Control Chain Protocol

From MOD Wiki

This page is a work in progress, the informations here can be modified any time. Do not rely on the content of this page.

Contents

About

Control Chain is an open hardware assignment protocol developed to reflect the LV2 controls properties. In other words the Control Chain enables you to control LV2 plugins control ports using a hardware device like Arduino.

Although the Control Chain has been created focused on MOD, it isn't a must to have it working with LV2 plugins. This can be done using, for example, an Arduino, connected via USB to your PC.

This page concentrates technical information of Control Chain. If you are looking for a startup guide to create new controllers, please refer to MOD Arduino Shield article, there you will find a tutorial and Arduino cases that will help you.

Physical interface

In theory, Control Chain does not requires a physical interface to deal with a host. But in practice is what happen and some of decisions about the protocol were taken looking at the physical interface limitations.

The MOD hardware uses the RS-485(TIA/EIA-485) electrical standard, over serial communication, accessible via the RJ45 connector, which allows the connection of external peripherals using ethernet cables. The communication is full-duplex and the speed used is 1Mbps.

Probably the better way to prototype Control Chain controllers is using an Arduino. In this case the

physical interface will be serial over USB.

Protocol

Control Chain uses a binary protocol which has your data encapsulation using SLIP (RFC 1055).

Message Structure

The below table shows the **elementary structure** of all messages.

field	size
header	6
data	N

The **header** contain the following informations: destination, origin, command,data size and check. The **data** field holds the data bytes related to command.

Expanding the fields:

field	subfield	size
	destination	1
	origin	1
header	command	1
	data size	2
	check	1
	data 0	1
data	data 1	1
	...	N

The **destination** and **origin** fields are addresses that says from where the message is going to and coming from, respectively. The value **0x00** stands for the host address and the devices addresses values must be between **0x80** and **0xFF**, inclusively. The **check** is the byte message verification. It's calculated XOR'ing all message bytes.

Size fields marked with *1B_str* stands for a string started with one byte, that indicates your size, and followed by your data characters. For example, the "hello" string will be represented by:

0x05	h	e	l	l	o

Commands

The valid commands are:

value	description
0x01	handshaking
0x02	device descriptor
0x03	control assignment
0x04	data request
0x05	control unassignment
0xFF	error report

Handshaking

Devices can only send the **handshaking** to host when the host isn't sending data. New devices doesn't have any **origin** address yet thus the origin, in the header message field, must contain the **0x00** value. Once that host received the **handshaking** it will generate an address to new device, this address value must be between 0x80 and 0xFF, inclusively, and must be informed to device using the **destination** field. The **data** field of original message must be given back to device untouched, allowing the device discover its address.

The **channel** field is used to differentiate devices of the same type, i.e. same device URI. Each device can choose its own way to select the channel.

The **protocol version** always must be verified by the host. In the case of device has newer protocol implementation, the host have to raise an **error report** message to device, and the device, by its side, can optionally, if possible, pop up the received error to user. And finally that device has to stop sending handshaking. The host have to support backwards compatibility.

device to host or *host to device* data field

field	size	description
device URI	1B_str	a string that identifies the device
channel	1	channel configured on device
protocol version (major)	1	major number version control
protocol version (minor)	1	minor number version control

Device descriptor

The **device descriptor** should contain all information about the actuators of the device. The host can optionally request the device descriptor of the device. Actually, only devices unknown by host need to be requested.

host to device data field

field	size
[none]	0

device to host data field

field	size	subfield 1	subfield 2	subfield 3	subfield 4	description
device label	1B_str					friendly device name
actuators count	1					amount of actuators of the device
actuators list	1	actuator 1	id			actuator id
	1B_str		name			actuator name
	1		modes count			amount of supported assignment modes
	1		modes list		relevant	relevant (mask 1)
	1			mode 1	mandatory	mandatory (mask 2)
	1B_str				label	mode label
	1		max assignments	...		amount of controls that can be assigned to this actuator
	1		steps count			amount of values in the steps list
	2		steps list	value 1		step value 1
	2			value 2		step value 2
		...		...		

The **actuator id** is number defined by device developer used as short way to indentify an actuator of the device.

Some devices developers can like to have a stacking assignment feature, which would be possible to have more than one assignment to same actuator. This have to be defined on the field **max assignments**. Using max assignments greater than one is possible, for example, use a button to navigate through of the assignments.

The **steps list** contains the values of steps that the device developer recommends to be used with that actuator.

The **modes** describe to the host the LV2 properties supported by the actuator. Each mode is composed by two masks and one label. Each bit of each mask represents one LV2 control property. The first mask, the **relevant**, it's used to define if the bits of **mandatory** mask has to be evaluated.

The **label** of mode is a friendly name to help the user understand what the mode does.

The masks structure is:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
integer	logarithmic	toggled	trigger	scale points	enumeration	tap tempo	bypass

Example:

The footswitch actuator could implement two operation modes: ON/OFF and PULSE. In this case the following masks can be used:

mode	mask	integer	logarithmic	toggled	trigger	scale points	enumeration	tap tempo	bypass
ON/OFF	relevant	0	0	1	1	0	0	0	0
	mandatory	0	0	1	0	0	0	0	0
PULSE	relevant	0	0	1	1	0	0	0	0
	mandatory	0	0	1	1	0	0	0	0

In the above example, in the ON/OFF case, the user can only assign a control to this actuator if this control port has "toggled" implemented but not "trigger". In the another case, the PULSE one, the assignment can be done only if both properties are implemented, "toggled" and "trigger".

The below table shows a realistic footswitch implementation:

mode	mask	integer	logarithmic	toggled	trigger	scale points	enumeration	tap tempo	bypass	mask value
ON/OFF	relevant	0	1	1	1	1	1	1	1	0x7F
	mandatory	0	0	1	0	0	0	0	0	0x20
PULSE	relevant	0	1	1	1	1	1	1	1	0x7F
	mandatory	0	0	1	1	0	0	0	0	0x30
TAP TEMPO	relevant	1	1	1	1	1	1	1	1	0xFF
	mandatory	0	0	0	0	0	0	1	0	0x02
ENUMERATION	relevant	0	1	1	1	1	1	1	1	0x7F
	mandatory	0	0	0	0	1	1	0	0	0x0C

- ON/OFF if **toggled** is implemented but not **logarithm**, **trigger**, **scale points**, **enumeration** and **tap tempo**. **integer** is irrelevant.
- PULSE if **toggled** AND **trigger** are implemented but not **logarithm**, **scale points**, **enumeration** and **tap tempo**. **integer** is irrelevant.
- TAP TEMPO if **tap tempo** is implemented and no other.

- ENUMERATION if **enumeration** AND **scale points** are implemented but not **logarithm**, **toggled**, **trigger** e **tap tempo**. **integer** is irrelevant.

Control assignment

The device will receive this command when the user assing a control to one of your actuators. The device must reply with a **error code** indicating whether was possible assigned the requested control to the actuator.

host to device

field	size	subfield 1	subfield 2	description
actuator id	1			actuator identifier
assignment id	1			assignment identifier
port mask	1			LV2 control port properties
chosen mode	1	relevant		relevant (mask 1) of chosen mode
	1	mandatory		mandatory (mask 2) of chosen mode
label	1B_str			control label
value	4			current control value
minimum	4			minimum control value
maximum	4			maximum control value
default	4			default control value
step	2			control step
unit render	1B_str			control unit reder
scale points count	1			amount of control scale points
scale points list	1B_str	scale point 1	label	scale point label
	4		value	scale point value
		...		

device to host

field	size	description
error code	1	indicates whether assignment was well done

The **assignment id** is a number that is short way to reference the assignments. It's used in the **data request** and **unassignment** commands. The **port mask** field defines the LV2 control port properties, the mask use the same structure as presented in the device descriptor. The **chosen mode** is used by device to indentify which of your modes has to be applied to this assignment.

The **unit render** is printf format string for rendering a value (eg. "%f dB"). The fields **value**, **minimum**, **maximum**, **default** and the **value** of **scale points** use the float standard IEEE 754.

Data request

This command is used to ask data of the devices. The host send **data request** to device and the devices must always send a response back to the host. Preferably, the response should have only the values of the modified actuators.

host to device data field

field	size
[none]	0

device to host data field

field	size	subfield 1	subfield 2	description
assignments count	1			amount of assignments in the list
assignments list	1 4	assignment 1 ... 	id value	assignment identifier current assignment value

The **value** uses the float standard IEEE 754.

Control unassignment

This command is sent to device as soon as the user remove the assignment (unassignment) of the actuator. The device response have no data field. In the case of the device not reply, the host must raise a timeout and understand the control as unassigned. Once that all actuators of one specific device are unassigned, the host must stop the data requisting in this device.

host to device

field	size	description
assignment id	1	assignment identifier

device to host

field	size
[none]	0

Error report

host to device or device to host

field	size	description
error on command	1	indicates which command raise the error
error code	1	a number that represents the error
error message	1B_str	string that describes the error

Operation

Initially host is waiting for devices handshaking, once the first device sends the handshaking, the host must response back informing its *device id* in the destination field. After that host can request the device descriptor and wait for device response or timeout. Since the device is identified on host, it should wait until the host send to it an assignment request.

The host start to polling the identified devices for data request as soon as an assignement is done. The polling period is determinated by the host considering the count of identified devices. In each polling devices cycle the host is responsible to reserve a short period of time to allow that new devices connect to it or send other requests, this because devices can only send messages to host when it isn't sending any message. The host must to watch the device time response, i.e. implement a timeout verification.

The host can control when the devices will message it keeping your transmission line sending data.

See also

- lv2 specifications
- lv2 integer

- lv2 logarithmic
- lv2 toggled
- lv2 trigger
- lv2 scale points
- lv2 enumeration

Retrieved from "https://wiki.moddevices.com/index.php?title=Control_Chain_Protocol&oldid=11353"
