LINN PRODUCT SOFTWARE

LINN PEKIN- RS232 ASCII Interface Specification And Commands
Version 1.0

Last Revision: 14th December 1998
Barry W Christie

REVISION HISTORY

Rev. No.	Description	Author	Date
1.0	First Issue	Barry W Christie	14/12/98
1.01	Revised for the Linn Pekin	Tracey Barclay	11/01/00

Table of Contents

INTRODUCTION 5

1:MESSAGE PROTOCOL 6

1.1: Overview 6

1.2: Message Syntax 6

1.3: Identifier Considerations 7

1.4: Syntax Of Commands And Responses 8
1.4.1: Command Syntax 8
1.4.1.1: Command Help 8
1.4.1.2: Command 8
1.4.2: Response Overview 9
1.4.2.1: Initial Response 9
1.4.2.1.1: Initial Response Failure 9
1.4.2.2: Final Response 9
1.4.2.2.1: Final Response Failure 9

2: SYSTEM COMMANDS 10

2.1: Identity Commands 10
2.1.1: ID 10
2.1.2: GID 10

2.2: Communication Commands 11
2.2.1: BAUD 11
2.2.2: RESET 11
2.2.3: ECHO 11
2.2.3.1: Power_Up Message 11

2.3: Polling Command 12
2.3.1: POLL 12
2.3.2: Polling Explained 13

2.4: Status Command 14
2.4.1: STATUS 14
2.4.1.1: Status Codes 14

3: LINN PEKIN COMMANDS 15

3.1: Command Help 15

3.2: System Commands 15

3.3 Other Commands 16
3.3.1: TUNE 16
3.3.2: SCAN 16
3.3.3: BAND 17
3.3.4: STEREO 17
3.3.5: MONO 17

3.3.6: COUNTRY 18
3.3.7: STORE 18
3.3.8: PRESET 18
3.3.9: MUTETHR 19
3.3.10: SIGNAL 19
3.3.11: MUTE 19
3.3.12: IR 20
3.3.13: INIT 20
3.3.14: OPTION 20
3.3.15: VERSION 20
3.3.16: CHECKSUM 21
3.3.17: COUNTER 21

APPENDIX A: COMMAND FORMAT AND DESCRIPTION 22

Parameter's shown as, `'p1 [p2 […]]'` means supply one or more values 22

Introduction

This document describes how to control the Linn Pekin 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 Linn Pekin to be used as part of a system driven through an RS232 interface.

3: Linn Pekin Commands

This section defines a list of commands for controlling **Linn Pekin**.

1:Message Protocol

1.1: Overview

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

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 KOLEKTOR 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

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

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

➦	**`$ID ?$`**
↳	`$ID identifier$`
🕮	*return product identifier*

2.1.2: GID

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

➦	**`$GID identifier$`**
↳	`$GID identifier$`
🕮	*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.

2.2: Communication Commands

2.2.1: BAUD

Select / Return baud rate setting

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

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

- The initial response will be at the current baud rate and the final response will be at the new baud rate.
- The baud rate defaults to 9600 when the product is initialised

2.2.2: RESET

Return product to a known state

☞	**`$RESET$`**
↳	`!$RESET$`
📖	*Clear communications buffer on product*

2.2.3: ECHO

Echo text

☞	**`$ECHO text$`**
↳	`!$ECHO <text>$`
📖	*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 that if no identifiers are supplied with this command, then all devices connected to a system will respond which may result in a communications 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 KOLEKTOR is as follows: **`!$KOLEKTOR$`**

Pressing the 'RECORD' key on the front panel and switching the product on activates this feature.

The product will now toggle the setting of the Power_Up message and display the result as follows:

Enabled:	'**`rS232 - on`**'	Disabled:	'**`rS232 - oFF`**'

2.3: Polling Command

2.3.1: POLL

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

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

➦	**`$POLL ID$`**
↳	`!$POLL ID pid$`
🕮	*Returns product identifier (`pid`)*

➦	**`$POLL SLEEP$`**
↳	`!$POLL SLEEP$`
🕮	*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

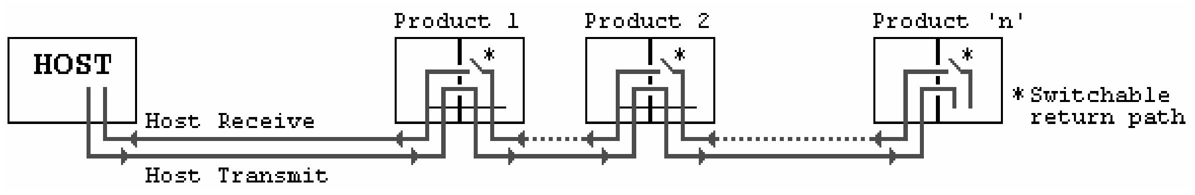
The '**`POLL SLEEP`**' command should be used with the product identifier returned by '**`POLL ID`**'.

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

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

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

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

	Code	**Description**
General:	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: Linn Pekin Commands

The following pages contain the command set for the Linn Pekin Tuner.

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 Linn Pekin and will give the host details for any given command.

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

3.2: System Commands

The system commands supported by the Linn Pekin 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: TUNE

`$TUNE p$` Select, alter or return the frequency setting

`$TUNE -$`
`!$TUNE value $`
Decrease current frequency setting by 1

`$TUNE +$`
`!$TUNE value $`
Increase current frequency setting by 1

`$TUNE n$`
`!$TUNE value $`
Select supplied frequency, MHz frequencies expressed without the decimal point e.g. 97.8 is entered as 9780

`$TUNE ?$`
`!$TUNE value $`
Return current frequency, MHz as above

3.3.2: SCAN

`$SCAN p (p)$` Select scan mode, start scanning or return current scan status

`$SCAN MODE ?$`
`!$SCAN MODE SEARCH$ or !$SCAN MODE SCAN$`
Return current scan mode

`$SCAN MODE SEARCH$`
`!$SCAN MODE SEARCH$`
Select single scan mode

`$SCAN MODE SCAN$`
`!$SCAN MODE SCAN$`
Select repeat scan mode

`$SCAN -$`
`!$SCAN -$`
Decrease frequency until next station is found. If in '`SCAN`' mode, listen to each station for 5 seconds before continuing to scan, repeat until '`$SCAN STOP$`'

`$SCAN +$`
`!$SCAN +$`
Increase frequency until next station is found. If in '`SCAN`' mode, listen to each station for 5 seconds before continuing to scan, repeat until '`$SCAN STOP$`'

`$SCAN STOP$`
`!$SCAN STOP$`
Stop current scan operation

➦	**`$SCAN ?$`**
↳	`!$SCAN STOP$ or !$SCAN PAUSED$ or !$SCAN SCANNING$`
🕮	*Return current status of scan mode*

3.3.3: BAND

`$BAND p$` Select or return frequency band

➦	**`$BAND AM$`**
↳	`!$BAND AM$`
🕮	*Select AM frequency band*

➦	**`$BAND FM$`**
↳	`!$BAND FM$`
🕮	*Select FM frequency band*

➦	**`$BAND ?$`**
↳	`!$BAND AM$ or !$BAND FM$`
🕮	*Return current frequency band*

3.3.4: STEREO

`$STEREO$` Return current stereo status

➦	**`$STEREO ?$`**
↳	`!$STEREO ON$ or !$STEREO OFF$`
🕮	*Return stereo status*

3.3.5: MONO

`$MONO p$` Set or return current mono status

➦	**`$MONO Y$ or $MONO ON$`**
↳	`!$MONO ON$`
🕮	*Enable mono state*

➦	**`$MONO N$ or $MONO OFF$`**
↳	`!$MONO OFF$`
🕮	*Disable mono state*

➦	**`$MONO ?$`**
↳	`!$MONO ON$ or !$MONO OFF$`
🕮	*Return mono status*

3.3.6: COUNTRY

`$COUNTRY$` Return country of operation

➦	**`$COUNTRY ?$`**
↳	`!$COUNTRY EUROPE$ or !$COUNTRY USA$ or !$COUNTRY JAPAN$`
🕮	*Return country of operation*

3.3.7: STORE

`$STORE p$` Store or return preset frequency

➦	**`$STORE n $`**
↳	`!$STORE value $`
🕮	*Store frequency as preset (1 to 80)*

➦	**`$STORE ?$`**
↳	`!$STORE value $`
🕮	*Return empty preset (1 to 80)*

➦	**`$STORE ALL$`**
↳	`!$STORE ALL$`
🕮	*Store all presets*

3.3.8: PRESET

`$PRESET p (p)$` Select or return preset frequency

➦	**`$PRESET CLEAR n$`**
↳	`!$PRESET CLEAR n$`
🕮	*Clear preset (1 to 80)*

➦	**`$PRESET CLEAR ALL$`**
↳	`!$PRESET CLEAR ALL$`
🕮	*Clear all presets*

➦	**`$PRESET -$`**
↳	`!$PRESET -$`
🕮	*Select previous preset*

➦	**`$PRESET +$`**
↳	`!$PRESET +$`
🕮	*Select next preset*

➦	**`$PRESET n$`**
↳	`!$PRESET n$`
🕮	*Select preset (1 to 80)*

➦	**`$PRESET ?$`**
↳	`!$PRESET ?$`
📖	*Return current preset*

3.3.9: MUTETHR

`$MUTETHR p$` Set or return mute threshold value

➦	**`$MUTETHR -$`**
↳	`!$MUTETHR n$`
📖	*Decrease mute threshold by 1*

➦	**`$MUTETHR +$`**
↳	`!$PRESET n$`
📖	*Increase mute threshold by 1*

➦	**`$MUTETHR n$`**
↳	`!$MUTETHR n$`
📖	*Set mute threshold value (0 to 31)*

➦	**`$MUTETHR ?$`**
↳	`!$MUTETHR n$`
📖	*Return current mute threshold*

3.3.10: SIGNAL

`$SIGNAL$` Return signal strength

➦	**`$SIGNAL ?$`**
↳	`!$SIGNAL value $`
📖	*Return signal strength*

3.3.11: MUTE

`MUTE p$` Set or return mute status

➦	**`$MUTE Y$ or $MUTE ON$`**
↳	`!$MUTE ON$`
📖	*Enable mute state*

➦	**`$MUTE N$ or $MUTE OFF$`**
↳	`!$MUTE OFF$`
📖	*Disable mute state*

➦	**`$MUTE ?$`**
↳	`!$MUTE ON$ or !$MUTE OFF$`
📖	*Return mute status*

3.3.12: IR

`$IR p$` Set or return the IR control status

➦	**`$IR Y$ or $IR ON$`**
↳	`!$IR ON$`
🕮	*Enable IR control of the product*

➦	**`$IR N$ or $IR OFF$`**
↳	`!$IR OFF$`
🕮	*Disable IR control of the product*

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

3.3.13: INIT

➦	**`$INIT$`**
↳	`!$INIT$`
🕮	*Resets product back to factory defaults*

3.3.14: OPTION

➦	**`$OPTION ?$`**
↳	`!$OPTION number(first) : setting … number(last): setting $`
🕮	*Return current settings of all options*

➦	**`$OPTION number ?$`**
↳	`!$OPTION number setting$`
🕮	*Return current setting of option 'number'*

➦	**`$OPTION number setting$`**
↳	`!$OPTION number setting$`
🕮	*Set option 'number' to 'setting' value*
🕮	*Note: The product will restart once a user option setting has been changed*

3.3.15: VERSION

➦	**`$VERSION SOFTWARE ?$`**
↳	`!$VERSION SOFTWARE SpppMMmm$`

🕮	*Return current software version.*
🕮	*Where: ‘ppp’ is product number, ‘MM’ is major version number and ‘ mm’ is minor version number*

➦	**`$VERSION HARDWARE ?$`**
↳	`!$VERSION HARDWARE PCAShhhLr$`
🕮	*Return current hardware version (audio board)*
🕮	*Where: ‘hhh’ is board number and ‘r’ is revision number*

3.3.16: CHECKSUM

➦	**`$CHECKSUM ?$`**
↳	`!$CHECKSUM hhhh$`
🕮	*Return current software checksum*
🕮	*Where:’ hhhh’ is a four digit hexidecimal value*

3.3.17: COUNTER

➦	**`$COUNTER POWER ?$`**
↳	`!$COUNTER POWER days:hours:minutes:seconds$`
🕮	*Return total powered time*
🕮	*Where: ‘days’ is 1 to 5 digits, ‘hours’ is 2 digits, ‘minutes’ is 2 digits and ‘seconds’ is 2 digits*

Appendix A: Command format and description

Commands are described using the following format:

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

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
• 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