TRINNOV AUDIO

DIGITAL CINEMA
AUDIO PROCESSORS

Ovation Processor Automation Protocol

Version 1.13

Rémy Bruno

November 22, 2016

www.TRINNOV.com

Contents

1 Introduction

The Ovation processor may be remotely controlled using TCP/IP or RS232 serial link. Both communication modes work very similarly using a bidirectional link, where the client sends *commands* and the Ovation answers these commands and sends *messages* providing its current state.

In both modes, the recommended use is to connect to the Ovation, possibly request its current state (see § 5), then alternately send commands and read back data it replies. As the Ovation sends a *message* for each state change (volume, loaded profile and preset, etc.), the client can know the state of the Ovation at each time.

2 General description of the protocol

Communication is achieved using ASCII characters. Each communication (*command* or *message*) consists of a text line ending with a *return* character.

Each line begins with a keyword consisting of alphanumerical characters and indicating the command or the message type, possibly followed with arguments separated by spaces.. The Ovation recognizes the three following *return* characters: `\n` (or `0x0A`), `\r` (or `0x0D`), and both characters `\r\n` (or `0x0D 0x0A`). This allows compatibility with Unix-like, Mac and Windows systems. *Commands* and their arguments are case sensitive, so you have to respect uppercase and lowercase (`command` and `COMMAND` are two different commands).

Since version 3.7.39 of the Ovation, a command may be ended using character "`;`". It is therfore possible to provide several commands on only one line without the need to use escape characters, and use command lines such as:

```
echo "id My Automation System; dvolume 1" | telnet 192.168.0.1 44100
```

Each *command* is followed by an answer *message* from the Ovation, which is either `OK` if the command succeeded, or `ERROR` followed by a description of the error that occurred. The processor may also send other *messages*, indicating for example a state change, depending on the effects of the *command*. You should note that the answer does not necessarily occur before the other possible messages from the Ovation.

3 TCP/IP protocol

This communication mode uses the available TCP/IP network connections of the Ovation.

3.1 Network configuration

The user manual explains how to configure the network in a detailed way, so details will not be considered here.

RJ45 network configuration of the Ovation is done using DHCP during the boot. The DHCP client of the Ovation sends an identifier (`dhcp-client-identifier` option of *dhclient*), which is `TRINNOV-SRP-<id>` where `<id>` is the identifier of the processor. This allows for example to automatically give it a specific IP address using an appropriate DHCP server configuration.

It is also possible to specify manual network settings in the *Network* (or *System Status* depending on the version) sub-menu of the *Setup* page of the Ovation.

3.2 TCP/IP protocol parameters

The communication uses TCP on port 44100.

This protocol is very similar to SMTP, FTP, or even HTTP. A first approach of this protocol may consist in connecting to the processor using a program like `telnet` in order to type command lines using the keyboard.

3.3 Connection

When a TCP connection is established on port 44100, the Ovation sends a welcome message, also providing the Ovation version and its unique identifier. This identifier (`ID` or `SRPID`) is a number which allows to get the serial number of the machine (provided by the 20 least significant bits of this identifier), and the machine type (provided by the most significant bits). The format of this line is as follows:

```
Welcome on Trinnov Optimizer (Version 3.8.7, ID 9437185)
```

This identifier 9437185 is written `0x900001` in hexadecimal, so the machine type is 9 (corresponding to the type Ovation) and the serial number is 1.

The Ovation then waits an identification from the client. This is achieved by sending an `id` command with an argument identifying the client. This identification is not an authentication, and only allows a possible specific behavior with certain clients. For example:

```
id my_TMS_system
```

Communication is then achieved according to the *commands* and *messages* mode presented in the preceding section 2 and detailed in the following section 5. You should note command processing only starts after the client has identified itself using the `id` command.

4 RS232 protocol

The Ovation can also be controlled using an RS232 link with a 9-pin *null modem* cable. The RS232 serial port parameters are as follows:

- 19200 bauds
- 1 stop bit
- 8 data bits
- no parity

As soon as a client is connected to the serial port, the Ovation is in the *commands* and *messages* mode presented in the preceding section 2 and detailed in the following section 5. You should not identify yourself, contrary to the TCP/IP protocol.

Please note that not all machines have an RS232 connector. Check the available connectors on your machine.

5 Commands

Here is a list of recognized commands. This list is not comprehensive but includes all commands useful for a normal use of the Ovation. Arguments for each command are represented between <...>.

- `volume` <*volume*>
  This command allows to change the main volume to the value <*volume*> dB. This is the *master level* as it appears in the GUI of the Ovation.

- `dvolume` <*delta*>
  This command allows to add <*delta*> dB to the main volume. This value may be positive or negative.

- `volume_ramp` <*target*> <*duration*>
  This command allows to apply a volume ramp starting from current volume and reaching <*target*> volume (in dB) after <*duration*> milliseconds. This command is available since version 3.7.3 of the Ovation.

- `volume_0_10` <*volume*>
  This command allows to change main volume to 0–10 V command value <*volume*>. This command is available since version 3.1.11 of the Ovation.

- `dvolume_0_10` <*delta*>
  This command allows to add 0–10 V command value <*delta*> to the main volume. This value may be positive or negative. This command is available since version 3.1.11 of the Ovation.

- `volume_ramp_0_10` <*target*> <*duration*>
  This command allows to apply a volume ramp starting from current volume and reaching <*target*> volume (in dB) after <*duration*> milliseconds (target volume is specified using 0–10 V command values). This command is available since version 3.7.3 of the Ovation.

- `mute` <*action*>
  This command allows to change *mute* state:
  - if <*action*> is 0, *mute* is disabled (sound is enabled),
  - if <*action*> is 1, *mute* is enabled (sound is disabled),
  - if <*action*> is 2, *mute* state is inverted.

- `dim` <*action*>
  This command allows to change *dim* state:
  - if <*action*> is 0, *dim* is disabled,
  - if <*action*> is 1, *dim* is enabled,
  - if <*action*> is 2, *dim* state is inverted.

- `bypass` <*action*>
  This command allows to change *bypass* state:
  - if <*action*> is 0, *bypass* is disabled,
  - if <*action*> is 1, *bypass* is enabled,
  - if <*action*> is 2, *bypass* state is inverted.

- `send_volume`
  This command requests that the Ovation send messages providing current state information related to the volume, in particular messages `VOLUME`, `MUTE`, `DIM` and `BYPASS` (see these messages in paragraph 6). This command is available since version 3.5 of the Ovation.

- `remapping_mode` <*mode*>
  This command allows to change remapping mode:
  - if <*mode*> is `none`, remapping is disabled,

  - if <*mode*> is `2D`, 2D remapping is enabled,
  - if <*mode*> is `3D`, 3D remapping is enabled,
  - if <*mode*> is `autoroute`, autorouting mode is enabled,
  - if <*mode*> is `manual`, manual remapping is enabled.

- `use_acoustics_correction` <*action*>
  This command allows to change acoustic correction state :
  - if <*action*> is 0, acoustic correction is disabled,
  - if <*action*> is 1, acoustic correction is enabled,
  - if <*action*> is 2, acoustic correction state is inverted.

- `use_level_alignment` <*action*>
  This command allows to change level alignment state :
  - if <*action*> is 0, level alignment is disabled,
  - if <*action*> is 1, level alignment is enabled,
  - if <*action*> is 2, level alignment state is inverted.

- `use_time_alignment` <*action*>
  This command allows to change time alignment state :
  - if <*action*> is 0, time alignment is disabled,
  - if <*action*> is 1, time alignment is enabled,
  - if <*action*> is 2, time alignment state is inverted.

- `quick_optimized` <*action*>
  This command allows to change optimization state :
  - if <*action*> is 0, optimization is disabled,
  - if <*action*> is 1, optimization is enabled,
  - if <*action*> is 2, optimization state is inverted.

- `change_page` <*delta*>
  This command changes the menu page currently displayed on the GUI. The value of <*delta*> indicates the number of pages to change, and may be positive or negative (1 for going to the next page, −1 for going to the previous page).

- `loadp` <*preset*|*file*>
  This command allows to load the preset number <*preset*>. User presets start with number 1 and preset 0 corresponds to the built-in preset. This syntax is available since version 3.3.4 of the Ovation.

  For older versions, you have to use the <*file*> syntax described below, where you provide a file name instead of a preset number. For compatibility with versions before 3.3.4, newer versions also support this <*file*> syntax. The <*file*> argument is the name of a file depending on the preset number to load and is formed as follows:

  `Config_`*n*`.xml`

  where *n* must be replaced with the preset number. For example, in order to load preset number 1, you should send the following command line:

  ```
  loadp Config_1.xml
  ```

  As command lines are *case sensitive*, you should use an upper case letter in the file name `Config_1.xml`. Note: this second syntax does not allow to load the built-in preset.

- `get_current_preset`
  This command sends back the current preset number. This command is available since version 3.3.4 of the Ovation.

- `get_label <n>`
  This command sends back the name of preset number `<n>`. This command is available since version 3.3.4 of the Ovation.

- `get_all_label`
  This command sends back the list of all presets available on the machine, along with their numbers and names. This command is available since version 3.3.4 of the Ovation.

- `tac_preset <preset>`
  This command allows to change the current source, as if you pressed a front panel button of the Ovation. Possible values for `<preset>` are `server`, `non_sync`, `mic`, `altern`, `dig1`, `dig2`, `dig3`, `dig4`, `dig5`, `ana1`, `ana2`, `line`, `user1`, and `user2`. It is also possible to provide the source number instead of its name (0 corresponding to the first source).

- `profile <profile>`
  This is an alias for command `tac_preset` available since version 3.4 of the Ovation.

- `get_current_profile`
  This command sends back the current source number. This command is available from version 3.5.0rc8 of the Ovation.

- `get_profile_name <n>`
  This command sends back the name of the source number `<n>`. This is the name as it appears in the source selection page of the GUI. This command is available from version 3.5.0rc8 of the Ovation.

- `bye`
  This command requests the server to close the connection. The server then replies `BYE` (instead of `OK`) and closes the connection. Commands `exit` and `quit` are synonyms. This command has no effect on the operation of the Ovation itself (no shutdown nor sound vanishing), but only on the client-server connection. This command must not be used with the RS232 protocol.

- `power_off_SECURED_FHZMCH48FE`
  This command allows to switch off the Ovation. Please note that it only switches off the PC part of the processor. The audio and front panel of the Ovation stay powered on. The purpose of this command is to allow a clean shutdown of the PC followed by a general power down. This command is available from version 3.7.32 of the Ovation.

6 Messages

Here is a list, non-comprehensive too, of the main *messages* sent by the Ovation to connected clients:

- `VOLUME <vol>`
  This message provides the current main volume (in dB).

- `DIM <dim>`
  This message indicates whether dim is active (if `<dim>` is 1) or not.

- `MUTE <mute>`
  This message indicates whether mute is active (if `<mute>` is 1) or not.

- `BYPASS <bypass>`
  This message indicates whether bypass is active (if `<bypass>` is 1) or not.

- `META_PRESET_LOADED <profile>`
  This message provides the active profile (or source).

- `SRATE <srate>`
  This message provides the current sampling rate of the Ovation.

- `AUDIOSYNC_STATUS <status>`
  This message indicates whether the Ovation is correctly synchronized with the audio source (if `<status>` is 1) or not.

- `AUDIOSYNC <mode>`
  This message provides the current audio synchronization mode of the Ovation, which can be *Master* or *Slave*.

- `SPEAKER_INFO <spk_number> <r> <θ> <φ>`
  This message is sent for each calibrated loudspeaker when a preset is loaded. The argument `<spk_number>` indicates which loudspeaker the following information relate to (loudspeaker numbers begin here with 0). Arguments `<r>`, `<θ>` and `<φ>` give the corresponding loudspeaker's position in spherical coordinates. (r in meters, θ in degrees from the ceiling (north pole), ϕ in degrees from the front and positive towards the left).

- `START_RUNNING`
  This message is sent when the Ovation is ready to make sound. This is the case for example short after a `loadp` command is sent (typically a few tenth of seconds after).

- `LABELS_CLEAR`
  This message indicates that the whole list of available presets will be sent by the Ovation just after this message. This tells the client that it should clear its internal preset list that it maintains, in case the client maintains such a list, and that a new list of available presets will follow right after this message. This message is typically sent after the `get_all_label` command and just before the presets list.

- `LABEL <n>: <name>`
  This message indicates that the preset number `<n>` exists and that its name is `<name>`. This message is typically sent as an answer to commands `get_label` and `get_all_label`.

It should be noted that connected clients must be ready to receive a message from the Ovation at any time. For example, when a user changes the current volume, whatever the way he does it, the Ovation sends a message indicating the new volume to all its connected clients. Also, when a calibration is started using the GUI, the Ovation sends loudspeaker coordinates to connected clients, as after the `loadp` command. This allows the Ovation to interact with its connected clients. This is therefore more a "dialog" between the client and the server than a "request/answer" protocol. In other words, the stream is constantly bidirectional rather than unidirectional alternately in each way.

Also, it is highly recommended not to *poll* the Ovation, namely regularly reading the state of the Ovation, a fortiori if this is done using frequent connections and disconnections to the machine. As the Ovation sends all state changes to the clients, it is useless and deleterious to constantly request the state of the machine. The recommended approach consists in connecting only once to the Ovation, read all state variables which are relevant for the client, and then remain connected, waiting for possible changes sent by the Ovation (such as the volume, for example). However, occasional connections and disconnections of a client to the machine, for example for a volume or source change during a movie, are fine.

7 Example

Here is a communication example using TCP/IP between the Ovation and a client. The client used here is simply the `telnet` program, that can be found for example on all Unix systems (including Linux). The first line is the *shell* command line used to start the client, lines starting with "`>`" are the lines input by the user (without "`>`"), the other lines are the answers of the Ovation or of the `telnet` client.

This example is transposable almost as is for the case of RS232 protocol, by replacing `telnet` with a program providing serial port communication and by removing the first three lines (namely the *Welcome* message from the Ovation, the *id* command, and the *OK* answer) as well as the last two lines of this communication (the *bye* command and the *BYE* answer).

```
$ telnet srp 44100
```

```
Trying 192.168.77.10...
Connected to srp.
Escape character is '^]'.
Welcome on Trinnov Optimizer (Version 3.8.7, ID 123456)
>id Trinnov Audio automation system
OK
>loadp config_1.xml
ERROR: non-existing or inaccessible config file
>loadp Config_1.xml
OK
SPEAKER_INFO 0 1.36485 102.091 -43.3817
SPEAKER_INFO 1 1.21479 100.781 28.5073
START_RUNNING
>volume -12
OK
VOLUME -12.000000
>dvolume 1
OK
VOLUME -11.000000
>mute 2
OK
MUTE 1
>mute 2
OK
MUTE 0
>volume_ramp -20 50
OK
VOLUME -11.000000
VOLUME -11.960000
VOLUME -12.920000
VOLUME -13.880000
VOLUME -14.840000
VOLUME -15.800000
VOLUME -16.760000
VOLUME -17.720000
VOLUME -18.680000
VOLUME -20.000000
>bye
BYE
Connection closed by foreign host.
```

8 Conclusion

This communication protocol is subject to changes, but, in case this happens, the evolutions will always remain compatible with the previous behaviour. This compatibility approach explains why some command names seem inconsistent regarding their nomenclature (for example `tac_preset` and `get_current_profile`).

Questions are welcome, and should be addressed to `remy.bruno@trinnov.com`.