Webserver interface user guide

ABSTRACT:
The document describes how to use webserver interface.
KEYWORDS:
Web server, boost, web socket, JS - JAVASCRIPT
APPROVED:

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	9-November-16	Daniel Lazar	First draft
0.2	9-January-17	Daniel Lazar	Updated function names
0.3	20-January-17	Daniel Lazar	Added JAVASCRIPT interface

Table of Contents

1 Introduction

The webserver interface allows sharing data between EVB and user's PC, mobile, TV or others. It helps to present output values like images or results and also it can be used for real-time setting of running application. The server side is written in C++. Client side can be implemented using language which supports web socket but preferred language is JAVASCRIPT. So client side will be web page. This document describes how to use web server interface that can be used for other applications where data has to be shared.

1.1 Purpose

The purpose of this document is to present the Web Server interface and help users to understand the main concepts of sharing data between server and clients.

1.2 Audience Description

This document is targeted at S32V234 Vision SDK users and Web Server interface for developers who intend to port for an application.

1.3 References

Id	*Title*	*Location*

Table 1: References

1.4 Definitions, Acronyms and Abbreviations

Term/Acronym	*Description*

Table 2: Acronyms

1.5 Document & source Location

VisionSDK: s32v234_sdk/docs/vsdk

VisionSDK: s32v234_sdk/libs/webserver

VisionSDK: s32v234_sdk/demos/other/web_server

2 Webserver Abstraction Layer

Header File webserver.hpp

API Call	*Description*
server_setup(char *address, char *port)	• Function for setup of server, address (IPV4) must be same as ip of device (EVB), port some number between 1 - 9999
run()	• Start running of server in new thread!
setOnInitEventListener(const SharedValuesListener listener)	• The server generates event when some new parameter appear. Parameter of function is pointer to some function where shared variable can be initialized
setOnUpdateEventListener(const SharedValuesListener listener)	• The server generates event when some parameter is updated from client side.
setOnCommandEventListener(const CommandListener listener)	• The server generates event when user wants to use some own communication.
getStringValueByName(std::string name, std::string default_val)	• Get **string** shared variable from server by name (HTML tag name for example). If something is wrong function returns user default value.
getBoolValueByName(std::string name, bool default_val)	• Get **bool** shared variable from server by name (HTML tag name for example). If something is wrong function returns user default value.
getIntValueByName(std::string name, int default_val)	• Get **int** shared variable from server by name (HTML tag name for example). If something is wrong function returns user default value.
getDoubleValueByName(std::string name, double default_val)	• Get **double** shared variable from server by name (HTML tag name for example). If something is wrong function returns user default value.
addOutValue(std::string name, T value)	• Prepare value for future send to clients. Parameter value can be any type (int, double, float …..)
sendOutSharedValues()	• Send every prepared values to all clients and clear buffer for next use!
streamBuffer(char* buffer, int size, std::string header)	• This function allow to stream buffer (image, own data). The user can also add some string header before buffer (for example name of canvas that will be used for drawing, width of image, height etc…)
sendCommand(std::string command)	• If some own communication has to be used. Command is sent to all clients

Table 3: API calls

2.1 JAVASCRIPT interface abstraction Layer

File evb_webserver.js

API Call	*Description*
var evb_websocket = function (wsUri, shared_inputs)	• Class for websocket connection with EVB
this.sendCommand = function (mess)	• Send own Command from client side
this.sendRaw = function (mess)	• This Function should be used in onOpen event handler
this.drawRGBimage = function (canvas, byteArray, imageWidth, imageHeight)	• Draw received RGB image
onOpen(evt, message)	• Event handler when websocket is opened
onClose(evt)	• Event handler when websocket was closed
onCommand(message)	• Event handler own Command received
onStreamReceived(header, byteArray)	• Event handler stream received
onUnrecognized(name, value)	• When some value received but any DOM not founded!
onError(evt)	• When something is wrong with websocket

3 Detailed Description

The webserver is using WebSocket communication. Used WebSocket protocol is **rfc6455.** This enables two-way communication between a client running untrusted code in controlled environment to remote host that has opted-in to communication from that code. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not reply on opening multiple HTTP connections.

The goal of using this technology for EVB is to create option how to share data which can be visualized in graphical form. Also this concept can be useful for developing image-processing algorithms because parameters can be modified immediately.

3.1 Boost library

The sources of webserver is using boost open source library. For installation few steps has to be followed. If **s32v234_sdk/3rdParty** already contains **boost** follow only step 7.

1. Download it from this link. (When this document was created, boost_1_62_0 was released)
2. Extract files to s32v234/3rdParty (So path will be **s32v234_sdk /3rdParty/boost_1_62_0/**)
3. Run bootstrap.sh for linux or bootstrap.bat (**Linux Terminal, Windows cmd**)
4. Open project-config.jam in some text editor. Find row **using gcc ;** and replace it with **"using gcc : arm : aarch64-linux-gnu-g++ ;"** this is cross compilation for **EVB S32V234.** See more
5. Run in terminal **b2 toolset=gcc-arm** (It takes a while maybe about 20 minutes)
6. Now just see s32v234_sdk/demos/other/web_server/BUILD.mk **where boost is included for compiler and linker !**
7. Now *.so.62.0 has to be copied from **<boost_dir>/stage/libs** to **<path_to_rootfs>/libs**

3.2 Communication protocol

This chapter describes communication protocol between client and server. This describes how web-socket payload looks not how rcf64555 works.

Rcf6455 basically allows two type of buffers that can be sent.

- Binary frame - buffer of bytes (0-255) so this allow to stream images for example.
- Text frame – basically it is buffer of bytes same as Binary frame but only characters can be included in (Asci table 0 – 127). If there is byte higher than 127 exception is thrown.

Every of frame starts with some string command. ***Table 4*** shows how data frame looks.

Table 4 communication protocol

Header	*Way*	*Data*	*Description*
shared_init:	Client → server	See ***Table 5***	Send init name and value to server. Server will init this inside.
shared_update:	Client → server	See ***Table 5***	Update some value
shared:	Server → client	See ***Table 5***	Server respond of value or if some ouput values has to be shared
<Own header>&	Server → client	Buffer array	Buffer streaming. Do not use **"&"** in header it is delimiter for server
Command:	Server ↔ client	Own data	Some own commands

Example of communication Strings:

Client → server – "shared_init:Tag1&100&Tag2&true&Tag3&300&". (Server adds these tags name and value to memory)

Client → server – "shared_update:Tag1&400&" (Server update Tag1 to value 400)

Server → client – "shared:Tag1&400&" (Server is sending respond to all clients, value changed!)

Server → client – "image;<width>;<height>,<format>:<buffer>"

Server ↔ client – "command:<some message> "

Table 5 shared data frame

Shared name (String)	*Delimiter*	*Shared value (String)*	*Delimiter*
Tag(1)	&	<Tag(1) value>	&
…	…	…	…
Tag(n)	&	<Tag(n) value>	&

3.3 API Description

3.3.1 Server Setup

This function has to be used first. Function setup server IP address and server port. Server can include different ports on same IP address. So for example this webserver can run under 7777 port and apache server can run under 80 port in same time.

Programmer Notes

Be sure to set IP address same as EVB is. If IP needs to be set use command: **setenv ipaddr "<IP>"** in uboot (immediately after evb start). Default IP address for EBV is 10.0.0.2 mask 255.255.255.248.

Syntax

Code 1: Server_setup syntax

```
31 void server::server_setup(char *address, char *port)
32 {
33     address_ = address;
34     port_ = port;
35 }
```

Input Parameters

Input Parameter Name	*Description*
Address	Ip address of server
Port	Port of server

3.3.2 Server run

This function starts webserver. So when this is called clients are able to connect a share data. The server is running in separated thread.

Programmer Notes

Server takes address and port from server_setup function. Default is 0.0.0.0 7777.

Syntax

Code 2: run syntax

```
1  void server::run()
2  {
3     run_t_ = boost::thread(boost::bind(&server::run_thread,this));
4     isRunning = true;
5  }
```

3.3.3 Set On Init Event Listener

This function sets on initialize listener. It means if unknown shared variable appears server generates this event. Listener is function where initialization should be done.

Programmer Notes

See input parameters.

Syntax

Code 3: setOnInitEventListener syntax

```
1  void room::setOnInitEventListener(const SharedValuesListener listener)
2  {
3      onInit.connect(listener);
4  }
```

Input Parameters

Input Parameter Name	*Description*
listener	Pointer to function. Function implementation has to be same as this template: **void <name of function>(std::string *name, std::string *value)**

3.3.4 Set On Update Event Listener

This function sets on update listener. It means some shared variable was updated by client server updates shared value inside and then generate this event. So application can react somehow (reconfiguration or something like that).

Programmer Notes

See input parameters.

Syntax

Code 4: set syntax

```
1  void room::setOnUpdateEventListener(const SharedValuesListener listener)
2  {
3      onUpdate.connect(listener);
4  }
```

Input Parameters

Input Parameter Name	*Description*
listener	Pointer to function. Function implementation has to be same as this template: **void <name of function>(std::string *name, std::string *value)**

3.3.5 Set On Command Event Listener

This function sets on command listener. It means if client sent some message that starts with "command:" server generates event.

Programmer Notes

See input parameters.

Syntax

Code 5: set on command event listener syntax

```
1  void room::setOnCommandEventListener(const CommandListener listener)
2  {
3      onCommand.connect(listener);
4  }
```

Input Parameters

Input Parameter Name	*Description*
listener	Pointer to function. Function implementation has to be same as this template: **void <name of function>(std::string *message)**

3.3.6 Get String Value by Name

This function returns string value of shared variable.

Syntax

Code 6: get string value by name syntax

```
1  std::string  room::getStringValueByName(std::string  name,  std::string
   default_val)
2  {
3     std::vector<shared_prop>::iterator item = getSharedProp(&name);
4     if(item == shared_properties.end()) return default_val;
5     return item->value;
6  }
```

Input Parameters

Input Parameter Name	*Description*
name	Name of shared variable
default_val	If variable cannot be find or something is wrong return default

Return Codes

Enumeration	*Description*
String	String value of shared variable or default value

3.3.7 Get Bool Value by Name

This function returns bool value of shared variable. Implementation is really same as previous chapter **3.3.6**.

3.3.8 Get Int Value by Name

This function returns int value of shared variable. Implementation is really same as previous chapter **3.3.6**.

3.3.9 Get Double Value by Name

This function returns double value of shared variable. Implementation is really same as previous chapter **3.3.6**.

3.3.10 Add out Value

This function prepares value for future send to all clients. It allows to share some results.

Programmer Notes

For send values function described in next chapter has to be used.

Syntax

Code 7: Add out value syntax

```
1  /// Template function for share output values
2  template <class T> void addOutValue(std::string name, T value)
3  {
4     out_shared_properties.push_back({name,
   boost::lexical_cast<std::string>(value)});
5  }
```

Input Parameters

Input Parameter Name	*Description*
Name	Name of out shared value
Value	Value (doesn't matter type)

3.3.11 Send out shared values

This function sends all prepared out value.

Syntax

Code 8: send out shared values syntax

```
1  void room::sendOutSharedValues()
   {
2    std::string message = "shared:";
3    for (uint i = 0; i < out_shared_properties.size(); i++) {
4          message += out_shared_properties.at(i).name + "&";
5          message += out_shared_properties.at(i).value + "&";
6    }
7
8    out_shared_properties.clear();
9
10   dataframe frm;
11   std::copy(message.begin(),message.end(),
   std::back_inserter(frm.payload));
12   std::for_each(participants_.begin(), participants_.end(),
13   boost::bind(&participant::deliver, _1, boost::ref(frm)));
14  }
```

3.3.12 Stream buffer

This function allows to stream buffer of bytes.

Syntax

Code 9: stream buffer syntax

```
1  void room::streamBuffer(char* buffer, int size , std::string header)
2  {
3     dataframe frm;
4
5     frm.stream = buffer;
6     frm.size = size;
7
8     header += "&";
9
   std::copy(header.begin(),header.end(),std::back_inserter(frm.payload));
10 std::for_each(participants_.begin(), participants_.end(),
11       boost::bind(&participant::deliver, _1, boost::ref(frm)));
12 }
```

Input Parameters

Input Parameter Name	*Description*
buffer	Array of chars

Input Parameter Name	*Description*
size	Size of buffer
header	Some header for buffer. Some information what buffer contains.

3.3.13 Send Command

This function allows to send some command.

Syntax

Code 10: send command syntax

```
1  void room::sendCommand(std::string command)
2  {
3     dataframe frm;
4
5     std::string header = "command:" + command;
6
7
   std::copy(header.begin(),header.end(),std::back_inserter(frm.payload));
8  std::for_each(participants_.begin(), participants_.end(),
9            boost::bind(&participant::deliver, _1, boost::ref(frm)));
10 }
```

3.3.14 JS evb websocket

This is class for communication with evb webserver.

Programmer Notes

Class in Javascript is written using function.

Syntax

Code 11: Add out value syntax

```
1  var evb_websocket = function (wsUri, shared_inputs)
2  {
3     ..
4  }
```

Input Parameters

Input Parameter Name	*Description*
wsUri	Address to EVB server, example: “ws://10.0.0.2:7777”
Shared_inputs	Array of inputs which are shared with evb. Here HTML <input> is used

3.3.15 JS evb sendCommand

Send own command message!

Syntax

Code 12: Add out value syntax

```
1  this.sendCommand = function (mess) {
2      var message = "command:" + mess;
3      websocket.send(message);
4   }
```

Input Parameters

Input Parameter Name	*Description*
mess	Message

3.3.16 JS evb sendRaw

Send raw message this function should be used in onOpen event handler.

Syntax

Code 13: Add out value syntax

```
1  this.sendRaw = function (mess)
2   {
3      websocket.send(mess);
4   }
```

Input Parameters

Input Parameter Name	*Description*
mess	Message

3.3.17 JS evb drawRGBimage

Draw RGB buffer. It can be used when evb is streaming RGB buffer. Format of RGB is same as OPENCV Mat.

Syntax

Code 14: Add out value syntax

```
1  this.drawRGBimage = function (canvas, byteArray, imageWidth, imageHeight)
   {
2      ..
3  }
```

Input Parameters

Input Parameter Name	*Description*
canvas	Canvas where image should be drawn
byteArray	Buffer of bytes where RGB image is stored
imageWidth	Width of image
imageHeight	Height of image

3.3.18 JS evb onOpen

Event handler when evb websocket is established. Here sendRaw(message) needs to be used. Because some other value pairs can be added. (`message += <name> + "&" + <value> + "&";`).

Syntax

Code 15: Add out value syntax

```
1  onOpen(evt, message){
2      .. // TODO implement code
3      <evb_websocket_instance_name>.sendRaw(message);
4  }
```

Input Parameters

Input Parameter Name	*Description*
evt	Websocket object
Message	Generated message by evb_websocket, if some other values needs to be added here check messaging style **3.2.**

3.3.19 JS evb onClose

Event handler when evb websocket is closed.

Syntax

Code 16: Add out value syntax

```
1  onClose(evt){
2      .. // TODO implement code
3  }
```

Input Parameters

Input Parameter Name	*Description*
evt	Websocket object

3.3.20 JS evb onCommand

Event handler when command is received.

Syntax

Code 17: Add out value syntax

```
1  onCommand(message){
2      .. // TODO implement code
3  }
```

Input Parameters

Input Parameter Name	*Description*
Message	Message

3.3.21 JS evb onStreamReceived

Event handler when evb is streaming some data.

Syntax

Code 18: Add out value syntax

```
1  onStramReceived(header, byteArray){
2      .. // TODO implement code
3  }
```

Input Parameters

Input Parameter Name	*Description*
Header	Header of buffer
byteArray	Buffer data

3.3.22 JS evb onUnrecognized

Event handler when name of value is not DOM object.

Syntax

Code 19: Add out value syntax

```
1  onUnrecognized(name, value){
2      .. // TODO implement code
3  }
```

Input Parameters

Input Parameter Name	*Description*
Name	Name of value
Value	Value

3.3.23 JS evb onError

Event handler when something wrong.

Syntax

Code 20: Add out value syntax

```
1  onError(evt){
2      .. // TODO implement code
3  }
```

Input Parameters

Input Parameter Name	*Description*
evt	Websocket object

3.4 Example of use

Location of demo: **S32V234_sdk/demos/other/web_server**

The function server_setup has to be called at first time with IP address and port. Than event listeners can be initialized. Later using function “run”, server starts.

The variable sharing and buffer streaming is shown in code below. This code is in server side. Demo generates simple 320x180x3 image buffer with random data. Next code shows how to get shared variables and also how to share output variables.

For running application this form of command has to be used.

./web_server <IP> <Port> so example: ./web_server 10.0.0.2 7777

Also be sure if html page stored in demo sources has same IP and Port in function OnLoad!!! And the adapter of client device IP address and port must be set to same submask and mask of network. **So for example IP 10.0.0.1 Mask: 255.255.255.248.**

During study of code notice the Tag names from ***Figure 1*** connection with code (notice red strings!).

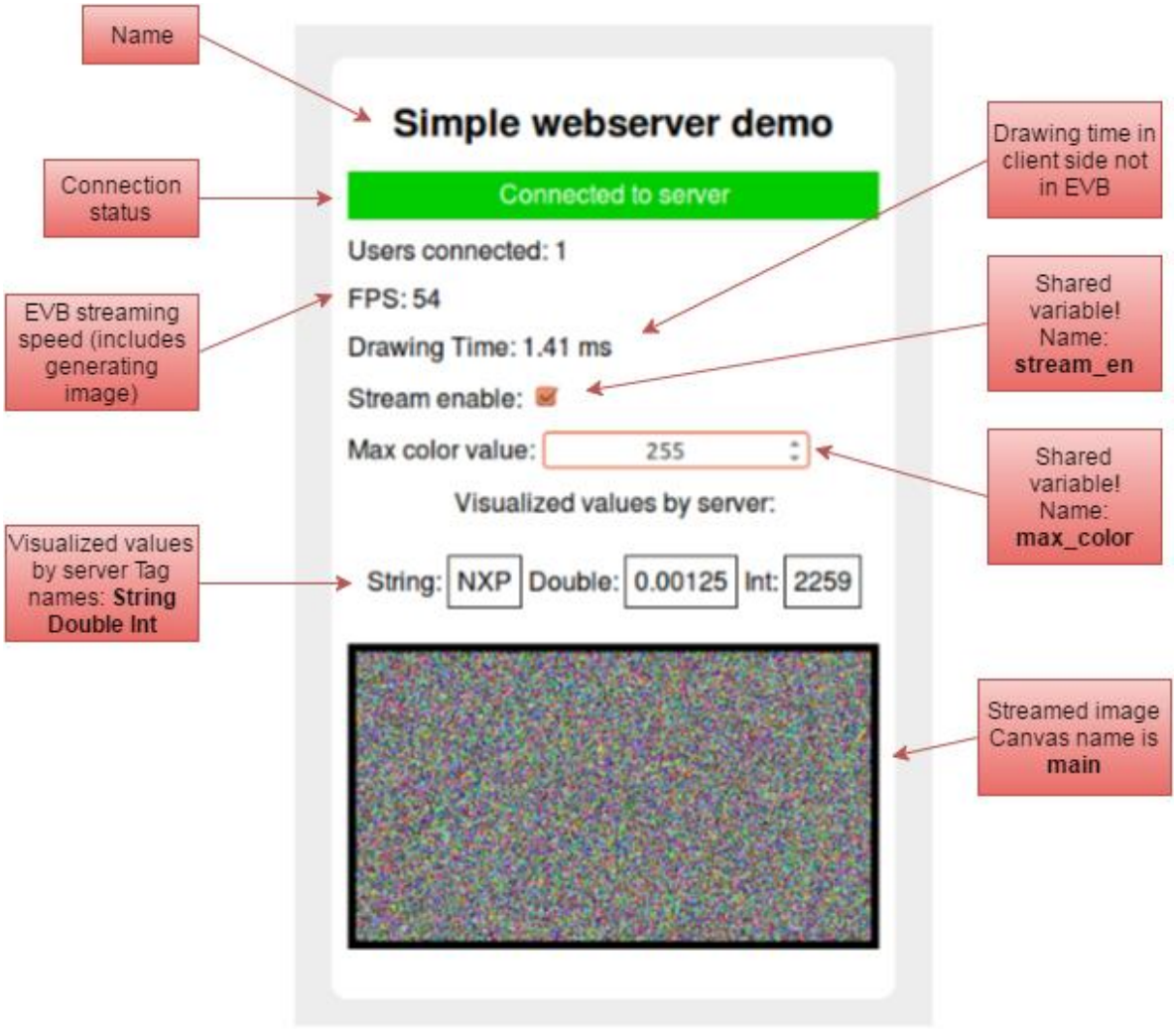

Figure 1 Web Page client

Code 21: webserver_part1 use example

```
#include "webserver.hpp"

void onInitMethod(std::string *name, std::string *value)
{
    // name compare with HTML tag
    if((*name).compare("max_color") == 0)
    {
        *value = "127";
        return;
    }
    if((*name).compare("stream_en") == 0)
    {
        *value = "true";
        return;
    }
}

void onUpdateMethod(std::string *name, std::string *value)
{
    /*
     *   Do something when some shared value was updated ! This is only event, the value was updated locally in server
     *   So use this only as trigger of your logic depends on updated value name !!!
     */}

void onCommandMethod(std::string *command)
{
    // Do something with own command !
}
```

Code 22: webserver_part2 use example

```
31 int main(int argc, char* argv[])
32 {
33   try
29   {
30     // Check command line arguments.
34     if (argc != 3)
35     {
36       std::cerr << "Usage: server <address> <port>\n";
37       std::cerr << "  For IPv4, try:\n";
38       std::cerr << "    server 0.0.0.0 7777\n";
39       std::cerr << "  For IPv6, try:\n";
40       std::cerr << "    server 0::0 7777\n";
41
42       return 1;
43     }
44
45     std::cout << "Server address: " << argv[1] << std::endl;
46     std::cout << "Server port: " << argv[2] << std::endl;
47     std::cout << "Server is starting..." << std::endl;
48
49     // Initialise the server.
50     webserver_setup(argv[1], argv[2]);
51
52     // Run the server until stopped.
53     webserver_run();
54
55     // Set listener for initialise value of shared value
56     webserver_setOnInitEventListener(&onInitMethod);
57
58     // Set listener if shared value is updated
59     webserver_setOnUpdateEventListener(&onUpdateMethod);
60
61     // Set listener if own command is used
62     webserver_setOnCommandEventListener(&onCommandMethod);
63
64     std::cout << "Server has started." << std::endl;
65     std::cout << "Press Ctrl+C (Ctrl+Break) to exit." << std::endl <<
   std::endl;
66   }
67   catch (std::exception& e)
68   {
69       std::cerr << "exception: " << e.what() << "\n";
70       return 1;
71   }
```

Code 23: webserver_part3 use example

```
// Simple image buffer 320 * 180 * 3
  int size = 172800;
  int frame_counter = 0;
  char* image = (char*)malloc(size);
  for (int i = 0; i < size; i++)
  {
      image[i] = 127;
  }

  for (;;)
  {
      // Value of HTML tag "max_color", second parametr if value max_color cant be read
      int max_color = webserver_getIntValueByName("max_color", 255);

      // Generata random image data
      for (int i = 0; i < size; i+=1)
      {
          image[i] = (char)(1 + (rand() % (int)(max_color - 1 + 1)));
      }

      // Stream image buffer if server isRunning and stream is enabled
      if(webserver_isRunning)
      {
          if (webserver_getBoolValueByName("stream_en", false))
          {
              webserver_streamBuffer(image, size, "main");
              frame_counter++;
          }
      }
       else
       {
         break; // Server is not isRunning break;
       }
       // Prepare vector with some output values that has to be shared
       std::string value = "NXP";
       webserver_addOutValue("String",value);
       webserver_addOutValue("Bool",true);
       webserver_addOutValue("Int",frame_counter);
       webserver_addOutValue("Double",0.00125);

       // Send output values to clients
       webserver_sendOutSharedValues();
  }

  return 0;
}
```

HTML, Javascript client code is shown below. Notice article part where HTML tags <input> this relates to function **onLoad**.

```
<article>

  <p id="status">Not connected</p>
  <p>Users connected: <span id="connected">0</span></p>
  <p>FPS: <span id="fps">0</span></p>
  <p>Drawing Time: <span id="drawT">0</span></p>
  <p>Stream enable: <input type="checkbox" name="stream_en" value="0" /></p>
  <p>Max color value: <input type="number" name="max_color" value="255" min="1" max="255" style="text-align:center" /></p>
  <p style="text-align:center">Visualized values by server: </p>
  <table style="margin:auto">
          <tr style="margin:0px">
                  <td><p style="text-align:center">String:</p></td>
                  <td><output style="text-align:center; border: 1px solid black; padding: 5px" name="String" disabled></output></td>
                  <td><p style="text-align:center">Double: </p></td>
                  <td><output style="text-align:center; border: 1px solid black; padding: 5px" name="Double" disabled/></td>
          <td><p style="text-align:center">Int: </p></td>
                  <td><output style="text-align:center; border: 1px solid black; padding: 5px" name="Int" disabled/></td>
          </tr>
  </table>
  <canvas name="main" width="320" height="180" style="border:5px solid #000000;"></canvas>
</article>

```

```
53      var evb_ws;
54
55      // When page is loaded
56      function onLoad()
57      {
58          // Ip address of server : port of server
59          var wsUri = "ws://10.0.0.2:7777";
60
61            /*
62             *      Select input elements in html which has to be shared
63             */
64            /*var inputs = Array();
65            inputs.push(document.getElementsByName("stream_en")[0]);
66            inputs.push(document.getElementsByName("max_color")[0]);
67            evb_ws = new evb_websocket(wsUri,inputs);*/
68
69            /*
70             *      All inputs in html is shared
71             */
72            evb_ws = new
    evb_websocket(wsUri,document.getElementsByTagName('input'));
73
74            /*
75             *      Setup evets listeners
76             */
77            evb_ws.onOpen = onOpen;
78            evb_ws.onClose = onClose;
79            evb_ws.onStreamReceived = onStreamReceived;
80            evb_ws.onError = onError;
81            evb_ws.onUnrecognized = onUnrecognized;
82      }
```

```
84  // On websocket Open
85     function onOpen(evt)
86     {
87          state.className = "success";
88          state.innerHTML = "Connected to server";
89
90           /*
91            * Here some other can be added if not just send message
92            */
93           evb_ws.sendRaw(message);
94     }
95
```

```
96      // On websocket Close
97      function onClose(evt)
98      {
99          state.className = "fail";
100         state.innerHTML = "Not connected";
101         connected.innerHTML = "0";
102     }
```

```
103     // When EVB is streaming some frame
104     function onStreamReceived(header, byteArray)
105     {
106       window.counter++;
107       var t0 = performance.now();
108       var imageWidth = 320, imageHeight = 180; // hardcoded width &
    height.
109
110       var canvas = document.getElementsByName(header)[0];
111       canvas.width = imageWidth;
112       canvas.height = imageHeight;
113       evb_ws.drawRGBimage(canvas, byteArray, imageWidth, imageHeight);
114
115       var t1 = performance.now();
116       drawT.innerHTML = (t1-t0).toFixed(2) + " ms";
117     }
118     // On websocket Error
119     function onError(evt)
120     {
121          state.className = "fail";
122          state.innerHTML = "Communication error";
123     }
124
125     function onUnrecognized(name, value)
126     {
127
128     }
```

4 Architecture (optional)

5 Tips and tricks

The webserver is implemented as **singleton asynchronous object** so methods can be called from any part in your code. For example out values can be added from different parts in code and then function “send output values” can be called from main or “streamBuffer” function can be called from anywhere too (so user can stream some image sub results).

This is same for “get values by name” function. Shared variables is accessible.

Another advantage are listeners. More than one object can set own listener. Server will connect each of them and when event is triggered all listeners are called.

If method of object will be used as listener method has to be used by next command **boost::bind(&<object_name>::<method_name>,this)**