LINN PRODUCT SOFTWARE

IKEMI - RS232 ASCII Interface Specification And Commands
Version 1.00

Last Revision: 28th April 1999
Barry W Christie

Table of Contents

Introduction

This document describes how to control the IKEMI through an RS232 interface.

There are three main sections to this document:

1: Message protocol

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

2: System Commands

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

3: IKEMI Commands

This section defines a list of commands for controlling **IKEMI**.

1:Message Protocol

1.1: Overview

The RS232 interface on the IKEMI allows it to be controlled by a touch screen, PC or any computer with an RS232 port. The IKEMI obeys the commands received through the RS232 interface and replies to confirm a successful or unsuccessful operation. The IKEMI is a slave device in that it does not transmit anything unless it first receives something, e.g. a task or status command.

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 IKEMI will echo as the destination for the replies.

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.

Notes:
Nesting of fields is not permissible, nor is the use of the special delimiter characters as part of the field strings themselves.

Spaces are permissible before and after an identifier, but are not allowed within the actual identifier.
For example, '**`#   recorddeck   #`**' is valid, but '**`#   record deck   #`**' is invalid.

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.

```
........... .......... ................ $message$      see note 1
........... .......... @destination_id@ $message$      see note 2
........... &group_id& ................ $message$      see note 3
........... &group_id& @destination_id@ $message$      see note 4
#source_id# .......... ................ $message$      see note 5
#source_id# .......... @destination_id@ $message$      see note 6
#source_id# &group_id& ................ $message$      see note 7
#source_id# &group_id& @destination_id@ $message$      see note 8
```

Notes:

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 outwith the group should ignore the task.
   * There are no replies from any boxes.

4. * All products within the group should attempt the task.
   * Products outwith 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 1, with the source identity becoming the destination identity in any replies.

6. * As for 2, with the source identity becoming the destination identity in any replies.

7. * As for 3. There are no replies.

8. * As for 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: **`$? cmnd$NL`**

where	**`$`**	= command start delimiter
	`?`	= request for help
	`cmnd`	= command
	`$`	= command end delimiter
	`NL`	= Line termination characters - carriage return, line feed.

Additionally, if '**`cmnd`**' is a '**`?`**' then the command set of the product will be provided, with each command being separated from the next by a space.

Note that command help is product dependent and is implemented on the IKEMI.

1.4.1.2: Command

This is the method by which the host controls the product

Syntax: **`$cmnd (param (param ……)) $NL`** perform some operation

where	**`$`**	= command start delimiter
	`cmnd`	= command string
	`param`	= parameter string
	`$`	= command end delimiter
	`NL`	= Line termination characters - carriage return, line feed.

Note that values contained within '**`(`**' and '**`)`**' may or may not be required, it is dependent on the command.

1.4.2: 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 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) !NL
```

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

The host should expect an initial response to the command within 10 ms.

Note that the identifiers may or may not be used, see section 1.3

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) !$FAIL n$NL
```

Where ‘n’ is a code specifying why the command was invalid, see section 2.4.1.1

Note that 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) !$Status_String$NL
```

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

Note that the identifiers may or may not be used, see section 1.3

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) !$FAIL n$NL
```

Where ‘n’ is a code specifying why the task could not be completed, see section 2.4.1.1

2: System Commands

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

2.1: Identity Commands

2.1.1: ID

Configure the product on a one to one basis

`$ID identifier$`	*Write product identifier*
↳ `$ID identifier$`	

`$ID ~ identifier$`	*Remove product identifier*
↳ `$ID $`	

`$ID ?$`	*Return product identifier*
↳ `$ID identifier$`	

2.1.2: GID

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

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

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

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

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.

2.2: Communication Commands

2.2.1: BAUD

Alter/Return the baud rate setting

`$BAUD baudrate$`	*Select new baudrate from the following: 2400, 4800, 9600, 19200 or 38400*
↳ `!$BAUD baudrate$`	

`$BAUD ?$`	*Returns current baud rate (see above)*
↳ `!$BAUD baudrate$`	

Note^{1} the initial response will be at the current baud rate and the final response will be at the new baud rate.
^{2} the baud rate defaults to 9600 when the product is initialised

2.2.2: RESET

Return product to a known state

`$RESET$`	*Clear comms buffer on product*
↳ `!$RESET$`	

2.2.3: ECHO

Echo text

`$ECHO text$`	*Echos the text back enclosed in < and >*
↳ `!$ECHO <text>$`	

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

2.2.3.1: 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 IKEMI is as follows: **`!$IKEMI$`**

This feature is activated by pressing the ‘PAUSE’ key on the front panel and switching the product on.

The product will now send the message before proceeding to normal operation.

2.3: Polling Command

2.3.1: POLL

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

`$POLL START$`	*Marks the start of polling*
↳ `!$POLL START$`	

`$POLL ID$`	*Returns product id (`pid`)*
↳ `!$POLL ID pid$`	

`$POLL SLEEP$`	*Product responding to this will ignore all further commands until '`POLL DONE`' is received*
↳ `!$POLL SLEEP$`	

`$POLL DONE$`	*All products will now return to active operation*
↳	

Important

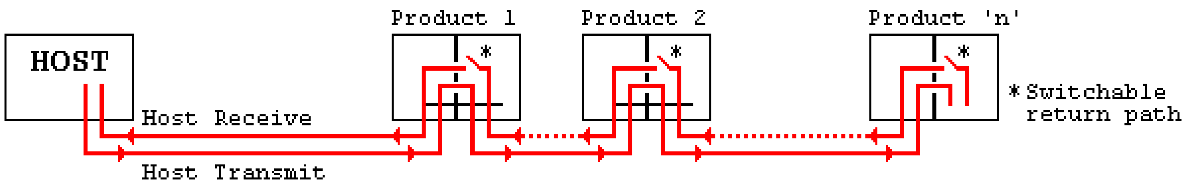
The '**`POLL SLEEP`**' command should be used with the product identifier returned by '**`POLL ID`**'.

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

2.3.2: Polling Explained

The RS232 interface hardware, via the **`POLL`** command, allows the return path on daisy-chained RS232 controlled devices to be isolated or in-circuit.

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

Hardware Note
If a product in the chain is switched off then the chain will be broken. If a product is removed then the chain must be re-established by use of a joining cable or by connecting the cable from the preceding product to the following product.

Note that on power_up all return path switches are closed.

2.4: 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.

2.4.1: STATUS

Return the last command status

$STATUS$	*Return the status of the last command*
↳ `!$STATUS number$`	

Where `number` is the returned status code. Codes are allocated on a block basis for each product with the first 48 codes reserved for general use.

2.4.1.1: Status Codes

General:

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 (0x1A) **upto** 47 (0x2F)	Reserved

3: IKEMI Commands

The following pages contain the command set for the IKEMI CD Player.

Important: Parameters must be separated from commands and each other by at least one space character

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

3.1: Command Help

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

for example: `$? TRACK$`
replies with: `!$? TRACK (?|+|-|int|+int|TOT)$`

3.2: System Commands

The system commands supported by the IKEMI are **`ID`**, **`GID`**, **`BAUD`**, **`RESET`**, **`ECHO`**, **`POLL`**, **`STATUS`** and Power_Up Message. These are all explained in section 2 of this document.

3.3: Other Commands

3.3.1: PLAY

$PLAY$	*Start playing disc or continue from current position if paused*
↳ `!$PLAY PLAYING$` ↳ `!$PLAY NODISC$`	

3.3.2: PAUSE

$PAUSE$	*Pause playing at current position*
↳ `!$PAUSE PAUSED$` ↳ `!$PAUSE NODISC$`	

3.3.3: STOP

$STOP$	*Stop playing the disc*
↳ `!$STOP STOPPED$` ↳ `!$STOP NODISC$`	

3.3.4: MODE

$MODE$	*Return current operational status*
↳ `!$MODE PLAYING$` ↳ `!$MODE PAUSED$` ↳ `!$MODE STOPPED$` ↳ `!$MODE NODISC$`	

3.3.5: TRACK

`$TRACK ?$`	*Return current track number*
↳ `!$TRACK number$`	
↳ `!$TRACK NODISC$`	

`$TRACK +$`	*Increment current track number by one*
↳ `!$TRACK number$`	
↳ `!$TRACK NODISC$`	

`$TRACK -$`	*Decrement current track number by one*
↳ `!$TRACK number$`	
↳ `!$TRACK NODISC$`	

`$TRACK number$`	*Select track number*
↳ `!$TRACK number$`	
↳ `!$TRACK NODISC$`	

`$TRACK TOT$`	*Return total number of tracks*
↳ `!$TRACK TOT number$`	
↳ `!$TRACK TOT NODISC$`	

3.3.6: INDEX

`$INDEX ?$`	*Return current index number*
↳ `!$INDEX number$`	
↳ `!$INDEX NODISC$`	

`$INDEX +$`	*Increment current index number by one*
↳ `!$INDEX number$`	
↳ `!$INDEX NODISC$`	

`$INDEX -$`	*Decrement current index number by one*
↳ `!$INDEX number$`	
↳ `!$INDEX NODISC$`	

`$INDEX number$`	*Select index number*
↳ `!$INDEX number$`	
↳ `!$INDEX NODISC$`	

3.3.7: INTRO

$INTRO ?$	*Return current status of intro play*
↳ !$INTRO PLAY$	
↳ !$INTRO STOP$	
↳ !$INTRO NO DISC$	

$INTRO [ON\|PLAY]$	*Start intro play*
↳ !$INTRO PLAY$	
↳ !$INTRO NODISC$	

$INTRO [OFF\|STOP]$	*Stop intro play*
↳ !$INTRO STOP$	
↳ !$INTRO NODISC$	

3.3.8: SEARCH

$SEARCH <$	*Search backwards through disc until STOP command is received (terminates automatically after approx 30 seconds)*
↳ !$SEARCH <$	
↳ !$SEARCH [NODISC\|BADSEARCH]$	

$SEARCH >$	*Search forwards through disc until STOP command is received (terminates automatically after approx 30 seconds)*
↳ !$SEARCH >$	
↳ !$SEARCH [NODISC\|BADSEARCH]$	

$SEARCH STOP$	*Stop searching disc*
↳ !$SEARCH STOP$	
↳ !$SEARCH [NODISC\|BADSEARCH]$	

3.3.9: DIGITAL

$DIGITAL ?$	*Return status of Digital Audio Output*
↳ !$DIGITAL ON$	
↳ !$DIGITAL OFF$	

$DIGITAL [Y\|ON]$	*Enable Digital Audio Output*
↳ !$DIGITAL ON$	

$DIGITAL [N\|OFF]$	*Disable Digital Audio Output*
↳ !$DIGITAL OFF$	

3.3.10: RANDOM

`$RANDOM ?$`	*Return status of random program*
↳ `!$RANDOM ON$`	
↳ `!$RANDOM OFF$`	
↳ `!$RANDOM NODISC$`	

`$RANDOM [Y\|ON]$`	*Turn random mode on which creates a random program and plays it*
↳ `!$RANDOM ON$`	
↳ `!$RANDOM NODISC$`	

`$RANDOM [N\|OFF]$`	*Turn random mode off which clears the current program*
↳ `!$RANDOM OFF$`	
↳ `!$RANDOM NODISC$`	

3.3.11: SHUFFLE

`$SHUFFLE ?$`	*Return status of shuffle program*
↳ `!$SHUFFLE ON$`	
↳ `!$SHUFFLE OFF$`	
↳ `!$SHUFFLE NODISC$`	

`$SHUFFLE [Y\|ON]$`	*Turn shuffle mode on which creates a random program and plays it*
↳ `!$SHUFFLE ON$`	
↳ `!$SHUFFLE NODISC$`	

`$SHUFFLE [N\|OFF]$`	*Turn shuffle mode off which clears the current program*
↳ `!$SHUFFLE OFF$`	
↳ `!$SHUFFLE NODISC$`	

3.3.12: PROGRAM

`$PROGRAM ?$`	*Return current program status*
↳ `!$PROGRAM OFF$`	
↳ `!$PROGRAM INCLUDE track [track […]]$`	
↳ `!$PROGRAM EXCLUDE track [track […]]$`	
↳ `!$PROGRAM NODISC$`	

`$PROGRAM INCLUDE track [track […]]$`	*Create new program list including tracks listed*
↳ `!$PROGRAM INCLUDE track [track […]]$`	
↳ `!$PROGRAM [NODISC\|BADTRACK\|PREVIOUSACTIVE]$`	

`$PROGRAM EXCLUDE track [track […]]$`	*Create new program list excluding tracks listed*
↳ `!$PROGRAM EXCLUDE track [track […]]$`	
↳ `!$PROGRAM [NODISC\|BADTRACK\|PREVIOUSACTIVE]$`	

`$PROGRAM CLEAR$`	*Clear current program list*
↳ `!$PROGRAM CLEAR$`	
↳ `!$PROGRAM NODISC$`	

3.3.13: REPEAT

`$REPEAT ?$`	*Return current repeat status*
↳ `!$REPEAT ON$`	
↳ `!$REPEAT OFF$`	

`$REPEAT [Y\|ON]$`	*Turn repeat on*
↳ `!$REPEAT ON$`	

`$REPEAT [N\|OFF]$`	*Turn repeat off*
↳ `!$REPEAT OFF$`	

`$REPEAT BEG$`	*Mark start of repeat section*
↳ `!$REPEAT BEG$`	
↳ `!$REPEAT BADREPEAT$`	

`$REPEAT END$`	*Mark end of repeat section and start to repeat*
↳ `!$REPEAT END$`	
↳ `!$REPEAT BADREPEAT$`	

3.3.14: IR

`$IR [Y\|ON]$`	*Enable IR control of product*
↳ `!$IR ON$`	

`$IR [N\|OFF]$`	*Disable IR control of product*
↳ `!$IR OFF$`	

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

Appendix A: Command format and description

Commands are described using the following format:

`$COMMAND parameters$`	*Brief command description*
↳ `!$COMMAND response 1$` ↳ `!$COMMAND response …$` ↳ `!$COMMAND response n$`	

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
`!$FAIL number$`	is the response to a failed command

All uppercase words are keywords (all commands must be supplied in uppercase)

All lowercase words represent a parameter, ie. `'number'` means supply a numeric value

A parameter listed as, `‘[p1|p2|p3]’`, means use one of these values

A parameter listed as, `‘p1 [p2 […]]’`, means supply one or more values

Appendix B: Product Specific Status Codes

Code	**Text**	**Description**
48	NODISC	*No disc present or a non-audio disc*
49	TRAYOPEN	*Tray is currently opened or is in the process of opening or closing*
50	BADTRACK	*Track number requested was not within the current valid range*
51	BADSEARCH	*Invalid search request*
52	BADTIME	*Time requested was not within the current valid range*
53	BADPROGRAM	*Invalid program request*
54	BADREPEAT	*Invalid repeat request*
55	PREVIOUSACTIVE	*Attempt to create a program while another program is still active*