which are stored in an Error structure internally and the machine transitions to error handling. A reference is then made to the internally stored error structure to determine nature of error. Any new operation on the state machine will first need to ensure that the error has been handled before transitioning to next state.

## Reliability

### Testing

The state machine operation includes several test cases to test most practical scenarios of data communication between server/client.

## Supportability

### Use standard software development conventions

Javadoc generated for entire codebase.

### Modular code development to facilitate maintenance

Revision control and modularity maintained for quick development.

## Licensing Requirements

* Ref: http://www.oracle.com/technetwork/java/javase/terms/license/index.html