# LINN PRODUCT SOFTWARE

**LR2 - RS232 Interface Specification**
**Version 1.0**

## Revision History

| *Revision* | *Description* | *Author* | *Date* |
|---|---|---|---|
| 1.00 | First Issue | Jason Newell, Barry Christie | 19/09/02 |
| 1.01 | Removed IR Translator references after review by Allan Stoneham. | Jason Newell | 7/10/02 |
| 1.02 | Renamed Line Receiver 2 as LR2 | Jason Newell | 23/04/03 |
| 1.03 | Updates following Design Changes | Andrew Gebbie | 14/05/03 |
| 1.04 | Updates for new commands | Andrew Gebbie | 10/07/03 |
|  |  |  |  |

## Table of Contents

# Introduction

This document describes how to control the LR2 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.

**2: System Commands**

- This section lists the commands, which allow the LR2 to be used as part of a system driven through an RS232 interface.

**3: LR2** Commands

- This section defines a list of commands for controlling the LR2.

# 1: Message Protocol

## 1.1: Overview

The RS232 interface on the LR2 allows it to be controlled by a touch screen, PC or any computer with an RS232 port. The LR2 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 LR2 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.

***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 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 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.<br>- A successful or unsuccessful final response will be issued subsequently.<br>- Products not recognising the command will remain silent.<br>- If no product recognises the command then there will be no reply.<br>- 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.<br>- Invalid commands will generate an error response.<br>- The replying product will transfer the destination to the source field on a reply.<br>- All products not matching the destination must remain silent and not attempt to handle the command.<br>- If two products have the same id, then a comms clash may occur. |
| 3 | - All products within the group should attempt the task.<br>- Products out with the group should ignore the task.<br>- There are no replies from any boxes. |
| 4 | - All products within the group should attempt the task.<br>- Products out with the group should ignore the task.<br>- Only the product, which matches the destination identity, should reply.<br>- Invalid commands will generate an error response.<br>- If there are more than two products in the group with the same destination identity then a comms clash may occur.<br>- 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 2.4.1.1: 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 2.4.1.1: 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:***
The LR2 will not generate unsolicited responses

# 2: System Commands

The following commands allow the LR2 to be part of a system driven through an RS232 interface.

## *2.1: Identity Commands*

### 2.1.1: ID

The Product ID is set by an internal ID chip. It therefore cannot be created , removed or modified. The only operation is to query its value. Note on power up if an RS232 terminal is active the product will send its ID. See Power up Message.

| |
|---|
| ⬇ **`$ID ?$`** |
| ⬆ `$ID identifier$` |
| 📖 *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

|  |
|---|
| ⬇ **`$BAUD baudrate$`** |
| ⬆ `!$BAUD baudrate$` |
| 🕮 *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.*

### 2.2.2: 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.*

#### *2.2.2.1: 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 LR2 is as follows: **`!$LR2nnnnnnnnnnnn$`**
Where nnnnnnnnnnnn is a 12 digit hexadecimal number generated by an internal ID chip. The number is unique to each product and is also printed on a label on the base of the unit. The number is used by the Linstall2 configuration program which needs the number in order to communicate with the product.

## *2.3: Polling Command*

### 2.3.1: POLL (Not implemented in LR2 due to lack of Hardware Support)

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**

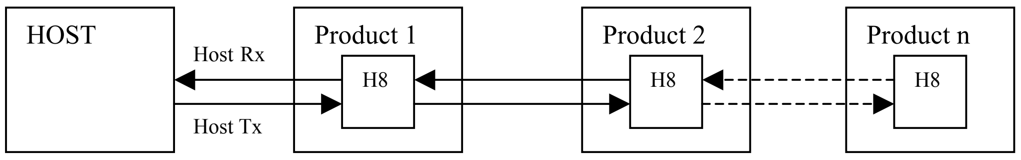
The '**`POLL SLEEP`**' command should be used with the product identifier returned by '**`POLL ID`**'.

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

## 2.3.2: 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

## 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

| |
|---|
| **$STATUS$** |
| `!$STATUS sc (sv)$` |
| *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.

### *2.4.1.1: 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 |

### 2.5: IR (Not Implemented)

```
$IR ?$
```
```
!$IR ON$
!$IR OFF$
```
*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*

### 2.6: INIT (not implemented)

```
$INIT$
```
```
!$INIT$
```
*Resets product back to factory defaults*

### 2.7: 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*

### 2.8: CHECKSUM

```
$CHECKSUM ?$
```
```
!$CHECKSUM hhhh$
```
*Return current software checksum*

*Where: 'hhhh' is a four digit hexidecimal value*

## 2.9: COUNTER

| |
|---|
| **`$COUNTER POWER$`** |
| `!$COUNTER POWER days:hours:minutes:seconds$` |
| *Returns total powered up (operational) time.* |

| |
|---|
| **`$COUNTER POWER 31051972$`** |
| `!$COUNTER POWER days:hours:minutes:seconds$` |
| *31051972 is a Unidisc specific password, which has no significance other than being available only to Linn staff.* |
| *Resets counter to zero and returns total powered up (operational) time.* |

| |
|---|
| **`$COUNTER MAINS$`** |
| `!$COUNTER MAINS days:hours:minutes:seconds$` |
| *Returns total mains connected time* |

| |
|---|
| **`$COUNTER MAINS 31051972$`** |
| `!$COUNTER MAINS days:hours:minutes:seconds$` |
| *31051972 is a Unidisc specific password, which has no significance other than being available only to Linn staff.* |
| *Resets counter to zero and returns total mains connected time.* |

# 3: LR2 Commands

The following pages contain the command set for the LR2 device.

**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

## *3.1: Command Help*

Command help is implemented by the LR2 and will give the host details for any given command.

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

Refer to section 1.4.1.1: Command Help for further information.

## *3.2: System Commands*

The system commands supported by the LR2 are **`ID`**, **`GID`**, **`BAUD`**, **`RESET`**, **`ECHO`**, **`POLL`**, **`STATUS,INIT,IR,CHECKSUM,VERSION,COUNTER`** and Power_Up Message. These are all explained in section 2: System Commands of this document.

## *3.3: Other Commands*

### 3.3.1: VOLUME

| |
|---|
| **`$VOL ?$`** |
| `!$VOL value$` |
| *Return current volume setting* |

| |
|---|
| **`$VOL +$`** |
| `!$VOL value$` |
| *Increase current volume setting by 1* |

| |
|---|
| **`$VOL -$`** |
| `!$VOL value$` |
| *Decrease current volume setting by 1* |

| |
|---|
| **`$VOL +number$`** |
| `!$VOL value$` |
| *Increase current volume setting by supplied value* |

| |
|---|
| **`$VOL -number$`** |
| `!$VOL value$` |
| *Decrease current volume setting by supplied value* |

| |
|---|
| **`$VOL = number$`** |
| `!$VOL value$` |
| *Set current volume setting to absolute value supplied* |

### 3.3.2: BALANCE

| |
|---|
| **`$BAL ?$`** |
| `!$BAL value$` |
| *Return current balance setting* |

| |
|---|
| **`$BAL +$`** |
| `!$BAL value$` |
| *Increase current balance setting by 1* |

| |
|---|
| **`$BAL -$`** |
| `!$BAL value$` |
| *Decrease current balance setting by 1* |

| |
|---|
| **`$BAL +number$`** |
| `!$BAL value$` |
| *Increase current balance setting by supplied value* |

| |
|---|
| **`$BAL -number$`** |
| `!$BAL value$` |
| *Decrease current balance setting by supplied value* |

| |
|---|
| **`$BAL = number$`** |
| `!$BAL value$` |
| *Set current balance setting to value supplied* |

### 3.3.3: BASS

| |
|---|
| **`$BAS ?$`** |
| `!$BAS value$` |
| *Return current bass setting* |

| |
|---|
| **`$BAS +$`** |
| `!$BAS value$` |
| *Increase current bass setting by 1* |

| |
|---|
| **`$BAS -$`** |
| `!$BAS value$` |
| *Decrease current bass setting by 1* |

| |
|---|
| **`$BAS +number$`** |
| `!$BAS value$` |
| *Increase current bass setting by supplied value* |

| |
|---|
| **`$BAS -number$`** |
| `!$BAS value$` |
| *Decrease current bass setting by supplied value* |

| |
|---|
| **`$BAS = number$`** |
| `!$BAS value$` |
| *Set current bass setting to value supplied* |

### 3.3.4: TREBLE

| |
|---|
| **`$TRB ?$`** |
| `!$TRB value$` |
| *Return current treble setting* |

| |
|---|
| **`$TRB +$`** |
| `!$TRB value$` |
| *Increase current treble setting by 1* |

| |
|---|
| **`$TRB -$`** |
| `!$TRB value$` |
| *Decrease current treble setting by 1* |

| |
|---|
| **`$TRB +number$`** |
| `!$TRB value$` |
| *Increase current treble setting by supplied value* |

```
$TRB -number$
!$TRB value$
```
*Decrease current treble setting by supplied value*

```
$TRB = number$
!$TRB value$
```
*Set current treble setting to value supplied*

## 3.3.5: MUTE

```
$MUTE ?$
!$MUTE ON$
!$MUTE OFF$
```
*Return current mute status*

```
$MUTE [Y|ON]$
!$MUTE ON$
```
*Enable mute*

```
$MUTE [N|OFF]$
!$MUTE OFF$
```
*Disable mute*

## 3.3.6: LISTEN (indirect commands)

```
$LISTEN ?$
!$LISTEN value$
```
*Return current source number ('value' is between 0  and 3)*

```
$LISTEN +$
!$LISTEN value$
```
*Select the next logical source*

```
$LISTEN -$
!$LISTEN value$
```
*Select the previous logical source*

```
$LISTEN number$
!$LISTEN value$
```
*Select absolute source 'number'*

**Note**: If a source is unavailable for any reason, then the number returned specifies the currently selected source.

## 3.3.7: LISTEN (direct commands)

```
$LISTEN SOURCE X$ - Where X is a source number (0-3)
!$LISTEN value$
```
*Select CD player source*

## 3.3.8: DEFAULTS

*Note DEFAULTS can be invoked individually. However its main use is within the Linstall2 upload protocol. Unpredictable results may occur if the command is used outside the context of Linstall2*

| |
|---|
| **`$DEFAULTS$`** |
| `!$DEFAULTS$` |
| *Reset the product settings to factory defaults.* |

## 3.3.9: XMLSTART

*Note XMLSTART can be invoked individually. However its main use is within the Linstall2 upload protocol. Unpredictable results may occur if the command is used outside the context of Linstall2*

| |
|---|
| **`$XMLSTART$`** |
| `!$XMLSTART$` |
| *Indicates that a number of $XML $ commands and an $XMLEND $ command will follow.* |

## 3.3.10: XML

*Note XML can be invoked individually. However its main use is within the Linstall2 upload protocol. Unpredictable results may occur if the command is used outside the context of Linstall2*

| |
|---|
| **`$XML 'xmldata' checksum$`** |
| `!$XML OK$` |
| `!$XML CORRUPT$ (= bad checksum)` |
| `!$XML INVALID$` |
| *Within the first set of single quotes is a line of XML containing product setting information.The LineReceiver 2 will extract tokens and parameters from the XML. If the grammar and parametes are valid, the LineReceiver 2 will store the parameters in flash. If they are not, it will return INVALID.* |
| *Preceding the closing $ is the arithmetic sum of all the characters in the line of XML, expressed as an ASCII decimal.* |

## 3.3.11: XMLEND

*Note XMLEND can be invoked individually. However its main use is within the Linstall2 upload protocol. Unpredictable results may occur if the command is used outside the context of Linstall2*

| |
|---|
| **`$XMLEND$`** |
| `!$XMLEND$` |
| *Indicates upload of product settings is complete.* |

## 3.3.12: RS232 TO RC5 PARSER

| |
|---|
| **`$PARSER X$`** |
| `!$PARSER X$ - where x is either ON or OFF` |
| *This function enables RS232 to RC5 conversion and back again on the selected channel.* |

| |
|---|
| **`$PARSER CHANNEL X$`** |
| `!$PARSER CHANNEL X$ - Where X is the chosen channel to send RC5 on` |
| *Send converted RS232 commands on channel ROOM, SUBZONE, REMOTE, MAIN* |

### 3.3.13: DATA

**`$DATA 'data' checksum$`**

```
!$DATA OK$
!$DATA CORRUPT$ (= bad checksum)
!$DATA INVALID$
```

*The format of the data within the ‘ ‘ is checked and converted into RC5 data. Preceding the closing $ is the arithmetic sum of all the characters in the line of DATA, expressed as an ASCII decimal. For an example of the command see Appendix F. The data is only sent as RC5 commands if RS232 to RC5 Parsing is enabled see PARSER above. This feature also works in reverse. When enabled all RC5 commands received will be converted to the DATA format and transmitted on the RS232. Note this is NOT in response to any RS232 command but is driven from the incoming RC5 message.*

### 3.3.14: RS232 ROUTING

*Note RS232PATH can be invoked individually. However its main use is within the Linstall2 upload protocol. Unpredictable results may occur if the command is used outside the context of Linstall2*

**`$RS232PATH$`**

```
!$RS232PATH$
```

*Sets the comms path for RS232 messages which are to be sent direct to the RCU through the LR2 Hardware. Used by Linstall to initiate LR2 RS232 switching from internal RS232 to external RS232 RCU comms. Used in conjunction with ENDUPLOAD – see $ENDUPLOAD$*

### 3.3.15: CONNECT ON/OFF

*Note CONNECTSTATE can be invoked individually. However its main use is within the Linstall2 upload Unpredictable results may occur if the command is used outside the context of Linstall2protocol.*

**`$CONNECTSTATE ?$`**

```
!$CONNECTSTATE X$ - where x is ON or OFF
```

*Displays the RS232 / Connect command interface useage.*

**`$CONNECTSTATE X$`**

```
!$CONNECTSTATE X$ - where x is ON or OFF
```

*Switches connect on or off to allow serial port 0 for RS232 command interface useage.*

### 3.3.16: TRIKEMODE ON/OFF

**`$TRIKEMODE ?$`**

```
!$TRIKEMODE X$ - where x is ON or OFF
```

*Displays the Current Trikemode.*

**`$TRIKEMODE X$`**

```
!$TRIKEMODE X$ - where x is ON or OFF
```

*Switches on and off the trike mode. Trike mode allows the main room to control the local room products*

### 3.3.17: ENDUPLOAD

*Note ENDUPLOAD can be invoked individually. However its main use is within the Linstall2 upload protocol. Unpredictable results may occur if the command is used outside the context of Linstall2*

**$ENDUPLOAD$**

```
!$ENDUPLOAD$
```

*Resets the comms path for RS232 messages which are to being sent direct to the RCU through the LR2 Hardware. Used by Linstall to inform the LR2 that uploads have finished and RS232 switching be changed from external RS232 RCU comms back to internal RS232. Used in conjunction with RS232PATH – see $RS232PATH$*

## 3.3.18: ORIGIN

**$ORIGIN ?$**

```
!$ORIGIN X$ - where x is ZONETYPE_ROOM or ZONETYPE_SUBZONE
```

*Displays the Current Origin*

**$ORIGIN X$**

```
!$ORIGIN X$ - where x is ZONETYPE_ROOM or ZONETYPE_SUBZONE
```

*This command is used to set the origin of the Volume, Bass, Treble etc instructions. In normal circumstances the location of the RCU/Skint is known by the hardware and set appropriately. However for RS232 commands which simulate RCU button presses there is no source component to inform the LR2. This command sets the Origin of the command to either ROOM(default) or SUBZONE. It is recommended that an ORIGIN ? command is sent first to determine the current setting.*

## 3.3.19: STANDBY

**$STANDBY ?$**

```
!$STANDBY x$ - where x is ON or OFF
```

*Displays the Standby state*

**$STANDBY X$**

```
!$STANDBY X$ - where x is ON or OFF
```

*This command is used to bring the unit out of Standby or put it back into Standby when finished. Note : if the unit is used with an external controller and there are NO RCUs connected to the product then STANDBY OFF must be sent before issuing any RS232 commands that change the source, volume, bass, treble or balance.*

## A. Format Of Command Table

Commands are described using the following format:

| | | |
|---|---|---|
| **`$COMMAND parameters$`** | - | actual command |
| `!$COMMAND response 1$`<br>… … … … … … … … … … …<br>`!$COMMAND response n$` | - | list of possible responses |
| 🕮 *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 |
| `!$FAIL sc fn$` | - | 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 |

## 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:***
*Characters within the ranges 0 to 31, and 128 to 159 should not be used.*

## C. Communications Settings

The LR2 uses the following communications settings:

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

## D. Product Specific Status Codes

| ***Code*** | ***Text*** | ***Description*** |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## E. Sequence Data Schema

The following is a sample of the XML sent by Linstall2.

```
<LR2>
  <MuteTrigger id="1" list="ZoneLineOutput, ZonePowerAmp"/>
  <MuteTrigger id="2" list="SubZoneLineOutput, SubZonePowerAmp"/>
  <MuteTrigger id="3" list="ZoneLineOutput, ZonePowerAmp"/>
  <ZoneRCU type="RCU"/>
  <SubZoneRCU type="None"/>
  <MainController type="Intersekt"/>
  <Trike enable="No"/>
</LR2>
```

## F. Data Command Format

The following is the format for the Data command: All the characters between the ‘ are subject to checksum which is sent in Decimal format and is the last element before the final $ symbol. All data starting with a ‘x’ character is a hex number and is converted to the binary equivalent for storing in the RC5 message.
Example

**Simple Command**
```
$DATA ‘<SYS>x14<COM>x3F’ 4321$
```

**Command with Extended Data**
```
$DATA ‘<SYS>x14<COM>x3F<EXD>x02x60x40x00x00’ 1234$
```