**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.

# Programs

## Introduction

Instructions are part of programs. A program is defined as:

## Example

Program Transpose5:

Note the : instead of ;

## Syntax

<program> <program\_id> “:”

<program>: @”Program”

<program\_id>: <identifier>

Identifier is defined in the triggers documents.

# 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 : 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.

Programs:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte 0: Mode** | **Byte 1: Value** | **Byte 2 : Duration** | **Byte 3: Pause duration** | **Mode Name** | **Meaning/comment** |
| Fixed | Fixed value | Logarithmic | Logarithmic | Fixed Color | Keep same color for duration. |
| Up | Number of Loops | Duration from min to max | Logarithmic | Up | Change color from current to 255. |
| Down | Number of Loops | Duration from max to min | Logarithmic | Down | Change color from current to 0. |
| Up/Down | Number of Loops | Duration per half loop | Logarithmic | Mode | Up and down, for ‘amount’ number of times, assuming start from max. |
| Down/Up | Number of Loops | Duration per half loop | Logarithmic | Mode | Down and Up, for ‘amount’ number of times, assuming start from max. |
| Strobo | Number of loops | Duration per half loop | Logarithmic | - | Switch between 0 and 255, start with 255 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. DMX

**Structure**

DMX uses 512 channels (per universe). Each channel can be mapped to a DMX program list. A DMX program can be a fixed value or e.g. up/down and various combinations. Each program has a duration and after the program finishes, the next program in the list will start. After the last program for that channel has finished, the first program starts again.

E.g. channel 1 is mapped to program list 20.

Program List 20 hash programs 2, 5 and 8.

Program 2 has a fixed value of 50 for 1 sec. Program 5 has an up/down duration for 2 seconds and program 8 has a fixed value of 80 for 1 sec.

**To preserve memory**

Each channel can have maximum 5 programs.

There can be max. 256 programs.

Assuming 1 universe of 512 channels, and each program has 3 bytes, this will cost this many bytes:

Channels: 512 channels \* 5 program indices/channel \* 1 byte/program index= 2,560 bytes

Programs: 256 programs \* 3 bytes = 768 bytes

This totals 3,328 bytes.

**Scenes based on notes**

Scenes can set all DMX channels are started when a DMX trigger comes in (after controller receives e.g. a Note On).

Each scene by default has all channels off. A scene has sets which define values for a range of DMX channels. A set contains the start channel (multiplier of 8), number of values (1-8), and the values (max 8).

Example: Scene 1 has set 1 and 2. Set 1 sets the following DMX channels (0: 50, 1: 60, 2: 70, 4: 80), This set is defined as (0, 5, 100, 101, 102, 103, 104) where 100 to 104 are programs setting the color to 50, 60, 70, 70 resp. 80.

**Memory**

There should be many scenes, like 256. Sets will be heavily shared, so 256 sets might be enough. Each scene should have multiple sets, on average one per DMX device, so assume 32 on average.

This means the following memory consumption is needed:

Scenes: 256 scenes \* 32 sets \* 1 byte/set = 8,192 bytes.

Sets: 256 sets \* 9 (start/length + program) bytes = 2,304 bytes

Programs: 256 programs \* 4 bytes/program = 1,024 bytes

DMX values: 512 channel values \* 2 (double buffering) = 1,024 bytes

Total: 12,548 bytes.

How long it takes to send 512 bytes (excl overhead) assuming 115K transmit speed (serial/CAN):

512 \* 8 / 115000 = 0.035 s = 35 ms… This is too long. Using 2 Mbps

512 \* 8 / 2,000,000 = 2 ms … still too long

Note comes in. Control runs programs, sends DMX 4

DMX gets 4. It checks what to do: running sets 1, 2 and 4.

Startup controller. Controller reads DMX file, and sends it to DMX slave.

Using nRF24L01, speed 250,000 bps.

Sending 10.000 bytes cost 10,000 \* 8 / 250,000 = 0.32 s which is acceptable. During startup no other slaves send info.

**DMX File**

Per DMX Trigger,1 program list:

* Program
  + 4 bytes

Memory average:

Trigger Program Index Start: 256 triggers \* 2 program-list-index

Program List Table: 1,000 programs

**Example**

**Scenes**

IMPORTANT: First 256 can be triggered by controller.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scenes** | **Scene** | **Instruction** | **Sub scenes or Program Param (4 bytes)** | **Comment** |
| 0 | Blackout | 255 |  | All channels 0 by default. |
| 1 | Front All Red | 256, 261 |  | Left front PARs red, Right front PARs red |
|  |  |  |  |  |
| 255 | All blackout | Program 0 | ??? |  |
| 256 | Left Front All Red | Scenes | 257, 258, 259, 260 | Separate LEDs |
| 257 | Left Front 1 Red | Program 0 (31) |  | Master + Red On |
| 258 | Left Front 2 Red | Program 0 |  |  |
| 259 | Left Front 3 Red | .. (diff channel) |  |  |
| 260 | Left Front 4 Red | .. (diff channel) |  |  |
| 261 | Left Right Red | 262, 263, 264, 265 |  |  |
| 262 | Left Front 1 Red | (Ch 33, program 0, Ch 35, program 0) |  | Master + Red On |
| 263 | Left Front 2 Red | .. (diff channel) |  |  |
| 264 | Left Front 3 Red | .. (diff channel) |  |  |
| 265 | Left Front 4 Red | .. (diff channel) |  |  |
|  |  |  |  |  |

Bytes: 11 (entries) \* (4 bits length + 1 bit type) + 1 byte length + #length \* 3 bytes (2b ch + 1b prg) = 11 \* (1 + 1 + 2 \* 3) = 88 bytes.

**Programs**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program #** | **Program Name** | **Mode** | **Value** | **Duration** | **Pause** | **Next** |
| 0 | Full On | Fixed | 255 | Inf | 0 s | Last |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

A program has 5 bytes. For each channel there are 5 programs, thus 512 \* 5 = 2,560 bytes.

Triggers: 2 bytes per trigger, pointing to program

Trigger 0: program 0: All Blackout

Trigger 1: program 1: Front Red

Trigger 2: program 4: Outside Front Red Down

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program #** | **Meaning** | **Intruction** | **Parameters** | **#Bytes** |
| 0 | All blackout | Nop | - | 1 |
| 1 | All front red | Run 2 Programs | 2, 3 | 5 |
| 2 | All front left red | Set Multi Values | 0 (start ch), 8 (interval), 4 (amount), 255, 0, 255, 0 | 6 |
| 3 | All front right red | Set Multi Values | 33 (start ch), 8 (interval), 4 (amount), 255, 0, 255, 0 | 6 |
| 4 | Outside front red down | Run 4 programs | 5, 6, 8, 9 | 5 |
| 5 | Intensity on | Max Single Channel | 33 | 2 |
| 6 | Red up down | Add Mode Single Channel | 35 (ch), 7 (mode) | 3 |
| 7 | Mode up/down | Mode | Down, Inf Loops, duration, pause | 4 |
| 8 | Intensity on | Max Single Channel | 81 | 2 |
| 9 | Red up down | Add Mode Single Channel | 83 (ch), 7 (mode) | 3 |
| Total |  |  |  | 18 |

Bytes: 4

**DMX Instructions**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Byte 0** | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Cons Bytes** | **Meaning** | **Value** | **Comment** |
| iiii ---- | END |  |  |  |  | Instruction | 0000 | Nop |
| iiii ---- |  |  |  |  |  | Instruction | 0001 | Run 2 Programs |
|  | Prg 1 MSB | Prg 1 LSB | Prg 2 MSB | Prg 2 LSB | END | Programs |  |  |
| iiii ---- | Prg 1 MSB | Prg 1 LSB | Prg 2 (offset from 1) | Prg 3 (offset from 2) | Prg 4 (offset from 3) | Instruction | 0010 | Run 4 programs, 0 = no program |
| iiii ---- |  |  |  |  |  | Instruction | 0011 | Set Single Value |
| ---- ##cc | cccc cccc |  |  |  |  | Channel | 1-1024 |  |
| ---- pp-- |  | Prg LSB |  |  |  | Program to run | 0-1024 | Should point to Append Mode Instruction |
| iiii --- |  |  |  |  |  | Instruction | 0100 | Set Multiple values |
| ---- ##ss | Start channel |  |  |  |  | Start Channel | 1-1024 |  |
|  |  | aaa- ---- |  |  |  | Amount | 1-8 |  |
|  |  | ---i iiii |  |  |  | Interval | 1-32 |  |
|  |  |  | Prg MSB | Prg LSB |  | Program to set mode |  | Should point to Append Mode instruction |
| **iiii ----** |  |  |  |  |  | Append Mode | 0101 |  |
| **---- tttt** |  |  |  |  |  | Type | 0000 | Fixed |
|  | vvvv vvvv |  |  |  |  | Value | 0-255 |  |
|  |  | dddd dddd |  |  |  | Duration | 0-255 | Logarithmic |
| **---- tttt** |  |  |  |  |  |  | 0001 | Fixed Min |
| **---- tttt** |  |  |  |  |  |  | 0010 | Fixed 50% |
| **---- tttt** |  |  |  |  |  |  | 0011 | Fixed Max |
|  | dddd dddd |  |  |  |  | Duration | 0-255 | Logarithmic |
| **---- tttt** |  |  |  |  |  | Type | 0100 | Down |
|  | llll llll |  |  |  |  | Loops | 0-255 | Number of loops |
|  |  | dddd dddd |  |  |  | Duration | 0-255 | Assuming from 255, logarithmic |
|  |  |  | pppp pppp |  |  | Pause | 0-255 | Duration, logarithmic |
| **---- tttt** |  |  |  |  |  | Type | 0101 | Up |
| **---- tttt** |  |  |  |  |  | Type | 0110 | Down/Up |
| **---- tttt** |  |  |  |  |  | Type | 0111 | Up/Down |
| **---- tttt** |  |  |  |  |  | Type | 1000 | Strobo |
|  | As above |  |  |  |  |  |  |  |

Programs are known beforehand, average size is 3 bytes, assuming 1024 programs: 3,072 bytes

For each channel there are 5 modes (containing of 4 bytes): 512 \* 5 \* 4 = 10,240 bytes

Values buffer 1 and 2 for 512 channels: 1,024 bytes

Total: 14,336 bytes.