Technical Specification

VisionLink Web Services

Document No: TS-0027

REV	DATE	REASON FOR ISSUE	PREPARED BY	CHECKED BY	APPROVED BY
1	2020-11-24	First revision	RL	JL	GAØ
2	2021-07-08	Add path option Extended 'SetTag' tags	RL		

Table of Contents

1 Introduction

1.1 Purpose

This document specifies the web services supported by VM-110 and VX-100 VisionLink library

- REST
- JSON
- WebSockets

These interfaces allow system integrators to build applications for remote units with development tools without bindings to Qt.

1.2 Definitions and Abbreviations

Refer to document QP-0005: Definitions and Abbreviations

1.3 References

- TS-0006 VM-110 UDP EG-Link Specification
- TS-0007 EC-Link Specification

2 Revision History

Revision	Description
1	Initial revision.
2	Add path to configuration, add more accessible tags in 'settag'

3 General

VisionLink WebServices allows VisionRemote application builders to use any development tool for their applications. Prior to WebServices support, applications had to be built with Qt Qml, preferable with Qt development tools.

VisionLink WebServices gives access to notification messages, the remote units internal data, base unit (BU) data and host data by HTTP REST API and Json over UDP and WebSockets. In addition the remote unit will serve as a simple webserver.These services may be combined with Qt C++ and Qml and the VisionLink library.

4 Configuration

The webservices configuration is done by adding two text files in the applications config folder.

4.1 Services

The <applicationpath>/config/webservices.cfg is used to configure the webservices.

All webservices are disabled if webservices.cfg is missing.

The webservices.cfg contains individual sections for each service, where each section contains enable flag, host address, port assignments and interval for periodic data if configured.

If the host value is empty, the service is running on localhost (default), else an ip address may be configured for the service to be accessible outside the remote unit (may be firewall dependent).

Port assignment may be changed in case of conflicts with other systems on the remote unit.

The interval value is interval in milliseconds between periodic data messages if no changes in data.

If restapi is enabled, the REST service will respond to HTTP GET request, where the path value is the web root folder.

Note. Setting host address to the wired or wifi interface address enables development of html/rest content on external computers but violates security restrictions for armed operations and should not be used in production.

4.1.1 Default content of webservices.ini

```
[restapi]
enabled=true
host=127.0.0.1
port=6258
path=../html

[websockets]
enabled=true
host=
port=6259
interval=2000

[udpserver]
enabled=true
host=
outport=6261
inport=6260
interval=200
```

4.2 Periodic data

The <applicationpath>/config/periodicdata.csv contains tags to be sent to the application whenever data changes or periodically if no changes.

This is a standard .csv text file, where the first two rows are required headers, the following rows are name of tag and an optional flag whether any change of tag value should trigger a message to the UdpSocket and WebSocket clients. Default all tag value changes trigger a new message, but this can be disabled by setting the flag to 0.

Rows may be commented out by prefixing the tag with "#" or "//".

Note that any new message will reset the periodic timer.

4.2.1 Example of periodictags.csv

Note commented win_ tags and "c_link_rssi" and "g_link_rssi" which do not trigger new messages when their value changes but is part of data in all messages.

```
tag;trigger
Tag;Trigger
ru_identity;
state;
c_link_status;
c_link_rssi;0
g_link_status;
g_link_rssi;0
g_link_cable;
docked;
disabled;
joystick_01;
joystick_02;
joystick_03;
joystick_04;
switch_01;
switch_02;
switch_03;
switch_04;
switch_05;
switch_06;
encoder_value;
encoder_switch;
io_error;
io_active;
roll;
pitch;
#win_ru_battery;
#win_ru_charge_state;
#win_power_available;
//win_battery_design_capacity;
//win_battery_cycles;
//win_battery_health;
win_ambient_light;
wlan0ip;
eth0ip;
touch_mode;
touch_state;
```

5 JSON Interface

JSON interface is supported as either WebSockets or UdpSocket broadcast service. The UDP broadcast has less overhead than TCP services and may be lost by the application but is not available in standard HTML and require other development tools.

The Json interface consists of two different types of messages, "processdata" and "event".

Process data are messages sent periodically or when data changes. It may contain relevant information about the remote, such as state, joystick positions and similar. The message content is application configurable in config/periodicdata.csv.

Event messages are meant for event-based communication. In practice a remote procedure call (RPC) system. It is used for sending messages between the application and the remote unit or the host system via the base unit, or from the host system to the application via the base unit.

5.1 Processdata message

A "processdata" message contains message header and paired tag name and its value in standard Json format. The processdata message is sent periodically or whenever the value if one of the tags in the message changes. The "trigger" value in the message tells the cause of message, either "periodicupdate" for period expired, or the name of tag which value changed.

The header contains message type, the trigger cause, a counter and the processdata itself. The msgtype identifies the message as a processdata message, the counter is an increasing unsigned byte value to separate messages (0..255 round robin), and payload the data itself, pairs of tag:value.

Example:

```
{
  "counter":3,
  "msgtype":"processdata",
  "trigger":"periodicupdate",
  "payload":{
    "Battery Charge State":0,
    "Battery Level":0,
    "Battery capasity":0,
    "C-Link Signal Strength":0,
    "C-Link Status":0,
    "Disabled Status":0,
    "Docking Status":0,
    "Encoder Switch":0,
    .
    .
    .
    "Remote Cable Status":0,
    "Remote ID":0,
    "Remote Pitch":0,
    "Remote Roll":0,
    "State":0,
    "Switch 1":10,
    "Switch 2":0,
    "Switch 3":0,
    "Switch 4":0,
    "Switch 5":0,
    "Switch 6":0,
    "Switch 7":0,
    "Switch 8":0
  }
}
```

5.2 Event message

VisionLink creates event messages when system events occur, when system state changes, when the stop button is pressed, if the remote unit is tilted etc. Each notification message must be acknowledged by the

application. In addition, the application may send event messages to VisionLink, typically to acknowledge notification messages, and to arm and disarm the remote unit.

5.2.1 Notification message

Notification messages are sent from VisionLink to client applications.

```
{
	"msgtype":"event",
	"notification":{
		"id":<int>,
		"message":<text>,
		"popup":<bool>,
		"reference":<int>,
		"severity":<int>
	}
}
```

"id": type of notifications, see table below

"reference": unique number to be used in acknowledging the notification

"severity": 0=info, 1=warning, 1=error

"popup": bool whether notification should be made explicit visible to user, typically by a dialog to be acknowledged by OK, before the message is acknowledged.

id	Description
0	Generic (not predefined) notification. RGP can use message text as an operator message.
1	Stop function active.
2	RU drop detected.
3	RU shock detected.
4	Select failed. RU does not match selected application.
5	Arm failed because stop function (has been) active.
6	Arm failed because RU has been dropped.
7	Arm failed because RU has been exposed to excessive shock.
8	Arm failed because communication with BU cannot be established.
9	Arm failed because BU is armed against another RU.
10	Armed OK. Ensure you have full control of the system and acknowledge to enable operation.
11	Disabled due to inactivity. Acknowledge to enable operation.
12	Disabled due to tilting. Place RU horizontally and acknowledge to enable operation.
13	External stop triggered because stop function was activated.
14	External stop triggered because RU was dropped or exposed to shock.
15	External stop triggered by internal error.
16	External stop triggered by loss of control (another application selected while armed).

17	External stop triggered by loss of communication with BU.
18	Power off in armed will trigger external stop function. Please disarm first.
19	RU is powering down.

Example:

```
{
	"msgtype":"event",
	"notification":{
		"id":9,
		"message":"Arm failed because BU is armed against another RU.",
		"popup":false,
		"reference":88,
		"severity":1
	}
}
```

5.2.2 Arm request message

Message to arm the remote unit. When attempting to arm the system, a notification message is sent from VisionLink which must be acknowledged by the application for the system to change state from armed to enabled.

```
{
	"msgtype":"event",
	"eventName":”arm”
}
```

5.2.3 Disarm request message

Message to disarm the remote unit when armed.

```
{
	"msgtype":"event",
	"eventName":”disarm”
}
```

5.2.4 SetTag message

Message to set a value in VisionLink tag database. Only a limited set of tags may be set. There is no acknowledge or error of message. The user may check result by reading tag value. Rarely uses as most tags are ReadOnly.

```
{
	"msgtype":"event",
	"eventName":”settag”,
	“name”:<tagname>,
	“value”:<tagvalue>
}
```

Tag	Description

touch_mode	Touch display mode, 0 = disable, 1 = normal, 2 = glove mode
backlightIntensity	Display backlight intensity, range 0..1, where 0 is dark, 1 is bright. Default value 0.5. May be reduced to reduce battery power consumption.

Available writeable tags

5.2.5 AckNotification message

Message to acknowledge a received notification from VisionLink. The "ackNotificationRef" value must be the value received in notification message, see 5.2.1

```
{
	"msgtype":"event",
	"eventName":"acknotification",
	"ackNotificationRef":<reference>
}
```

5.2.6 CloseApp message

Message to request the launcher application to close the application. The launcher will start the dashboard application on success.

```
{
	"msgtype":"event",
	"eventName":"closeapp"
}
```

6 REST API

The REST API give access to the VisionLink tag database, alarms, Remote Unit's applications and users. It consists of HTTP GET, PUT and POST URI's depending on which service to execute.

All requests using JSON by setting the **Content-Type** to **application/json** in the header. The response is also JSON.

The REST API service is by default running at **localhost:6258/api/**

REST API configuration may be changed in webservices.ini in Applications folder.

6.1 Get data

6.1.1 Request all tag values

URL: http://localhost:port/api/1.0/tags

VERB: GET

Returns [200, payload] where payload is json data, pairs of tag name and value for each tag in VisionLink tag database.

```
{
	"payload":{
		"alarmText":"",
```

```
            "armed_ru":0,
            "bat_cap":0,
            "bu_identity":5019,
            "bu_state":16,
            "win_ambient_light":54,
            .
            .
            .
            "win_battery_cycles":0,
            "win_battery_design_capacity":0,
            "win_battery_health":0,
            "win_power_available":0,
            "win_ru_battery":0,
            "win_ru_charge_state":0,
            "windSpeed":0,
            "wlan0ip":"No IP Address"
      }
}
```

6.1.2 Request specific tag values

URL: http://localhost:port/api/1.0/tags/id=<tag1>,<tag2>…,<tagN>

VERB: GET

Returns [OK, payload], where payload is json data, pairs of tag name and value for each requested tag. If undefined tags are requesters, return is [403, error], where error is undefined tags as json data. Same format of data as above.

6.1.3 Request all alarms

URL: http://localhost:port/api/1.0/alarms

VERB: GET

Returns [200, payload] on success, else [403, error]. Payload is a Json array of alarms items, where each item contains alarm data.

```
[
  {
    "id":<int>,
    "when":<timestamp>,
    “reference”:<int>,
    “severity”:<int>,
     “msg”: <text>,
     “acked”: <bool>,
    “popup”: <bool>
  },
  .
  .
  .
  }
    "id":<int>,
    "when":<timestamp>,
    “reference”:<int>,
    “severity”:<int>,
     “msg”: <text>,
     “acked”: <bool>,
    “popup”: <bool>
  },
]
```

Timestamp is fixed format of length 19 characters, "YYYY-MM-DD HH:MM:SS", leading zeroes for all fields, 0..23 hours.

Reference is an increasing number from when the Remote Unit was powered up and is generated by the RemoteController subsystem. If the alarm is an old alarm read from the notification database, the reference is -1 and indicates this is from another power up session.

Example:


```
{
  "payload":{
    "alarms":[
      {
        "acked":false,
        "id":9,
        "msg":"Arm failed because communication with BU cannot be established.",
        "popup":false,
        "reference":1,
        "severity":0,
        "when":"2020.11.18 18:06:14"
      },
      .
      .
      .
      {
        "acked":false,
        "id":9,
        "msg":"Arm failed because communication with BU cannot be established.",
        "popup":false,
        "reference":-1,
        "severity":0,
        "when":"2020.11.18 14:03:08"
      }
    ]
  }
}
```


6.1.4 Request all alarms of specific severity

URL: http://localhost:port/api/1.0/alarms/?id=<info><warning><alarm>

VERB: GET

Returns [200, payload] if ok, else [403, error]. Payload is Json array of alarms items of specified severity. Same format as above.

6.2 Set data

Setting of data is rarely used as most of the data is ReadOnly. There are a few exceptions as touch_mode and backlight.

Returns [OK, <errors>]

6.2.1 Set tag value

URL: http://localhost:port/api/1.0/tags

VERB: PUT

payload : {<tag1>:<value>, <tag2>:<value>, … ,<tagN>:<value>}

Returns [200, OK] if all tags accepted, else [401, error]

Note: only a few internal VisionLink tags are writeable

6.3 Commands

Commands equals Json Interface events.

6.3.1 Arm request

URL: http://localhost:port/api/1.0/arm

VERB: POST

BODY: none

Returns [200, OK] if request ok, else [403, error]

Note that the arm request does not arm directly but is a request only. VisionLink will generate an ackNotification request which has to be acknowledged before the state changes (bu_state tag)

6.3.2 Disarm request

URL: http://localhost:8080/api/1.0/disarm

VERB: POST

BODY: none

Returns [200, OK] if request ok, else [403, error]

The disarm request does not disarm directly, but requests VisionLink to disarm. The result of command may be seen in "state" tag.

6.3.3 CloseApp request

URL: localhost:8080/api/1.0/closeapp

VERB: POST

BODY: none

Returns [200, OK] if request ok, else [403, error]

The command will request the launcher application to close current application and start the dashboard application.

6.3.4 AckNotify request

URL: localhost:8080/api/1.0/acknotify

BODY: { reference: <int>}

VERB: POST

Returns [200, OK] if request ok, else [403, error]

Data packets of less than 16 bytes will be padded. The data content is specified by the host system and transferred transparently to the host 'AS IS'.

6.3.5 SendMsg request

URL: localhost:8080/api/1.0/sendmsg

BODY: { msg: <data>}

VERB: POST

Returns [200, OK] if request ok, else [403, error]

Data packets may be up to 16 bytes, packages of less than 16 bytes will be padded zero. The data content is specified by the host system and transferred transparently to the host 'AS IS'.

7 WebSockets

WebSockets may be used in HTML pages to receive periodic data and notifications from VisionLink. Data content is equal to Json interface described above, but the WebSocket interface only supports acknotify commands, any other interaction with VisionLink must be by REST interface.

The WebSockets service is by default running at **localhost:6259**

WebSocket configuration may be changed in webservices.ini in Applications folder.

7.1 Periodic data

Periodic data is defined in the current applications config/periodicdata.csv. Each row holds a tag name defined in the VisionLink tag database. The first row is header and should always be "tag;".

If empty or missing file no periodic data messages will be sent, only notification events will be sent to connected clients.

The message content is equal to the JsonInterface processdata message, see 5.1

7.2 Notifications

Notifications are sent to all connected WebSocket clients. All notifications must be acknowledged by the application. Some requests also must be acknowledged to fulfil, like arming the remote unit. The message contains a boolean property 'popup', and when true, notifications should be notified to the user by a popup window. If popup is false, they may be acknowledged automatic by the application.

The message content is equal to the JsonInterface notification message, see 5.2.1.

7.3 UdpSockets

The UdpSocket server broadcasts udp periodic data messages and notifications. Clients may subscribe to theses messages instead of WebSockets. Udp messages has a lower footprint but may be lost by the application. To send messages to VisionLink, VisionLink is listening to command and notification acknowledge on another UdpSocket port.

There is no support for UdpSockets in HTML, but some web platforms, as jnode, supports UdpSockets. The Json Interface equals WebSockets, se above section.

7.4 HTML Examples

Below is some examples for HTML usage. CSS styles is used but not a scope for this document. The examples use simple HTML and java scripts just to show usage, it is not meant for real applications.

7.5 Show all tags and their value in a table

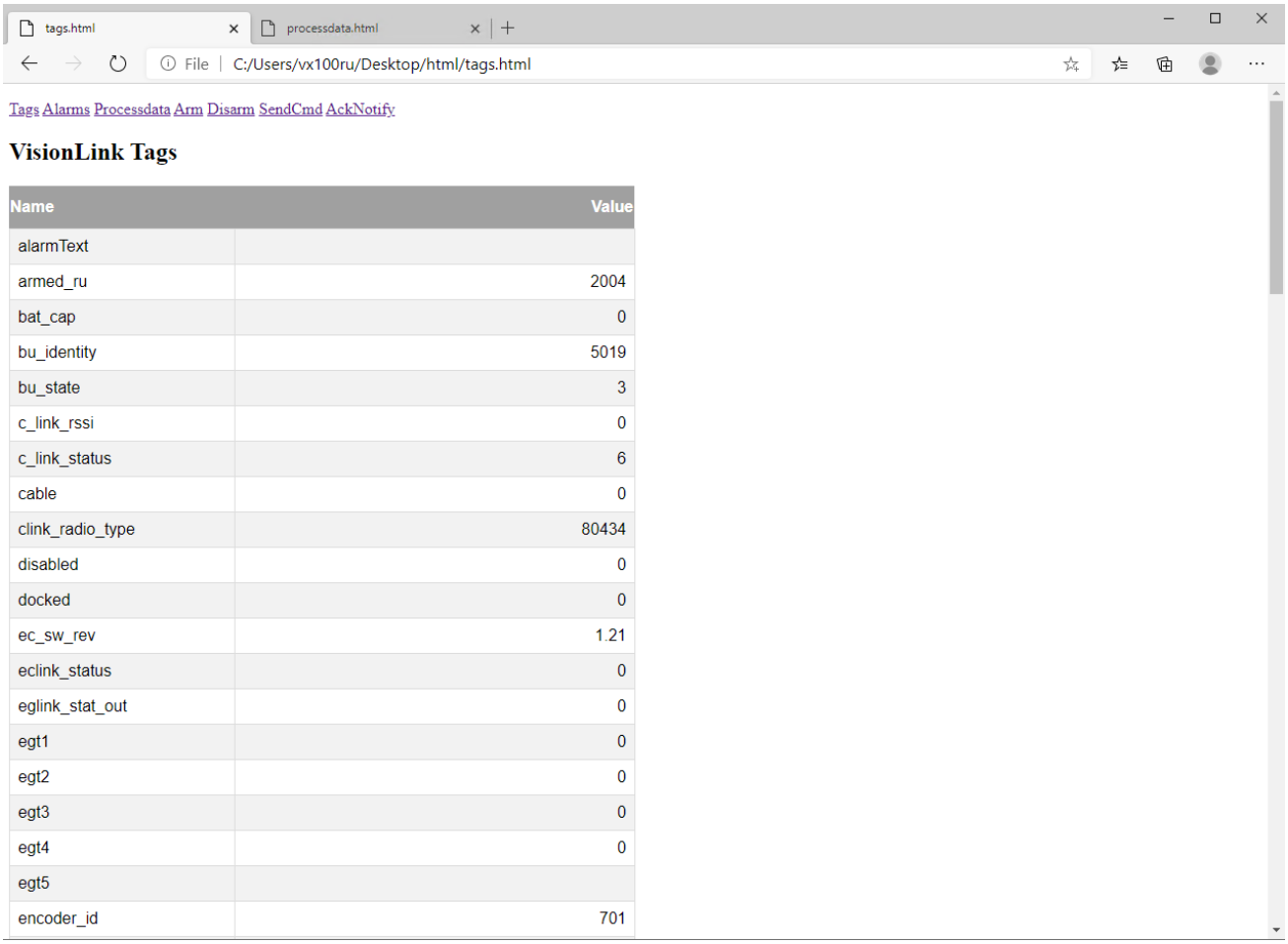
The example uses REST API to request all tags and their value and show them in a table.

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="css/style.css">
    <p>
        <a href="./tags.html">Tags</a>
        <a href="./alarms.html">Alarms</a>
        <a href="./processdata.html">Processdata</a>
        <a href="./arm.html">Arm</a>
        <a href="./disarm.html">Disarm</a>
        <a href="./sendcmd.html">SendCmd</a>
        <a href="./acknotify.html">AckNotify</a>
    </p>
</head>

<body>
    <h2>VisionLink Tags</h2>

    <p id="tags"></p>

    <script>
        var xmlhttp, msg, x, txt = "";
        xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function () {
            if (this.readyState == 4 && this.status == 200) {
                msg = JSON.parse(this.responseText);
                txt += "<table class=tables>"
                txt += "<tr><th>Name</th><th style=text-align:right>Value</th></tr>";
                for (x in msg.payload) {
                    txt += "<tr><td width=30%>" + x + "</td><td style=text-align:right>"
+ msg.payload[x] + "</td></tr>";
                }
                txt += "</table>"
                document.getElementById("tags").innerHTML = txt;
            }
        }
        xmlhttp.open("GET", "http://localhost:6258/api/1.0/tags", true);
        xmlhttp.send();
    </script>
</body>
</html>
```



7.6 Show all alarms in a table

The example uses REST API to show all alarms in a table. If the table should contain only one severity kind, add severity kind to the URL http://localhost:6258/api/1.0/alarms/?id=warning

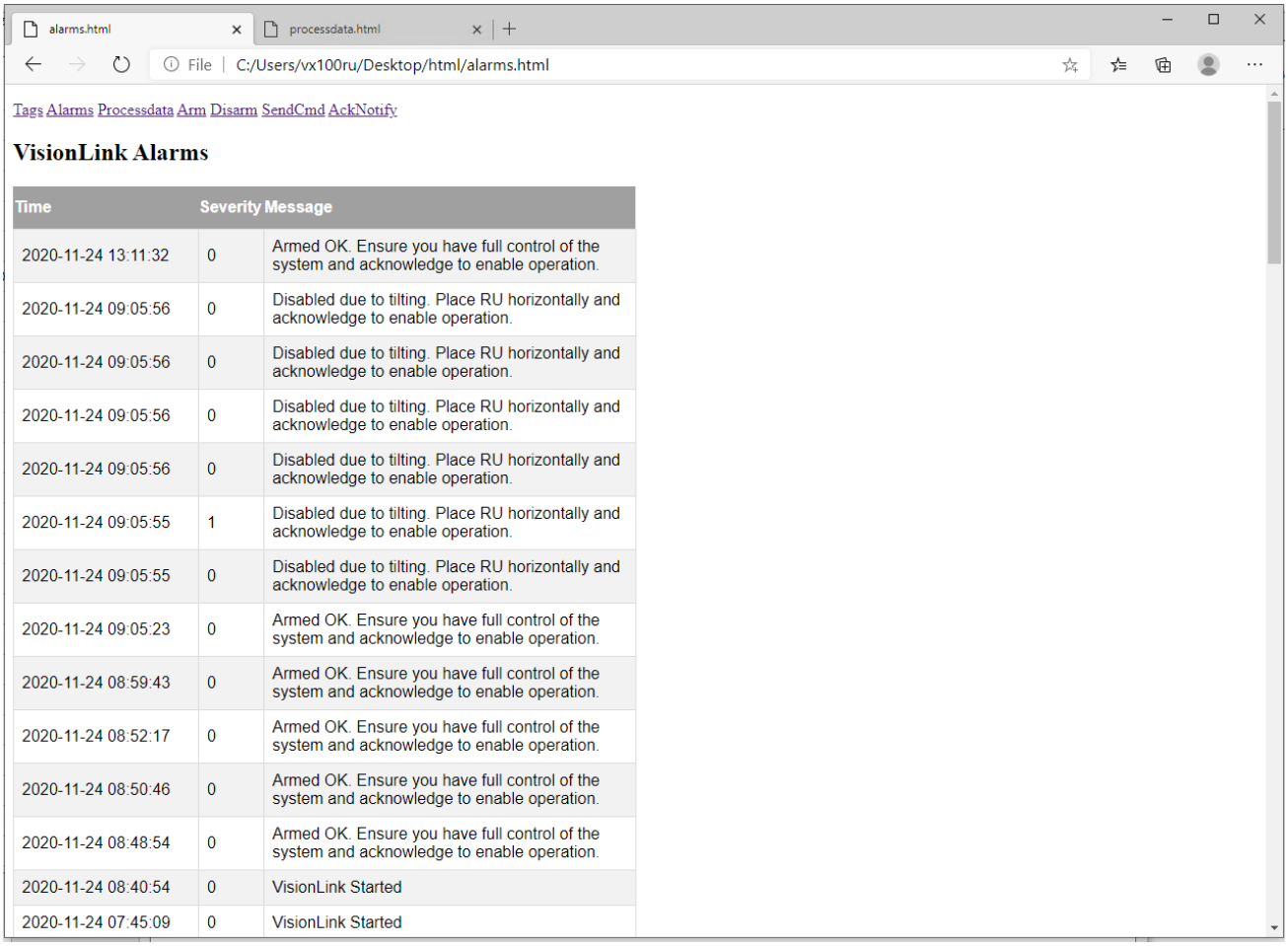
```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="css/style.css">
    <p>
        <a href="./tags.html">Tags</a>
        <a href="./alarms.html">Alarms</a>
        <a href="./processdata.html">Processdata</a>
        <a href="./arm.html">Arm</a>
        <a href="./disarm.html">Disarm</a>
        <a href="./sendcmd.html">SendCmd</a>
        <a href="./acknotify.html">AckNotify</a>
    </p>
</head>

<body>
    <h2>VisionLink Alarms</h2>

    <p id="alarms"></p>

    <script>
```

```
        var xmlhttp, msg, i, txt = "";
        xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function () {
            if (this.readyState == 4 && this.status == 200) {
                msg = JSON.parse(this.responseText);
                txt += "<table class=tables> <colgroup><col style=width:30% text-
align=center><col style=width:10%><col style=width:60%></colgroup>"
                txt += "<tr><th>Time</th><th>Severity</th><th>Message</th></tr>";
                for (i in msg.payload["alarms"]) {
                    var alarm = msg.payload["alarms"][i];
                    txt += "<tr><td>" + alarm["when"] + "</td><td>" + alarm["severity"]
+ "</td><td>" + alarm["msg"] + "</td></tr>";
                }
                txt += "</table>"
                document.getElementById("alarms").innerHTML = txt;
            }
        }
        xmlhttp.open("GET", "http://localhost:6258/api/1.0/alarms", true);
        xmlhttp.send();
    </script>
</body>
</html>
```



Time	Severity	Message
2020-11-24 13:11:32	0	Armed OK. Ensure you have full control of the system and acknowledge to enable operation.
2020-11-24 09:05:56	0	Disabled due to tilting. Place RU horizontally and acknowledge to enable operation.
2020-11-24 09:05:56	0	Disabled due to tilting. Place RU horizontally and acknowledge to enable operation.
2020-11-24 09:05:56	0	Disabled due to tilting. Place RU horizontally and acknowledge to enable operation.
2020-11-24 09:05:56	0	Disabled due to tilting. Place RU horizontally and acknowledge to enable operation.
2020-11-24 09:05:55	1	Disabled due to tilting. Place RU horizontally and acknowledge to enable operation.
2020-11-24 09:05:55	0	Disabled due to tilting. Place RU horizontally and acknowledge to enable operation.
2020-11-24 09:05:23	0	Armed OK. Ensure you have full control of the system and acknowledge to enable operation.
2020-11-24 08:59:43	0	Armed OK. Ensure you have full control of the system and acknowledge to enable operation.
2020-11-24 08:52:17	0	Armed OK. Ensure you have full control of the system and acknowledge to enable operation.
2020-11-24 08:50:46	0	Armed OK. Ensure you have full control of the system and acknowledge to enable operation.
2020-11-24 08:48:54	0	Armed OK. Ensure you have full control of the system and acknowledge to enable operation.
2020-11-24 08:40:54	0	VisionLink Started
2020-11-24 07:45:09	0	VisionLink Started

7.7 Send ARM request

The example sends an arm request to the remote unit. This page only shows how to implement arm request from a button click event, but do not respond to the required acknowledge notification sent from VisionLink. See the "WebSockets - subscribe to processdata and notifications" example for how to respond to notifications.

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="css/style.css">
    <p>
        <a href="./tags.html">Tags</a>
        <a href="./alarms.html">Alarms</a>
        <a href="./processdata.html">Processdata</a>
        <a href="./arm.html">Arm</a>
        <a href="./disarm.html">Disarm</a>
        <a href="./sendcmd.html">SendCmd</a>
        <a href="./acknotify.html">AckNotify</a>
    </p>
</head>

<body>

    <h2>VisionLink ARM</h2>

    <button id="bttn" onclick="bttnClicked()">Arm</button>

    <p id="info"></p>

    <div id="loader"></div>

    <script>
        function bttnClicked() {
            var xmlhttp;
            xmlhttp = new XMLHttpRequest();
            xmlhttp.onreadystatechange = function () {
                if (this.readyState == 1) {
                    document.getElementById("loader").style.display = "block";
                    document.getElementById("info").innerHTML = "ARMING...";
                } else if (this.readyState == 4) {
                    document.getElementById("loader").style.display = "none";
                    if (this.status == 200) {
                        document.getElementById("info").innerHTML = "SYSTEM IS ARMED";
                    } else {
                        document.getElementById("info").innerHTML = "ARM FAILED";
                    }
                }
            };
            xmlhttp.open("POST", "http://localhost:6258/api/1.0/arm", true);
            xmlhttp.send();
        }
        /* initial hide loader */
        document.getElementById("loader").style.display = "none";
    </script>

</body>
</html>
```

7.8 Send ack notification

This example is like the arm example, but it applies data to the message sent to VisionLink.

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="css/style.css">
    <p>
        <a href="./tags.html">Tags</a>
        <a href="./alarms.html">Alarms</a>
        <a href="./processdata.html">Processdata</a>
        <a href="./arm.html">Arm</a>
        <a href="./disarm.html">Disarm</a>
        <a href="./sendcmd.html">SendCmd</a>
        <a href="./acknotify.html">AckNotify</a>
    </p>
</head>

<body>
    <h2>VisionLink AckNotification</h2>

    <p id="info"></p>
    <text>
        Enter reference number to acknowledge
    </text>
    <input id="edit" type="number" value="1">
    <br><br>
    <button id="bttn" onclick="bttnClicked()">Send</button>
    <p id="result"></p>

    <script>
        function bttnClicked() {
            var xmlhttp;
            xmlhttp = new XMLHttpRequest();
            xmlhttp.onreadystatechange = function () {
                var txt = "";
                if (this.readyState == 4) {
                    if (this.status == 200) {
                        document.getElementById("result").innerHTML = "Send OK";
                    } else {
                        document.getElementById("result").innerHTML = "Send FAILED";
                    }
                }
            };
            xmlhttp.open("POST", "http://localhost:6258/api/1.0/acknotify", true);
            xmlhttp.send(JSON.stringify({ reference:
document.getElementById("edit").value }));
        }
    </script>
</body>
</html>
```

7.9 WebSockets - subscribe to processdata and notifications

This example shows the processdata and notification messages sent from VisionLink. Note that in a HTML application there always should be a one and only open WebSocket responding to notification messages from VisionLink.

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="css/style.css">
    <p>
        <a href="./tags.html">Tags</a>
        <a href="./alarms.html">Alarms</a>
        <a href="./processdata.html">Processdata</a>
        <a href="./arm.html">Arm</a>
        <a href="./disarm.html">Disarm</a>
        <a href="./sendcmd.html">SendCmd</a>
        <a href="./acknotify.html">AckNotify</a>
    </p>
</head>

<body>
    <script type="text/javascript">
        var ws = new WebSocket("ws://localhost:6259");

        ws.onopen = function () {
            console.log("Connected")
            document.getElementById("status").innerHTML = "Connected"
        };

        ws.onmessage = function (event) {
            var msg = JSON.parse(event.data);
            if (msg.msgtype == "processdata") {
                var txt = "";
                txt += "<table class=tables>"
                txt += "<tr><th>Name</th><th style=text-align:right>Value</th></tr>";
                for (x in msg.payload) {
                    txt += "<tr><td width=30%>" + x + "</td><td style=text-align:right>"
+ msg.payload[x] + "</td></tr>";
                }
                document.getElementById("processdata").innerHTML = txt;
            } else if (msg.msgtype == "event") {
                console.log(msg.msgtype, msg.eventtype, msg.notification);
                var table = document.getElementById("notifications");
                var row = table.insertRow(1);
                var cell1 = row.insertCell(0);
                var cell2 = row.insertCell(1);
                var cell3 = row.insertCell(2);
                cell1.innerHTML = msg.notification.reference;
                cell2.innerHTML = msg.notification.severity;
                cell3.innerHTML = msg.notification.message;
                if (msg.notification.popup == true) {
                    alert(msg.notification.message);
                };
                //ack received notification
                ws.send(JSON.stringify({ "msgtype": "event", "event": { "eventName":
"acknotification", "notificationAckRef": msg.notification.reference } }));
            };
        };
```

```
        ws.onclose = function () {
            console.log("Connection is closed...");
            document.getElementById("status").innerHTML = "Disconnected"
        };

        ws.onerror = function () {
            console.log("Connection error");
            document.getElementById("status").innerHTML = "Connection error"
        }
    </script>

    <h2>VisionLink WebSockets</h2>
    <p id="status">Connecting...</p>

    <p id="processdata"></p>

    <table id="notifications" class="tables" width=80% border='1'>
        <colgroup><col style=width:20%><col style=width:20%><col
style=width:60%></colgroup>
        <tr><th>Reference</th><th>Severity</th><th>Message</th></tr>
    </table>
</body>
</html>
```