Linn KINOS^{TM} - RS232 ASCII Interface Specification and Commands

Linn KINOS™ - RS232 ASCII Interface Specification and Commands

Revision 1.01

Revision History

Revision	Description	Author	Date
0.1	Initial Draft (modified from Kisto V1.06)	Ian Wilson	13th February 2004
1.01	Released documentation	Ian Wilson	24th February 2004

Table of Contents

Introduction

This document describes how to control the Kinos System Controller and peripherals through an RS232 interface.

Please note that the information contained in this document is preliminary and will be subject to change.

There are three main sections to this document:

1: Message Protocol

- This section describes how commands are constructed and how they may be used.

0 System Commands

- This section lists the commands, which allow the Kinos to be used as part of a system driven through an RS232 interface.

2: Kinos Commands

- This section defines a list of commands for controlling Kinos. This section is further subdivided into the subsets of the Kinos commands.

1: Message Protocol

1.1: Overview

The RS232 interface on the Kinos allows it to be controlled by a touch screen, PC or any computer with an RS232 port. The Kinos obeys the commands received through the RS232 interface and replies to confirm successful or unsuccessful operation.

The RS232 interface uses an initial response then final response method to acknowledge receiving the command and then completing the task. The interface also supports device and group identifiers to allow a number of units to be connected together. The controlling device can also supply a source identification, which the Kinos will echo as the destination for the replies.

Previous products were termed as slave devices, in regards that nothing was transmitted until something was received, e.g. a task or status command. The Kinos however, will transmit unsolicited messages when something within the product changes, e.g. the volume changes.

1.2: Message Syntax

The general syntax is as follows: `(Source_ID)(Group_ID)(Destination_ID) Command NL`

Where:

`Source_ID`   Syntax: **`#Source_ID#`**

is a unique identifier, used to denote the source of the message. Enclosed by the '**#**' delimiter, the maximum identifier size is 20 ASCII alphanumeric characters (excluding spaces).

`Destination_ID`   Syntax: **`@Destination_ID@`**

is a unique identifier, used to denote the destination of the message. Enclosed by the '@' delimiter, the maximum identifier size is 20 ASCII alphanumeric characters (excluding spaces).

`Group_ID`   Syntax: **`&Group_ID&`**

is a unique identifier, used to denote a specific group of products. Enclosed by the '&' delimiter, the maximum identifier size is 20 ASCII alphanumeric characters (excluding spaces).

`Command`   Syntax: **`$Command$`**

is the command from the host for the product. Enclosed by the '$' delimiter.

`NL`   Syntax: 13dec and 10dec (0Dhex and 0Ahex)

are the line termination characters, carriage return and line feed.

Note:
Nesting of fields is not permissible, nor is the use of the special delimiter characters as part of the field strings themselves, unless they are expressed as an escape sequence (see Appendice B : Escape Sequences).

Spaces are permissible before and after an identifier, but are not allowed within the actual identifier, unless they are expressed as an escape sequence (see Appendice B : Escape Sequences).

For example, **`# recorddeck #`** *is valid*   *whereas* **`# record deck #`** *is invalid.*

By using an escape sequence, the second example becomes valid, i.e. **`# record\0x20deck #`**

1.3: Identifier Considerations

The full transmission format uses four fields as shown.

#Source_ID# &Group_ID& @Destination_ID@ $Message$

Where fields are omitted the results are defined in the following notes.

```
...........  ..........  ................  $Message$        refer to note 1

...........  ..........  @Destination_ID@ $Message$         refer to note 2

..........  &Group_ID&  ................  $Message$          refer to note 3

..........  &Group_ID&  @Destination_ID@ $Message$           refer to note 4

#Source_ID#  ..........  ................  $Message$          refer to note 5

#Source_ID#  ..........  @Destination_ID@ $Message$           refer to note 6

#Source_ID#  &Group_ID&  ................  $Message$          refer to note 7

#Source_ID#  &Group_ID&  @Destination_ID@ $Message$           refer to note 8
```

Note	Details
1	- A product recognising the command will issue an initial response and try to perform the task.
	- A successful or unsuccessful final response will be issued subsequently.
	- Products not recognising the command will remain silent.
	- If no product recognises the command then there will be no reply.
	- If more than one product recognises the command then there may be a comms clash on the replies.
2	- The destination product is responsible for all replies.
	- Invalid commands will generate an error response.
	- The replying product will transfer the destination to the source field on a reply.
	- All products not matching the destination must remain silent and not attempt to handle the command.
	- If two products have the same id, then a comms clash may occur.
3	- All products within the group should attempt the task.
	- Products out with the group should ignore the task.
	- There are no replies from any boxes.
4	- All products within the group should attempt the task.
	- Products out with the group should ignore the task.
	- Only the product, which matches the destination identity, should reply.
	- Invalid commands will generate an error response.
	- If there are more than two products in the group with the same destination identity then a comms clash may occur.
	- The destination identity becomes the source identity in any reply traffic.
5	- As for note 1, with the source identity becoming the destination identity in any replies.
6	- As for note 2, with the source identity becoming the destination identity in any replies.
7	- As for note 3. There are no replies.
8	- As for note 4, with the source identity becoming the destination identity in any replies.

1.4: Syntax of Commands and Responses

1.4.1: Command Syntax

The command message has two variations:

1.4.1.1: Command Help
This allows the host to find out what type of parameters the command requires.

Syntax: **`$? Command$NL`**

Where:
`$`	is the command start delimiter
`?`	is a request for help
`Command`	is the command help request is for
`$`	is the command end delimiter
`NL`	are the line termination characters - carriage return, line feed.

Additionally, if '**`Command`**' is a '**`?`**' then the command set of the product will be provided, with an initial response followed by a final response for each command supported by the product.

This is a change to the previous method, where the command set of the product was output as a single response, with each command being separated from the next by a space and no help text was included.

Note:
Command help is product dependent and is implemented on the Unidisc.

1.4.1.2: Command
This is the method by which the host controls the product

Syntax: **`$Command (Param (Param ……))$NL`**

Where:
`$`	is the command start delimiter
`Command`	is the command string
`Param`	is the parameter string (0 or more)
`$`	is the command end delimiter
`NL`	are the line termination characters - carriage return, line feed.

Note:
Parameters required are command dependent

1.4.2: Solicited Response Overview

When replies are made an initial response and final response are issued. It is unwise for the host to issue further commands until the final response has been received. Section 1.3: Identifier Considerations, describes the action of identifiers on these replies and specifies rules which may also suppress the replies.

1.4.2.1: Initial Response

This will be given on receipt of a valid command and for a positive acknowledge will be of the form:

`(Source_ID)(Group_ID)(Destination_ID)!`

In this way, the host quickly knows that the destination has received and understood the command.

1.4.2.1.1: Initial Response Failure

This will be given on receipt of an invalid command and will be of the form:

`(Source_ID)(Group_ID)(Destination_ID)!$FAIL sc fn$`

Where '**sc**' is a status code (see section 1.8.1.1: Status Codes) specifying why the task could not be completed, and '**fn**' specifies which field was responsible.

Note: There is no final response.

1.4.2.2: Final Response

This will be given on completion of the task and will be of the form:

`(Source_ID)(Group_ID)(Destination_ID)!$Status_String$`

The status string will be a unique response to the originating command.

1.4.2.2.1: Final Response Failure

This will be given where a task could not be completed and will be of the form:

`(Source_ID)(Group_ID)(Destination_ID)!$FAIL sc fn$`

Where '**sc**' is a status code (see section 1.8.1.1: Status Codes) specifying why the task could not be completed, and '**fn**' specifies which field was responsible.

Note:
[1] *In all cases, identifiers will only be returned as part of the response if supplied as part of the command (refer to section 1.3: Identifier Considerations for further details).*

[2] *Fields are numbered from left to right, starting at 1.*

1.4.3: Unsolicited Response Overview

Unsolicited responses are an addition to the RS232 protocol, and are generated automatically by the product to inform the host of a change to the products status.

1.4.3.1: Unsolicited Response

This will be given at any time during the operation of the product and will be of the form:

```
(Source_ID)$Status_String$
```

The major differences between solicited and unsolicited responses are as follows:

1. Unsolicited messages can occur at any time (if activated).
2. Source identifier, if present within product settings, will always form part of the message.
3. No exclamation mark is included before the command delimiter.

Note:
Refer to user guide regarding activation of unsolicited responses (user option '*Enable RS232 Events*').

1.4.3.2: Unsolicited Response Events

Should the Kinos have parameters changed for any of the following events, then an unsolicited response should be generated.

1. Volume increment/decrement from IR / RC5 command or front panel keypad.
2. Input Source profile selection from IR / RC5 command or front panel keypad.
3. Record Path changes from IR / RC5 (including Knekt driver commands) or from front panel keypad.
4. Decode algorithm change from IR / RC5 / front panel keypad or stream change.
5. Standby toggle from IR / RC5 command or front panel keypad.
6. Mute toggle from IR / RC5 command or front panel keypad.

These are not available at present within the product.

System Commands
The following commands allow the Kinos to be part of a system driven through an RS232 interface.

1.5: Identity Commands

1.5.1: ID

Configure the product on a one to one basis

```
$ID identifier$
$ID identifier$
Write product identifier
```

```
$ID ~identifier$
$ID$
Remove  product identifier
```

```
$ID ?$
$ID identifier$
Return  product identifier
```

1.5.2: GID

Configures a product as part of a group so that it can be accessed a number of ways

```
$GID identifier$
$GID identifier$
Write group identifier (product now becomes part of a group of products)
```

```
$GID ~identifier$
$GID identifier [identifier […]]$
Remove a product from a particular group
```

```
$GID ?$
$GID identifier [identifier […]]$
Return list of currently defined group identifiers from product
```

Notes on Groups:

A product can be a member of at most 5 groups to allow it to be addressed in a variety of ways.

While in group mode, products with the same group ID will react in the same way to product specific commands sent to them using the Group_ID syntax (&group_id&).

In addition, products in Group Mode will not acknowledge receipt of commands from the host. This is to avoid all products in the group potentially responding at the same time.

Each product can be polled individually at the end of a group mode command to check they have all been updated correctly.

1.6: Communication Commands

1.6.1: BAUD

> ⚓ **`$BAUD baudrate$`**
>
> ⚓ `!$BAUD baudrate$`
>
> 📖 *Select new baud rate from the following: 4800, 9600, 14400^3, 19200, 28800^3, 38400, 57600^3, 115200^3, 230400^3*

> ⚓ **`$BAUD ?$`**
>
> ⚓ `!$BAUD baudrate$`
>
> 📖 *Returns current baud rate (see above)*

Note:

[1] *Initial and final responses will be at the current baud rate, before the new baud rate is implemented.*

[2] *Baud rate defaults to 9600 when the product is initialised.*

[3] *New baud rates supported by this product.*

[4] *2400 baud rate not supported by this product.*

1.6.2: RESET

Return product comms buffers to a known state

> ⚓ **`$RESET$`**
>
> ⚓ `!$RESET$`
>
> 📖 *Clear communications buffer on product*

1.6.3: ECHO

> ⚓ **`$ECHO text$`**
>
> ⚓ `!$ECHO <text>$`
>
> 📖 *Echo's the text back enclosed in < and >*

This command is used ease the burden of initial set-up of host-product communications, the product will echo the parameter provided back to the host.

Note:

If no identifiers are supplied with this command, then all devices connected to a system will respond, which may result in a comms clash.

1.6.3.1: Power_Up Message

A power up message is provided which is transmitted to the host in order to verify that the host / product link is working.

The power up message on the Kinos is as follows: **`!$KINOS$`**

This feature is enabled / disabled via a user option (see ??? in the user manual)

1.7: Polling Command

1.7.1: POLL

Polling is used to extract details of all products connected to the host

```
⬇ $POLL START$
⚓ !$POLL START$
📖 Marks the start of polling
```

```
⬇ $POLL ID$
⚓ !$POLL ID product identifier$
📖 Returns product identifier
```

```
⬇ $POLL SLEEP$
⚓ !$POLL SLEEP$
📖 Product responding to this will ignore all further commands until 'POLL DONE' is received
```

```
⬇ $POLL DONE$
⚓ No response to this command
📖 All products will now return to active operation
```

Important

The 'POLL SLEEP' command should be used with the product identifier returned by 'POLL ID'.

If this is not done then all the products will stop responding and the polling sequence will fail.

1.7.2: Polling Explained

The RS232 interface hardware, via the **POLL** command, allows communication to daisy-chained RS232 controlled devices. The devices must be capable of buffering data for transmission as required.

Using this feature allows the host to 'auto-detect' the slave products on the RS232 link.

By taking advantage of this, it is possible to identify what is on the link using the following type of algorithm:

$POLL START$

- opens return-path switches in all devices, so only first device in chain can respond

$POLL ID$

- all devices respond but only response from first device reaches host

@dest_1_id@$POLL SLEEP$

- where '**dest_1_id**' is the result of the previous '**POLL ID**'
- matching product closes its switch
- product will not respond to any command now until '**POLL DONE**' command received.

$POLL ID$

- second device can now respond with it's ID

@dest_2_id@$POLL SLEEP$

- where '**dest_2_id**' is the result of the previous '**POLL ID**'
- matching product closes its switch
- product will not respond to any command now until '**POLL DONE**' command received.

The '**POLL ID**' and '**POLL SLEEP**' commands are issued repeatedly until all products have been queried and there is no response from the last '**POLL ID**' command.

$POLL ID$

- no response since all product id's read, so time-out

$POLL DONE$

- resync all products on link again

1.8: Status Command

The status command has been provided as a debugging aid, i.e. the host can find out why a command was not processed.

1.8.1: STATUS

> ⚓ `$STATUS$`
> ⚓ `!$STATUS sc (sv)$`
> 📖 *Return the status of the last command*

Where '`sc`' is the returned status code and '`sv`' is the status value (only used with code 25 for now). Codes are allocated on a block basis for each product with the first 48 codes reserved for general use.

1.8.1.1: Status Codes

The following table lists the General Status Codes which all products support.

Code		*Description*
00 (0x00)	-	No error
01 (0x01)	-	Unexpected termination of command line
02 (0x02)	-	Unrecognised or misplaced character in command line
03 (0x03)	-	Corrupted command message (within $....$)
04 (0x04)	-	Start of another source identifier, identifier has already been supplied
05 (0x05)	-	Start of another group identifier, identifier has already been supplied
06 (0x06)	-	Start of another destination identifier, identifier has already been supplied
07 (0x07)	-	Source identifier is too large, maximum of 20 characters
08 (0x08)	-	Group identifier is too large, maximum of 20 characters
09 (0x09)	-	Destination identifier is too large, maximum of 20 characters
10 (0x0A)	-	Source identifier corrupted
11 (0x0B)	-	Group identifier corrupted
12 (0x0C)	-	Destination identifier corrupted
13 (0x0D)	-	Unknown group identity
14 (0x0E)	-	Unknown destination identity
15 (0x0F)	-	Unknown command
16 (0x10)	-	Unknown command parameter
17 (0x11)	-	Parameter missing from **ID** command
18 (0x12)	-	Unknown product identifier, cannot delete
19 (0x13)	-	Parameter missing from **GID** command
20 (0x14)	-	Cannot delete group identifier, unknown
21 (0x15)	-	Cannot add new group identifier, already exists
22 (0x16)	-	Cannot add new group identifier, list full
23 (0x17)	-	Polling must be activated by the **POLL START** command
24 (0x18)	-	Only **POLL ID, SLEEP** or **DONE** commands accepted during polling
25 (0x19)	-	Message exceeded maximum allowable length
	-	'**sv**' defines maximum length (upto and including CR, and excluding LF)
26 (0x1A) up to 47 (0x3F)	-	Reserved

1.9: IR

⚓ **`$IR ?$`**
⚓ `!$IR ON$`
⚓ `!$IR OFF$`
📖 *Return current IR control status*

⚓ **`$IR [Y|ON]$`**
⚓ `!$IR ON$`
📖 *Enable IR control of product*

⚓ **`$IR [N|OFF]$`**
`!$IR OFF$`
📖 *Disable IR control of product*

1.10: INIT

⚓ **`$INIT$`**
⚓ `!$INIT$`
📖 *Resets all product parameters back to factory defaults*

1.11: VERSION

⚓ **`$VERSION SOFTWARE ?$`**
⚓ `!$VERSION SOFTWARE H8 v PPC v DSP0 v DSP1 v DIGFPGA v VIDFPGA v$`
📖 *Return current versions of system software.*

Notes:

The software versions returned are as follows ...

Field ...	Software for ...	Current version ...
H8	H8 Processor	?
PPC	Power PC	S1790102
DSP0	Digital Signal Processor 0	S1210101
DSP1	Digital Signal Processor 1*	S1220101
DIGFPGA	Digital FPGA	S0870101
VIDFPGA	Video FPGA	P1160201

Note: *DSP1 is only valid for the Kinos+

The format of '**v**' *is* '**tpppvvvv**'

> *where:* '**t**'   is the software release type ('**P**' = prototype and '**S**' = release)
> '**ppp**'   is the software identifier (3 digits)
> '**vvvv**'   is the software version (4 digits)

The software versions should be read from the appropriate storage area each time this command is received, and should not be stored as hard coded strings as they will then be incorrect when software is updated.

1.12: COUNTER

$COUNTER POWER ?$

!$COUNTER POWER days:hours:minutes:seconds$

Returns total powered up (operational) time.

$COUNTER MAINS ?$

!$COUNTER MAINS days:hours:minutes:seconds$

Returns total mains connected time

2: Kinos Commands

The following pages contain the command set for the Kinos System Controller.

Important:

[1] Parameters must be separated from commands and each other by at least one space character

[2] Where a command can be enabled or disabled then

> **Y** or **ON** will enable (turn on) the setting and **N** or **OFF** will disable (turn off) the setting

2.1: Command Help

Command help is implemented by the Kinos and will give the host details for any given command.

> for example: **$? SEARCH$**

> replies with: **!$? SEARCH (?|+|-|int|+int|-int|<|>|STOP)**

2.2: System Commands

The system commands supported by the Kinos are **ID**, **GID**, **BAUD**, **RESET**, **ECHO**, **POLL**, **STATUS,INIT, IR,VERSION,COUNTER** and Power_Up Message. These are all explained in section 0 System Commands of this document.

2.3: Other Commands

2.3.1: STANDBY

```
📥 $STANDBY ?$
📤 !$STANDBY [ON|OFF]$
📖 Return current standby status.
```

```
📥 $STANDBY [Y|ON]$
📤 !$STANDBY [ON|OFF]$
📖 Enter standby.
```

```
📥 $STANDBY [N|OFF]$
📤 !$STANDBY [ON|OFF]$
📖 Exit standby.
```

```
📥 $STANDBY TOGGLE$
📤 !$STANDBY [ON|OFF]$
📖 Toggle standby.
```

2.3.2: MUTE

```
📥 $MUTE ?$
📤 !$MUTE [ON|OFF]$
📖 Return current mute status.
```

```
📥 $MUTE [Y|ON]$
📤 !$MUTE [ON|OFF]$
📖 Mute on.
```

```
📥 $MUTE [N|OFF]$
📤 !$MUTE [ON|OFF]$
📖 Mute off.
```

```
📥 $MUTE TOGGLE$
📤 !$MUTE [ON|OFF]$
📖 Toggle mute.
```

2.3.3: OSG (On Screen Graphics)

⚓ **`$OSG ?$`**
⚓ `!$OSG [ON|OFF]$`
📖 *Return current OSG status.*

⚓ **`$OSG [Y|ON]$`**
⚓ `!$OSG ON$`
📖 *Turn OSG on.*

⚓ **`$OSG [N|OFF]$`**
⚓ `!$OSG OFF$`
📖 *Turn OSG off.*

⚓ **`$OSG TOGGLE$`**
⚓ `!$OSG [ON|OFF]$`
📖 *Toggle OSG.*

2.3.4: QUIET (Midnight Movie)

⚓ **`$QUIET ?$`**
⚓ `!$QUIET [ON|OFF] level$`
📖 *Return current quiet mode status and level, where level is 1, 2 or 3 (1 = low, 2 = medium, 3= high)*

⚓ **`$QUIET [Y|ON]$`**
⚓ `!$QUIET ON level$`
📖 *Turn quiet mode on (using current level).*

⚓ **`$QUIET [N|OFF]$`**
⚓ `!$QUIET OFF level$`
📖 *Turn quiet mode off.*

⚓ **`$QUIET level$`**
⚓ `!$QUIET [ON|OFF] level$`
📖 *Set quiet level (on/off state remains the same).*

2.3.5: VOLUME

$VOLUME ?$

!$VOLUME value$

Return current setting.

$VOLUME [+|-]$

!$VOLUME value$

Increase or decrease current setting by one.

$VOLUME [+|-]value$

!$VOLUME value$

Increase or decrease current setting by supplied value.

$VOLUME = [+]value$

!$VOLUME value$

Set to absolute value supplied.

$VOLUME LIMITS$

!$VOLUME LIMITS min value max value$

Return minimum and maximum settings (0 to 100 inclusive).

Note: The Kinos volume can be specified in half steps, for example, a volume of 75.5 is now permissible.

2.3.6: BALANCE

2.3.6.1: BALANCE / BALANCE_LR (left/right)

⚓ **`$[BALANCE|BALANCE_LR] ?$`**

⚓ `!$[BALANCE|BALANCE LR] value$`

📖 *Return current setting (-ve is left biased, +ve is right biased).*

⚓ **`$[BALANCE|BALANCE_LR] [+|-]$`**

⚓ `!$[BALANCE|BALANCE LR] value$`

📖 *Increase or decrease current setting by one.*

⚓ **`$[BALANCE|BALANCE_LR] [+|-]value$`**

⚓ `!$[BALANCE|BALANCE LR] value$`

📖 *Increase or decrease current setting by supplied value.*

⚓ **`$[BALANCE|BALANCE_LR] = [+|-]value$`**

⚓ `!$[BALANCE|BALANCE LR] value$`

📖 *Set to absolute value supplied.*

⚓ **`$[BALANCE|BALANCE_LR] LIMITS$`**

⚓ `!$[BALANCE|BALANCE LR] LIMITS min value max value$`

📖 *Return minimum and maximum settings (-10 to +10 inclusive).*

2.3.7: LIPSYNC

⚓ **`$LIPSYNC ?$`**

⚓ `!$LIPSYNC value$`

📖 *Return current setting.*

⚓ **`$ LIPSYNC [+|-]$`**

⚓ `!$LIPSYNC value$`

📖 *Increase or decrease current setting by one.*

⚓ **`$ LIPSYNC [+|-]value$`**

⚓ `!$LIPSYNC value$`

📖 *Increase or decrease current setting by supplied value.*

⚓ **`$ LIPSYNC = [+]value$`**

⚓ `!$LIPSYNC value$`

📖 *Set to absolute value supplied.*

⚓ **`$ LIPSYNC LIMITS$`**

⚓ `!$ LIPSYNC LIMITS min value max value$`

📖 *Return minimum and maximum settings (0 to 255 inclusive, 0 means off).*

2.3.8: SPEAKER

Speaker 'CALIBRATION' allows for the fine adjustment of each speaker volume, ensuring that each speaker provides the same relative volume to the listener, relative to it's location in the room.

Calibration should only need to be done on the initial setup of a system, or when some aspect of the system changes, i.e. new speakers or a moving the system into a different room. The calibration settings are then used as the 'base' settings for the system and are retained at all times.

Speaker 'TRIM' allows for on-the-fly adjustment of speaker pairs, in order to provide a temporary method of boosting or curtailing the audio when required. Speaker trim values will be lost when the system is put into standby, or when the system is "normalised".

2.3.8.1: SIZE

⚓ `$SPEAKER SIZE speaker ?$`

⚓ `!$SPEAKER SIZE speaker size$`

📖 *Return current speaker size setting.*

⚓ `$SPEAKER SIZE speaker size$`

⚓ `!$SPEAKER SIZE speaker size$`

📖 *Set speaker size.*

---

where **speaker** and **size** parameters are …

	speaker	*size*
front left and right	**FRONT**	**[SMALL|LARGE]**
front centre	**CENTRE**	**[NONE|SMALL|LARGE]**
surround left and right	**SURROUND**	**[NONE|SMALL|LARGE]**
surround back left and right	**SURROUND_BACK**	**[NONE|SMALL|LARGE]**
subwoofer	**SUBWOOFER**	**[NONE|PRESENT]**

2.3.8.2: CALIBRATE

⚓ **$SPEAKER CALIBRATE speaker ?$**

⚓ !$SPEAKER CALIBRATE speaker calibration$

📖 *Return current setting.*

⚓ **$SPEAKER CALIBRATE speaker [+|-]$**

⚓ !$SPEAKER CALIBRATE speaker value$

📖 *Increase or decrease current setting by one.*

⚓ **$SPEAKER CALIBRATE speaker [+|-]calibration$**

⚓ !$SPEAKER CALIBRATE speaker calibration$

📖 *Increase or decrease current setting by supplied value.*

⚓ **$SPEAKER CALIBRATE speaker = [+|-]calibration$**

⚓ !$SPEAKER CALIBRATE speaker calibration$

📖 *Set to absolute value supplied.*

⚓ **$SPEAKER CALIBRATE LIMITS$**

⚓ !$SPEAKER CALIBRATE LIMITS min calibration max calibration $

📖 *Return minimum and maximum settings.*

---

where **speaker** and **calibration** parameters are …

	speaker	*calibration (min)*	*calibration (max)*	*step size*
front left	**FRONT_LEFT**	−30	30	0.5
front centre	**FRONT_CENTRE**	−30	30	0.5
front right	**FRONT_RIGHT**	−30	30	0.5
surround left	**SURROUND_LEFT**	−30	30	0.5
surround right	**SURROUND_RIGHT**	−30	30	0.5
surround back left	**SURROUND_BACK_LEFT**	−30	30	0.5
surround back right	**SURROUND_BACK_RIGHT**	−30	30	0.5
subwoofer	**SUBWOOFER**	−30	30	0.5

2.3.8.3: TRIM

⚓ **`$SPEAKER TRIM speaker ?$`**

⚓ `!$SPEAKER TRIM speaker trim$`

📖 *Return current setting.*

⚓ **`$SPEAKER TRIM speaker [+|-]$`**

⚓ `!$SPEAKER TRIM speaker value$`

📖 *Increase or decrease current setting by one.*

⚓ **`$SPEAKER TRIM speaker [+|-]trim$`**

⚓ `!$SPEAKER TRIM speaker trim$`

📖 *Increase or decrease current setting by supplied value.*

⚓ **`$SPEAKER TRIM speaker = [+|-]trim$`**

⚓ `!$SPEAKER TRIM speaker trim$`

📖 *Set to absolute value supplied.*

⚓ **`$SPEAKER TRIM LIMITS$`**

⚓ `!$SPEAKER TRIM LIMITS min trim max trim$`

📖 *Return minimum and maximum settings.*

---

where **`speaker`** and **`trim`** parameters are …

	speaker	*trim* (min)	*trim* (max)	*step size*
front centre	`CENTRE`	**-10**	**10**	**0.5**
surround left and right	`SURROUND`	**-10**	**10**	**0.5**
surround back left and right	`SURROUND_BACK`	**-10**	**10**	**0.5**
subwoofer	`SUBWOOFER`	**-10**	**10**	**0.5**
low frequency effects	`LOW_FREQUENCY_EFFECTS`	**-10**	**0**	**0.5**

2.3.9: SURROUND

⚓ `$SURROUND ?$`

⚓ `!$SURROUND mode [mode [...]]$`

📖 *Return current surround mode, and a list of currently available surround modes.*

⚓ `$SURROUND [+|-]$`

⚓ `!$SURROUND mode$`

📖 *Select next or previous (available) surround mode*

⚓ `$SURROUND mode$`

⚓ `!$SURROUND mode$`

📖 *Select absolute surround mode.*

📖 *Reply will return actual surround mode selected, i.e. requested mode may not match returned mode.*

Full list of surround **mode** parameters …

Mode	Algorithm	Mode	Algorithm
0	AUTO	21	DOLBY DIGITAL
1	STEREO	22	DOLBY DIGITAL EX
2	ANALOG STEREO	23	DOLBY PRO LOGIC II
3	SACD STEREO	24	DOLBY PRO LOGIC II MUSIC
4	DVD-A STEREO	25	DOLBY PRO LOGIC II EX
5	STEREO SUB	26	DOLBY PRO LOGIC II MUSIC EX
6	ANALOG STEREO SUB	27	DOLBY HEADPHONES
7	SACD STEREO SUB	28	DOLBY HEADPHONES ROOM 1
8	DVD-A STEREO SUB	29	DOLBY HEADPHONES ROOM 2
9	PHANTOM	30	DOLBY HEADPHONES ROOM 3
10	ANALOG PHANTOM	31	DTS CD
11	SACD PHANTOM	32	DTS DIGITAL SURROUND
12	DVD-A PHANTOM	33	DTS ES MATRIX
13	3 STEREO	34	DTS ES DISCRETE
14	ANALOG 3 STEREO	35	DTS 96/24
15	SACD 3 STEREO	36	MPEG STEREO
16	DVD-A 3 STEREO	37	MPEG SURROUND
17	MULTI-CHANNEL	38	AAC STEREO
18	ANALOG MULTI-CHANNEL	39	AAC SURROUND
19	SACD MULTI-CHANNEL	40	LIMBIK PARTY
20	DVD-A MULTI-CHANNEL	41	LIP SYNC

2.3.10: INPUT

2.3.10.1: PROFILE

⚓ `$INPUT PROFILE ?$`

⚓ `!$INPUT PROFILE number (name)$`

📖 *Return currently selected profile (user defined audio, video or audio/video input)*

⚓ `$INPUT PROFILE [number|name]$`

⚓ `!$INPUT PROFILE number (name)$`

📖 *Select absolute profile (NONE means select no profile).*

📖 *The profile can be selected by using the appropriate profile number or (user defined) name*

⚓ `$INPUT PROFILE +$`

⚓ `!$INPUT PROFILE number (name)$`

📖 *Select next available profile (circular, including NONE)*

⚓ `$INPUT PROFILE -$`

⚓ `!$INPUT PROFILE number (name)$`

📖 *Select previous available profile (circular, including NONE)*

⚓ `$INPUT PROFILE LIMITS$`

⚓ `!$INPUT PROFILE min value max value$`

📖 *Return minimum and maximum profiles available*

⚓ `$INPUT PROFILE NAME number$`

⚓ `!$INPUT PROFILE NAME number name$`

📖 *Return name of profile number supplied*

2.3.10.2: AUDIO

⚓ **`$INPUT AUDIO ?$`**

⚓ `!$INPUT AUDIO name$`

📖 *Return currently selected audio input*

⚓ **`$INPUT AUDIO name$`**

⚓ `!$INPUT AUDIO name$`

📖 *Select audio input*

---

where **`name`** is one of the following

`NONE`	Select no audio input
`ANALOG[1…6]`	Select analogue audio input 1-6
`ANALOGGROUP`	Select analogue audio input group: ANALOGGROUP = Analogue inputs 1, 2 and 3
`ANALOGAUX`	Select auxiliary analogue audio input
`ANALOGSCART[TV|VCR]`	Select scart analogue audio input
`ANALOGKNEKT`	Select knekt analogue audio input
`TOSAUX`	Select auxiliary TOS input
`TOS[1…3]`	Select TOS input 1-3
`SPDIF[1…3]`	Select SPDIF input 1-3

2.3.10.3: VIDEO

⚓ **`$INPUT VIDEO ?$`**

⚓ `!$INPUT VIDEO name$`

📖 *Return currently selected video input*

⚓ **`$INPUT VIDEO name$`**

⚓ `!$INPUT VIDEO name$`

📖 *Select video input*

---

where **name** is one of the following

`NONE`	Select no video input
`SVIDEO[1…4]`	Select S-Video video input 1-4
`SVIDEOSCART`	Select S-Video scart input
`SVIDEOAUX`	Select S-Video auxiliary input
`COMPONENTSCARTVCR` `[RGB|RGBHV|YPRPB]`	Select component video scart input.
`COMPOSITE[1…12]`	Select composite video input 1-12
`RGBGROUP[1…4]` `[RGB|YPRPB|YPRPBHDV]`	Select RGB video input group … 　　　　RGBGROUP1 = RGB inputs 1, 2 and 3 　　　　RGBGROUP2 = RGB inputs 4, 5 and 6 　　　　RGBGROUP3 = RGB inputs 7, 8 and 9 　　　　RGBGROUP4 = RGB inputs 10, 11 and 12
`RGBHVGROUPRGBHV`	Select RGB-HV video input group = inputs 1,2,3,4 & 5
`COMPOSITESCARTTV`	Select composite video scart TV input
`COMPOSITESCARTVCR`	Select composite video scart VCR input
`COMPOSITEAUX`	Select composite video auxiliary input

2.3.11: VIDEO

2.3.11.1: PROGRESSIVE_SCAN

> ⬇ **`$VIDEO PROGRESSIVE_SCAN [?|mode]$`**
> ⬆ `!$VIDEO PROGRESSIVE SCAN mode$`
> 📖 *Set and/or return current progressive scan setting*

where **mode** is one of the following: ON, OFF or AUTO

2.3.11.2: WATCH_DEFAULT

> ⬇ **`$VIDEO WATCH_DEFAULT [?|mode]$`**
> ⬆ `!$VIDEO WATCH DEFAULT mode$`
> 📖 *Return default watch path*

where **mode** is one of the following: COMPOSITE, RGB, RGBHV, YPRPB, S-VIDEO

2.3.12: RECORD

📥 **$RECORD ?$**

📥 !$RECORD [OFF|NONE|input TO output]$

📖 *Return current record path details.*

📥 **$RECORD OFF$**

📥 !$RECORD OFF$

📖 *Disable current record path.*

📥 **$RECORD ON$**

📥 !$RECORD [NONE|input TO output]$

📖 *Enable current (last) record path.*

📥 **$RECORD input TO output$**

📥 !$RECORD [INVALID_INPUT|INVALID_OUTPUT|input TO output]$

📖 *Set up a record connection from specified input(s) to record output.*

Where **input** is one of the following …

        [profile_input|audio_input|video_input|audio_input video_input]

profile_input	… *either name or number of a profile*
audio_input	… *name of an audio input*
video_input	… *name of a video input*
audio_input video_input	… *names of an audio and video input pair*

        refer to INPUT AUDIO *and* INPUT VIDEO *commands for a list of relevant input names.*

and **output** is one of the following …

ANALOG	… *analog output*
ANALOGSCART	… *analog scart output*
DIGITAL_44.1	… *44.1kHz digital output*
DIGITAL_48.0	… *48.0kHz digital output*
DIGITAL_88.2	… *88.2kHz digital output*
DIGITAL_96.0	… *96.0kHz digital output*
DIGITAL_ASINPUT	… *output frequency matches input frequency, allowing signal processing*

2.3.13: NORMALISE

⚓ **$NORMALISE$**
⚓ !$NORMALISE$
📖 *Normalise the audio settings*

2.3.14: PINKNOISE

⚓ **$PINKNOISE ?$**
⚓ !$PINKNOISE [AUTO\|ON\|OFF] (speaker)$
📖 *Return current pink noise status.*

⚓ **$PINKNOISE AUTO$**
⚓ !$PINKNOISE AUTO speaker$
📖 *Turn pink noise on and cycle round available speakers continuously (2 seconds per speaker).*

⚓ **$PINKNOISE speaker$**
⚓ !$PINKNOISE ON speaker$
📖 *Turn pink noise on for supplied speaker (cancels auto mode).*

⚓ **$PINKNOISE [+\|-]$**
⚓ !$PINKNOISE [AUTO\|ON] speaker$
📖 *Select next or previous (available) speaker.*

⚓ **$PINKNOISE [N\|OFF]$**
⚓ !$PINKNOISE OFF$
📖 *Turn pink noise off.*

Refer to **SPEAKER CALIBRATE** for speaker parameters.

2.3.15: SYSTEM

2.3.15.1: VOLUME

```
$SYSTEM VOLUME ?$
!$SYSTEM VOLUME level$
```
Return current maximum system volume.

```
$SYSTEM VOLUME level$
!$SYSTEM VOLUME level$
```
Set maximum system volume.

2.3.15.2: STATUS

```
$SYSTEM STATUS$
!$SYSTEM STATUS volume source mute surround$
```
Return current system status.

Where … volume   is the current volume
         source   is the current source (audio/video/profile)
         mute   is the current mute status **[MUTED|UNMUTED]**
         surround   is the current surround mode

2.3.16: 12V_TRIGGER

```
$12V_TRIGGER number [ON|OFF|time|?]$
!$12V_TRIGGER number [ON|OFF|time]$
```
Set and/or return current 12v trigger status.
If a time (ms) is specified then the trigger is turned on and, after the specified time, it is turned off.
Used during production to ensure 12v trigger lines are operating correctly.

2.3.17: FAN

```
$FAN [ON|OFF|?]$
!$FAN [ON|OFF]$
```
Set and/or return fan status.

Appendice A : Format Of Command Table

Commands are described using the following format:

⚓	**$COMMAND parameters$**	-	actual command
⚓	!$COMMAND response 1$	-	list of possible responses
⚓	… … … … … … … … … …		
⚓	!$COMMAND response n$		
📖	*Description*	-	brief description of command

Each table describes one variation of the command, therefore, for a command with five variations there will be five tables.

In cases of a command where there may be more than one form of response, all forms of the response will be listed.

The following conventions apply:

$COMMAND parameters$	-	is the command variation
!$COMMAND response$	-	is the response to a command
$response$	-	is an unsolicited response
!$FAIL number field$	-	is the response to a failed command
All uppercase words are keywords	-	all commands and system parameters must be supplied in uppercase
All lowercase words represent a parameter	-	ie. **number** means supply a numeric value
Parameter's shown as, '[p1|p2|p3]'	-	means use one of these values
Parameter's shown as, 'p1 [p2 […]]'	-	means supply one or more values

Appendice B : Escape Sequences

Previous implementations of the RS232 protocol, excluded the use of specific characters within identifiers (#, $, &, @ and spaces) and the command itself. These characters may now be included by using the escape sequence **\xHH**, where **HH** is a two digit hexadecimal code representing the actual ASCII code of the character.

This, for example, allows identifiers and command field data to contain spaces, which would otherwise be treated as field separators.

For example,   **#Record Deck#**
now becomes   **#Record\x20Deck#**

and   **!$ARTIST name of artist$**
becomes   **!$ARTIST name\x20of\x20artist$**

The following (ASCII) characters must be encoded, if they are to be included as part of an identifier or as part of a command.

- 32 (0x20)   space   - field separator
- 35 (0x23)   #   hash sign   - source identifier delimiter
- 36 (0x24) $ dollar sign   - command delimiter
- 38 (0x36)   &   ampersand   - group identifier delimiter
- 64 (0x40)   @   commercial at sign   - destination identifier delimiter
- 92 (0x5C)   \   backslash   - escape sequence

- Additionally, top-bit set (ASCII codes 128-255) characters can now also be included, using the same method.

Note:
[1] *Characters within the ranges 0 to 31, and 128 to 159 should not be used.*

[2] *The main use of escape sequences will be to output Album, Artist and Track names. It is therefore suggested that hosts do not use escape sequences unless absolutely necessary, thereby limiting any possible problems when this product is connected to a daisy chain of products which includes previous incarnations of Linn equipment (software in these products will simply treat them as raw ASCII data).*

Appendice C : Communications Settings

The Kinos uses the following communications settings:

- 7 bits data
- 1 stop bit
- even parity
- baud rate specified by host (initially 9600)