**Mestra**

**Combined Controller**

Instructions

Michel Keijzers, © 2017

# History

Table : History

|  |  |
| --- | --- |
| **Date/period** | **Actions** |
| Nov 8, 2017 | Initial version |

# Table of Contents

Contents

[History 1](#_Toc498126522)

[Table of Contents 2](#_Toc498126523)

[List of Tables 2](#_Toc498126524)

[List of Figures 2](#_Toc498126525)

[1 Introduction 3](#_Toc498126526)

[2 Folder Structure 4](#_Toc498126527)

# List of Tables

[Table 1: History 1](#_Toc498126528)

# List of Figures

# Introduction

This document describes the instructions for the combined Controller and MIDI/DMX slave.

Properties always start with #

Variables always start with $

Signed values always start with + or –

16 bits values end with S

# Predefined Properties

The following properties are predefined:

#Byte: A specific (unsigned) byte inside a property, mostly used for SysEx data, these are 8 bit values where the byte number can be between 0 and 31.

#Word: A specific (unsigned) word inside a property, mostly used for SysEx data, these are 32 bit values where the byte number can be between 0 and 31.

#ByteS: A specific signed byte inside a property, mostly used for SysEx data, these are 8 bit values where the byte number can be between 0 and 31.

#WordS: A specific signed word inside a property, mostly used for SysEx data, these are 32 bit values where the byte number can be between 0 and 31.

#MIDIChannel /

#MidiCh/ #MC: MIDI Channel, least 4 significant bits of byte 0

#PitchBend / #PB: Bytes 1/2 (14 bits) of a Pitch Bend command

#NoteNr / #Note Byte 1 of a Note On/Off command

#CcNr / #Cc Byte 1 of a Cc command

#CcValue / #CcVal: Byte 2 of a Cc command

#Velocity / #Vel: Byte 2 of a Note On command

#ReleaseVelocity /

#RelVel: Byte 2 of a Note Off command

#Nr/#Number: Byte 1 of a Cc/Note command

#Val/#Value: Byte 1 or 2, depending on type.

# Instruction Types

## Set

### Introduction

This instruction sets a variable with the value from a variable, property or a value.

### Examples

Set $Temp = 10;

Set $Temp16 = +1000;

Set $Temp8s = -50s;

Set $Temp16s = -500s;

Set $Temp = Temp2;

Set $Temp = #Byte 1; // Set Temp to current message, byte 1

Set #1 = 20; // 1 = byte number, from current message

Set #Byte 1 = 10S; // S = signed, from current message

Set #MC = #PB ; // Set MC of current message

Set #PB = +10000; // Set Pitch bend (14 bits value)

Set $Number = $MC; // Set Number of current message (e.g. note number)

### Syntax

Syntax:

<Set> <variable> | <property> ) ‘=’ ( <value> | <variable> | <property> )

<set>: @ ”Set”

<variable>: “$” <identifier>

<property>: <byte> | <property\_name>

<byte>: [ “#” @ “Byte”] <value32>

<property\_name>: # @ (“Note” | “MC” | ”PB” | “Number” “Value” )

<value32>: ( “0”..”32 )

<value>: [ “-“ | “+” ] ( “65535” .. “65535” ) [ S ]

### Memory

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Byte 0** | **Dst / Src** | **Byte 2**  **Src Value** | **Byte 3 Src Value** | **Byte 4/5 Dest Value** | **Name** | **Value(s) (nnnn)** | **Meaning/comment** |
| 0000 #### |  |  |  |  | Instruction | 0000 | Var (set) |
|  | nnnn nnnn | Value | - | As byte 2/3 | Value Type + values | 0000 | **Value 8 bit unsigned value** |
|  |  | Value | - | As byte 2/3 | Value Type + values | 0001 | Value 8 bit signed value |
|  |  | Value MSB | Value LSB | As byte 2/3 | Value Type + values | 0010 | Value 16 bit unsigned value |
|  |  | Value MSB | Value LSB | As byte 2/3 | Value Type + values | 0011 | Value 16 bit signed value |
|  |  | - |  | As byte 2/3 | Value Type + values | 0100 | **Var 8 bit unsigned value** |
|  |  | - |  | As byte 2/3 | Value Type + values | 0101 | Var 8 bit signed value |
|  |  | Value | Value LSB | As byte 2/3 | Value Type + values | 0110 | Var 16 bit unsigned value |
|  |  | Value MSB | Value LSB | As byte 2/3 | Value Type + values | 0111 | Var 16 bit signed value |
|  |  | Byte Nr (0-31) | - | As byte 2/3 | Value Type + values | 1000 | **Prop Byte** |
|  |  | MC Nr (0-127) | - | As byte 2/3 | Value Type + values | 1001 | Prop MC |
|  |  | Number (0-127) | - | As byte 2/3 | Value Type + values | 1010 | Prop Number |
|  |  | Pitch Bend MSB | Pitch Bend LSB | As byte 2/3 | Value Type + values | 1011 | Prop Pitch Bend (14 bits value) |
|  |  | Value (0-127) | - | As byte 2/3 | Value Type + values | 1100 | Prop Value |
|  |  | - |  | - |  | 1101 | Reserved |
|  |  | - |  | - |  | 1110 | Reserved |
|  |  | - |  | - |  | 1111 | Reserved |

Bytes with ‘-‘ are shifted left.

## Operators

### Introduction

This instruction performs a function on a property or variable, where the property or variable to change comes first and the operand second.

### Examples

Add #NoteNr 5;

Add #NoteNr #Byte 1;

Add #Byte 1 #Byte 2;

Add #Temp #NoteNr;

Subtract #NoteNr 5;

Multiply #Velocity 3;

Divide #Velocity 3;

Modulo #Velocity 3;

Percentage #Velocity 3;

Clip #Velocity 3 10;

Map #Velocity 3 100 10 20;

Sign #Velocity # Sets -1, 0 or 1 depending on sign

< #Velocity 10 # Sets 0 if False, max value if true

<= #Velocity 10

> #Velocity 10

>= #Velocity 10

== #Velocity 10

<> #Velocity 10

& #Velocity 10

| #Velocity 10

~ #Velocity 10 # Not: 0 or max value

^ #Velocity 10 # Xor

Later:

Sin #Velocity

Cos #Velocity

Tan #Velocity

Exp #Velocity

Log #Velocity

Log10 #Velocity

Power #Velocity

Sqrt #Velocity

### Syntax

Syntax:

<operator> (<set\_operand> | <property> ) <operand> <operand> <operand> <operand>

<operator>: @ ( “Add” | “Subtract” | “Sub” | “Multiply” | “Mult” | “Divide” | “Div” |

“Modulo “ | “Mod” | “Percentage” | “Perc” | “Clip” | “Map” )

<set\_operand>: < variable > | <property>

<operand> < variable > | <property> | <value>

### Memory

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte 0** | **Byte 1 Operand Types** | **Data** | **Name** | **Value(s) (nnnn)** | **Meaning/comment** |
| 0001 ---- |  |  | Instruction | 0001 | Int/Single Operators |
| ---- 0ppp |  |  | Operator | 0000 | Add |
|  |  |  |  | 0001 | Subtract |
|  |  |  |  | 0010 | Multiply |
|  |  |  |  | 0011 | Divide |
|  |  |  |  | 0100 | Modulo |
|  |  |  |  | 0101 | Percentage |
|  |  |  |  | 0110 | Clip |
|  |  |  |  | 0111 | Map |
| ---- 1ppp | - |  | Operator | 1000 | Sign |
|  |  |  |  | 1001 | < |
|  |  |  |  | 1010 | > |
|  |  |  |  | 1011 | <= |
|  |  |  |  | 1100 | >= |
|  |  |  |  | 1101 | == |
|  |  |  |  | 1110 | <> |
|  |  |  |  | 1111 | Reserved |
|  | As in Set |  |  |  | 4 bits PER OPERAND (thus 2 bytes for Clip) |
|  |  | Data bytes |  |  | Operands, 1 or 2 bytes per operand. |
| --10 pppp | - |  | Binary Operator | 0000 | & |
|  |  |  |  | 0001 | && |
|  |  |  |  | 0010 | | |
|  |  |  |  | 0011 | || |
|  |  |  |  | 0100 | ^ |
|  |  |  |  | 0101 | ~ |
|  |  |  |  | Others | Reserved |
|  | As in Set |  |  |  | 4 bits PER OPERAND (thus 2 bytes for Clip) |
|  |  | Data bytes |  |  | Operands, 1 or 2 bytes per operand. |
| 0011 ---- |  |  | Instruction | 0010 | Float Operators |
| ---- pppp |  |  | Operator | 0000 | Sin |
|  |  |  |  | 0001 | Cos |
|  |  |  |  | 0010 | Tan |
|  |  |  |  | 0011 | ASin |
|  |  |  |  | 0100 | ACos |
|  |  |  |  | 0101 | ATan |
|  |  |  |  | 0110 | Log |
|  |  |  |  | 0111 | Log10 |
|  |  |  |  | 1000 | Exp |
|  |  |  |  | 1001 | Xy |
|  |  |  |  | 1010 | X 1/y |
|  |  |  |  | 1011 | Sqrt |
|  |  |  |  | 1100 | ^2 |
|  |  |  |  | Others | Reserved |
|  | As in Set |  |  |  | 4 bits PER OPERAND (thus 2 bytes for Clip) |
|  |  | Data bytes |  |  | Operands, 1 or 2 bytes per operand. |

E.g. a Clip operator will have 1 instruction/operator byte + 2 operand type bytes + 8 (if all 16 bit values) = 11 bytes.

## Send

This instruction sends a message. This does not necessarily have to be a MIDI command.

### Examples

Send // Send current command

Send 1 // Sends command with specified number

### Syntax

Syntax:

<send> [ index ]

<send>: @ “Send”

<index>: ( “0”..”31” )

### Memory

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte 0** | **Byte 1 Operand Types** | **Data** | **Name** | **Value(s) (nnnn)** | **Meaning/comment** |
| 0010 ---- |  |  | Instruction | 0010 | Send |
| ---- iiii |  |  | Index | 0..31 | Index of command (0 = current message, default) |

## CreateMessage

This instruction create a new message. This does not necessarily have to be a MIDI command.

### Examples

Create 2 MIDI NoteOn; // Create a new MIDI Note On command at index 2

Create 2 MIDI NoteOff;

Create 2 MIDI PitchBend

Create 2 MIDI Aftertouch

Create 2 MIDI PolyAft

…

Create 2 MIDI SysEx 12 // Create a new MIDI SysEx message with 12 bytes at index 2

Create 2 DMX // Create a DMX message for 1 channel at index 2

Create 2 DMX 8 // Create a DMX message for x channels at index 2

### Syntax

Syntax:

<create> <index> <type>

<create> @ “Create”

<index>: ( “0”..”31” ) # For MIDI Sysex message only 0..7 is allowed

<type>: ( < midi\_type > | <dmx\_type> )

<midi\_type>: @ “MIDI” <midi\_sub\_type>

<midi\_sub\_type>: @ (“NoteOn” | “On” | “NoteOff” | “Off” | “Aftertouch” | “After”… | <sysex> )

<sysex>: @ “Sysex” <length>

<length>: “0” .. “31”

<dmx\_type> @ “DMX” [ <dmx\_channels> ] [ <spacing> ]

<dmx\_channels> “0” .. “31”

<spacing> “0” .. “31”

### Memory

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte 0** | **Byte 1** | **Byte 2** | **Name** | **Value(s)** | **Meaning/comment** |
| 0011 ---- |  |  | Instruction | 0010 | Create Message |
| ---- cccc |  |  | Creation Type | 0000 | MIDI |
|  | mmmm ---- |  | Creation subtype | 0000 | Note On |
|  |  |  |  | 0001 | Note Off |
|  |  |  |  |  | See … TODO |
|  | 1111 ---- |  |  | 1111 | Real Time messages |
|  |  | ###i iiiii | Index | 0-31 | Index of message to create |
|  | ---- rrrr |  | Real Time Type | 0000 | Sysex, see …TODO |
|  |  | iii- ---- | Index | 0-7 | Index of message to create |
|  |  | ---l llll | Sysex Length | 0-31 |  |
| ---- cccc |  |  |  | 0001 | DMX |
|  | iiii i--- |  | Index | 0-31 | Index of message to create |
|  | ---- -ddd |  |  |  |  |

$ means end of instruction.

# Remote Triggers

## Examples

trigger Remote ID 1 Button 4 Transpose5 disabled;

trigger Remote 1 4 Transpose5;

## Syntax

trigger <remote\_type> [@”ID”] <id> [@ “Button” ] <button> <program\_name> <enabled>;

@ means: case insensitive

<id> ( “0” … “16” )

<button> ( “0” .. “ 16” )

## Memory

A trigger always has 5 bytes. The first three bytes depend on the type. Bytes 3 and 4 contains the program ID, 0-65535.

Table 1: Triggers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte 0** | **Byte 1** | **Byte 2** | **Name** | **Value(s)** | **Meaning/comment** |
| 1--- ---- |  |  | Enabled | 1/0 | Enabled/disabled |
| -010 ---- |  |  | Trigger Type | 0000 | Remote |
| ---- iiii |  |  | Remote ID | 0-15 | Max 16 remotes |
| ---- ---- | bbbb #### | #### #### | Button ID | 0-15 | Max 16 buttons |

# Instructions

## If

### Introduction

This instruction are for conditionally jump back or forward.

### Examples

Set $Var 1 #NoteNr;

!= $Var C4;

If $VAR Skip; // Skip if not C4 note

Add #NoteNr 5

Label Skip;

End;

### Syntax

Syntax:

@ “If” ( <variable> | <property> ) <label>;

@ “Label” <label>;

<label>: <identifier>

### Memory

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte 0** | **Byte 1** | **Byte 2** | **Name** | **Value(s)** | **Meaning/comment** |
| 0100 ---- |  |  | Instruction | 0100 | Conditional |
| ---- sss# |  |  | Sub instruction | 000 | If |
|  | t--- ---- |  | Type | 0 | Variable |
|  |  |  |  | 1 | Property |
|  | -### llll |  | Label | 0-15 | Label |
|  |  | Vvvv vvvv | Variable/Property | 0-255 | Variable or property |
| ---- sss# |  |  | Sub instruction | 001 | Label |
|  | #### llll |  | Label | 0-15 | Label |
| ---- sss# |  |  | Sub Instruction | 010 | For (Later, TODO) |
| ---- sss# |  |  | Sub Instruction | 011 | While (Later, TODO) |
| ---- sss# |  |  | Sub Instruction | 011 | Until (Later, TODO) |

## End (of program)

### Introduction

This instruction sets the end of a program.

### Examples

End;

### Syntax

Syntax:

@ “End”;

### Memory

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte 0** | **Name** | **Value(s)** | **Meaning/comment** |
| 1111 #### | Instruction | 1111 | End |

## DMX \_\_\_\_\_

**Duration**

Property for duration should be logarithmic:

255 values, from 0.001 s to 500 s.

Use exp( 0.. 255) / 20.21950 for a range of 1 ms to 5 seconds.

**Memory for storage**

512 channels

Per channel a list with a length with tuples with various modes using 3 bytes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Byte 0: Mode** | **Byte 1: Value** | **Byte 2 : Duration** | **Mode Name** | **Meaning/comment** |
| Fixed Color | Fixed value | Logarithmic | Fixed Color | Keep same color for duration. |
| Up | Number of Loops | Duration from min to max | Up | Change color from current to 255. |
| Down | Number of Loops | Duration from max to min | Down | Change color from current to 0. |
| Up/Down | Number of Loops | Duration per half loop | Mode | Up and down, for ‘amount’ number of times, assuming start from max. |
| Down/Up | Number of Loops | Duration per half loop | Mode | Down and Up, for ‘amount’ number of times, assuming start from max. |
| Strobo | Number of loops | Duration per half loop | - | Switch between 0 and 255, start with 255 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |