WinCan

ProTouch

# ProTouch API

Version: 1.1

Date: 1.06.2022

# Table of contents

# Overview

## Document purpose

This document is meant to be a documentation of the ProTouch API that gives our users the possibility to connect with ProTouch, interact, display texts on video and control vendor's device using our application's user interface. ProTouch API is a network API to control device that do not certainly needs to be available on the same machine. There is a possibility to control devices in the same network that our application is connected to.

## Current Version

Version: 2022.1.1

## Data format

Line-delimited JSON

## Protocol

Standard TCP will be used as a communication protocol over configurable port (available on ProTouch settings – Device Control tab).

# API Documentation

## Messages struct

A standard message should have a following structure:

```
type MessageName = "OBJECT_STATUS_IND" |
                   "METER_COUNTER_STATUS_IND" |
                   "CHANGE_OBJECT_VALUE_REQ" |
                   "CHANGE_OBJECT_VALUE_RESP" |
                   "MOVE_OBJECT_REQ" |
                   "MOVE_OBJECT_RESP" |
                   "ACTION_OBJECT_TRIGGER_REQ" |
                   "ACTION_OBJECT_TRIGGER_RESP" |
                   "CREATE_FREE_TEXT"


type MessageType = "CONTROL" | "MONITORING" | "OSD" | "VIDEO"


type Message = {
    header: {
        messageId: uint // unique id for message
        messageName: MessageName // one of supported message names
        messageType: MessageType // one of supported message types
    },
    payload: {
        // specific for concrete message
    }
}
```

## Message names

Every single message has its name. This name informs ProTouch about the reason for it being sent. Available message names are:

- **OBJECT_STATUS_IND** (Message type: *Monitoring*) – this message is sent from client to server to update object status when some object's value on client side has changed.
    - Payload:
        - **object** – <string> id of object, which is to be updated, should be one of supported object, others will be ignored
        - **value** – value for updated object, negative will be ignored, server is not responsible for a validation, only displays what receives.
    - Supported objects:
        - **LevelIndicator** – <int> you can change value on indicator in percent (range 0 – 100%)
        - **Spinbox** – <int> you can change value on spinbox in percent (range 0 – 100%)
        - **SwitchButton** – <bool> you can change value on switch in boolean (true or false)
        - **Label** – <text> you can write text on label
        - **Text** – <text> you can write text on text element
    - Example message (json):
        - *{\"header\":{\"messageId\":102,\"messageName\":\"OBJECT_STATUS_IND\",\"messageType\":\"MONITORING\"},\"payload\":{\"object\":\"levelIndicator1\",\"value\":57}}*
- **METER_COUNTER_STATUS_IND** (Message type: *Monitoring*) – this message is sent from client to server to update meter counter value when meter counter's value on client side has changed.
    - Payload:
        - **value** – <float> meter counter value in specified unit – meter or feet
        - **unit** – <text> unit of value - "meter" or "feet"
    - Example message (json):
        - *{\"header\":{\"messageId\":102,\"messageName\":\"METER_COUNTER_STATUS_IND\",\"messageType\":\"MONITORING\"},\"payload\":{\"value\":320,\"unit\":\"meter\"}}*
- **CHANGE_OBJECT_VALUE_REQ** (Message type: *Control*) – this message is sent from server to client to update object value, when user click on spinbox buttons ("+" or "-") the value is changed and a new value is sent to client
    - Payload:
        - **value** – <int> new value of object
        - **object** - <string> object id which value has changed
    - Example message (json):
        - *{\"header\":{\"messageId\":3,\"messageName\":\"CHANGE_OBJECT_VALUE_REQ\",\"messageType\":\"CONTROL\"},\"payload\":{\"object\":\"spinBox2\",\"value\":6}}*
- **CHANGE_OBJECT_VALUE_RESP** (Message type: *Control*) – this message is sent from client to server to response on message CHANGE_OBJECT_VALUE_REQ. If request is accepted by client response should have status ok. If not, response should contain error.
    - Payload:

        - status – <bool> if message was accepted by client it should be true, if error status should be false
        - errorCode - <int> if message was accepted by client it should be 0, if error occurred, errorCode should be on of available error codes
        - error - <string> if message was accepted by client it should be empty, if error occurred, error should contain error description.
    - Example message (json):

        - *{\"header\":{\"messageId\":3,\"messageName\":\"CHANGE_OBJECT_VALUE_RESP\",\"messageType\":\"CONTROL\"},\"payload\":{\"error\":\"errorDescription2\",\"errorCode\":2,\"status\":false}}*

- **MOVE_OBJECT_REQ** (Message type: *Control*) – this message is sent from server to client to update object angle, when user click on a joystick the value is changed and new angle is sent to client
    - Payload:
        - **angle** – <int> angle in degrees.
        - **power** - <double> power in range 0.0 – 1.0
        - **object** - <string> object id which angle has changed
    - Example message (json):

        - *{\"messageId\":36,\"messageName\":\"MOVE_OBJECT_REQ\",\"messageType\":\"CONTROL\"},\"payload\":{\"angle\":97,\"object\":\"joystick1\",\"power\":0.4729}}*

- **MOVE_OBJECT_RESP** (Message type: *Control*) – this message is sent from client to server to response on message MOVE_OBJECT_REQ. If request is accepted by client response should have status ok. If not, response should contain error.
    - Payload:
        - **status** – <bool> if message was accepted by client it should be true, if error status should be false
        - **errorCode** - <int> if message was accepted by client it should be 0, if error occurred, errorCode should be on of available error codes
        - **error** - <string> if message was accepted by client it should be empty, if error occurred, error should contain error description.
    - Example message (json):

        - *{\"header\":{\"messageId\":1,\"messageName\":\"MOVE_OBJECT_RESP\",\"messageType\":\"CONTROL\"},\"payload\":{\"error\":\"\",\"errorCode\":0,\"status\":true}}*

- **ACTION_OBJECT_TRIGGER_REQ** (Message type: *Control*) – this message is sent from server to client to trigger action on object, when user click on a button the message is sent to client
    - Payload:
        - **object** - <string> object id which button was clicked
    - Example message (json):

        - *{\"header\":{\"messageId\":2,\"messageName\":\"ACTION_OBJECT_TRIGGER_REQ\",\"messageType\":\"CONTROL\"},\"payload\":{\"object\":\"button1\"}}*

- **ACTION_OBJECT_TRIGGER_RESP** (Message type: *Control*) – this message is sent from client to server to response on message ACTION_OBJECT_TRIGGER_REQ. If request is accepted by client response should have status ok. If not, response should contain error.
    - Payload:
        - **status** – <bool> if message was accepted by client it should be true, if error status should be false

        - **Error code** - <int> if message was accepted by client it should be 0, if error occurred, error code should be on of available error codes
        - **error** - <string> if message was accepted by client it should be empty, if error occurred, error should contain error description.
    - Example message (json):
        - *{\"header\":{\"messageId\":2,\"messageName\":\"ACTION_OBJECT_TRIGGER_RESP\",\"messageType\":\"CONTROL\"},\"payload\":{\"error\":\"errorDescription2\",\"errorCode\":2,\"status\":false}}*
- **CHANGE_METER_COUNTER_VALUE_REQ** (Message type: *Control*) – this message is sent from server to client everytime when meter counter value in ProTouch is changed.
    - Payload:
        - **value** – <double> new meter counter value
        - **unit –** <string> unit of meter counter can be “ft” or “m”
    - Example message (json):
        - *{\"header\":{\"messageId\":20,\"messageName\":\"CHANGE_METER_COUNTER_VALUE_REQ\",\"messageType\":\"CONTROL\"},\"payload\":{\"unit\":\"ft\",\"value\":24}}*
- **CHANGE_METER_COUNTER_VALUE_RESP** (Message type: *Control*) – this message is sent from client to server to response on message CHANGE_METER_COUNTER_VALUE_REQ. If request is accepted by client response should have status ok. If not, response should contain error.
    - Payload:
        - **status** – <bool> if message was accepted by client it should be true, if error status should be false
        - **Error code** - <int> if message was accepted by client it should be 0, if error occurred, error code should be on of available error codes
        - **error** - <string> if message was accepted by client it should be empty, if error occurred, error should contain error description.
    - Example message (json):
        - *{\"header\":{\"messageId\":20,\"messageName\":\"CHANGE_METER_COUNTER_VALUE_RESP\",\"messageType\":\"CONTROL\"},\"payload\":{\"error\":\"\",\"errorCode\":0,\"status\":true}}*
- **CREATE_FREE_TEXT** (Message type: *OSD*) – this message is sent from client to server to display free text using software OSD on the live video stream in ProTouch.
    - Payload:
        - **text**: <string> content of the message,
        - **x**: <number> integer informing about the column in which the text should appear,
        - **y**: <number> integer informing about the row in which the text should appear,
        - **visible**: <number> range [1;10] informing about the time that value should display for,
        - **textColor**: <Color> color of text,
        - **backColor**: <Color> color of background.
    - Example message (json):
        - *{\"header\":{\"messageId\":2,\"messageName\":\"CREATE_FREE_TEXT\",\"messageType\":\"OSD\"},\"payload\":{\"text\":\"messageText76\",\"x\":2,\"y\":4,\”visible\":6,\"textColor\":\"black\",\"backColor\":\"white\"}}*
- **START_VIDEO_STREAMING_REQ** (Message type: Video) – this message is sent from client to server when client wants to start UDP video streaming on ProTouch.
    - Payload:

        - **port** – <int> port for video streaming
    - Example message (json):
        - *{{\"header\":{\"messageId\":102,\"messageName\":\"START_VIDEO_STREAMING_REQ\",\"messageType\":\"VIDEO\"},\"payload\":{\"port\":5000}}*
- **START_VIDEO_STREAMING_RESP** (Message type: Video) – this message is sent from server to client to response on message START_VIDEO_STREAMING_REQ with IP on which streaming can be started.
    - Payload:
        - **ip** – <string> IP of ProTouch on which video can be streamed by UDP
    - Example message (json):
        - *{\"header\":{\"messageId\":102,\"messageName\":\"START_VIDEO_STREAMING_RESP\",\"messageType\":\"VIDEO\"},\"payload\":{\"ip\":\"127.0.0.1\"}}*

## Message types

There are three types of messages that can be sent. Those can be:

- Control,
- Monitoring,
- OSD,
- VIDEO

Every type is being used for a different purpose.

### Control

**Control** message is the one that is being used for device parameters **manipulation**. The user can use those messages to change values of the previously registered parameters. In the initialization phase of communication application should inform ProTouch about the parameters that should be controlled via our predefined control panel. In this moment ProTouch waits for all the elements on that panel to be initialized. Panel elements that correspond to the given parameter will only be enabled if they are previously registered.

### Monitoring

**Monitoring** message is the one that is being used for device parameters **display.** The connected device should inform ProTouch about every value change of the given parameters for e.g.:

- Meter counter value,
- Light's state and value,
- Sonde value.

*Warning! Value placeholders will display "n/a" label until the device provides the initial value.*

### OSD

**OSD** message is the one that is being used for displaying overlay texts on live video. With a help of that message the device is able to add and manage the overlay texts content, position, color and display duration.

```
type Color = "Black" | "DarkBlue" | "DarkGreen" | "DarkCyan" |
"DarkRed" | "DarkMagenta" | "Brown" | "Gray" | "DarkGray" | "Blue"
| "Green" | "Cyan" | "Red" | "Magenta" | "Yellow" | "White

type CreateFreeText = {
    header: {
        messageName: "CREATE_FREE_TEXT"
        messagetType: "OSD"
    },
    payload: {
        text: string,
        x: number, //integer
        y: number,
        visible: number //range [1;10]
        textColor: Color
        backColor: Color
    }
}
```

### Video

**Video** message is the one that is being used for configuration and management of video streaming in ProTouch.

## Dynamic UI API

Dynamic UI is orthogonal to ProTouch API, however it is required to define controls for CONTROL part of ProTouch API. It uses two ZMQ sockets:

1. Configuration, ROUTER, TCP, port 52943.
   It is used to define controls, using the following JSON shape:

```
type Joystick = {
    id: string,
    type: "Joystick"
}

type Button = {
    id: string,
    type: "Button",
    properties: {
        text: string
    }
}

type Label = {
    id: string
    type: "Label"
    properties: {
        text: string
    }
}
```

```
type SpinBox = {
    id: string
    type: "SpinBox"
    properties: {
        value: number
        stepSize: number
    }
}

type LevelIndicator = {
    id: string
    type: "LevelIndicator"
    properties: {
        chargeLevel: number
    }
}

type Switch = {
    id: string
    type: "Switch"
    properties: {
        status: boolean
    }
}

type MeterCounter = {
    id: "meterCounter1"
    type: "MeterCounter"
}

type PanelConfig = array<Joystick | Button | Label | SpinBox
| LevelIndicator | Switch | MeterCounter>

type DynamicUiSetupMsg = {
    leftPanel: PanelConfig
    rightPanel: PanelConfig
    bottomPanel: PanelConfig
}
```

2. Events (from controls) PUB, TCP, port 52944.
   Events from that socket are also forwarded in ProTouch API in CONTROL messages.

Reference implementation of this API can also be found in Simulator.

## Simulator

To integrate your application with ProTouch there is a possibility to use our open-source simulator available at:

https://github.com/WinCan/ProTouchApiSimulator

This is a very simple application that allows you to connect with ProTouch and control it with a help of ProTouch API. Windows executable and the entire source code is available on our GitHub. The application contains 4 main parts:

1. First one which is: "**Connection box**". In there you can connect to server. By default, the connection parameters are: **localhost** (127.0.0.1) on port **8095** (if they are not changed there is no need to provide them). After a click on the "Connect to server" button the application will attempt to connect into ProTouch and the status on status information bar will update.
2. The second box is for "**Monitoring messages**". There is a possibility to update every object value and send a new meter counter value to ProTouch.
    a. **OBJECT_STATUS_IND** with parameters object and value is used to update a value that is available on the addition device control panels,
    b. **METER_COUNTER_STATUS_IND** with value and unit parameters parameter is used for the meter counter value update.
3. Last box "Control message's response" is responsible for simulating response messages such as:
    a. CHANGE_OBJECT_VALUE_RESP,
    b. MOVE_OBJECT_RESP,
    c. ACTION_OBJECT_TRIGGER_RESP,
    d. CHANGE_METER_COUNTER_VALUE_RESP.

    You can create payload for these messages by providing status, errorCode and error. You can also ignore messages by clicking "Ignore messages" for example to test error handling timeout, etc.

4. The last box is "Video Message" for simulating **START_VIDEO_STREAMING_REQ** with port as parameter.
5. There is a box "Create free text" where you can send **CREATE_FREE_TEXT** message to OSD on server.
6. On the bottom there is text area where client can see messages, which are sent from server

To enable panels on ProTouch app, please use dynamic_gui_tester.py script available on the simulator repository.

1. Make sure that in ProTouch -> Settings -> Device Control -> ProTouch Api Panels you selected Dynamic UI Panels
2. Start the script and initialize panels by typing '0' ("Init panels").
3. Then enter '1' ("Show panels") to show panels.
4. Remember that you need to always do steps '2' and '3' to enable panels.

To test if video streaming by UDP is correct you can ProTouch's GStreamer libraries via UDP Video tab in simulator or call gstreamer command (below), where IP should be the same as in START_VIDEO_STREAMING_RESP and port from START_VIDEO_STREAMING_REQ:

```
gst-launch-1.0 autovideosrc is-live=true do-timestamp=true ! autovideoconvert !
queue leaky=downstream ! x264enc tune=zerolatency ! h264parse ! rtph264pay pt=96
! udpsink host=127.0.0.1 port=5000
```


Client simulator
Status information
STATUS: CONNECTED
MESSAGE WILL BE HANDLED
Connection
IP: 127.0.0.1
Port: 8095
Connect to server
Disconnect from server
Monitoring messages
Object:
Value:
Send OBJECT_STATUS_IND
Meter Counter Value:
Meter Counter Unit: meter
Send METER_COUNTER_STATUS_IND
Control message's response
status true
errorCode 0
error
Ignore messages
Video Message
Port: 5000
Start UDP video streaming
Create free text
Text
X
Y
Visible [s]
Text Color
Back Color
Black
Black
Send
Messages received:
[08:45:42][127.0.0.1:8095]: {"header":{"messageId":25,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":314,"object":"joystick1","power":0.0442}}
[08:45:42][127.0.0.1:8095]: {"header":{"messageId":26,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":0,"object":"joystick1","power":0.0313}}
[08:45:42][127.0.0.1:8095]: {"header":{"messageId":27,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":45,"object":"joystick1","power":0.0442}}
[08:45:42][127.0.0.1:8095]: {"header":{"messageId":28,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":63,"object":"joystick1","power":0.0699}}
[08:45:42][127.0.0.1:8095]: {"header":{"messageId":29,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":75,"object":"joystick1","power":0.1288}}
[08:45:43][127.0.0.1:8095]: {"header":{"messageId":30,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":78,"object":"joystick1","power":0.1593}}
[08:45:43][127.0.0.1:8095]: {"header":{"messageId":31,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":84,"object":"joystick1","power":0.3452}}
[08:45:43][127.0.0.1:8095]: {"header":{"messageId":32,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":90,"object":"joystick1","power":0.3438}}
[08:45:43][127.0.0.1:8095]: {"header":{"messageId":33,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":90,"object":"joystick1","power":0.375}}
[08:45:43][127.0.0.1:8095]: {"header":{"messageId":34,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":97,"object":"joystick1","power":0.4729}}
[08:45:43][127.0.0.1:8095]: {"header":{"messageId":35,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":97,"object":"joystick1","power":0.4729}}
[08:45:43][127.0.0.1:8095]: {"header":{"messageId":36,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":97,"object":"joystick1","power":0.4729}}
[08:45:43][127.0.0.1:8095]: {"header":{"messageId":37,"messageName":"MOVE_OBJECT_REQ","messageType":"CONTROL"},"payload":{"angle":0,"object":"joystick1","power":0}}