TFlow Web App

|  |  |
| --- | --- |
| *Document #* | *TFLOW-WEB-APP-001* |
| *Author* | *Andrejs Vasiljevs. REEDL Tech.* |
| *Date* | *07.02.2024* |
| *Revision* | *V0* |

Table of Contents

[Purpose 3](#_Toc158243912)

[Overview 3](#_Toc158243913)

[General requirements 3](#_Toc158243914)

[Application modules 3](#_Toc158243915)

[Application layout 4](#_Toc158243916)

[Camera 5](#_Toc158243917)

[Preview panel 5](#_Toc158243918)

[Capture settings panel 11](#_Toc158243919)

# Purpose

The document describes the behavior of TFlow Web application its requirements as well as specification to the back end.

# Overview

TFlow Web application is JS based application that provides user control for TFlow camera. The camera and its control application are not supposed for regular customers, but for other developers, OEM integrators and manufacturers. Accordingly, the application style and logic should be adopted for engineers, programmers, tester, but not for dummy customers.

The Application main functions are:

* Configure camera and its modules parameters.
* Playback and management of previously recorded videos.
* General system info, settings and diagnostics.
* \*Network subsystem configuration.

The main purpose of the application is to simplify Machine Vision algorithms development and debugging based on TFlow camera, therefore the basic functions of Camera itself are quite distinct from existing cameras’ interfaces, while other functions like System settings, File management and Network configuration are quite similar.

# General requirements

The application should be implemented using HTML Responsive Web Design technics. It should be easy and convenient to use on different platforms such as PC, Tablet and Smart Phone. For Tablets and Smartphones only most popular models can be considered. I.e. no obsolete, rare and antique models support is required.

The Application should be implemented in accordance with the provided design style guide (color scheme, controls, etc.).

# Application modules

The application consists of the following main modules:

* Camera
* \*Network
* \*System
* \*Files

Current document describes the Camera module configuration only. Other modules will be implemented in the next development waves.

## Application layout

Application should contain an extendible side bar like on the pictures below:



Figure 1. Application layout.

The side bar contains buttons or tabs for each particular module of the application:

* Camera
* Network
* System

## Camera

The Camera module is responsible for video image capture, Machin Vision processing algorithms, debug info visualization, stream recording to the local filesystem and streaming over a network.

Also, Camera tab has Video preview block with some control button and switches.

As the Camera preview block should be always in front of user, while all Camera setting may not fit to the view are, it is proposed to split Preview and Camera Settings into two panels and add a scroll bar to the Camera Setting (see Figure 2).



Figure 2. Camera module layout and responsiveness behaviour.

### Preview panel

The preview panel consists of the following sub-panels:

* Video Preview.
* Preview control.
* Playback control.

Video Preview

*[Note: as there is no an optimal solution for Preview implementation, it should agreed between backend and frontent developers].*

Video Preview is an element that decodes video stream received over ?WS or WebRTC ?.

The stream can be either MPEGTS (preferred) or if not possible, then Fragmented MP4 (ISO variant). The video is encoded by H.265 codec (aka HEVC). The H.264 and MJPEG (in really worst case scenario) can be used if necessary, but HEVC is more preferred. The input stream must be decoded and displayed as soon as possible with minimum possible delay – no buffering, dejitter and other smoothing techniques allowed. In case of network bandwidth limitation or delay it may happens that several packets arrive at the same time. In that case the only most recent frame should be displayed.

The video stream pace control is fully on the server side.

The Preview performance must be able to proceed @60FPS with max delay <100ms. The target delay is ~40ms.

*Note:* [*https://github.com/xqq/mpegts.js*](https://github.com/xqq/mpegts.js) *is it worth considering? It receives MPEGTS and convert to FMP4. Optionaly we can stream in fMP4 format directly, but then how to handle metadata?*

The MPEG-TS stream along with video data contains so called Meta Data. The Machine Vision processing module uses these Meta Data as a transport for debug info. Accordingly, if configured by a user, the debug info should be extracted from the MPEG-TS stream and depicted over the video image (https://www.oreilly.com/library/view/html5-canvas/9781449308032/ch06.html). The info is in form of primitives description - circle, rectangle, rhomb, line, dot, arrow and text with specific color and optional text. The detailed Meta Data format description is in the Chapter ???.

For that project the frame’s native sizes are 320x240, 640x512 and 384x288. On a desktop PC native size video looks too small and thus there should be an option to increase Preview window scale to 1:2 and 1:4.

2024-14-10 using wed decode can solve the problem. There is a latency config option as well (https://www.w3.org/TR/webcodecs/#enumdef-latencymode).

https://w3c.github.io/webcodecs/samples/video-decode-display/

Preview Control



Figure 3. Preview Control panel.

iP1 – Enlarge video preview window size icon.

The button toggles preview window scale 1:1🡪 1:2🡪 1:4.

iP2 – Reduce video preview window size icon.

The button toggles preview window scale 1:4🡪 1:2🡪 1:1.

iP3 – Freeze/Unfreeze video preview window icon. The icon is active in “Live” capture mode only.

Meta – Show Meta data toggle switch.

Source Selector – Source for the Video Preview panel.

Upon Enlarge/Reduce (iP1 and iP2) button press the window scale changed internally on the Web Client side. No interaction with the server is supposed.

Upon Freeze (iP3) button press the Video Preview window get freeze and not refreshed until the button is unpressed. No interaction with the server is supposed.

“Meta” – is toggle switch that enables/disables the Meta Data rendering over the Video Preview window.

Upon switch toggle its state should be set to disabled and the following request should be sent:

uri: /api/preview/set

method: post

body: {“meta” : <0|1> }

The response will contain the actual “meta” value.

The toggle switch state should be enabled back with its actual value received in response. In case of timeout or an error the toggle switch enabled back without its value update.

Source Selector – there are two options:

* Capture. Preview window shows the output of Capture module
* MVision. Preview window show the output of Machine Vision module.

Request:

uri: /api/preview/set

method: post

body: {“source” : <“capture” | “mvision”> }

Playback Control

The Playback control is an optional panel that allows simulate video capturing using previously recorded media files from the local filesystem (i.e. onboard files only).

The panel becomes active then the Capture sub-module (see below ???) is switched to Playback mode. While the Capture is in “Live” mode the panel is hidden.

Upon creation the application sends the api/playback/get request and initialize playback controls accordingly. In case of file name is not specified or there is an error, then all controls except “File open” should be disabled. The error message should be displayed in “Capture file name” field as ???.



Controls:

 **File open**.

On press application send the /api/playback/get\_dir request. The backend responds with the content of the directory specified in the request. The application shows the directory content in a modal window.

Upon click on a directory item in the window, the application sends another /api/playback/get\_dir request for a newly selected directory and window refreshed with new items from the response.

Request:

uri: /api/playback/get\_dir

method: post

body:

{

“dir” : <dir\_name>,

“mask” : <linux\_style\_file\_mask>,

}

Response:

body:

{

“name” : “my\_dir\_name”,

“items” : [

{ “name” : “a1”, “type” : <”file”|”dir”>, “size” : 100500 },

{ “name” : “a2”, “type” : <”file”|”dir”>, “size” : 100500 },

{ “name” : “a3”, “type” : <”file”|”dir”>, “size” : 100500 },

…

]

}

Upon click on a file item, the application closes the directory modal dialog and sends the /api/playback/open\_file request.

Request:

uri: /api/playback/open\_file

method: post

body:

{

“name” : <file\_name>,

}

Response:

body:

{

“name” : <file\_name>,

“error” : “”, // Omitted if no error. In case of error other

// field will be omitted.

“frames\_num” : 100500,

“frame\_rate” : 25

}

Upon response receive the application reset the Frame# field. The “Frame rate” field should not be changed upon new file open.

 **Play/Pause**.

Single button with dual state “Play” () and “Pause” ().

The button is active only when media file is open and no errors.

If in “Play” state, upon the button press, the application disables other playback controls and sends the “Pause” request (the “frame” field should be omitted):

Request:

uri: /api/playback/set

method: post

body:

{

“state” : “pause”

}

Response:

body:

{

“file\_name” : “abc”,

“error” : “”, // Omitted if no error. In case of error other

// field will be omitted.

“state” : “pause”,

“frame” : 100500, // The frame number where stream is stopped

“frames\_num” : 100500,

“frame\_rate” : -1.0, // -1.0, 0.5, 1.0, 2.0

}

Upon response the application updates Play/Pause, Frame#, Navigation bar and Frame rate Controls using the response data.

If in “Pause” state, upon the button press, the application disables other playback controls and sends the “Play” request:

Request:

uri: /api/playback/set

method: post

body:

{

“file\_name” : “abc”,

“state” : “play”

“frame” : 100500, // The frame number to start from

“frame\_rate” : -1, // -1.0, 0.5, 1.0, 2.0

}

Response:

Same as for “Pause”, except the state set to “Play”

During the playback the application should periodically send empty /api/playback/get requests and update the Playback control from the response info. The response format is the same as for /api/playback/set request.

While in “Play”mode the “Frame#” field should be disabled for edit.

 **Step**.

If in “Play” state the functionality is the same as for “Pause” button.

If in”Pause” state, then sends another “Pause” request with frame number +1.

**Navigation bar**

Upon the click the application sends the Pause request. The backend responds with the total number of frames. Then, new Navigation bar’s value calculated to the frame number and then application sends the Play request with the calculated frame number.

**Frame#**

The field is an edit box. The field is disabled while in “Play”state.

In “Pause” state the field is editable.

Upon field update the application sends the same “Pause” request as specified above with the newly entered frame number value.

**Frame rate**

The button that toggles over the states that are correspond to the following values:

[-1.0, 0.5, 1.0, 2.0]

Where:

-1 – As fast as possible.

0.5…2.0 – Normalized playback speed.

In “Pause” state – do nothing, just updates icon state.

In “Play” state – sends the “Play” request with omitted “frame” field.

### Camera Sub-modules

The TFlow backend side consists from the several modules which are connected by native linux IPC means (fifo, pipes). The simplified diagram is depicted below (Figure 4).

Due to different reasons the modules may not be always available and thus the application must follow their states.



Figure 4. Camera module internal diagram.

The Camera consists of following sub-modules:

* Capture

Captures input video stream and distribute it over other sub-modules. The input source for the capture can be either TFlow hardware sensor or previously recorded video. In case of capture from the sensor the dedicated Capture Setting panel allows configure the video driver’s parameters. In case

* Processing  
  Receives video frames from the Capture module, process them and optionally send to the streaming.

The result of processing algorithm can be either rendered directly to the frame of send to the Streaming module in form of OSD Meta data.

OSD Meta data is in form primitive description – triangle, circle, rectangle, text, etc. with some attributes like color and size.

* Streaming  
  The Streaming module, depend on configuration, receives data either from the Capture or from the Processing module. The input frames optionally mixed with rendered OSD Metadata, encoded and packed into MPEG-TS stream. Another option – OSD Metadata is not rendered, but added to the MPEG-TS stream according to the standard.

Finally the prepared stream uni/multi-casted to the local network using UDP/IP and to the connected web clients using ??? WS or WebRTC???.

Optionally the prepared MPEG-TS can be stored on a local filesystem.

* Control

The Control module is the HTTP server for the Web Clients and IPC client for other modules which are acting as IPC servers.

Control module receives WEB request, parse them, forward to the corresponding sub-module and reply with the received result.

Settings for each module combined into a collapsable panel (Figure 5).



Figure 5. Sub-module control panel.

Where:

 **Module State icon**

 – Error. Module is not connected.

 – Grey. Connected but not active.

 – Red blinking. Recording.

 – Green.

Blank – Connected and active.

**Quick Access**

The function depends on the module.

 **Extend/Collapse panel**

Self evident.

Capture module settings



Figure 6. Capture settings.

The capture module may works in two modes – Live and Playback. In Live mode the video stream is fetched from the Linux V4L2 sub-system. In Playback mode the video is captured from the device local file system. The purpose of Playback mode is not just re-play the video file, but pass video content through the Processing module.

The state “Disabled” is used to switch-off the camera and release its driver.

The configuration of the Capture module settings should be obtained from the backend. I.e. control elements added to the panel dynamically.

Each control is defined with the following JSON description.

{

“name” : <literal>,

“label” : <literal>,

“type” : <literal>,

“state” : <number>,

“value” : <depends on type>,

“size” : <number>

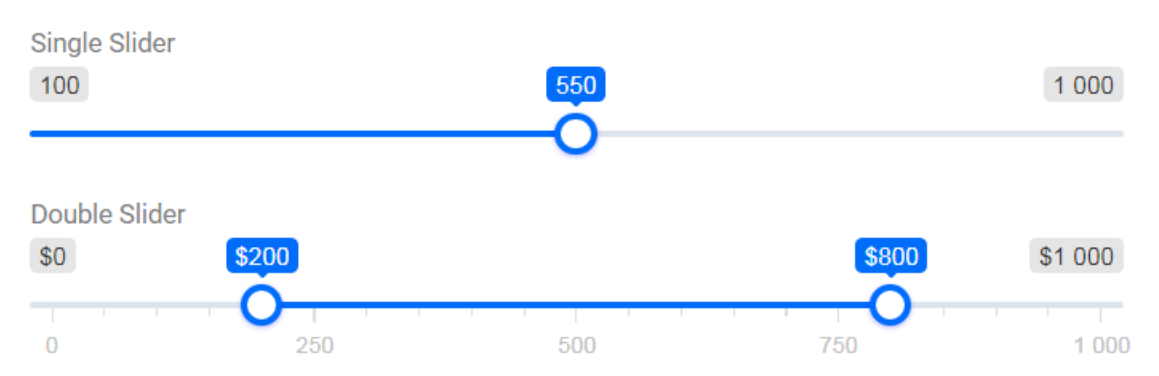
}

“name” - Control name. Not exposed to a user. For internal use only.

“label” - Text label shown in settings panel.

“type” - Type of the control:

* “edit” – edit box the value passed and stored as literals.
* “switch” – a regular switch (checkbox). The value is 0 or 1.
* “button” – The value is 1 if button is presed.
* “dropdown“ – dropdown list. The value is an array with literals, where 1st element contains current control value, while other elements are the list members.
* “slider\_v” or “slider\_h” – Slider bar vertical or horizontal. The value is an array of integer [current, min, max, size]
* “slider2\_v” or “slider2\_h” – Dual slider bar vertical or horizontal. The value is an array of integer [current\_1, current\_2, min, max, size].



“state” - State of the control. 1 – enabled, 0 – disabled.

Request:

uri: /api/capture/get

method: post

body: {}

Response:

body:

{

“hdr\_control” : { <ctrl\_0> }

“controls” : [ <ctrl\_1>, <ctrl\_2>, … ]

}

Where:

“controls” - items are JSON objects in format described above.

“hdr\_control” – the control that placed in panel upper side and accessible when a panel is collapsed.

Upon parameters change the App sends the set request with name and a current value only.

Request:

uri: /api/capture/set

method: post

body: {}

Response:

body:

{

“hdr\_control” : { <ctrl\_0> }

“controls” : [ <ctrl\_1>, <ctrl\_2>, … ]

}

In addition, the Capture module setting panel has a dedicated control – “Compression” which can be allocated in a separate collapsible panel inside the Capture settings.



Figure 7. Compression control.

The Compression panel consist of the following controls:

Current mode indicator – Text field. “8bit” if compression is enabled; “14bit” if disable.

Enable – checkbox. If compression is disabled, then there are no other controls.

Hist – checkbox. If enabled, then the histogram should be displayed on the graph under the compression line.

iC2 – Compression reset button. Returns the compression handlers back to initial state.

Grid – Shows current values of blue and green handlers. The X axis - min 0, max 16384. The Y axis – min 0, max 256.

Handlers – Movable handlers. Each handler has X and Y coordinates according to Grid axes. The handlers coordinates also shown in the editable fields.

During the handlers dragging the following rules must be followed:

1. Blue X < Green X
2. Blue Y < Green Y
3. Green X > 0
4. Blue X < 16382
5. Green pushes blue to the left if Green X == Blue X + 1
6. Blue pushes green to the right if Blue X == Green X – 1
7. Same as 5, 6 but on Y axis. Green pushes blue down, blue pushes green up.

Compression control response:

{

“name” : “compression\_v0”,

“label” : “Compression”,

“type” : “custom”,

“state” : 1,

“value” :

[

{

“name” : “compr\_en”,

“label” : “Enable”,

“type” : “switch”,

“state” : 1,

“value” : 0

},

{

“name” : “hist\_en”,

“label” : “Enable”,

“type” : “switch”,

“state” : 1,

“value” : 1

},

{

“name” : “hist”,

“label” : “Hist”,

“type” : “custom”,

“value” : [0, 10, 20, 10, 5...] // array of 16 integers

},

{

“name” : “reset”,

“label” : “Reset”,

“type” : “button”,

“state” : 1,

“value” : 1,

},

{

“name” : “h\_blue”,

“type” : “custom”,

“value” : [<x>, <y>],

},

{

“name” : “h\_green”,

“type” : “custom”,

“value” : [<x>, <y>],

}

]

}

Compression control request:

{

“name” : “compression\_v0”,

“value” :

[

{

“name” : “compr\_en”,

“value” : 0

},

{

“name” : “hist\_en”,

“value” : 1

},

{

“name” : “reset”,

“value” : 1,

},

{

“name” : “h\_blue”,

“value” : [<x>, <y>],

},

{

“name” : “h\_green”,

“value” : [<x>, <y>],

}

]

}

MVision module settings

Streaming module settings

Recording module settings

The modules setting obtained from the backend

Request:

uri: /api/<mvision|streaming|recording>/set

method: post

body: {}

Response:

body:

{

“hdr\_control” : { <ctrl\_0> }

“controls” : [ <ctrl\_1>, <ctrl\_2>, … ]

}



For ex.:

Request:

uri: /api/streaming/get

method: post

body: {}

Response:

body:

{

“hdr\_control” : {

{

“name” : “stream\_en”,

“label” : “”,

“type” : “switch”,

“state” : 1,

“value” : 1

},

“controls” :

[

{

“name” : “dest\_ip”,

“label” : “Deatination IP”,

“type” : “edit”,

“state” : 1,

“value” : “192.168.2.2”

“size” : 15

},

{

“name” : “encoder”,

“label” : “Encode”,

“type” : “list”,

“state” : 1,

“value” : [“H.265”, “H.265”, “H.264”],

},

{

“name” : “quant”,

“label” : “Quantun”,

“type” : “edit”,

“state” : 1,

“value” : “5”

“size” : 3

},

]

}

# Designers

https://99designs.com/profiles/threezeroone