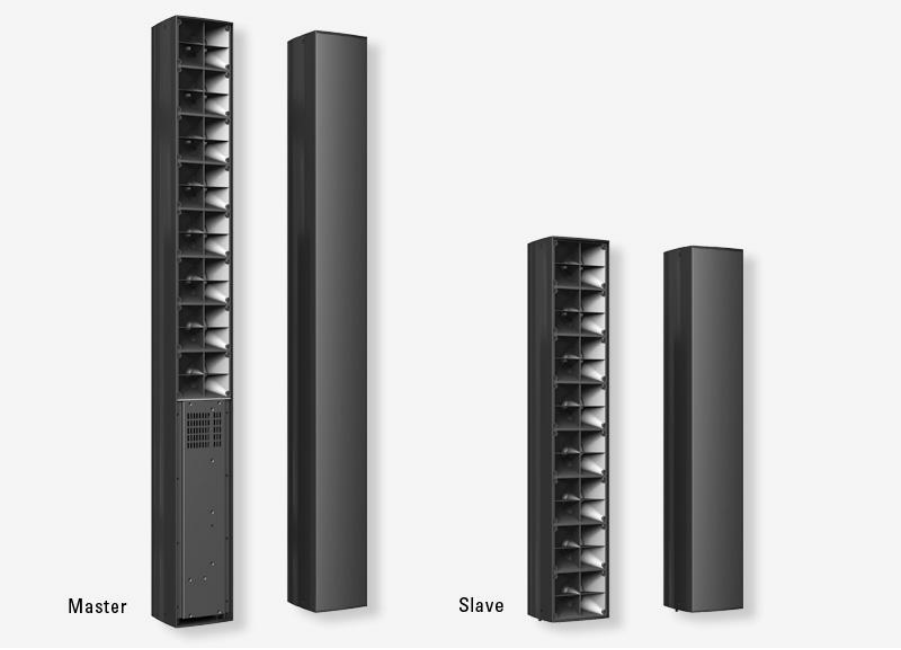
API Documentation
i Beam



Version 1.0

01.07.2022

SEEBURG
acoustic line

1 Content

1 Content 2

2 General Information 3

2.1 UDP/IP via LAN 3

2.2 Testing 3

2.3 Communication via UDP/IP 3

2.4 BasicHeader 4

2.5 DeviceData request 5

2.6 PresetRecall command 6

2.7 Acknowledgement message 7

2.8 Wait message 7

2.9 Error message 8

3 Example 9

3.1 Recall the 3rd preset 9

Core Module Line Array (CMLA) Control Protocol

This document describes how to remote control the CMLA device with external media control units (MCU) like e.g. Crestron®.

2 General Information

This document refers to Line Array devices based on the Four Audio Core Module. It is based on Firmware Version 0.3.0.6, but the fundamental commands in the protocol rarely change.

2.1 UDP/IP via LAN

CMLA can be addressed via **UDP Port 5001**. Reply messages from CMLA to your controller will be sent back to the same port the controller used for sending.

The IP address can be configured either in the Four Audio System Software (**Device/Network Settings** in the device menu) or any vendor specific software derived from it. There are two modes of IP configuration for the device:

- **DHCP (the default)** will try to obtain an IP from a local DHCP server, such as a standard IP router. If this fails, it will fall back to a Link Local IP address in the range 169.254.0.0/16, following the specification of RFC5735 (https://tools.ietf.org/html/rfc5735).
- **Static IP** will assign a fixed IP address to the device. When choosing this option, please ensure that the subnet is correctly chosen and that there are no address conflicts in the network. Otherwise, the device may become unreachable via LAN.

2.2 Testing

You may test the communication with your CMLA device using your PC and a software like Packet sender for UDP/IP http://packetsender.com/ , or using a scripting language like python (see .

2.3 Communication via UDP/IP

This section explains the CMLA network control protocol in some detail. If you just need to know how to trigger a **Preset Recall** via network, you may skip right to the examples section (page 6).

All network communication between the CMLA and the rest of the world is done with messages in the form of binary UDP packets. Each message consists of a 12 byte BasicHeader, followed by command specific information.

All numbers are sent in Little Endian byte order, i.e. the least significant bit (LSB) comes first. For example, the value 770 – or hexadecimal 0x0302 - is sent as 02 03.

2.4 BasicHeader

Each message needs to be preceded by a header in the following format:

Byte #	1	2	3	4	5	6	7	8	9	10	11	12
Meaning	Message Type	Protocolld, always 0x01	Status		Device UniqueId				Message Sequence Number		ComponentId	Reserved (leave 0)

MessageType (1 byte)

This is the most important byte as it shows what the message is all about. The following values are interesting for media control devices:

Value (hex)	**Description**
0x00	*Ping*, can be used to check if the device is still answering. A Ping consists just of the BasicHeader and is not followed by any more data.
0x02	*DeviceData,* used to query general information, such as device name or static IP address.
0x04	*PresetRecall*, either recalls a preset on device, or queries which preset is currently active.

Status (2 bytes)

These two bytes determine the basic communication role of the message. The following values may be relevant for basic media control programming. Please remember that byte order is little endian, so 0x0002 is sent as “02 00”. Any message with an unknown Status must be ignored.

Value (hex)	**Description**
0x0002	*Command*, used for commands that actually change something on the device.
0x0006	*Request*, queries some parameters from the device.
0x0001	*Response*, sent from the device as reply to a command or request.
0x0009	*Error*, sent from the device if something has gone wrong.
0x0041	*Wait*, tells the receiver that the requested procedure is initialized but may take some time.

DeviceUniqueId (4 bytes)

Is an identifier that will be filled by your CMLA device, allowing you to uniquely identify the device, e.g. to guarantee that you are talking to the right device in a DHCP setting, where the IP may change. In messages being sent *to* the device, it can remain 0.

MessageSequenceNumber (2 bytes)

This is an identifier for your message. The CMLA will always reply to the message referring to it with the same sequence number. You may start with an arbitrary number here, but it is crucial to use a *different number for each new message*. Otherwise, errors will result when trying to send commands in quick succession, as the network failure recovery mechanisms will get confused.

ComponentId (1 byte)

This byte is used to address a specific component in multi-component devices, i.e. stacked devices that use more than one FA Core Module.

Value (hex)	Description
0x00-0xFD	Address specific DSP components in the device directly, normally counted from bottom to top. This normally isn't required when performing standard tasks of a media control device.
0xFE	*No Forward*. Address only the component that is connected to the network. This is useful for requesting information from the device.
0xFF	*Broadcast*. Send the command to all components in the stack. This is e.g. useful for switching presets.

2.5 DeviceData request

A DeviceData query follows a BasicHeader with a MessageType of 0x02 (=DeviceData) and a Status of 0x0006 (=request). In order to request the standard DeviceData structure described below, just fill the DeviceData command with 4 zero bytes:

Byte #	13	14	15	16
Meaning	CrtFlags (leave 0)	OptFlags (leave 0)	Reserved (leave 0)	Reserved (leave 0)

As an answer, you will receive the BasicHeader acknowledgement, followed by the following structure:

Byte #	Meaning
13	CrtFlags (must be 0x00)
14	OptFlags (ignore these)
15..16	Device type id (0x0111 for CMLA)
17	Subnet Prefix length (in case of static IP)
18	Self diagnostic state (a value other than 0 indicates some problem)
19..22	Firmware Version
23..24	Serial no.
25..28	Reserved
29..32	Gateway IP
33..36	Static IP (set to 0 when DHCP is active)
37..40	Hardware Features (bitfield)
41	Start preset Id (note that this refers to the preset Id, not position!)
42..47	Reserved
48..79	Device Name (Latin-1 encoded)
80	Vendor ID
81..82	Reserved

2.6 PresetRecall command

A preset recall command follows a BasicHeader with a MessageType of 0x04 (=PresetRecall) and a Status of 0x0002 (=command). The message consists of 4 bytes, structured as follows:

Byte #	13	14	15	16	17	18
Meaning	CrtFlags	OptFlags (leave 0)	Index/ Position	Reserved (leave 0)	Preset Bank (leave 0)	Reserved (leave 0)

CrtFlags (1 byte)

Value (hex)	Description
0x00	*Recall by Preset Index.* The number given in byte 15 is the internal index of the Preset. The index of a preset may be obtained by requesting the preset list beforehand (this requires a block transfer and is not described in this document).
0x02	*Recall by Preset Position.* The number given in byte 15 is simply the position of the Preset in the device preset list. Recommended for use in media control devices.
0x04	*Return to Encoder Settings.* Ignores the Index/Position given in the packet and resets the device to the setting of the hardware encoder.

The CrtFlags are used to chose variants of the command. If these flags aren't understood by the device, the command will be rejected. The following values are of general use:

Index/Position (1 byte)

Specifies the preset to be recalled. The method of addressing the preset is determined by the value of CrtFlags. Note that Position is zero-based, i.e. the first preset has Position 0, the second Position 1, and so on.

2.7 Acknowledgement message

An acknowledgement is sent by the CMLA whenever a command has been successfully executed. It consists of the BasicHeader with the MessageType and MessageSequenceNumber of the corresponding command, and Status set to 0x0001.

2.8 Wait message

A wait message is sent by the device to acknowledge that it has received a command, but processing may take more than 500 ms. The message may be ignored by the receiver, displayed to the user, or used to extend a resend timeout.

The message consists of a BasicHeader with the MessageType and MessageSequenceNumber of the corresponding command, and Status set to 0x0041. After the header, the expected maximum time to wait is indicated by the following structure:

Byte #	13	14	15	16
Meaning	CrtFlags (should be 0)	OptFlags (ignore these)	TimeToWait (in 1/100s)	

2.9 Error message

If for some reason the device is not able to comply to a message sent to it, it will answer with an error message. The error message consists of a BasicHeader with the Status field set to 0x0009; the fields MessageType and MessageSequenceNumber correspond to those of the message the error refers to. After the BasicHeader, the following structure informs you about details of the error:

Byte #	13	14	15	16
Meaning	ErrorCode		Reserved	Reserved

The ErrorCode takes on one of the following values:

Value (hex)	**Description**
0x0001	*Bad Request.* The message sent to it is not understood by the device.
0x0002	*Unknown Resource.* E.g. trying to recall a preset that doesn't exist.
0x0003	*Busy.* E.g. trying to access the device from two controls at once.
0x0004	*Out of Resource.* E.g. trying to upload a new preset to a device that has no free preset slots left.
0x0005	*Internal.* This code should not occur in normal operation. If it happens repeatedly, it hints at an inconsistent device state.
0x0006	*Inconsistent Bootloader.* Should not occur in normal operation.
0x0007	*Sync Lost.* Should not occur in normal operation with an MCU.

3 Example

Here's a simple examples for switching presets on the device. If that is all you're interested in, then you may just copy the packets from this example and modify only the marked fields.

3.1 Recall the 3rd preset

Preset positions are zero-based, so recalling the 3rd preset means Position = 02.

To achieve this, simply send the following packet to the device via UDP port 5001. The fields that may vary for your purpose are highlighted in the table.

Byte #	1	2	3	4	5	6	7	8	9	10	11	12
Value (hex)	04	01	02	00	00	00	00	00	ee	01	ff	00
Explanation	Message Type: Preset Recall	X	Status: Command		DeviceUniqueId: leave empty				Message Sequence Number: Change for each command!		ComponentId: Broadcast to all components in the stack	

Byte #	13	14	15	16	17	18
Value (hex)	02	00	02	00	00	00
Explanation	CrtFlags: Recall by Position	OptFlags	Position: 2	Reserved	Reserved	Reserved

The CrtFlags are set to 0x02, indicating that we want to recall a specific preset addressed by its position in the preset list at byte #15. If instead we want to reset the device to the setting of a built-in hardware encoder, CrtFlags should be set to 0x04. In this case, byte #15 would be ignored.

Now, you should expect something like the following response from the device, sent to the UDP port from which you sent the command and from device port 5001:

Byte #	1	2	3	4	5	6	7	8	9	10	11	12
Value (hex)	04	01	01	00	6a	00	02	00	ee	01	00	00
Explanation	Preset Recall	x	Status: Response (Success)		DeviceUniqueId: Identifies the device				The same sequence number as in your command			

Here, Status is probably the most interesting field. Only status 0x0001 means that the device has fully executed the command. 0x0009 (Error) signals an error, and 0x0041 (Wait) just means that the device may take some time to process your request. As a PresetRecall may take some time, it is common to receive a Wait message before the final acknowledgement.

To get you started, we end with a simple code snippet, written in python, to send a preset recall command to a device with a known IP. Note that it does not wait for an answer or handle possible errors – it is simply “fire-and-forget”. Also, MessageSequenceNumber and the recalled Position are hardcoded.

```
import socket, struct

CMLA_IP = "192.168.1.232"
UDP_PORT = 5001
FMT_BASIC_HEADER = b"<BBHIHBB"
FMT_PRESET_RECALL = b"<6B"
bh = struct.pack(FMT_BASIC_HEADER, 0x04, 0x01, 0x02, 0x00, 0x01ee, 0xff, 0x00)
cmd = struct.pack(FMT_PRESET_RECALL, 0x02, 0x00, 0x02, 0x00, 0x00, 0x00)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

sock.sendto(bh + cmd, (IBEAM_IP, UDP_PORT))
```

API Documentation

Irrtum bei Beschreibung
sowie technische
Änderungen vorbehalten.

Alle SEEBURG acoustic line
Produkte sind nur für den
gewerblichen Einsatz bestimmt.

All specifications are
current at the time of publishing
but are subject to change.

SEEBURG acoustic line GmbH

Auweg 32
D-089250 Senden-Freudenegg

Fon: +49 (0)7307 97 00- 0
Fax: +49 (0)7307 97 00- 29

www.seeburg.com
info@seeburg.net