Midra™

TPP
Programmer's Guide

For version v02.00.15

ANALOG WAY®
Pioneer in Analog, Leader in Digital

Table of contents

Pictures index

1 INTRODUCTION

This document provides information and guidance to control Midra™ products directly from controllers without using the standard *RCS*^{2}. A basic knowledge of the machine is necessary, using the *RCS*^{2}.

TPP stands for Third Party Protocol.

1.1 References

• Midra_TPP_commands_for_v02-00-15.zip	Midra™ v02.00.15 TPP command set
• Midra™ v02.00.15 firmware updater	(on ANALOG WAY web site)
• EIKOS2 reference manual	(on ANALOG WAY web site)
• EIKOS2 quick start guide	(on ANALOG WAY web site)

1.2 Notices

Pictures and drawings are non-contractual.

Specifications are subject to change without prior notice.

2 CONTROLING MIDRA™

2.1 Introduction

Midra™ products are usually controlled by the user friendly RCS^2 or other high-end remote controller, but a programming interface is also provided for automation applications.

A good practice is to setup the machine with the RCS^2 and then control it with a few basic commands like “preset recalling” and “layer input change”.

2.2 Physical interfaces

Picture 1 : RJ45 leds colors definition

Left LED color	Definition	Right LED color	Definition
Off	No link	Off	No activity
Amber	10 Mbps	Amber	Half-Duplex
Green	100 Mbps	Green	Full-Duplex

Midra™ can be controlled through its rear Ethernet **RJ45** plug :

- labeled “**ETHERNET**”
- 10Base-T or 100Base-TX (Auto-Sensing)
- **MDI** connection (which need a crossover cable to connect it directly to a computer)

Midra™ can also be controlled through its rear RS232 DB9 female plug :

- labeled “**RS-232**”
- 1200Bds up to 115200 Bauds
- 3 wires straight cable, 8 data bits, 1 stop bit, no parity bit, no flow.

2.3 Protocol

Supported protocol is **TCP/IP**, parameters can be set up with front panel or RCS^2 configuration menus.

Default values are :

- Protocol : **TCP or UDP**
- DHCP client : **no**
- IP address : **192.168.2.140**
- IP mask : **255.255.255.0**
- Gateway : **192.168.2.1**
- TPP port : **10500**

Note : the simultaneous connection number is limited to 1.

2.4 Command principle and structure

2.4.1 Midra™ control principle

Midra™ functionalities are controlled through **commands**. Those commands allow **reading** or **writing** in machine **registers**.

The product should be considered as a state machine controlled by writing to its registers. Writing into registers modifies machine state. Current state of the machine is always available by reading its registers.

Registers structure and size can range from a simple bit up to multidimensional array of 32 bits words. As a result, writing in such a multidimensional register requires providing **indexes values** in addition to the **register value**.

Each register have a unique name, only made of **five letters**, upper case or lower case. (2 exceptions, 1 letter name command)

A command is made only of displayable ASCII characters (ranging from 0x21 up to 0x7E).

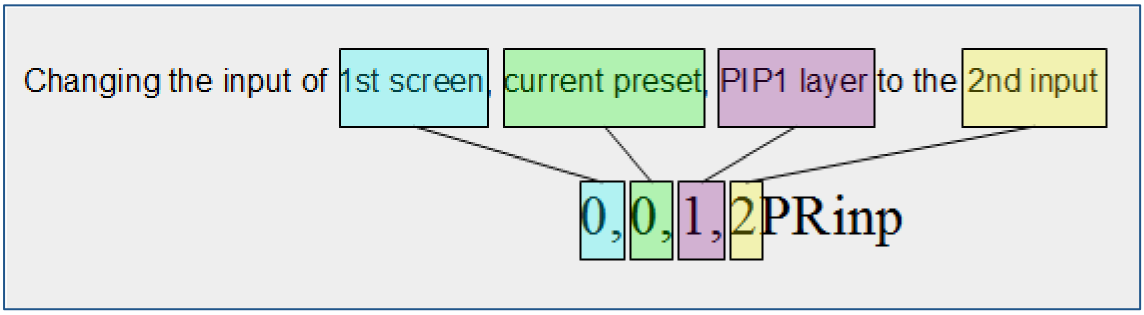
Commands are of 2 types: read commands or write commands, using the same syntax.

- a write command is made of indexes values, followed by the register value and ended by the register name.
- a read command uses exactly the same syntax, except the register value that is omitted.

2.4.2 Write command structure

A write command is made of numeric values separated by comma, followed by a group of up to 5 letters defining the command.

Picture 2 : write command example



The last numerical field is the value to be set by the write command.

The first numerical fields are “indexes values”, specifying on which the command relates. The number of indexes can range from 0 to 5 depending on the command.

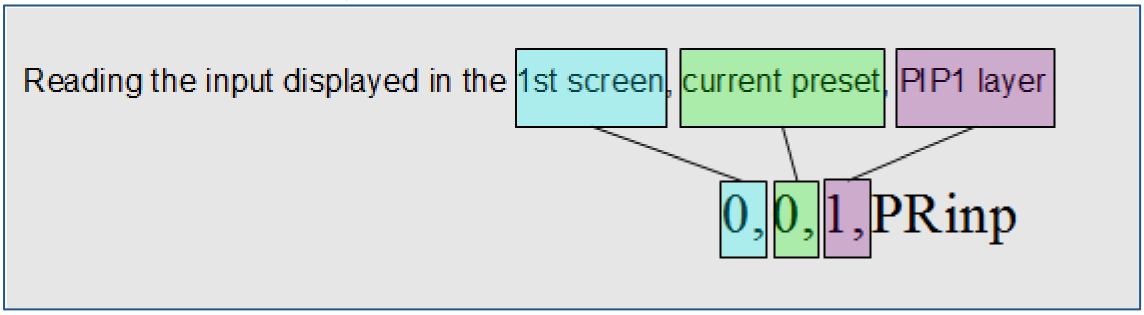
(details in chapter §Command indexes and Command values)

Each command is ended with the last command character.

2.4.3 Read command structure

A read command follows the same structure than the write command, simply with the value field omitted. Please note that an index value is always followed by a comma character.

Picture 3 : read command example

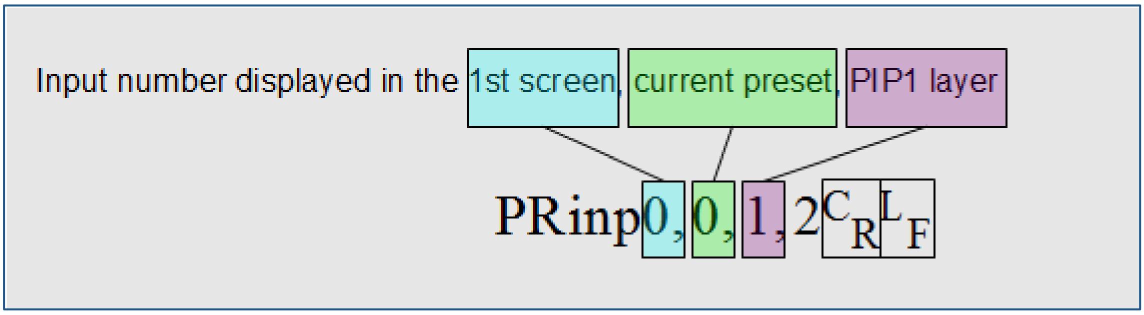


2.4.4 Valid answer structure

When a read or write command is **valid**, the machine answers, giving the current register value. The answer structure is symmetrical with the write or read command.

An answer is made of a group of letters (most often the same as the command) followed by numeric values separated by comma, and is ended with CR LF characters. (ASCII 0x0D and 0x0A)

Picture 4 : valid answer structure

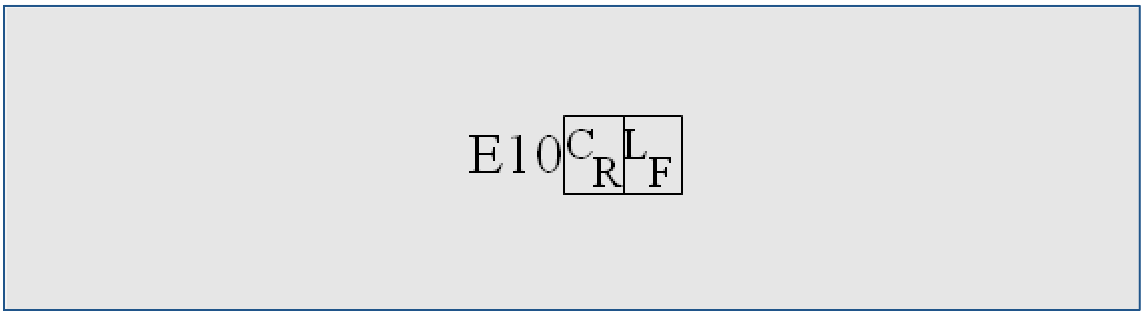


An answer starts generally by the same group of letters than in the initial read or write command, followed by indexes values, then command value and ending with CR LF characters.

2.4.5 Error answer

When an **invalid** read or write command string is received, the machine ignores it, answers with the previous value (unless in case of command error) and immediately answers with one of the following error string.

Picture 5 : error answer example



Error message structure :

An error message is made of the capital letter E followed by a 2 digits value depending on the error and is ended with ␍␊ characters. (ASCII 0x0D and 0x0A)

Here are returned error code and conditions covered :

- E10 : means “command name error”. It is usually due to a command field (i.e. five letters) that does not match any legal command string.
- E11 : means “index value out of range”. It is usually due to a wrong index value.
- E12 : means “index number error”. It is usually due to an incorrect number of indexes, too or not enough.

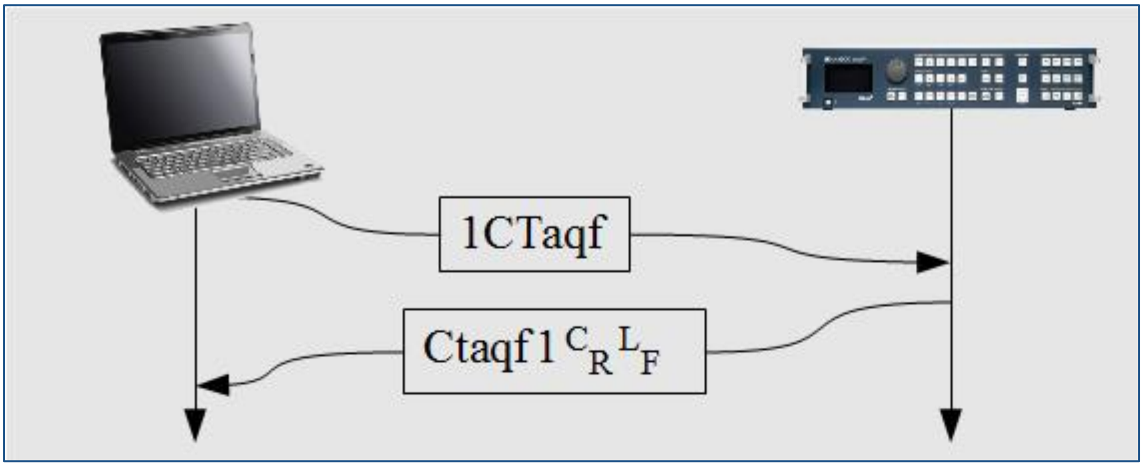
2.5 Commands sequencing

Complete command sequences is made of a read or write command issued by the controller and by the answer of the machine. The answer can be used as an acknowledgment. As the input commands processing is asynchronous, the answer time is nor constant nor predictable. On the other side, this allows to send multiple commands in advance as long as they are independent.

A good practice is to check commands acknowledgment before sending new block of commands.

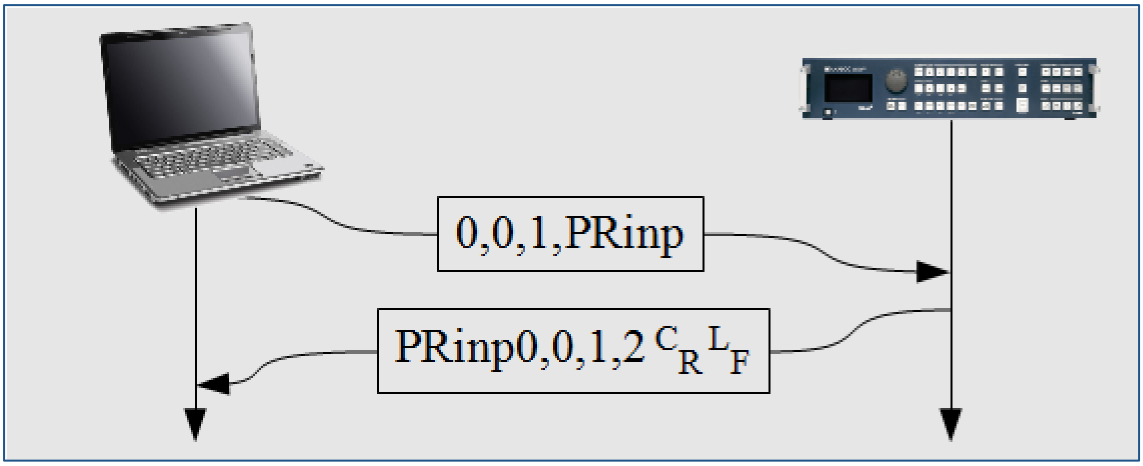
2.5.1 Write sequence

Picture 6 : write command sequence



2.5.2 Read sequence

Picture 7 : read command sequence



2.6 Command indexes and Command values

As explained in chapter §Midra control principle, Midra™ commands allow reading or writing values in multidimensional registers. For these, indexes values must be supplied. If a register is not multidimensional, its reading or writing commands does not need indexes values.

Indexes values : Depending on the command, you can have to specify from 0 to 5 indexes values. They indicate on which the command relates. For example, the "*OSCREEN_MAX_LAYERS*" command, which gives the number of live layers, requires an index value to indicate one of the device's screen.

No wildcard exists, all required indexes values shall be supplied. Some indexes values have names starting with "*DIM_*", meaning dimension. For example, the "*OSCREEN_MAX_LAYERS*" command giving the number of live layers in a screen, always requires a "*DIM_SCREEN*" index value indicating the screen.

Indexes values are detailed in the "Midra_TPP_commands_for_v02-00-15.xls" document.

Command value: This is the register value. In a write command, It indicates the new value that you want to be applied. In a read answer, it indicates the current state of the command (current register value). A write command is distinguished from a read command due to the presence of the numerical command value just before the command letters.

A value written in a register remains until modified by a new write command or by the device itself. This allows options to be written only once.

All registers have a ***default value***, noted in the detailed tables.

You must be careful on value range, which depends on multiple factors, like device type, device configuration or current situation. Value range have names starting with "*ENUM_*", else, if no enumeration name exist, value must be comprise between given "min value" and "max value".

Commands values are detailed in the "Midra_TPP_commands_for_v02-00-15.xls" document.

2.7 Multiple controllers

Multiple controllers are not allowed with TCP protocol, due to the connection number limited to one. It is allowed with UDP protocol, please refer to the relevant note.

With multiple controllers, no priority exists, in case of simultaneous writing of the same command, the machine applies the last received. In all cases, controllers must take in account the last answer received.

User must particularly be careful with compound commands (multiple commands sent sequentially). This can cause commands mixture with 2 controllers sending simultaneously their own commands.

2.8 Multiple machines

This configuration can only use UDP protocol, please refer to the relevant note.

3 COMMON MIDRA™ USE CASES

The following common actions can be remotely controlled:

- Establishing a connection with a Midra™, including :
    - Socket opening
    - Device type checking
    - Command set version checking
    - Midra™ registers read back
- Keeping a connection alive with a Midra™
- Changing a source displayed in a layer
- TAKE : Transitioning a Preview screen onto a Program screen
- TAKE ALL : Transitioning from Preview onto the Program for all screens (matrix mode)
- Loading a Preset from Master memory to a single Program screen
- Loading a Preset from Master memory and changing the source displayed in a layer
- Displaying/Hiding the Quick Frame for one single screen
- Displaying/Hiding the Quick Frames for all screens (matrix mode)

3.1 Establishing a connection with a Midra™

3.1.1 usage

This example gives you the proper way to establish the connection with a Midra™ device. It is made of four recommended steps: socket opening, device type checking, command set version checking and Midra™ registers read back.

3.1.2 summary of the commands sequence

socket opening	initial step, TCP/IP connection and device ready wait for.
device type checking	verifying the right expected device.
command set version checking	verifying matching of controller commands and machine commands.
Midra™ registers read back	machine current state retrieval.

3.1.3 detailed commands sequence

- socket opening : As indicated in §CONTROLING MIDRA chapter, TCP/IP must be used to control Midra™. The device acts as a server. Once the TCP connection is established, the controller shall check that the device is ready, by reading the **READY** status, until it returns the value 1.

  Syntax : `*`

  Answer : `*<value>␍␊` The controller shall wait and retry until it receives the value **1**.

- device type checking : This read only command gives the device type.

  Syntax : `?`

  Answer : `DEV<value>␍␊` The numerical <value> gives the connected device type. Other values match other Analog Way devices.

<value>	device
257	Eikos2
258	Saphyr
259	Pulse2
260	SmartMatriX2
261	QuickMatriX
262	QuickVu
282	Saphyr - H
283	Pulse2 - H
284	SmartMatriX2 - H
285	QuickMatriX – H

- command set version checking : This read only command gives the version number of the command set. It is recommended to check that this value matches the one expected by the controller.

  Syntax : `VEvar`

  Answer : `VEvar1␍␊` Meaning that the device is using a command set in version 1.

- Midra™ registers read back : This step is recommended to initialize the controller. Various methods exist, depending on controller software architecture.

  To ease this initialization step, the device features the **DIESE** command, to enumerate (read back) all its registers current values. This produces a huge amount of data that can saturate the controller. A command parameter allows reducing this volume by sending only register values different from their default value. If the volume is still too high, the controller should enumerate himself all the required registers, at its own rate.

- Read back using DIESE command :

  At first, the controller should wait a possible current DIESE command to finish.

  sending: `#` The controller asks for the current state of the DIESE command.

  Answer: `#<value>␍␊` The controller must wait that the numerical <value> equals **0**, meaning that no enumeration is running.

  sending: `1#␍␊` or `3#␍␊`

  When <value> is equal to **1**, the device will enumerate all its registers values, for all indexes in case of multidimensional registers.

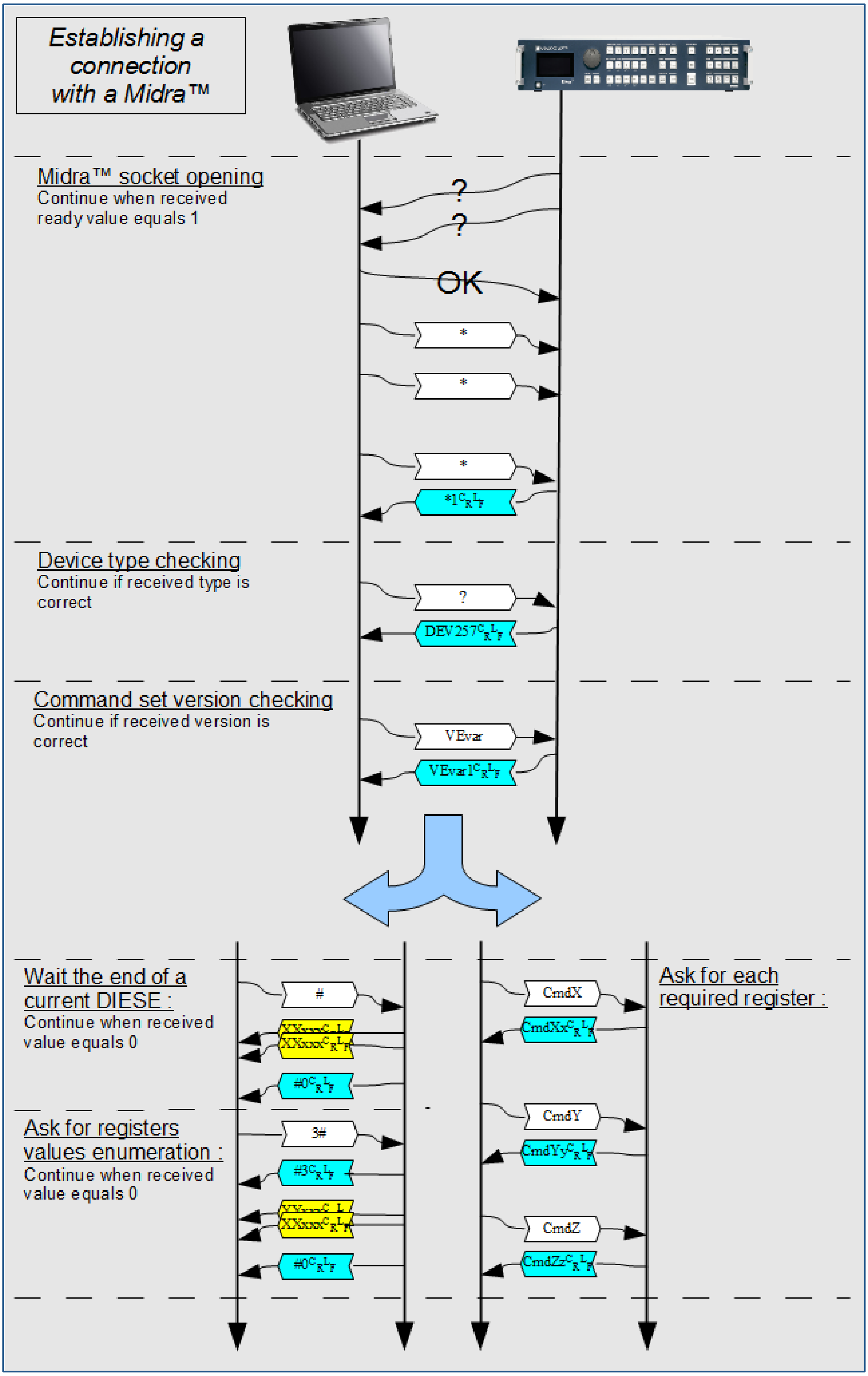
  When <value> is equal to 3, the device works the same way, except that it will not enumerate registers having their default value, saving some amount of data.

- Registers read back managed by the controller :

  The controller should read all used registers, slowly enough to avoid being saturated.

3.1.4 example of connection establishment

Picture 8 : example of connection establishment


Establishing a connection with a Midra™
Midra™ socket opening
Continue when received ready value equals 1
?
?
OK
*
*
*
*1CRLF
Device type checking
Continue if received type is correct
?
DEV257CRLF
Command set version checking
Continue if received version is correct
VEvar
VEvar1CRLF
Wait the end of a current DIESE :
Continue when received value equals 0
#
XXxxxCRLF
XXxxxCRLF
#0CRLF
Ask for registers values enumeration :
Continue when received value equals 0
3#
#3CRLF
XXxxxCRLF
XXxxxCRLF
#0CRLF
Ask for each required register :
CmdX
CmdXxCRLF
CmdY
CmdYyCRLF
CmdZ
CmdZzCRLF

3.2 Keeping a connection alive with a Midra™

3.2.1 usage

This example gives you the proper way to maintain the connection alive with a Midra™ device. It is recommended to use this method for future developments. It uses a single command allowing testing the connection and that the device is alive.

The sending rate should be slow enough to avoid link overload, typically at "human rate".

It is advised to "ping" the device only when no communication occurs.

3.2.2 detailed command sequence

Syntax: `<val1>SYpig` <val1> is any numerical value that will be used and returned inverted by the device, showing that it is alive.

Answer: SYpig<val2>$^{C}_{R}$$^{L}_{F}$ <val2> is computed by binary inverting <val1>.

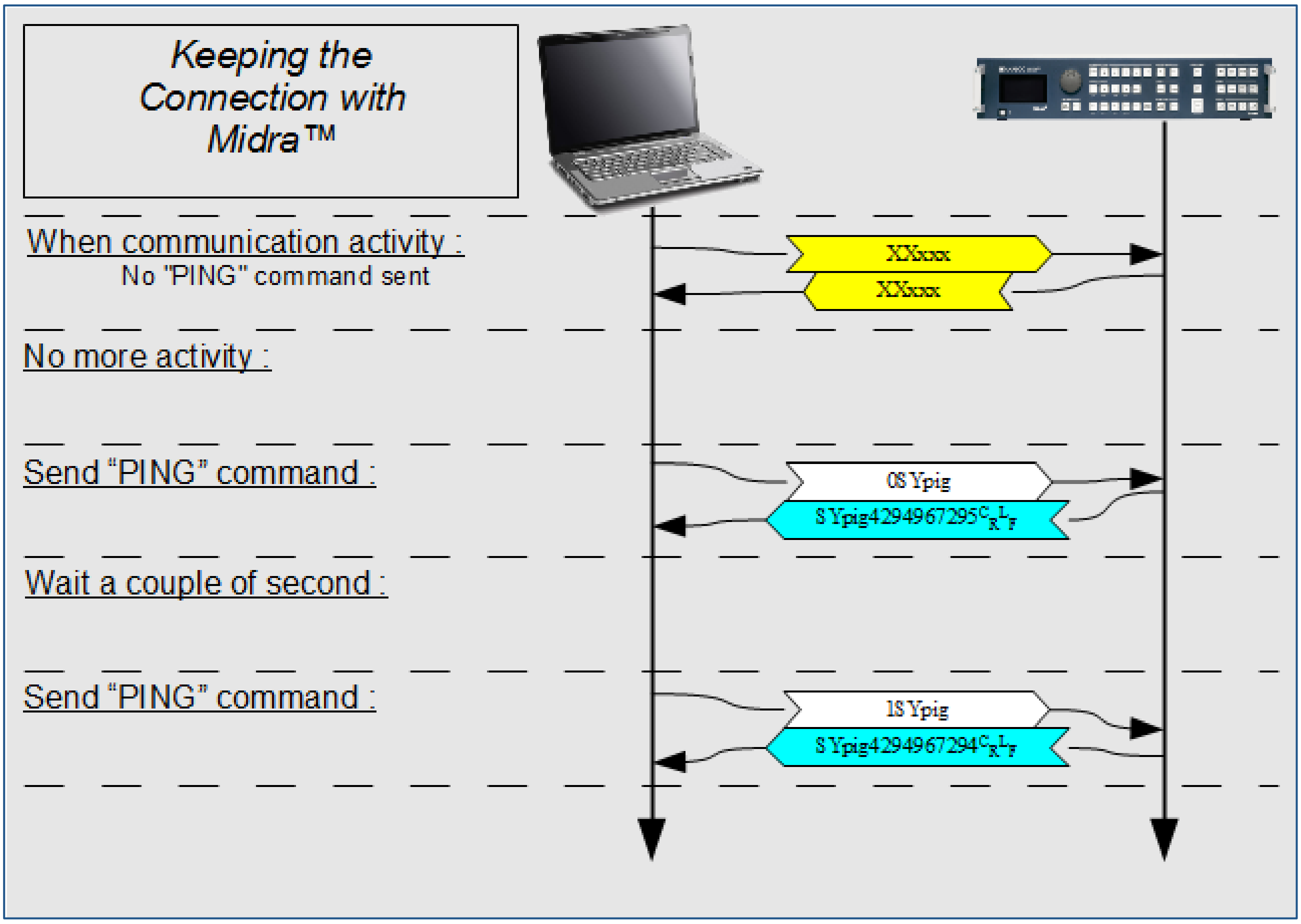
Examples: sending 0 (0x0000 0000) will return 4.294.967.295 (0xFFFF FFFF)

sending 1 (0x0000 0001) will return 4.294.967.294 (0xFFFF FFFE)

sending 170 (0x0000 00AA) will return 4.294.967.125 (0xFFFF FF55)

3.2.3 example of PING command

Picture 9 : example of keeping a connection alive

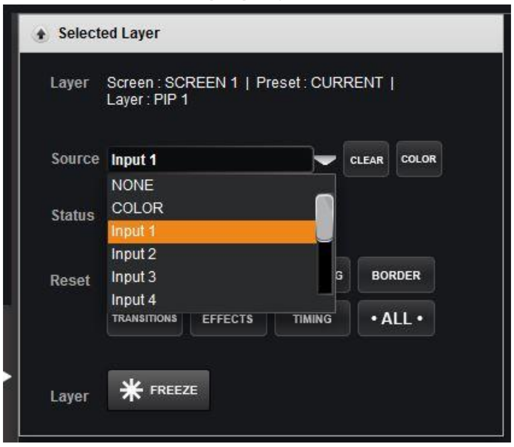

Keeping the Connection with Midra™
When communication activity :
No "PING" command sent
XXxxx
XXxxx
No more activity :
Send "PING" command :
0SYpig
SYpig4294967295$^{C}_{R}$$^{L}_{F}$
Wait a couple of second :
Send "PING" command :
1SYpig
SYpig4294967294$^{C}_{R}$$^{L}_{F}$

3.3 Changing a source displayed in a layer

3.3.1 usage

This example indicates how to change a source displayed in a layer of a screen, on the Program or Preview output. Source type can be "live sources", frames, logos or color.

Picture 10 : Changing layer source - RCS²



3.3.2 Midra™ layers reminder

Pictures are displayed on a screen in overlapping layers. Each layer can only contain a predefined picture type, either "Frame", live input or "Logo", depending on the layer depth.

The first layer is the lowest, it can be covered by any upper layer and can only display still frame.

The following layers, named "PIP", can only display live inputs. Their number depends on device type and mode and it can be known using the *OSCREEN_MAX_LAYERS* command.

The following upper layers must be used to display logos.

The last invisible layer is dedicated to audio, allowing it to be processed synchronously with video and also to be memorized in preset memories.

<layer>	name
7	Audio
6	Logo2
5	Logo1
4	PIP4
3	PIP3
2	PIP2
1	PIP1
0	Frame

3.3.3 detailed command parameters

Syntax : <scrn>,<ProgPrev>,<layer>,<src>PRinp

<scrn> is the RCS² screen number minus 1.

<ProgPrev> is 0 for Program, 1 for Preview.

<layer> is a value representing the destination Layer.

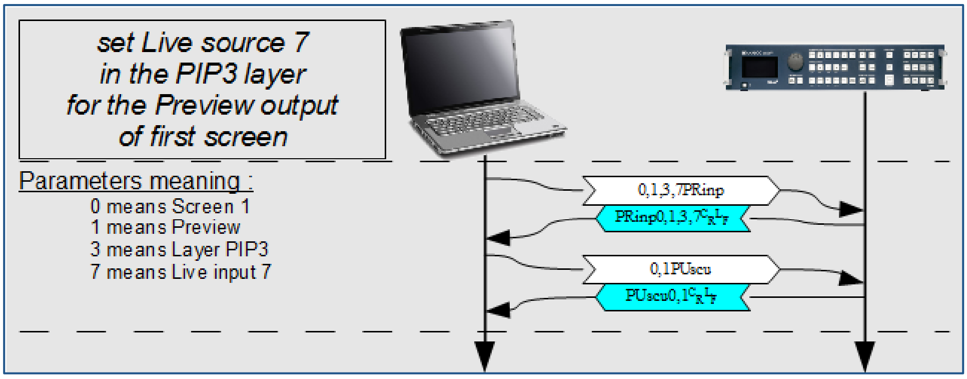
<src> is a value representing the input source.

Note that this command shall be followed by an update command, to be taken in account, as any other preset modification.

<src>	name
0	No input
1	Input 1 or frame 1
2	Input 2 or frame 2
3	Input 3 or frame 3
4	Input 4 or frame 4
5	Input 5 or frame 5
6	Input 6 or frame 6
7	Input 7 or frame 7
8	Input 8 or frame 8
9	Input 9 or frame 9
10	Input 10 or frame 10
11	Color or Black fill

3.3.4 example of Preview layer source change

Picture 11 : example of Preview layer source change



set Live source 7
in the PIP3 layer
for the Preview output
of first screen

Parameters meaning :
0 means Screen 1
1 means Preview
3 means Layer PIP3
7 means Live input 7

```
0,1,3,7PRinp
PRinp0,1,3,7␍␊
0,1PUscu
PUscu0,1␍␊
```

3.4 TAKE : Transitioning a Preview screen onto a Program screen

3.4.1 usage

The “TAKE” action allows transitioning, for a single screen, the “Preview” content (next state) to the “Program” (current state) using the current transition.

3.4.2 Midra™ transitions reminder

Picture modifications can be made directly on the program screen (in the “current preset”) or can be prepared on the preview screen (in the “next preset”) and applied synchronously through the *TAKE* command. The device will try to apply all preset changes in one step, using user parameters. Depending on the complexity of preset changes, it can lack resources and be obliged to use two steps or to sequence the transition. This status can be read through the *TAKEINFO* command.

Before launching a *TAKE* command, the *TAKEAVA* status shall always be read to wait for the device to be ready.

3.4.3 detailed commands sequence

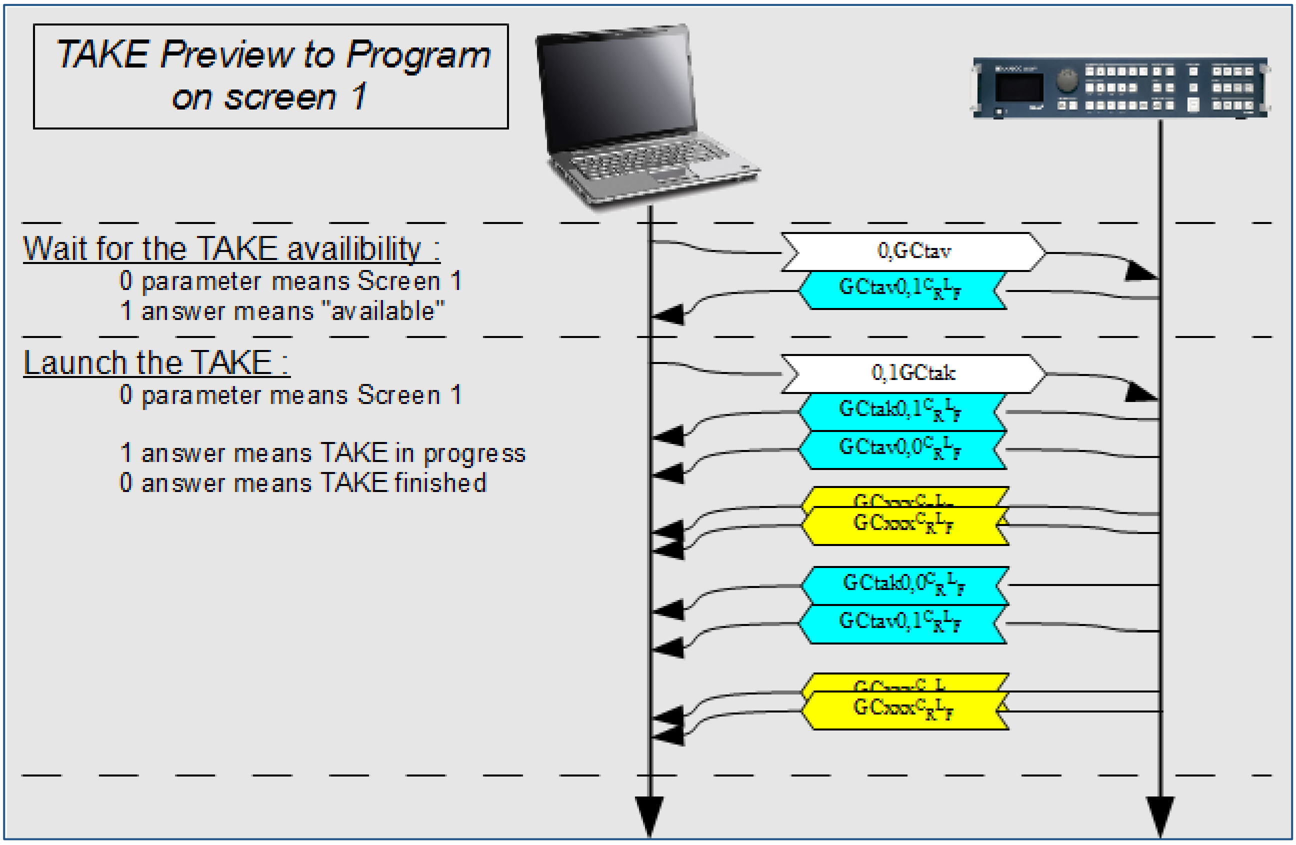
- **Wait for the TAKE availability :**
- **Launch the TAKE action** : Launch the TAKE by writing the value 1.

Syntax: `<scrn>,1GCtak`$^{L}_{F}$ `<scrn>` is the RCS² screen number minus 1.

Only value 1 is allowed, machine will immediately acknowledge the command, then will do the transition and last will answer with the 0 value after the end of the TAKE.

3.4.4 example of TAKE

Picture 12 : example of TAKE


TAKE Preview to Program on screen 1
Wait for the TAKE availibility :
0 parameter means Screen 1
1 answer means "available"
Launch the TAKE :
0 parameter means Screen 1
1 answer means TAKE in progress
0 answer means TAKE finished
0,GCtav
GCtav0,1$^{C}_{R}{}^{L}_{F}$
0,1GCtak
GCtak0,1$^{C}_{R}{}^{L}_{F}$
GCtav0,0$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
GCtak0,0$^{C}_{R}{}^{L}_{F}$
GCtav0,1$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$

3.5 TAKE ALL : Transitioning from Preview onto the Program for all screens (matrix mode)

3.5.1 usage

In Matrix mode, the “TAKE ALL” action allows transitioning for all screens, the “Preview” content (next state) to the “Program” (current state) using the current transition.

3.5.2 Midra™ transitions reminder

Picture modifications can be made directly on either program screens (in the “current preset”) or can be prepared on the preview screens (in the “next preset”) and applied synchronously through the *TAKE ALL* command. The device will try to apply all preset changes in one step on all screens, using user parameters. Depending on the complexity of preset changes, it can lack resources and be obliged to use two steps or to sequence the transition. This status can be read through the *TAKEINFO* command for screen 1 and for screen 2.

Before launching a *TAKE ALL* command, the *TAKEAVA* status shall always be read for screen 1 AND screen 2 in order to wait for the device to be ready.

3.5.3 detailed commands sequence

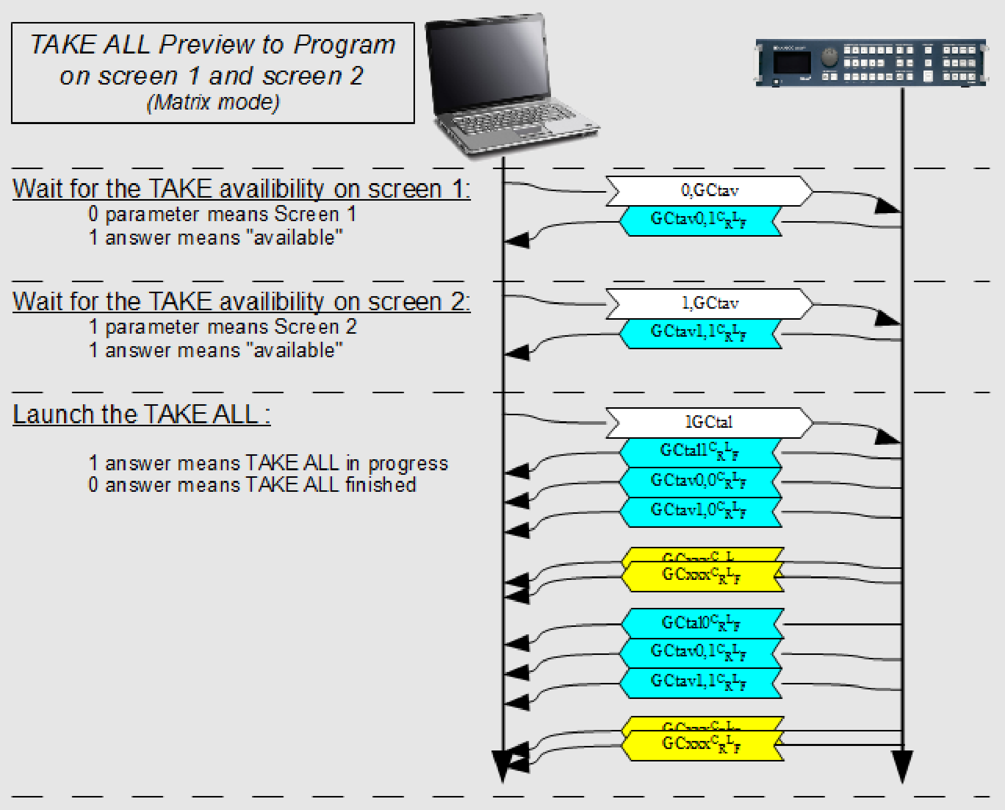
- **Wait for the TAKE availability on screen 1 and on screen 2 :**
- **Launch the TAKE ALL action** : Launch the TAKE ALL by writing the value 1.

Syntax: 1,GCtal$^{L}_{F}$

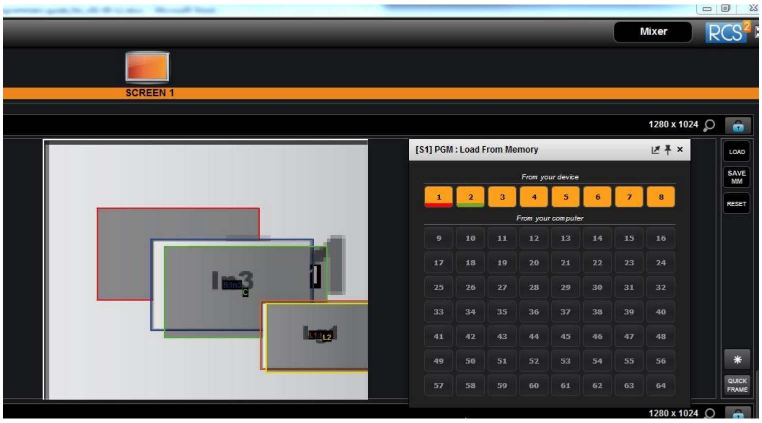
Only value 1 is allowed, machine will immediately acknowledge the command, then will do the transition on both screens and last will answer with the 0 value after the end of the TAKE ALL command.

3.5.4 example of TAKE ALL

Picture 13 : example of TAKE ALL


TAKE ALL Preview to Program
on screen 1 and screen 2
(Matrix mode)
Wait for the TAKE availility on screen 1:
0 parameter means Screen 1
1 answer means "available"
0,GCtav
GCtav0,1$^{C}_{R}{}^{L}_{F}$
Wait for the TAKE availility on screen 2:
1 parameter means Screen 2
1 answer means "available"
1,GCtav
GCtav1,1$^{C}_{R}{}^{L}_{F}$
Launch the TAKE ALL :
1 answer means TAKE ALL in progress
0 answer means TAKE ALL finished
1GCtal
GCtal1$^{C}_{R}{}^{L}_{F}$
GCtav0,0$^{C}_{R}{}^{L}_{F}$
GCtav1,0$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
GCtal0$^{C}_{R}{}^{L}_{F}$
GCtav0,1$^{C}_{R}{}^{L}_{F}$
GCtav1,1$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$

3.6 Loading a Preset from Master memory to a single Program screen



Picture 14 : Loading Preset from memory - RCS²

3.6.1 usage

A simple way to control a Midra™ product is to record the entire screen contents in its memories, using the RCS², during the initial setup. These memories can then be remotely recalled by a controller.

The “recall Preset from memory” action is made of only one command, allowing setting parameters like memory number, destination screen, filters.

A filter allows choosing to not recall some elements, like source or layers positions and sizes, etc. It can be used, for example to include or exclude displayed input from memory recall.

3.6.2 Midra™ “Master preset Memories” reminder

Midra™ allows memorization of up to 8 “Master Memories” inside the device, usable by a controller. The RCS² software gives 56 other memories, stored in the computer running this software. They are not usable by a controller.

A “Master Memory” slot contains all preset elements of the 2 screens of a device. Depending on device type and device mode, only one screen can be available. In this case, the memory part for the second screen still exists, but nothing is stored in it during saving.

The “*PRESET_LOAD_REQUEST*” single command allows to reload a single destination screen, from one of the two part (origin screen) of a “Master Memory” slot.

3.6.3 detailed command parameters

Syntax: `<scrnF>,<mem>,<scrnT>,<progPrev>,<fltr>,1GClrq`

- **<fltr> parameter** : This parameter allows excluding some preset elements from recalling.

The <fltr > value can be calculated by adding the values associated with each element to exclude (filtered elements). Legal values are from 0 up to 1023. Value 0 means that all preset elements will be recalled.

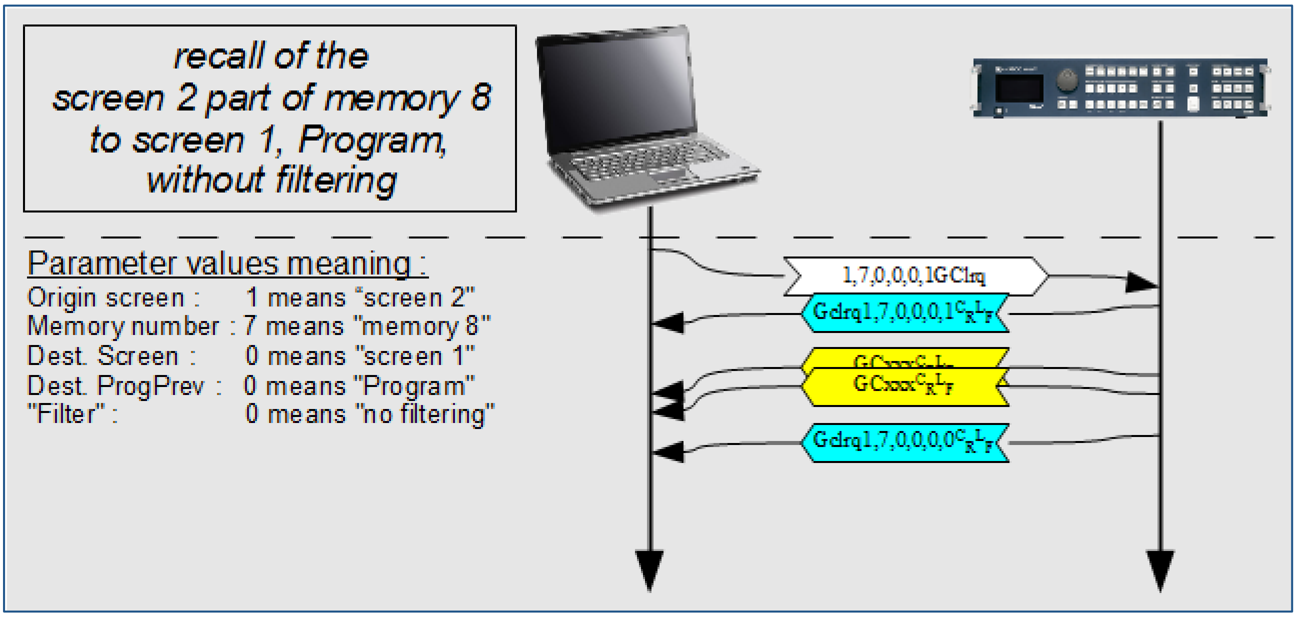
element to recall	value
Auto-Scale	1
source	2
Pos/Size	4
Transparency	8
Cropping	16
Border	32
Transition	64
Timing	128
Effects	256
Audio layer	512
No filter	0

- **<progPrev> parameter :** This parameter gives the destination preset number, either Program (current preset) or Preview (next preset). The recalled preset is immediately displayed after end of recall. If the displayed input has to be changed after memory recall then the Preview preset destination shall be used, followed by the changes of displayed inputs, ended with the TAKE or with the TAKE ALL command. (see the example bellow)

- **<scrnT> parameter :** This parameter gives the destination screen number. If only one screen is available, due to device type or device mode, the screen number **0** shall be used.
- **<mem> parameter :** This parameter gives the memory slot number to load. The allowed range of values is 0 to 7, corresponding to memories 1 to 8.
- **<scrnF> parameter :** This parameter gives the origin screen number, the one from which was recorded the preset. Used with the <scrnT> parameter, they allow loading in a screen a preset stored from the other, in matrix mode.

3.6.4 example of memorized preset recall

Picture 15 : example of memorized preset recall


recall of the
screen 2 part of memory 8
to screen 1, Program,
without filtering
Parameter values meaning :
Origin screen : 1 means "screen 2"
Memory number : 7 means "memory 8"
Dest. Screen : 0 means "screen 1"
Dest. ProgPrev : 0 means "Program"
"Filter" : 0 means "no filtering"
1,7,0,0,0,1GClrq
Gclrq1,7,0,0,0,1CRLF
GCxxxCRLF
GCxxxCRLF
Gclrq1,7,0,0,0,0CRLF

3.7 Loading a Preset from Master memory and changing the source displayed in a layer

3.7.1 usage

This example combines previous examples showing how to use master memories as a "layout template", plus changing a layer source before being displayed on the program output.

3.7.2 detailed commands sequence

- **Load master memory to preview** : See the previous example for parameters explanation. The filter parameter (<fltr>) can be used to avoid recalling the layer sources. In this case, the memory recall will be able to change layer positions and sizes, layer borders, without changing layer content.

  Syntax: `<scrnF>,<mem>,<scrnT>,<prgPrv>,<fltr>,1GClrq`

- **Change the layer source** : See this example for parameters explanation. The *PE_INPUTNUM* command shall be followed by a *PU_SCREEN_UPDATE* command to update the screen.

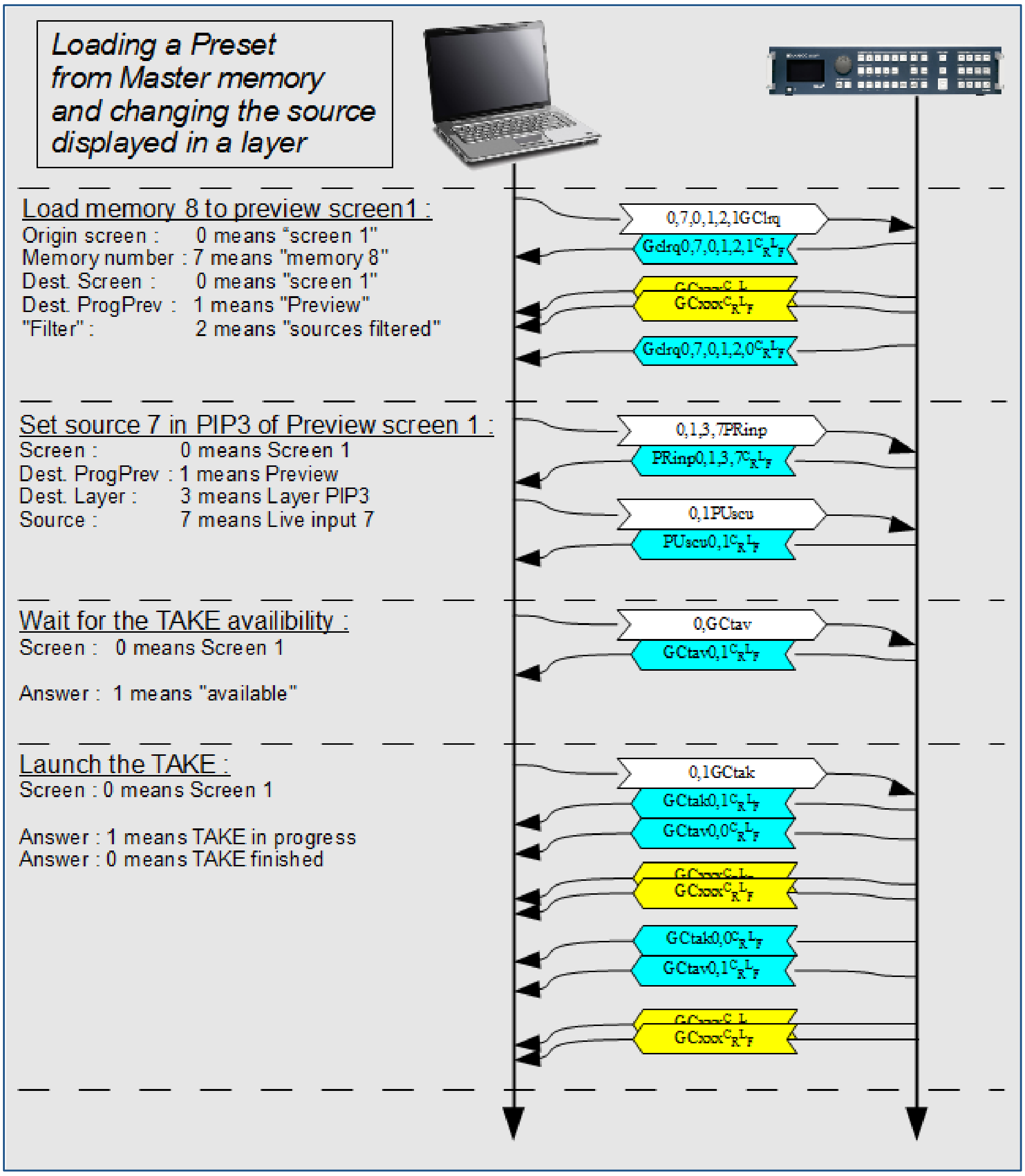
  Syntax: `<scrn>,<progPrev>,<layer>,<src>PRinp`

- **TAKE, launch the transition from preview to program** : See the previous example for parameters explanation. The TAKEAVA status validity shall be tested before launching the *TAKE* command.

  Syntax: `<scrn>,1GCtak␊` <scrn> is the RCS² screen number minus 1.

3.7.3 example of Master Memory preset recall

Picture 16 : example of Master Memory preset recall


Loading a Preset from Master memory and changing the source displayed in a layer
Load memory 8 to preview screen1 :
Origin screen : 0 means "screen 1"
Memory number : 7 means "memory 8"
Dest. Screen : 0 means "screen 1"
Dest. ProgPrev : 1 means "Preview"
"Filter" : 2 means "sources filtered"
0,7,0,1,2,1GClrq
Gclrq0,7,0,1,2,1$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
Gclrq0,7,0,1,2,0$^{C}_{R}{}^{L}_{F}$
Set source 7 in PIP3 of Preview screen 1 :
Screen : 0 means Screen 1
Dest. ProgPrev : 1 means Preview
Dest. Layer : 3 means Layer PIP3
Source : 7 means Live input 7
0,1,3,7PRinp
PRinp0,1,3,7$^{C}_{R}{}^{L}_{F}$
0,1PUscu
PUscu0,1$^{C}_{R}{}^{L}_{F}$
Wait for the TAKE availibility :
Screen : 0 means Screen 1
Answer : 1 means "available"
0,GCtav
GCtav0,1$^{C}_{R}{}^{L}_{F}$
Launch the TAKE :
Screen : 0 means Screen 1
Answer : 1 means TAKE in progress
Answer : 0 means TAKE finished
0,1GCtak
GCtak0,1$^{C}_{R}{}^{L}_{F}$
GCtav0,0$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
GCtak0,0$^{C}_{R}{}^{L}_{F}$
GCtav0,1$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$
GCxxx$^{C}_{R}{}^{L}_{F}$

3.8 Displaying/Hiding the Quick Frame for one single screen

3.8.1 usage

The Quick Frame is the emergency frame that can be displayed in front of every layer for a given screen. This example shows how to enable the Quick Frame for Screen 1.

3.8.2 detailed command sequence

- **Displaying the Quick Frame**

Syntax: `<scrn>,1CTqfa` <scrn> is the RCS^2 screen number minus 1.

Answer: `CTqfa<scrn>,1CRLF` 1 indicating that the frame configured as the Quick Frame for <screen> is now displayed above all layers.

- **Hiding the Quick Frame**

Syntax: `<scrn>,0CTqfa` <scrn> is the RCS^2 screen number minus 1.

Answer: `CTqfa<scrn>,0CRLF` 0 indicating that the frame configured as the Quick Frame for <screen> is now hidden.

3.9 Displaying/Hiding the Quick Frames for all screens (matrix mode)

3.9.1 usage

Two different frames can be defined as Quick Frames (or emergency frames) respectively for screen 1 and screen 2. This example shows how to enable the configured Quick Frames simultaneously on Screen 1 and Screen 2.

3.9.2 detailed command sequence

- **Displaying the Quick Frames on both screens simultaneously**

Syntax: `1CTqfl`

Answer: `CTqfl1CRLF` 1 indicating that the Quick Frames configured for Screen 1 and Screen 2 are now displayed above all layers on both screens.

- **Hiding the Quick Frames on both screens simultaneously**

Syntax: `0CTqfl`

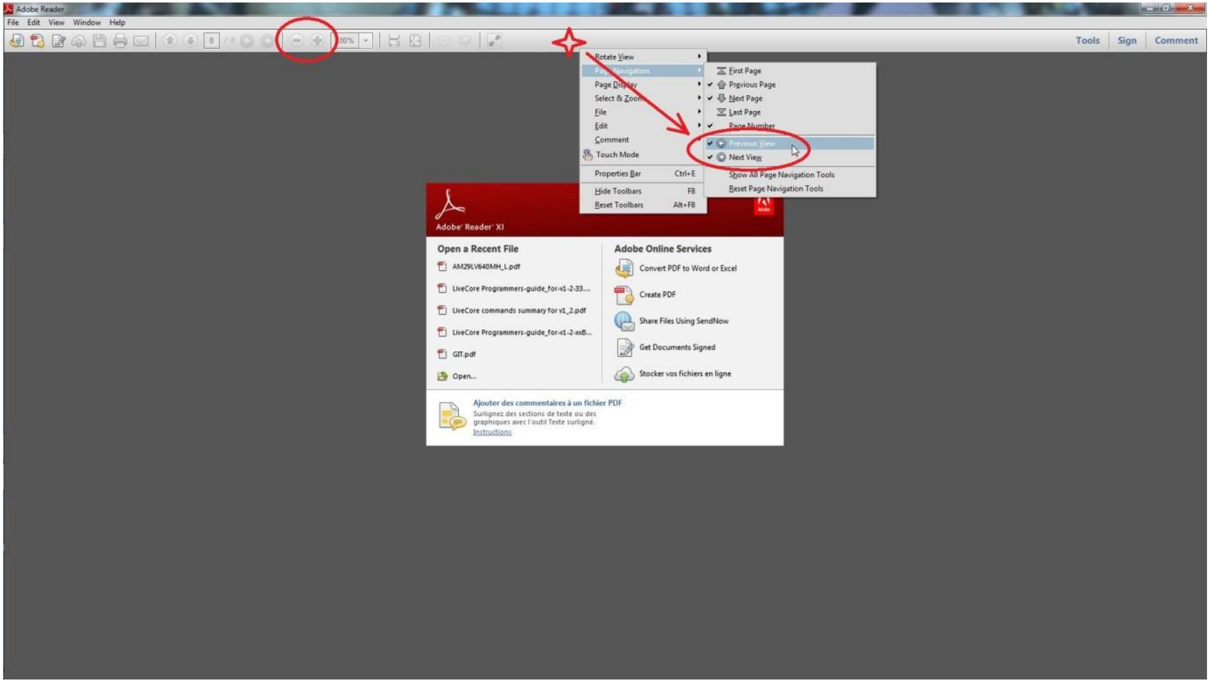
Answer: `CTqfl0CRLF` 0 indicating that the Quick Frames configured for Screen 1 and Screen 2 are now hidden on both screens.

4 NOTES

4.1 Using this document

This document contains many internal links. You can improve your navigation by using the “previous page” function, as in the following example:

Picture 17 : PDF reader, Previous and Next page buttons



Surlignez des sections de texte ou des graphiques avec l'outil Texte surligné.

Follow us

ANALOG WAY®
Pioneer in Analog, Leader in Digital

V1.00 – Nov/03/2017