1G RESTful API DESIGN DOCUMENT

PREPARED BY

Alexander Tyutyarev

alexander.tyutyarev@macnica.com

APR 10, 2019

Document revision history

Version	Date	Initial
0.1	10 April 2019	Document created
1.0	11 April 2019	Initial version of document
1.1	17 April 2019	Updates

Table of Contents

1. Introduction

This document describes a RESTful API and Web UI from the point of interaction with Low Level API in context of defined user stories. Since the Web UI is being built directly on top of the RESTful API which supports it, it makes sense to review them together. The document contains the following:

- Related requirements
- Derived use cases
- Set of RESTful API calls
- Low Level API class proposals

In the future the design will be extended to cover all the use cases, not only LL API-specific.

1.1 References

1G Module User Stories - https://macnica.app.box.com/file/405102090923

RESTful API description document - https://macnica.box.com/s/icl5sbb1y7qejyofnkipyprdh9xh9x7z

LL API Class full definition (for devs) - https://gitlab.com/natcheztrace/pro-av-/blob/ll_api_class/low-level-api/api/source/ll_api_class.h

2. Related requirements

Even though current number of requirements exceeds 15, list of requirements directly related to RESTful API interaction with LL API is not that big:

1. Main menu:
   As the IT professional/End User, and immediately after I've logged into the system, I want to:
   1.1. Be brought to the Main menu;
   1.2. See the ID name of the product;
   1.3. See the current status of the product, including the current state and the general configuration of the unit;
   1.4. See my options for navigating the web application;
   1.5. Have access to help;

So that I can get a general view of how the module is working, where to go next if I want to accomplish a task and if I'm lost, how to get some help.

2. Set RX/TX Mode:
   As a User/API consumer, I want to:
   2.1. Configure the product to receive HDMI from the HDMI port and send ST2110 essence flows;
   2.2. Configure the product to receiver ST-2110 essence flows and send HDMI to connected HDMI display;
   2.3. View/Get the current TX/RX mode of the device;

   So that I can plug the product into an HDMI source and send that content over my network, or I can receiver content from another module and send it to an HDMI display.

3. Get List of Available Sources:
   As a User/API consumer, I want to:
   3.1. Get/View all of the available essence streams on my network;
   3.2. See the profile of the essence streams at each device;
   3.3. See a thumbnail of the video content, updated once periodically (5-15 seconds), if available;

4. Connect to available source:
   As a User/API consumer, I want to:
   4.1. Select/post one or more compatible ST-2110 essence stream(s) from the list of available sources, up to the max quantity of the device output for each essence stream type (ancillary, video and audio);
   4.2. Easily select all of the essence streams from a single device;
   4.3. Not be able to select/connect to essence streams that are incompatible with the display device;
   4.4. Connect and display the selected sources;

   So that I can choose the content that I want to receiver from the network and show it on my HDMI display without accidentally requesting something that won't work. Usually that means grabbing the video/content from a single remote source/device.

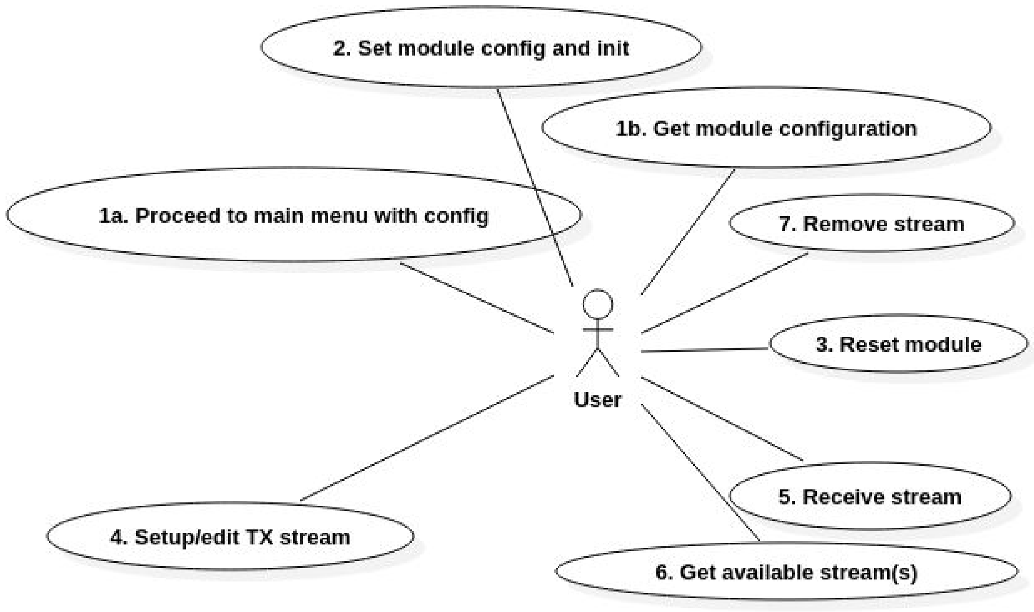
5. HDMI Content as Essense Streams
   As a Product User / API consumer, I want:
   5.1. HDMI video to be discoverable on the network
   5.2. For remote modules to be able to connect to and receive my HDMI video;

   So that when my HDMI cable is plugged in, remote modules can receive video and show it on their displays.

6. Configuration serialization:
   As a IT Admin/Integrator/API consumer, I want to:
   6.1. Get a human readable string in a standard format (example: JSON, CSV) that represents entire configuration of the module in a non-nested structure;
   6.2. Post the same string back into the system and have it apply that configuration data, ignoring missing configuration settings, retaining whatever settings were already applied;
   6.3. Post/Get the configuration payload as CSV string (with escaped returns and quotes properly handled with Row 1 as the list of keys and empty fields interpreted as no value); (?do we need it right now?)

   So that I can configure lots of systems quickly by using scripts that receive send configuration data (often from spreadsheets, which is why the config data should not be in a nested structure and the need to accept CSV) to automate configuring systems in a human readable format, is simple to use and debug.

3. Use Cases


2. Set module config and init
1b. Get module configuration
1a. Proceed to main menu with config
7. Remove stream
3. Reset module
User
4. Setup/edit TX stream
5. Receive stream
6. Get available stream(s)

This diagram describes following use cases for Web GUI/RESTful API piece.

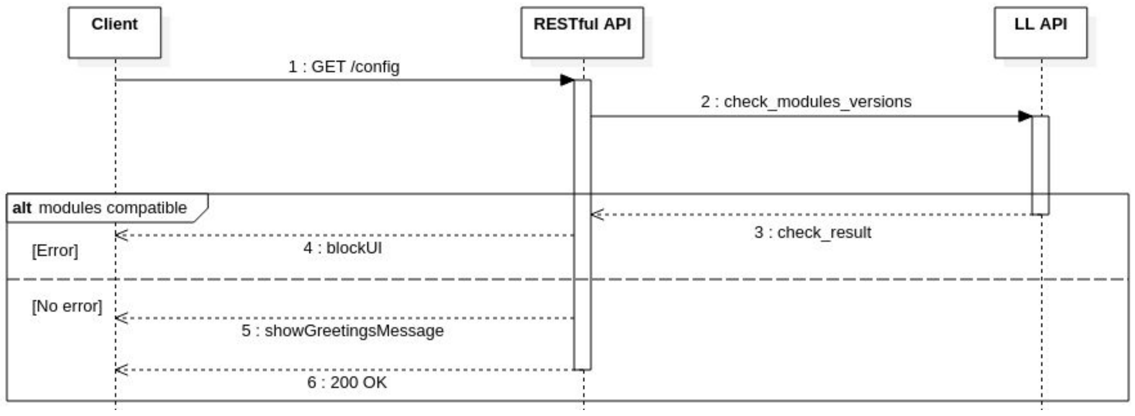
#	Use Case	Device mode	Related RESTful calls	Brief description	Related User Stories
1	a. Proceed to main menu with config (GUI) b. Get JSON object containing module configuration	any	GET /config		1.1-1.5, 6.1, 2.3
2	Set module config and init	any	POST /config		6.2, 6.3, 2.1, 2.2
3	Reset module	any	POST /reset		
4	Setup/edit TX stream	TX	PUT /stream		5.1, 5.2
5	Receive stream	RX	PUT /stream		4.1-4.4
6	Get list of available streams	RX	GET /stream		3.1-3.3
7	Remove stream	any	DELETE /stream		

4. Sequence diagrams

All the LLAPI functions mentioned below are belonging to ll_api_class for 1G module, which is basically an interface helping us utilize LL API module functions in our own needs.

4.1 Proceed to main menu with configuration

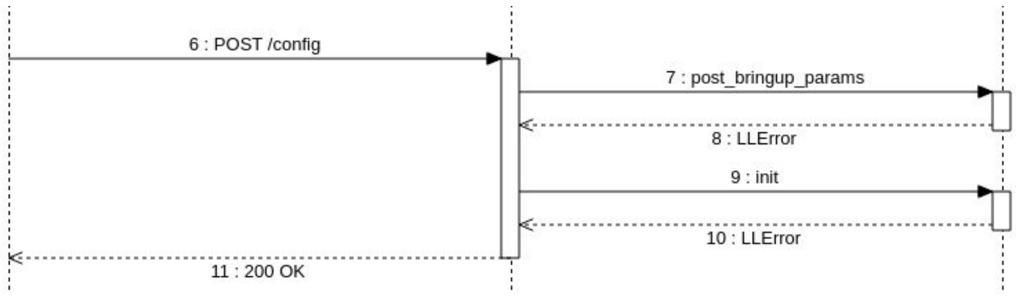
Sequence for such an initial action is rather simple and contains the following:



Goal is to let the user see the main page with current board configuration. After that it's done user can either load his own config or proceed with default and init the board:

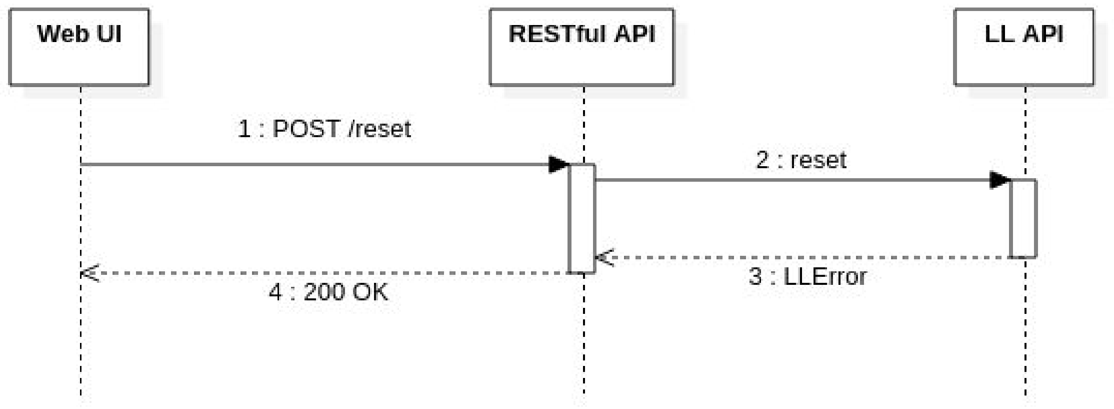
4.2 Set module config and init

The question is what we need to do after config has been changed (in particular, if device mode has been changed from TX to RX or otherwise)? Most logical way to deal with it would be call an init function, which should clean up all the previous work on module and basically give you a clean sheet. Otherwise, if we don't want to go through whole initialization routine, we might want to track exact changes in configuration and create handler function which will utilize LL API functions to apply those configurations (i.e deleting RX flows when configuration changes from RX to TX mode)

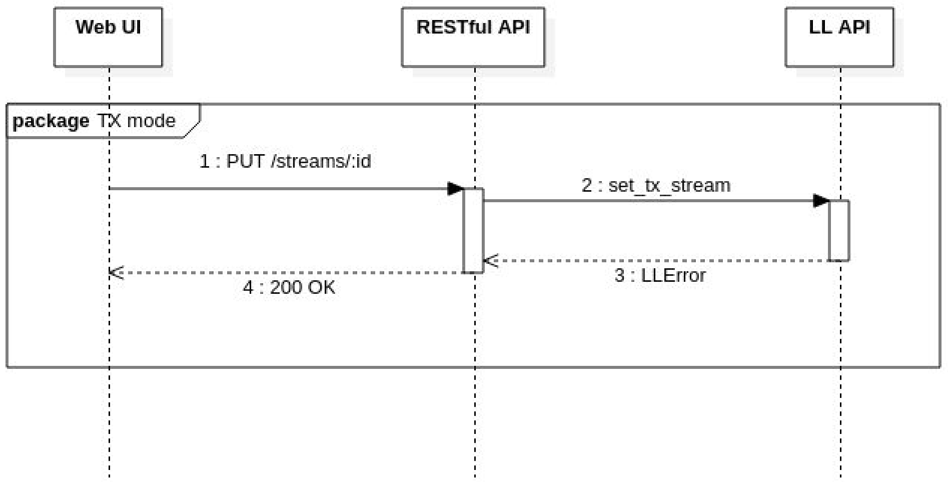


If sequence ends with successfully, user may continue further work with Web UI and 1G module. If no config chosen then system will try to run with default one.

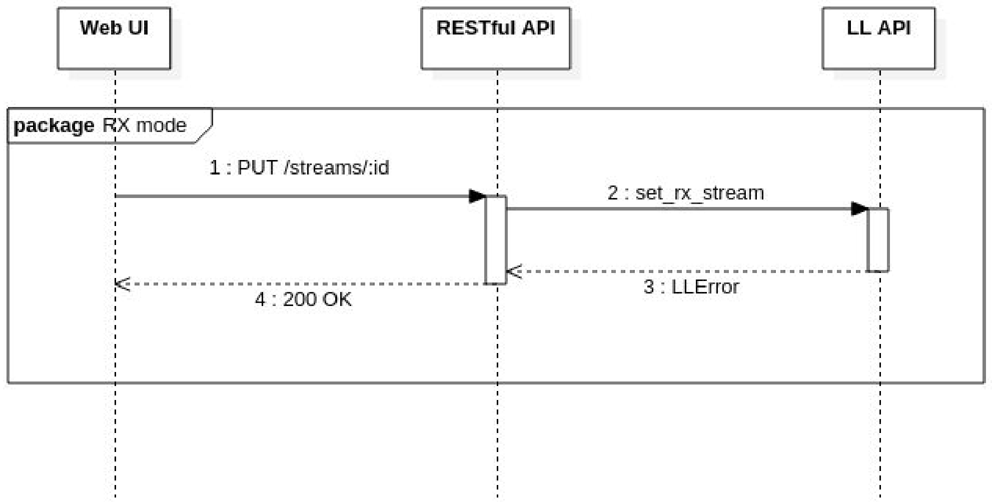
4.3 Reset module



4.4 Setup/Edit TX stream

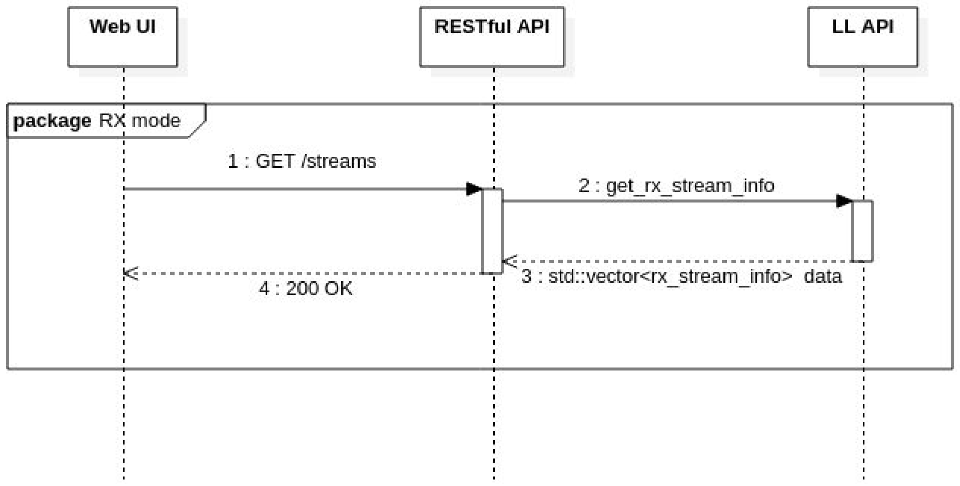


4.5 Receive stream

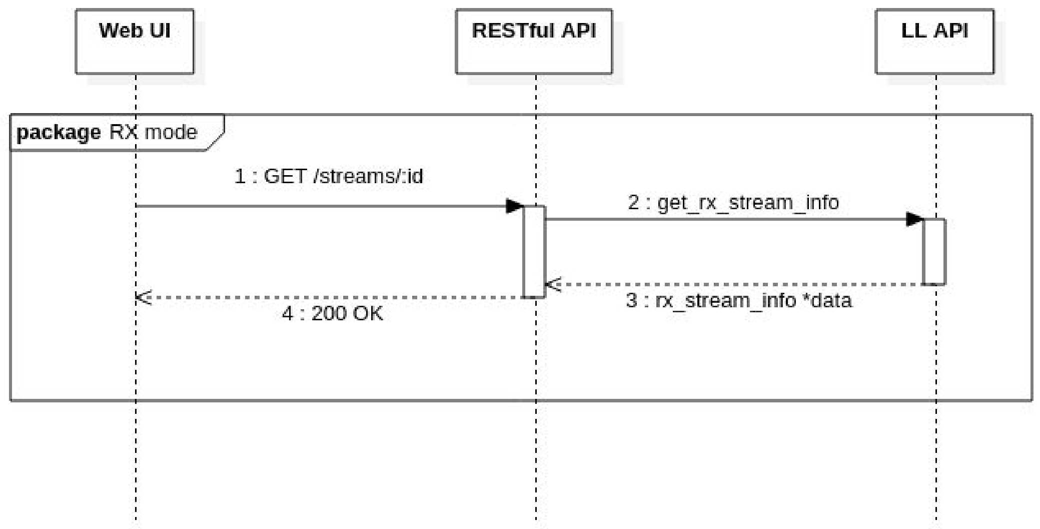


4.6 Get available streams

If we want to get a whole list of streams:

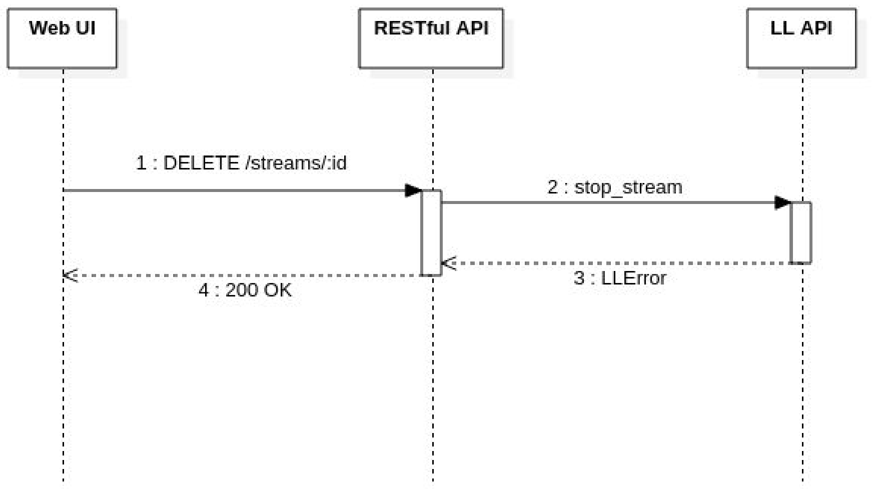


If we want to get single stream:



Action sequence with TX streams works the same way, except we will have to run it while device is in the TX mode.

4.7 Remove stream



5. LL API functions

Change format here to be Doxygen compatible. Full code should be specified with error return. For Example:

`LL_ERROR check_modules_version(unsigned int *check_result)` - check all firmware and software modules for compatibility

post_bringup_params(init_params params) - set of parameters for board initialization

```
struct init_params {
        float init_hw_clk_rate; (?)
        unsigned int tx_port;
        unsigned int rx_port;
        char src_ip[MAX_MAC][16];   // 'ff:ff:ff:ff:ff'
        char src_mac[MAX_MAC][20];  // '255.255.255.255'
        int dev_number; (?)
};
```

init() - LL API modules initialization function

LLError reset() - reset all modules;

LLError get_version(unsigned int *firmware_ver, unsigned int *software_ver) - returns software/firmware modules versions;

LLError set_tx_stream(tx_stream_info data),

LLError set_rx_stream(rx_stream_info data) - setup RX/TX stream with parameters obtained from tx_stream_info/rx_stream_info:

```
struct tx_stream_info {
        unsigned int id;
        unsigned int udp_sp;    // Source Port
        char ip_da[16];         // IP Destination Addr
        unsigned int udp_dp;    // Destination UDP port
        int vlan;               // -1 noVLAN (?ask Andrew: do we need that?)
        int pgroup_size; (?)
        unsigned int ttl;           // TTL for all streamed packets
```

```
            unsigned int video_pt; (?)
            unsigned int audio_pt; (?)
            unsigned int media_chan;   // Associated SDI input Channel
            unsigned int smpte_type;   // Stream format
    };
    struct rx_stream_info {
            unsigned int id;
            char ip_da[16];      // MC IP Addr of received stream
            unsigned int udp_dp; // UDP Port of received stream
            char pri_ip_sa[16];      // Ignored unless IGMPV3 src flt
            unsigned int smpte_type; // Stream format
            char vid_format[16];         // 1080i, 1080p 720p etc
            unsigned int frame_rate;     // Frame rate(e.g, 30,50,60...
            unsigned int playout_delay;  //  Playout delay in ms
            int aud_chan;                //  Number of audios for audio
            unsigned int media_chan;      // Output sdi channel
    };
```

LLError stop_stream(unsigned int ip_address, unsigned int udp_port,

unsigned_int sdi_chan) - function to stop RX/TX stream;

LLError get_tx_stream_info(std::vector<tx_stream_info> *tx_streams),

LLError get_rx_stream_info(std::vector<rx_stream_info> *rx_streams) - Get TX/RX stream info (?thumbnail?)

LLError validate_receiver(tx_stream_info sender, bool *result) - check if current device (in RX mode) is able to receive some specific stream from network.

6. Testing strategy

This paragraph contains information related to project testing, such as types of testing, instruments involved, etc.

6.1 Instruments

From the fact that subject of testing is RESTful API come the following criteria for the instrument:

- Ability to make HTTP requests and process responses;
- Maintainer should be able to run tests target platform as well (1G module);
- Ability to utilize JSON schemas to verify the response bodies;

Thus **Mocha & Chai** are exact Javascript libraries which would allow us to sufficiently test the RESTful API since they are easily implemented, can be automated and return may be configured to return test results in any way needed.

6.2 Tests to be run

To ensure that code developed correctly and customer expectations are met, it makes sense to make the following types of testing:

- Unit testing of whole API including every RESTful call;
- Use Case testing which will verify that functionality of each Use-Case in this design is correct after every iteration of code implementation.

Since the tests have a dependency on the functionality of the hardware two modes will be supported for testing.

1) Simulated Hardware Mode
2) Live System Mode

Simulated mode will use a Test Implementation of the LL-API that returns mocked up data where required and successful or errors based on the specific test being run.

The following tests will be run (Numbering sequence refers to the PRO AV Product Requirement document which lists RESTful API as Feature Requirement 15.):

Test Number	WB-15.1
Test Objective:	To verify all RESTFul API calls execute and return with no error
Technique:	Chai unit test framework will create a single test that calls each RESTful API with appropriate parameters and verify response
Completion Criteria:	Each RESTful API returns OK error status. There are no crashes and no ERROR or WARNING messages printed to a log or to the console.
Special Considerations:	Should be included as part of the requirement for shipped autotests and code examples Req 22 - Published tests.

Test Number	WB-15.2
Test Objective:	Verification of Use-Case 3: Reset Module
Technique:	Chai unit test framework will create a single test that calls follows sequence diagram use-case 3 and reports success of failure
Completion Criteria:	Module has been correctly reset.
Special Considerations:	Should be included as part of the requirement for shipped autotests and code examples Req 22 - Published tests.

Test Number	WB-15.3
Test Objective:	Verification of Use-Case 1: Get module configuration
Technique:	Chai unit test framework will create a single test that calls follows the Sequence Diagram Use-Case x.x and reports success of failure
Completion Criteria:	Correct configuration JSON returned
Special Considerations:	Should be included as part of the requirement for shipped autotests and code examples Req 22 - Published tests.

Test Number	WB-15.4
Test Objective:	Verification of Use-Case 2: Set module config and init
Technique:	Chai unit test framework will create a single test that calls follows the Sequence Diagram Use-Case x.x and reports success of failure
Completion Criteria:	Correct configuration is sent and applied
Special Considerations:	Should be included as part of the requirement for shipped autotests and code examples Req 22 - Published tests.

Test Number	WB-15.5
Test Objective:	Verification of Use-Case 4: Setup/edit stream
Technique:	Chai unit test framework will create a single test that calls follows the Sequence Diagram Use-Case x.x and reports success of failure
Completion Criteria:	New TX stream is created or some parameters of the existent changed
Special Considerations:	Should be included as part of the requirement for shipped autotests and code examples Req 22 - Published tests.

Test Number	WB-15.6
Test Objective:	Verification of Use-Case 5: Receive stream
Technique:	Chai unit test framework will create a single test that calls follows the Sequence Diagram Use-Case x.x and reports success of failure
Completion Criteria:	Stream is being received on device
Special Considerations:	Should be included as part of the requirement for shipped autotests and code examples Req 22 - Published tests.

Test Number	WB-15.7
Test Objective:	Verification of Use-Case 6: Get available stream(s)
Technique:	Chai unit test framework will create a single test that calls follows the Sequence Diagram Use-Case x.x and reports success of failure
Completion Criteria:	List of streams or single stream with its information returned
Special Considerations:	Should be included as part of the requirement for shipped autotests and code examples Req 22 - Published tests.

Test Number	WB-15.8
Test Objective:	Verification of Use-Case 7: Remove stream
Technique:	Chai unit test framework will create a single test that calls follows the Sequence Diagram Use-Case x.x and reports success of failure
Completion Criteria:	Stream is not being received/transmitted, related JSON object has been deleted
Special Considerations:	Should be included as part of the requirement for shipped autotests and code examples Req 22 - Published tests.

7. Discussion topics

- Is mentioned RESTful/LL API combination is emplementable and enough to suffice customer needs for 0.8 Alpha version?
- How and when do we implement ll_api_class?
- What items marked with (?) needed for? Shall we use them in 1G module APIs or not?
- Do we actually need thumbnails acquired periodically? How?
- ...