



XLuminate - Software User Manual

Document No. 80-20387 Issue 5

Current Issue :-	Issue 5, 4 September 2009
Previous Issue :-	Issue 4, 29 January 2009
	Issue 3, 22 December 2008
	Issue 2, 12 November 2008
	Issue 1, 27 July 2008

If your query is not covered in this User Manual, or you require further information, please email Heber Customer Support: support@heber.co.uk

The latest version of this User Manual and other technical information can be found on the Heber website: www.heber.co.uk

Copyright © Heber Ltd. 2009. All rights reserved. This document and the information contained therein is the intellectual property of Heber Ltd. and must not be disclosed to a third party without consent. Copies may be made only if they are in full and unmodified.

The information contained in this User Manual is believed to be accurate and reliable. However, Heber Ltd. assumes no responsibility for its use, and reserves the right to revise the documentation without notice.

Precise specifications may change without prior notice.

All trademarks are acknowledged.

HEBER LIMITED
Belvedere Mill
Chalford Stroud
GL6 8NT
UK

Tel +44 (0) 1453 886000
Fax +44 (0) 1453 885013
Email support@heber.co.uk
Website www.heber.co.uk

CONTENTS

1	INTRODUCTION.....	3
1.1	PRODUCT OVERVIEW:	3
1.1.1	<i>The hardware view:</i>	3
1.1.2	<i>Types of RGB Led packages:</i>	4
1.1.3	<i>The programmer's or user's view:</i>	5
1.1.4	<i>The LEDMAP:</i>	5
1.1.5	<i>Patterns:</i>	6
1.1.6	<i>Sequences:</i>	6
1.1.7	<i>Expandability:</i>	7
1.1.8	<i>The XLine Pattern Generator Application:</i>	7
2	API FUNCTIONS	8
	FUNCTION RETURN VALUES:	8
2.1	FUNCTION NAME: XLUMINATE ()	8
2.2	FUNCTION NAME: XLUMINATE (XLUMINATE &)	8
2.3	FUNCTION NAME: OPERATOR = (XLUMINATE &)	8
2.4	FUNCTION NAME: ~XLUMINATE ()	8
2.5	FUNCTION NAME: INITUSBBOARD()	8
2.6	FUNCTION NAME: INIT()	9
2.7	FUNCTION NAME: CLOSE()	9
2.8	FUNCTION NAME: GETBOARDSPEED()	9
2.9	FUNCTION NAME: GETPRODUCTVERSION()	9
2.10	FUNCTION NAME: GETAPIVERSION()	10
2.11	FUNCTION NAME: GETFIRMWAREVERSION()	10
2.12	FUNCTION NAME: GETLASTERROR()	10
2.13	FUNCTION NAME: CLEARERRORS()	10
2.14	FUNCTION NAME: GETERRORLINENUMBER ()	10
2.15	FUNCTION NAME: GETDIPSWITCHES()	11
2.16	FUNCTION NAME: SETDISPLAYCONFIG()	11
2.17	FUNCTION NAME: ENABLEDISPLAY()	13
2.18	FUNCTION NAME: FADEDISPLAY()	13
2.19	FUNCTION NAME: CLEARDISPLAY()	13
2.20	FUNCTION NAME: DISABLEDISPLAY()	13
2.21	FUNCTION NAME: GETDISPLAYSTATUS()	14
2.22	FUNCTION NAME: SELECTPATTERN()	14
2.23	FUNCTION NAME: WRITEPATTERN()	15
2.24	FUNCTION NAME: CLEARPATTERN()	16
2.25	FUNCTION NAME: LOADPATTERNFILE()	16
2.26	FUNCTION NAME: WRITSEQUENCEDATA()	16
2.27	FUNCTION NAME: STARTSEQUENCE()	17
2.28	FUNCTION NAME: SETNEXTSEQUENCE()	17
2.29	FUNCTION NAME: STOPSEQUENCE()	18
2.30	FUNCTION NAME: WRITEDIGIT()	18
2.31	FUNCTION NAME: WRITEDIGITSTRING()	19
2.32	FUNCTION NAME: TESTDISPLAYS()	19
3	API RETURN CODES	20
4	PATTERN FILES.....	22
4.1	OVERVIEW:	22
4.2	PATTERN FILE SYNTAX:	22
4.3	DISPLAY CONFIGURATION SECTION:	22
4.3.1	<i>Number of displays:</i>	22
4.3.2	<i>Number of rows:</i>	23
4.3.3	<i>Number of columns:</i>	23

4.3.4	<i>RGBMode:</i>	23
4.3.5	<i>Transpose:</i>	23
4.3.6	<i>Refresh:</i>	24
4.4	LEDMAP DEFINITION SECTION	24
4.5	PATTERN DEFINITION SECTION	25
4.6	SEQUENCE DEFINITION SECTION	26
4.7	TROUBLESHOOTING PATTERN FILES:	26
4.8	SAMPLE APPLICATION - ANALOG CLOCK:	27
4.8.1	<i>Overview:</i>	27
4.8.2	<i>Programmers view:</i>	28
4.8.3	<i>Sample Pattern File:</i>	28

1 INTRODUCTION

This document contains technical details about how to connect an XLuminate Led/Lamp control board and operate the board through the supplied API software library.

The XLuminate board is a member of the XLine family of USB based gaming peripherals; this particular variant of the family has been specifically designed to control leds/lamps, allowing brightness control of individual lamps or groups of lamps to form display patterns, collections of patterns may be grouped into sequences, and display sequences repeated at regular time intervals if required.

1.1 Product Overview:

The XLuminate board is designed to provide flexible control of multiple light sources. The individual light sources can be conventional filament lamps, monochrome leds, bi-coloured leds or RGB (red/green/blue) leds. The brightness level (and/or colour) of each light source can be individually controlled.

This section of the manual sets out the display terminology used throughout the manual, presenting the underlying electronic hardware used to drive the lamps and the ways in which the lamps/leds can be connected (the hardware perspective); the view of the product as seen from the programmer's or user's perspective and how the LEDMAP structure links the view as seen by the programmer to the underlying hardware. The section ends with an outline description of the Xline Pattern Generator program, a software application to greatly simplify and automate the production of XLuminate applications.

1.1.1 The hardware view:

The lamp/led driving signals generated by the XLuminate board consist of 16 row driving signals and 16 column driving signals, forming an array of 256 matrix intersection points. A lamp or led connected at a matrix intersection point will be illuminated when both row and column driving signals to which it is connected are switched on. The brightness of the lamp/led can be controlled by controlling the length of time the signals are switched on for, and by systematically scanning all matrix intersection points over a short period of time, the full two-dimensional display can be built up. See simplified diagram of matrix display in Figure 1.1.

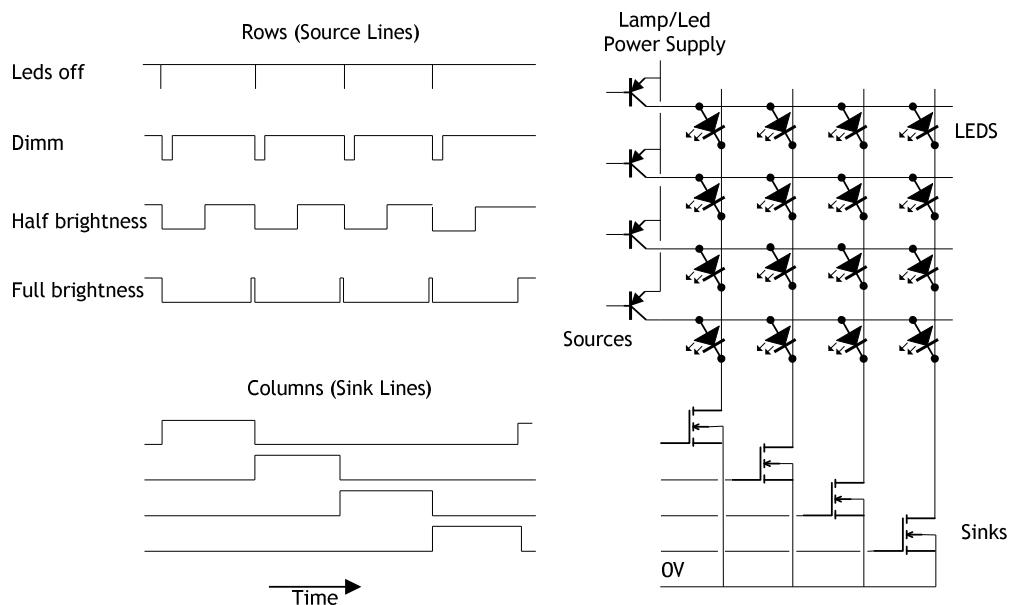


Figure 1.1

For the XLuminate board, groups of 16 leds are connected to each column driving line or “sink” line, and there are a total of 16 sink lines. Each of the 16 sink lines is activated in turn, and each line remains active for approx 1mS. Each of the 16 leds connected to a single sink line is driven by a row driving line or ‘source’ line; these signals remain active for varying amounts of time dependent on the brightness required for that lamp/led (pwm) up to the limit of the approx 1mS for which the sink line is active. By arranging the delivery of the appropriate 16 row brightness values on a column by column basis the full display image is constructed in approx 16mS, which is fast enough to avoid display “flicker”. This 16 row by 16 column (ie. 256-intersection) matrix arrangement is described as a 2-dimensional display “plane”.

If using lamps or monochrome leds, a single row/column intersection is all that is required to drive that type of light source; if on the other hand the light source is an RGB led, it will require 3 row/column intersections (one each for red, green and blue) to control the light source. The term “logical-cell” is used to describe the position of the light source in a way that is independent of what type of resource it is (monochrome, two-colour or RGB led). A table called the LEDMAP then connects logical-cells to physical row and column driving signals.

An XLuminate board on its own can control up to 256 monochrome cells, in effect the brightness control of 256 individual light diode elements or lamps. Alternatively the XLuminate could control up to 85 RGB cells (3 matrix intersection points per RGB cell). This XLuminate configuration can be expanded by the addition of up to two Multiplex Expansion boards, each of which adds a further display “plane”, ie. The driving signals needed for a further 256 monochrome cells. Fully configured therefore, an XLuminate plus two Multiplex Expansion boards can drive up to 768 monochrome leds/lamps across 3 display planes, or up to 256 RGB cells.

1.1.2 Types of RGB Led packages:

Monochrome leds require just 2 wires, one wire each for the anode and cathode of the led. RGB leds on the other hand come in a range of different packages, some of which place restrictions on how they can be connected to the XLuminate’s matrix driving signals.

The simplest type of RGB led packaging is the 6-wire device; 2 wires for anode and cathode of each of red, green and blue RGB elements. This type of packaging can be connected in any way the designer wishes to the XLuminate row/column driving signals.

A second type of RGB led packaging is the “common-anode” device; here the anode connections of red, green and blue elements are linked internally in the package leading out to a single external package pin. Three further external wires connect to the individual cathodes of red, green and blue elements. This type of package must connect the anode connection to a matrix row (source) line and each of red, green and blue cathode lines to different column (sink) lines - ie. the RGB led is connected horizontally “across the columns”.

The final type of RGB package is the “common cathode” device; here the 3 cathodes are joined internally while separate wires go to the red, green and blue anode connections. This type of package must connect the common cathode connection to a matrix column (sink) line and each of red, green and blue anode lines to different row (source) lines - ie. the RGB led is connected vertically “down the rows”.

The scenarios described above refer to RGB led connections within the same display plane. It is also possible to connect RGB leds “across display planes”, for example where plane-0 supplies the red brightness, plane-1 the green and plane-2 the blue brightness. Cross-plane connection with 6-wire RGB leds is straightforward. For common cathode RGB leds, the same column (sink) line must be specified for the RGB led across all 3 planes in the LEDMAP and the common cathode wire will connect to just one of the 3 sink transistor (from plane 0, or 1 or 2) and not to all 3 sinks.

Common anode RGB leds can not normally be connected across display planes. Consult Heber Technical Support staff for further information on cross-plane common-anode possibilities and limitations.

Figure 1.2 illustrates the different connection scenarios for common-anode, common-cathode and 6-wire RGB packages.

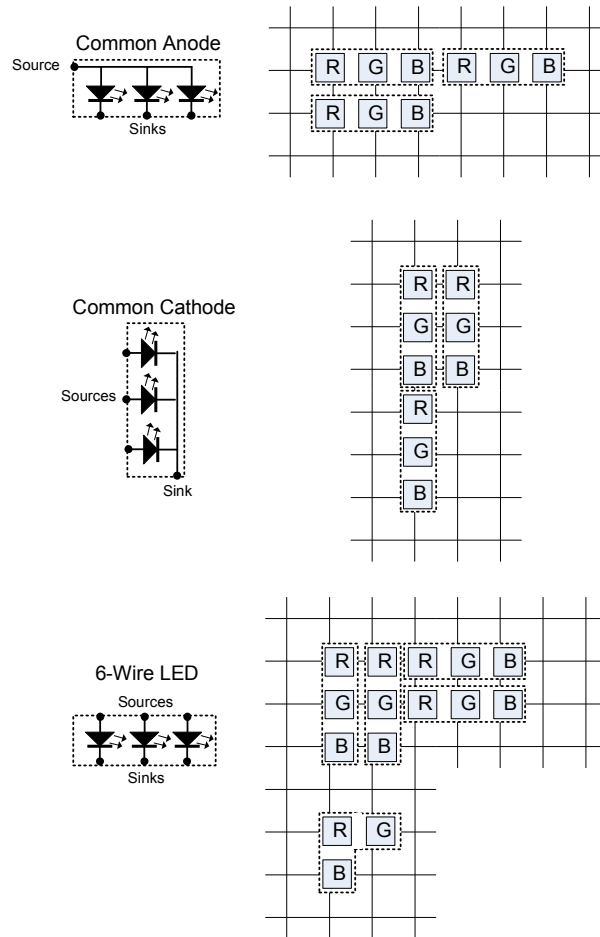


Figure 1.2

1.1.3 The programmer's or user's view:

While the hardware perspective is concerned with the wiring of matrix row and column driving signals to individual lamp/led devices, the programmer/user will wish to have a more abstract view of the application, that reflects the visible layout of the light resources of each particular application. The programmer/user achieves this by labelling the visible layout of the application in some form of sequential order that is easiest for the programmer/user to work with. The numbering/labelling of these logical display points or 'logical cells' may be a simple contiguous numbering scheme from zero to the maximum logical cell defined, or it may be some form of grid arrangement of cells (but totally separate from matrix driving row/column lines). A display 'pattern' consists of stored brightness values for each logical cell in the display, hence a stored display pattern is a series of brightness bytes stored in the same order as logical cells were defined.

The Simple Application Example of implementing an analogue clock face with RGB leds is described in the chapter on Pattern Files.

The final stage is to link the programmer/user view (of logical-cells) to real hardware driving signals, the underlying matrix row and column driving signals. This is the role of the LEDMAP.

1.1.4 The LEDMAP:

The LEDMAP is a table to link each logical-cell in the programmer's or user's view of an XLuminate application to the underlying physical lamp or led resources producing the light output. The table offers considerable flexibility in the choice and mix of leds used in the application (monochrome and/or RGB), type of RGB led used (6-wire, common-anode, common-cathode or mixture) and how the led wiring loom for the specific application is designed.

The first requirement is that the XLuminate board know whether the display resources are exclusively monochrome lamps/leds (by selecting monochrome-mode) or is employing one or more RGB leds (by selecting RGB-mode). This in turn determines which of the two types of LEDMAP entry (see below) will be required.

The XLuminate display resources are sub-divided into display-planes, each of which can drive up to 256 lamps or monochrome leds, or that number divided by 3 if using RGB leds throughout. Therefore the LEDMAP table will need to define: which display-plane, matrix row and matrix column each lamp/led is connected to.

A filament lamp or monochrome led LEDMAP entry is as follows:

Logical Cell	Number of Elements	Display	Row	Column
15	1	0	5	6

This entry specifies that logical cell '15' is a monochrome resource and is connected between matrix row-5 and column-6 of display plane-0.

Where the logical cell is an RGB light source, then it will be necessary to define the three light resources making up the RGB logical cell, ie. three entries for: plane, row and column, as in the following example.

Logical Cell	Number of elements	Display	Row	Column	Display	Row	Column	Display	Row	Column
12	3	0	2	3	0	2	4	0	2	5

This entry defines logical-cell '12' as an RGB resource and it is clear the device is connected 'across the columns' (because the column number is incrementing) so may be either a '6-wire' or 'common-anode' type RGB device.

The maximum logical cell number that can be defined depends of the total number of matrix row/column intersection points available (for XLuminate: 16 rows times 16 columns times 3 display planes = 768) and whether monochrome leds (one matrix intersection point per led), bi-chrome leds (two intersection points per led) or RGB leds (three intersection points per led) are being used. The 'number of elements' parameter in the above table defines the number of intersection points and hence how many sets of 'display-row-column' parameters need to be supplied.

Once the LEDMAP structure has been defined programmatically, the structure, together with other essential display parameters (number of rows, number of columns, etc.) are sent to the XLuminate board with API function SetDisplayConfig(). Alternatively, the table and display parameters can be defined much more easily using the XLine Pattern Generator Application, and the resulting pattern file sent to the XLuminate board with API function LoadPatternFile().

1.1.5 Patterns:

The set of data required to define the brightness values of up to 256 cells, whether monochrome or RGB, is referred to as a *pattern* and is stored as up to a 256-byte data block for monochrome or up to 768 bytes for RGB. Patterns may be pre-defined and loaded along with sequences from a pattern file, or patterns may be dynamically generated and activated at any time from the application program using supplied API functions. The XLuminate board has on-board storage of 94464 bytes set aside for pattern storage. Thus 255 (the maximum number of patterns) separate 256-byte monochrome patterns could be stored concurrently or if using the full complement of 256 RGB leds in RGB mode (768 bytes per pattern) 123 complete patterns may be stored concurrently.

1.1.6 Sequences:

Recognising that there may be a desire to repeat the display of particular light patterns in a particular order at regular intervals leads to the concept of *sequences*. A sequence is a data structure that defines which of the currently loaded patterns should be displayed, in what order, and for what interval of time. The

sequence data structure allows up to 127 pattern-number/display-time pairs of values to be combined together into a single sequence definition. The XLuminate board can store up to 16 such sequences. In monochrome-mode, each display plane can independently run a pattern, so up to 3 sequences could be run concurrently across the 3 display planes. In RGB-mode a total of one sequence can be run across all three planes at any one time.

The execution of pattern sequences is handled entirely by XLuminate hardware, so once started, running sequences presents no software overhead to the application program or system, so there are no variations in pattern timing caused by system activity. As in the case of defining patterns, sequences can be pre-defined and loaded from the pattern file, or loaded dynamically at any time.

To facilitate the pre-definition of display configuration, patterns and sequences via 'pattern files', Heber supply the Windows-based XLuminate Pattern/Sequence Generator Application.

1.1.7 Expandability:

With suitable additional USB hubs and ports, a total of eight XLuminate boards, with or without additional Multiplex Expansion boards, may be connected to a single PC system.

1.1.8 The XLine Pattern Generator Application:

The XlinePatternGenerator application makes it easy to define and model any XLuminate application scenario: to drag-and-drop icons representing the different light source positions of the application into a visual representation of the real product, label the light sources in a meaningful way and associate each light source to the hardware 'source' and 'sink' matrix lines driving it. Once this map of light sources to physical led connections is complete, the XLine Pattern Generator allows the user to easily create patterns of these light sources, where the brightness (and/or colour) of each light source can be individually defined. Finally the pattern generator allows groups of these patterns to be linked together into sequences where the display time of each pattern in the sequence can be defined. The pattern generator produces a 'pattern file' as output which is then directly loadable and executable by the XLuminate board. Chapter 4 of this manual described the structure and syntax of pattern files.

2 API FUNCTIONS

Function Return Values:

All functions in this API library, with the exceptions of `GetLastError()` and `GetErrorLineNumber()`, return a Boolean result value: true indicates command success, and false that the function has failed. In the event of error, use `GetLastError()` to obtain the Error Code identifying the cause of the error. If an error is encountered when loading a pattern file with function `LoadPatternFile()`, the line number of the pattern file triggering the error can be obtained using `GetErrorLineNumber()`.

The different error codes that can be generated and their meaning are listed in chapter 3 of this document.

2.1 Function name: `XLuminate ()`

This function is called automatically when instantiating a class `XLuminate` object. Initialises internal class variables.

Function Prototype

```
XLuminate( )
```

2.2 Function name: `XLuminate (XLuminate &)`

A library supplied copy constructor for `XLuminate` objects. Provides proper copy construction behaviour, and allows temporary `XLuminate` objects to be passed to functions by value or reference.

Function Prototype

```
XLuminate (XLuminate &)
```

2.3 Function name: `operator = (XLuminate &)`

Overloaded equals operator. Allows the use of assignment on `XLuminate` objects.

Function Prototype

```
XLuminate & operator = (XLuminate &)
```

2.4 Function name: `~XLuminate ()`

This is called automatically when an `XLuminate` object is deleted or goes out of scope and is not called directly. Cleans up internal class variables in addition to closing the `XLuminate` Board if it is still open.

Function Prototype

```
~XLuminate( )
```

2.5 Function name: `InitUSBBoard()`

Initialises an `XLuminate` board. The `boardNumber` variable supplied specifies which of up to 8 `XLuminate` boards connected to the system should be initialised. On each `XLuminate` board, DIL switch pack (SW1) bits 0-2 (switches 1-3) are used to set the identity of each board. All switches closed (on) corresponds to address 0. All switches open (off) corresponds to address 7.

All further function calls following this initialisation will be made to the board with matching ID.

The `XLuminate` board is initialised to operate in monochrome display mode.

Function Prototype

```
bool InitUSBBoard( unsigned char boardNumber );
```

2.6 Function name: Init()

Initialises the first XLuminate board found. This is a simplified version of the function InitUSBBoard() and can be used when only one XLuminate board is connected to the system.

The XLuminate board is initialised to operate in monochrome display mode.

Function Prototype

```
bool Init( );
```

Programming Considerations

Do not use this function if more than one XLuminate board is connected to the system.

2.7 Function name: Close()

As the application program completes operation and terminates, it should issue a call to the Close() function to release memory used by the application and exit the program in an orderly way.

Function Prototype

```
bool Close( );
```

2.8 Function name: GetBoardSpeed()

The XLuminate board can work at two different speeds: if the board is plugged into a USB 1.1 hub then it will work at “full speed”, and if it is plugged into a USB 2 host/hub then it will work at “high speed”. High speed is considerably faster than full speed and this allows many XLuminate functions to perform faster.

This function returns the speed at which the currently fitted board is working at.

The boardSpeed parameter must be a pointer to an unsigned char which will be set to the speed at which the currently fitted board is working. Three possible values can be returned:

- UNKNOWN_SPEED : The speed is unknown, i.e. the board is not behaving correctly.
- USB_1_1_FULL_SPEED : The board is working at full speed.
- USB_2_0_HIGH_SPEED : The board is working at high speed.

Function Prototype

```
bool GetBoardSpeed( unsigned char *boardSpeed );
```

2.9 Function name: GetProductVersion()

This function reports the XLuminate Development Suite Product version.

Function Prototype

```
bool GetProductVersion( unsigned char *versionProduct );
```

Programming Considerations

Writes a string of not more than 10 characters including the null terminator via the supplied string pointer.

2.10 Function name: GetAPIVersion()

This function reports the version of the API that is providing access to the XLuminate board. It may be used to check that the correct version has been installed and is being called. This is an enquiry about the installed software revision and does not require a physical XLuminate board to be connected at the time the function is issued.

Function Prototype

```
bool GetAPIVersion( unsigned char *versionAPI );
```

Programming Considerations

Writes a string of not more than 10 characters including the null terminator via the supplied string pointer.

2.11 Function name: GetFirmwareVersion()

This function reports the version number of the firmware that has been downloaded to the XLuminate board during power-up, or at any time the XLuminate board is re-connected to USB.

Function Prototype

```
bool GetFirmwareVersion ( unsigned char *firmware_version );
```

Programming Considerations

Writes a string of not more than 10 characters including the null terminator via the supplied string pointer.

2.12 Function name: GetLastError()

This function returns the error code for the last function call that failed and updated the error code. See also function `ClearErrors()`.

Function Prototype

```
xluminateErrorCode GetLastError( );
```

Return value

A variable of type `xluminateErrorCode` that describes the reason for failure, see also chapter 3 on *API Return Codes*.

Programming Considerations

Be aware that some API function calls report errors. Therefore an error code reported by `GetLastError()` will relate to the last API function that actually set a `LastError` value.

2.13 Function name: ClearErrors()

This function clears any previous recorded error code, restoring `RETCODE_SUCCESS` as the default error code. This function also resets the pattern file error line number to 0.

Function Prototype

```
bool ClearErrors( );
```

2.14 Function name: GetErrorLineNumber ()

This function returns additional error information only when errors are encountered in the `LoadPatternFile` and `SetDisplayConfig` functions. This value is not valid at any other time.

- Where the error message includes the words `_PATTFILE_` or `_CONFIG_` `GetErrorLineNumber` reports the line number in the pattern file triggering the error condition.

- Where the error message includes the word `_LEDMAP_ GetErrorLineNumber` reports the logical cell in the LEDMAP definition that is triggering the error condition.

The value returned may need to be interpreted to arrive at the real cause of an error. See chapter 4 for hints on troubleshooting pattern files and related error messages.

Function Prototype

```
int GetErrorLineNumber( );
```

Return value

Location where the error occurred.

2.15 Function name: GetDIPSwitches()

This command returns the value of the 8-position DIL switch on the board. The three least significant switches are used for board ID. These may be used in conjunction with function `InitUSBBoard()` to initialise a specific XLuminate board. Switches in the 'ON' position return a '1' bit value; 'OFF' switches return '0'.

Function Prototype

```
bool GetDIPSwitches(unsigned char *switch data);
```

2.16 Function name: SetDisplayConfig()

This function has very important effects on the operation of the XLuminate board:

- It sets display characteristics eg. number of display planes, rows and columns, etc.
- It defines whether the XLuminate board operates in monochrome or RGB-mode.
- It is used to load the LEDMAP, linking logical cells to underlying physical light sources.
- Defines the refresh rate; the frequency at which matrix rows/columns are scanned.
- Where 7-segment displays share cell space with RGB cells, define where the 7-digits are mapped in to the RGB space, and how many digits there are.

Various function parameters have maxima, minima and default values as defined in the table below.

The purpose of several of this function's parameters is reasonably self explanatory: eg. number of displays, rows, columns, etc.

Refresh rate defines how frequently the display hardware scans the pattern image out onto the display leds. In each refresh period, 16 row brightness values are presented onto a display column of the display, then the column number advanced to the next column for the next refresh period. As the XLuminate has 16 columns in total, and the default XLuminate board `rfsh_rate` of 1mS, it takes 16mS to refresh the full XLuminate display. Note: all 3 XLuminate display planes are refreshed in parallel.

`TransposeFlag` indicates that row number and column number supplied in each LEDMAP entry should be swapped over. The need for this depends on whether display hardware scans out the display image row-at-a-time - building the image from top to bottom, or column-at-a-time - building the image from left to right. For the XLuminate hardware this parameter is 'set' by default, ie. the image is scanned column-at-a-time - building the image from left to right.

Where 7-segment display digits are mapped into the same cell space as used for RGB leds (for example the Heber RGB-Evaluation board), `digitCellOffset` and `num7SegDigits` define the cell start address of the digit segments, and how many digits there are.

`DisplayMode` can be set to `MONO_MODE=(0)` or `RGB_MODE=(1)` which in turn determines how LEDMAP data is interpreted, and how several other display related API functions (e.g. `EnabledDisplay`) behave. See individual function descriptions for details.

Lastly come two parameters about the LEDMAP, the count of LEDMAP entries being supplied (each element is an unsigned short - 3 elements for each RGB cell) and a pointer to these LEDMAP entries.

Example of one LEDMAP Entry for an RGB led:

		Red	Red		Green	Green		Blue		
Cell	Number of elements	Display	Row	Column	Display	Row	Column	Display	Row	Column
15	3	0	5	6	0	5	7	0	5	8

Example of LEDMAP Entry for a monochrome led if operating in RGB mode:

			Mono							
Cell	Number of elements	Display	Row	Column	Display	Row	Column	Display	Row	Column
15	1	0	2	3	0	0	0	0	0	0

Logical cell values do not have to start at zero, nor do all logical cell values have to be defined - the highest cell number defined in the LEDMAP sets the maximum logical cell, and therefore effectively the pattern size, up to a limit of NumRows * NumColumns. LEDMAP entries do not have to be entered in numeric order although doing so makes the table easier to read. Gaps in the table of logical cells are automatically marked undefined and do not influence the displayed pattern.

The SetDisplayConfig function expects all mapping values supplied to be binary values and not strings. Checks performed on the entered LEDMAP values are:

- Check for duplicate cell definitions
- Check that display number, row number and column numbers in LEDMAP do not exceed the maximum values specified in the SetDisplayConfig command.

Troubleshooting errors in LEDMAP data entry can be tricky. See chapter 4 on Pattern Files for help with finding configuration/LEDMAP errors.

Parameter minimum, maximum, and default values:

Variable	Range
Display panes	Minimum = 1, Maximum = 3, Default = 3
Rows	Minimum = 3, Maximum = 64, Default = 16
Columns	Minimum = 3, Maximum = 64, Default = 16 (Note: Total of Rows * Columns must not exceed 256)
Refresh Rate (uS)	Minimum = 500uS, Maximum = 5000uS, Default = 1000uS
Transpose	Default = True
Digit Cell Offset	Default = 0
7 Segment Digits	Maximum = 32, Default = 0
Operating Mode	Default = Monochrome

Function Prototype

```
bool SetDisplayConfig ( unsigned char num_displays,
                        unsigned char num_rows,
                        unsigned char num_columns,
                        int rfsh_rate,
                        unsigned char transpose_flag,
                        int digitCellOffset,
                        unsigned char num7SegDigits,
                        unsigned char DisplMode,
                        int LEDMAPDataLen,
                        unsigned char *LEDMAPDataPtr );
```

See the example program (fades_patt.cpp) supplied with the XLuminate Development Kit for examples of LEDMAP definition and SetDisplayConfig() function use.

2.17 Function name: EnableDisplay()

This command enables the specified display. Issuing this function call on its own does not result in any data pattern being displayed, a pattern will also need to be loaded using the WritePattern function and selected using the SelectPattern function in order for the pattern to be visible.

When operating in monochrome display mode, the specified display_plane alone is enabled.

When operating in RGB mode, all three display planes are automatically enabled by this call, therefore the display_plane value supplied does not matter provided it is a legal display_plane value.

Use function GetDisplayStatus() to discover if a display plane is enabled or not.

Function Prototype

```
bool EnableDisplay( unsigned char display_plane );
```

2.18 Function name: FadeDisplay()

This command slowly dims the brightness of all the cells of the selected display plane from their initial brightness levels down to zero, one brightness step per display update cycle . It will take approx 4 seconds to dim the display from full brightness level if using default number of columns and display refresh rate.

Once this command has been issued, the dimming process cannot be halted part way through dimming. However the displayed pattern can be refreshed or changed using the SelectPattern() function at any time during the dimming process.

When operating in monochrome display mode, the selected display_plane alone is faded.

When operating in RGB mode, all three display planes are faded by this call, the display_plane value specified in the call does not matter provided it is a legal display_plane value.

Function Prototype

```
bool FadeDisplay(unsigned char display_plane);
```

2.19 Function name: ClearDisplay()

This command sets the brightness level of all of the cells of the selected display plane to zero.

When operating in monochrome display mode, the selected display_plane alone is cleared.

When operating in RGB mode, all three display planes are cleared by this call, therefore the display_plane value specified in the call does not matter provided it is a legal display_plane value.

Function Prototype

```
bool ClearDisplay(unsigned char display_plane);
```

2.20 Function name: DisableDisplay()

This command stops any pattern sequences running on the specified display and disables the display (meaning no further pattern data will be output to that display). To reverse this action and once again display information on the display, use the EnableDisplay() function.

All display-planes are disabled by default when the XLuminate board is powered-up.

This function also cancels the effects of issuing the TestDisplays function.

Use the function GetDisplayStatus to discover if a display plane is enabled or not.

When operating in monochrome display mode, the selected display_plane alone is disabled.

When operating in RGB mode, all three display planes are disabled by this call, therefore the display_plane value specified in the call does not matter provided it is a legal display_plane value.

Function Prototype

```
bool DisableDisplay(unsigned char display_plane);
```

2.21 Function name: GetDisplayStatus()

This command reports on the status of the specified display plane: whether display is enabled or not, what pattern is being displayed, is a pattern sequence running, etc. Information is returned by the function to a DisplayStatusStruct the members of which are described below.

When operating in monochrome display mode, information about the specified display_plane alone is returned.

When operating in RGB mode, all three display planes will report the same information, hence the display_plane value specified in the call does not matter provided it is a legal display_plane value.

Function Prototype

```
bool GetDisplayStatus( unsigned char display_plane,
                      DisplayStatusStruct *retDispStat );
```

DisplayStatusStruct:

unsigned char displayEnabled	Specified display plane is enabled (TRUE) or not (FALSE).
unsigned char displayMode	Either MONO_MODE(=0) or RGB_MODE(=1).
unsigned char activePatternNo	Pattern number currently being displayed.
unsigned char sequencerStatus	Either SEQ_IDLE(=0) or SEQ_RUNNING(=1)
unsigned char activeSequenceNo	What sequence number is selected (may be running or not)
unsigned char nextSequenceNo	What sequence number follows at end of current sequence.
int logicalCellsDefined	Highest numbered cell defined in LEDMAP (=size of pattern)
unsigned char maxPatterns	Computed from available SRAM memory and current pattern size (max patterns=255).

2.22 Function name: SelectPattern()

This command loads the specified pattern data from SRAM to the specified display for viewing. Note an error is returned if the display plane has not already been enabled using function EnableDisplay(). Similarly an error is returned if the display plane has a sequence running at the time this function is called.

When operating in monochrome display mode, a monochrome fixed-size 256-cell pattern is activated on the selected display_plane_specified in the call.

When operating in RGB mode, depending on the LEDMAP definition, potentially all three display planes are employed to activate the RGB pattern. Furthermore the display_plane value specified in the call does not matter in RGB mode, provided it is a legal display_plane value.

Function Prototype

```
bool SelectPattern(unsigned char display, unsigned char pattern_number);
```


2.23 Function name: WritePattern()

This command allows a display pattern, or part of a pattern to be written to SRAM on the XLuminate board, ready for subsequent display either by using the SelectPattern function or by including the pattern as part of a pattern sequence.

How many distinct patterns can be stored in the XLuminate board depends on the maximum pattern size (which in turn depends on the number of logical cells defined via LEDMAP and the display operating mode - mono or RGB). Assuming a display plane with 256 monochrome leds, a total of 255 patterns can be stored (255 is also the maximum number of patterns allowed). If the display consisted of 256 RGB leds then 123 patterns can be stored. The actual pattern size, and the number of patterns of that size that can be stored on the XLuminate board are reported by the GetDisplayStatus function.

The pattern data written by the WritePattern function consists of a series of brightness values for each logical cell in the pattern (a single byte for each logical cell if operating in monochrome mode, 3 bytes for each logical cell in RGB mode - in the order Red, Green and Blue brightness values).

Brightness values can range from 0 (off) through 255 (maximum brightness). Be aware that the light levels emitted by leds can vary widely, between colours with RGB leds, and from one led manufacturer to the next. A brightness lookup table might be added to the application program to help linearise the brightness profiles.

An entire pattern may be written by specifying first-cell(=0), last cell(=logicalCellsDefined-1), and supplying the appropriate amounts of cell brightness data (1-byte per logical cell in monochrome, 3 per cell in RGB mode). Alternatively just a limited range of cell positions from 'firstlamp' through 'lastlamp' inclusive can be modified.

Where a limited range of cells is being modified, the programmer has the choice of merging this limited range of cells into any existing pattern data (clear_first=false) or else replacing all of the existing data in that pattern with just the limited range of cells specified (clear_first=true).

Monochrome example:

```
unsigned char patt_data[15] = { 120 }; // 15 cells at brightness 120
WritePattern(12, 20, 34, patt_data, true);
SelectPattern(1, 12);
```

This will modify pattern number 12, setting lamps 20 to 34 inclusive to brightness level 120, will clear any existing pattern first and will show the pattern on display-1

RGB example:

```
// 3 RGB-cells: cell-10=max red, cell-11=max green, cell-12=max blue
unsigned char rgb_data[9] = { 255,0,0, 0,255,0, 0,0,255 };
WritePattern(5, 10, 12, rgb_data, true);
SelectPattern(0, 5);
```

This will modify pattern number 5, setting cells 10 through 12 to maximum brightness red, green and blue respectively, and then show on the display.

Somewhat similar functions oriented to generating and writing 7-segment display pattern data are WriteDigit() and WriteDigitString().

Function Prototype

```
bool WritePattern( unsigned char pattern_number,
                  unsigned char firstcell,
                  unsigned char lastcell,
                  unsigned char *cell_data,
                  bool clear_first );
```

2.24 Function name: ClearPattern()

This function will clear the specified monochrome pattern if operating in monochrome mode, or the specified RGB pattern if operating in RGB mode.

Function Prototype

```
bool ClearPattern( unsigned char pattern_number );
```

2.25 Function name: LoadPatternFile()

This function allows pattern files generated with the Heber XLine Pattern Generator Application or created manually with a text editor to be loaded into the XLuminate board. Pattern files allow the XLuminate board configuration to be set, a LEDMAP to be loaded, and display patterns and pattern sequences to be defined, ready to be invoked with SelectPattern() or StartSequence functions.

The 'patternFileName' parameter can point to a simple file name or can include relative or absolute path components with the filename. The error code RETCODE_CANNOT_OPEN_PATT_FILE will be returned if the file cannot be found or is already in use by another application.

Function Prototype

```
bool LoadPatternFile( unsigned char *patternFileName );
```

2.26 Function name: WriteSequenceData()

A sequence consists of a collection of pattern numbers and display times that define the display sequence. The WriteSequenceData() function is used to write this sequence information into XLuminate SRAM. The StartSequence() function is subsequently used to select which sequences are to be executed and on which display plane.

The sequence data consists of up to 127 pairs of bytes specifying a pattern-number and a display time. A maximum of 16 distinct sequences can be specified.

When writing the sequence data with this function, the display time values are specified in units of 10 milliseconds (precision 10mS) so that a reasonable span of pattern time can be accommodated in a byte storage unit; thus a time range from zero through to a maximum of 2.5 seconds per pattern can be specified. Note however that the sequence time values specified via a pattern file (loaded with LoadPatternFile) are in more intuitive mS units.

Example:

```
unsigned char seq_array[] = { 2, 10, 3, 7, 1, 5 };
WriteSequence( 4, seq_array, 6 );
StartSequence( 0, 4, 0 );
```

This will define sequence 4 as being: pattern 2 for 100mS; pattern 3 for 70mS; pattern 1 for 50mS, then starts the sequence on display-0, and runs through sequence just once (loops=0).

The sequence data written by this command is exactly the same whether operating in monochrome mode or RGB mode, this data simply identifies pattern numbers and durations.

Function Prototype

```
bool WriteSequence( unsigned char sequenceNum,
                    unsigned char *sequence_array,
                    unsigned char sequenceLength );
```

2.27 Function name: StartSequence()

This command specifies which sequence of patterns, specified earlier using `WriteSequence()`, is to be presented for display on the specified `display_plane`. The parameter 'num_sequence_loops' is used to define how many times the entire sequence of patterns is to be repeated (0 = display sequence just once, 1-254 = number of loops through sequence, 255 = reserved value to indicate that sequence should be repeated continually until stopped by the `StopSequence()` function.

When a sequence finishes, the final pattern of the sequence will remain on the display until the display is cleared or disabled.

`StartSequence()` will return an error if the selected display plane is disabled. Use `GetDisplayStatus()` to discover if a particular display plane is enabled or not. No error is reported if a sequence had been already running on the display, the new sequence simply starts and executes as if the previous sequence had never existed.

In monochrome mode, separate sequences can be run on each of the three XLuminate display planes.

When in RGB mode, all three display planes are being used for RGB pattern data, therefore just one RGB sequence can be run at a time. In RGB mode the `display_plane` value specified in the call does not matter provided it is a legal `display_plane` value.

Function Prototype

```
bool StartSequence(      unsigned char display_plane,
                        unsigned char sequence_number,
                        unsigned char num_sequence_loops);
```

2.28 Function name: SetNextSequence()

This function allows two sequences of patterns to be executed one after the other without time gaps or glitches in the transition from one sequence to the next, independent in time of the application program. A typical application program might start a sequence, set a next-sequence, and then periodically check the `nextSequenceNo` parameter obtained using `GetDisplayStatus`. When this value switches to `NO_NEXT_SEQUENCE_MARKER` (0xFF) value, the first sequence has completed and the second sequence has become active. The application then has the duration time of the second sequence to decide if it wishes to chain yet another sequence to follow the currently active sequence by issuing a further `SetNextSequence` function.

The parameters supplied with `SetNextSequence` are identical to those supplied with `StartSequence`, and identical error conditions may be reported (illegal display number, display disabled, or illegal sequence number)

In the event that `SetNextSequence` is called when the sequencer is not busy running another sequence, ie. is IDLE, the `SetNextSequence` function acts like the `StartSequence` function and starts the sequence running immediately, leaving the programmer then free to specify another next-sequence.

The `SetNextSequence` values for sequence number and sequence loops can be overwritten at any point up until the moment the sequence becomes the current active sequence.

Note that issuing a `StopSequence` function will not only halt the currently running sequence, if any, but will also clear any `NextSequence` programmed at that point.

As for the `StartSequence` function, in monochrome mode a next-sequence can be specified for each of the three XLuminate display planes.

When in RGB mode, all three display planes are being used for RGB pattern data, therefore just one next-sequence can be specified. In RGB mode the `display_plane` value specified in the call does not matter provided it is a legal `display_plane` value.

Function Prototype

```
bool SetNextSequence(    unsigned char display_plane,
                        unsigned char next_sequence_number,
                        unsigned char num_next_sequence_loops);
```

2.29 Function name: StopSequence()

This function stops the pattern sequence currently running on the specified display_plane. Note that after calling `StopSequence()`, and whatever pattern from the sequence was being displayed at the moment the `StopSequence` function was called will remain on the display until cleared by `ClearDisplay()` or `DisablDisplay()`.

A sequence, once stopped, can not be resumed at the point where it was stopped; the full sequence can be re-started from the beginning again by issuing `StartSequence` or `SetNextSequence`.

The `StopSequence` function also clears any pending `NextSequence` queued for execution.

In monochrome mode, a sequence running on any one of the three distinct display_planes can be individually stopped. As all 3 display_planes are being used simultaneously in RGB mode, the display_plane value specified in the call does not matter provided it is a legal display_plane value.

Function Prototype

```
bool StopSequence( unsigned char display_plane );
```

2.30 Function name: WriteDigit()

7-segment display information is treated as a special form of display pattern data, as might be written using function `WritePattern`. The `WriteDigit` and `WriteDigitString` functions simplify the generation of 7-segment pattern data based on the value of the digit required for display and at which digit position it should be displayed.

Display characters are the digits 0-9, hexadecimal A-F or a-f, decimal point and the 'space' character.

Operation of this command is heavily dependent on the design of the LEDMAP led layout information and whether the XLuminate board is operating in monochrome or RGB mode; digits must be defined in contiguous logical cells starting at 'digitCellOffset' specified with `SetDisplayConfig()` function, 8 logical cells per digit in the following sequence: top-bar, upper-right, lower-right, bottom-bar, lower-left, upper-left, centre-bar, and decimal point; in RGB-mode the RED component of RGB values drives the segment.

The number of digit positions will depend on the display board being driven: early XLuminate Monochrome Evaluation boards implemented 32 digits using an entire display_plane in monochrome mode, the later XLuminate RGB Evaluation board implemented 8 digits which allowed integrated displays of RGB leds and 7-segment digits at the same time.

When the 7-segment pattern data has been written using this function, use `SelectPattern()` to activate the new pattern on the display.

Example:

```
WriteDigit( 2, 4, "F", 44 );
```

This will write a hex-F character at digit position 4 within pattern-2 with a brightness level of 44.

Function Prototype

```
bool WriteDigit(    unsigned char pattern_number,
                   unsigned char digit_position,
                   unsigned char value,
```

```
unsigned char brightness);
```

2.31 Function name: WriteDigitString()

This function will add the pattern data representing the supplied hexadecimal text string to the specified display pattern, starting at the specified digit position, for the number of digits requested.

This function can be regarded as multiple WriteDigit operations, LEDMAP and mono/RGB mode selection considerations are the same as for WriteDigit function.

Legal display characters are the same as those for WriteDigit() function, but with two additional facilities:

- a '\0' encountered in the supplied string before the requested number of digits have been displayed allows the function to terminate after output of a partial string.
- if first_digit_position plus number_of_digits exceeds the available digit positions, no error message is generated; the excess digits are ignored.

When the 7-segment pattern data has been written using this function, use the SelectPattern function to select the new pattern data for display.

Example:

```
WriteDigitString( 2, 2, 3, "12.34 56", 94);
```

This will attempt to write a digit string of 9 characters, starting at digit position 2, into display pattern 2, with a brightness level of 94 for all segments. If run on the 8-digit RGB-Evaluation board, this example will also demonstrate that the end of the string extending past the end of the available digits will be dropped and the string will appear as "12.34 " from digit-position 2 onward.

Function Prototype

```
bool WriteDigitString( unsigned char pattern_num,
                      unsigned char first_digit_position,
                      unsigned char number_of_digits,
                      unsigned char *digit_values,
                      unsigned char brightness );
```

2.32 Function name: TestDisplays()

The TestDisplays function is a useful troubleshooting tool to confirm that all lamps or leds are connected to physical source/sink matrix driving lines, so is completely independent of the LEDMAP mapping mechanism. All lamps/leds on all display planes are turned on at full brightness.

The test can be launched without the need to enable display planes first; all displays remain on until a DisableDisplay function is issued to any display plane, or the board is reset. All displays are disabled following this test.

Function Prototype

```
bool TestDisplays( );
```

3 API RETURN CODES

The following table describes the XLuminate API return codes (as obtained using `GetLastError()`). The codes are defined in file `xluminatedefs.h`.

No	Return Code	Description
0	RETCODE_SUCCESS	No error - successful operation.
1	RETCODE_BOARD_NOT_INITIALISED	The XLuminate board is failing to initialise. Try disconnecting and reconnecting the USB cable.
2	RETCODE_INVALID_POINTER	A NULL pointer was passed in an API function.
3	RETCODE_PARAMETER_OUT_OF_RANGE	An API function parameter is invalid.
4-21	Reserved	
22	RETCODE_USB_DEVICE_WRITE_ERROR	The API function is unable to write data over USB. Try disconnecting and reconnecting the USB cable.
23	RETCODE_USB_MSG_LENGTH_ERROR	An API function is sending an incorrect packet size to the board. Check that the XLuminate Firmware version matches the API/DLL version.
24	RETCODE_USB_DEVICE_RESPONSE_ERROR	This often means that the API/drivers do not match the firmware on your XLuminate board.
25	RETCODE_USB_CMD_NOT_RECOGNISED	An API function is sending an unrecognised USB command to the board. Check that the XLuminate Firmware version matches the API/DLL version.
26	RETCODE_USB_REPLY_MSG_LEN_ERR	The length of the reply to a USB command is incorrect. Check that the XLuminate Firmware version matches the API/DLL version.
27-35	Reserved	
36	RETCODE_SRAM_READWRITE_FAILURE	Failure to access 128k XLuminate on-board SRAM
37	RETCODE_SRAM_ADDRESS_RANGE_ERROR	Attempt to access SRAM outside available 128k
38	RETCODE_DISPLAY_DISABLED	Attempt to SelectPattern or StartSequence while display disabled.
39	RETCODE_SEQUENCE_ALREADY_RUNNING	Attempt to SelectPattern while sequence running.
40	Reserved	
41	RETCODE_SEQUENCE_LENGTH_ERROR	Sequence data must be between 1 and 127 pattern-time pairs long.
42	RETCODE_SEQUENCE_WRITE_ERROR	Failed to transfer sequence to XLuminate SRAM
43	RETCODE_CANNOT_OPEN_PATT_FILE	File named in LoadPatternFile function not found.
44	RETCODE_PATTFILE_UNKNOWN_KEYWORD	Unrecognised keyword in pattern file.
45	RETCODE_PATTFILE_BAD_NUMERIC	Numeric conversion of input number string failed.
46	RETCODE_PATTFILE_BAD_VALUE	Value of a numeric parameter is outside allowed range
47	RETCODE_PATTFILE_WRONG_NUM_PARAMS	Wrong number of parameters supplied with keyword.
48	RETCODE_PATTERN_LENGTH_ERROR	Wrong amount of pattern data supplied with 'P' keyword.
49	RETCODE_PATTERN_NUMBER_OVER_MAX	Pattern number is greater than max available.
50	RETCODE_INVALID_7SEG_CHAR	Legal characters: 0-9, a-f, A-F, 'space' and 'period'
51	RETCODE_CONFIG_WRONG_DISPLAY_MODE	Bi-colour and/or RGB leds require RGB-mode.
52	RETCODE_CONFIG_NUM_DISPLAYS_ILLEGAL	Number of displays requested by pattern file is illegal.
53	RETCODE_CONFIG_NUM_ROWS_ILLEGAL	Number of rows requested by pattern file is illegal.
54	RETCODE_CONFIG_NUM_COLUMNS_ILLEGAL	Number of columns requested by pattern file is illegal.
55	RETCODE_CONFIG_ROWS_X_COLS_ILLEGAL	Product of Rows x Columns in pattern file is illegal.
56	RETCODE_CONFIG_ILLEGAL_REFRESH	Display refresh time in pattern file illegal
57	RETCODE_CONFIG_DIGIT_BASE_ILLEGAL	7-seg digit base address beyond max logical cell
58	RETCODE_CONFIG_NUM_DIGITS_ILLEGAL	Insufficient logical cells to fit full 7-seg display
59	RETCODE_CONFIG_PARAM_OVER_RANGE	Parameter in SetDisplayConfig API function illegal.

No	Return Code	Description
60*	RETCODE_LEDMAP_UNKNOWN_LED_TYPE	Led type in LEDMAP must be 'R', 'B' or 'M' only
61*	RETCODE_LEDMAP_MORE_ROW_COL_DATA	Not enough disp/row/column data supplied for mode
62*	RETCODE_LEDMAP_DUP_LOGICAL_CELL	Logical cell number already defined in LEDMAP
63*	RETCODE_LEDMAP_LOG_CELL_OVER_MAX	Logical cell number exceeds rows x columns
64*	RETCODE_LEDMAP_DISP_NUM_OVER_MAX	Display number in disp/row/column address illegal.
65*	RETCODE_LEDMAP_ROW_NUM_OVER_MAX	Row number in disp/row/column address illegal.
66*	RETCODE_LEDMAP_COL_NUM_OVER_MAX	Column number in disp/row/column address illegal.
67*	RETCODE_LEDMAP_WRITE_ERROR	Failure while writing LEDMAP to XLuminate board.
68	RETCODE_PATT_ADDR_TAB_WRITE_ERROR	Failed to write pattern address lookup table.
69	RETCODE_PATT_ADDR_TAB_READ_ERROR	Failed to read pattern address lookup table.

* GetErrorLineNumber function returns failing logical cell in LEDMAP for these error codes.
 Otherwise, when loading pattern files, GetErrorLineNumber reports text line in file causing error.

4 PATTERN FILES

4.1 Overview:

The XLine Pattern Generator application eliminates much of the typing drudgery involved in creating pattern files manually. However for the rare occasions where the user wishes to create a pattern file manually, and/or to understand or modify pattern files produced by the XLine Pattern Generator, the following paragraphs describe the structure and contents of pattern files.

Pattern files consist of 4 main sections:

- Display configuration information - numbers of rows, columns, displays etc.
- LEDMAP definition that describes which light resources are controlled by which matrix drive lines.
- Definitions of display patterns.
- Definitions of sequences of patterns that may be executed with a timeline.

All parameters have default values so that only those parameters that need to be changed need appear in the pattern file. Others will assume their default values.

Although the order of the different sections as they appear in the pattern file is not strictly policed, be aware that the order listed above is strongly recommended. The reasons are that settings in display configuration information (numbers of rows, columns, monochrome or RGB mode of operation) have fundamental impacts on the definition of the LEDMAP, which in turn influences pattern size, which in turn impacts the number of patterns that can be stored hence available patterns to be specified in sequences.

4.2 Pattern File Syntax:

A pattern file is a simple text file containing instructions to define the operation of an XLuminate board; the files may be examined/edited with a wide range of text editing programs.

Pattern files use the '#' character to indicate comments: with one exception comments are ignored in the processing of the pattern file. Exceptions are lines containing "# META_LEDMAP ...". These lines contain special information used exclusively by the XLine Pattern Generator application and should not be altered. Comments can begin anywhere on a line and result in the '#' character and all remaining characters through to the end of that line being ignored.

The remainder of the pattern file consists of 'keywords', complete words or single letters reserved by the application that inform the XLuminate board which parameter is being altered.

Keywords are case neutral and can be entered in upper-case, lower-case or a mixture of the two. Pattern file keywords may be followed by zero or more 'parameters' depending on the keyword, and inform the XLuminate board how it should perform certain tasks, and the data it performs the task upon.

Numeric values for parameters generally use decimal values; hexadecimal values can also be used by preceding the hex value with "0x".

Keywords are separated from parameters and further keywords by white-space. White-space is defined as one or more 'space' characters, tab characters, commas, periods, equal signs, hyphens or CR (end-of-line) characters.

4.3 Display configuration section:

4.3.1 Number of displays:

An XLuminate board on its own includes enough lamp/led driving transistors to control up to 256 leds wired in a matrix arrangement. This 256-led entity is described as a 'display plane'. Up to two further 'display

planes' can be added to the XLuminate configuration by the addition of up to two Multiplex Expansion boards. This parameter specifies the total number of display planes available.

```
# Example:
Displays = 3           # using 3 display planes
```

The default setting is 3 displays.

4.3.2 Number of rows:

It is convenient to view the arrangement of 256 led driving signals of a display plane as a matrix of 16 rows (vertical dimension) and 16 columns (horizontal dimension). In terms of hardware matrix driving signals the rows are 'sources' of electrical current directing power from the power supply to the leds and controlling the brightness of the leds. For XLuminate product this parameter has a default value of 16.

```
# Example:
Rows = 16             # 16 rows
```

The default setting is 16 rows.

4.3.3 Number of columns:

In terms of matrix driving signals the columns are current 'sinks' completing the circuit to power supply return (0v). The XLuminate is continually cycling through each sink line in turn, activation each for a fixed period of time, in order to build the 2-dimensional rows and columns display. For XLuminate this parameter has a default value of 16.

```
# Example:
Columns = 16          # 16 columns
```

The default setting is 16 columns.

4.3.4 RGBMode:

XLuminate applications can be constructed using single-colour lamps or leds, and/or with leds of two or 3-colour (RGB) elements per display cell. To cater for the use of single-colour and multi-colour leds, the XLuminate has two possible operating modes:

- monochrome mode -- where only single-colour leds are being used, display resources consist of up to 3 display planes of 256 monochrome leds each (XLuminate board plus two Multiplex Expansion boards);
- RGB-mode -- where RGB leds or a mix of RGB and monochrome leds are being used, all display resources are concentrated into just one display plane but this display plane can consist of up to 256 RGB leds (with an XLuminate board plus 2 Multiplex Expansion boards).

By default the XLuminate board starts in monochrome mode.

```
# Example:
RGBMode = 1           # RGB mode selected
```

The default setting is RGBMode = 0 (monochrome mode).

4.3.5 Transpose:

If considering how to generate a 2-dimensional row/column display array, it is most natural to compare the process to the way we read a printed page -- reading from top to bottom. In the context of driving lamp/led arrays this requires generating brightness data for all columns of a single row and outputting this information to the row, then advancing to successive rows until the full display has been generated. Implicit in this description is that pattern data be stored to match the scanning process, ie. row-at-a-time.

However the designers of display hardware may have various reasons to design the hardware to generate the display signals column-at-a-time instead (electrical characteristics, compatibility with older products, etc.). The XLuminate board is one such example.

In order to allow data patterns to be stored in the way that is most natural to users - row-at-a-time, yet will be displayed correctly on a device using column-at-a-time control signals requires that the image be 'transposed' to column order. That is achieved by setting the transpose parameter to '1' which is also the default value for the XLuminate product.

```
# Example:
Transpose = 1           # row/column transpose is active
```

Implicit in the above description is that the LEDMAP (see below) describing the logical arrangement of light resources (logical-cells) and linkage to the underlying lamps/leds was created in row-at-a-time order. If the LEDMAP is designed to have column-at-a-time order instead, then there is no need for the transpose parameter.

The default setting is Transpose = 1.

4.3.6 Refresh:

The displayed image is created by presenting 16 lines of brightness information (row information) for one column in the display, and activating that column for one 'Refresh' interval. This process then repeats for each column in turn, ensuring the appropriate brightness information for each column is presented at the correct time. This parameter defines the amount of time spent displaying each column worth of brightness information.

The default setting is 1000uS (ie. 1mS), therefore scanning all 16 columns takes 16mS. Values between 500uS(0.5mS) and 5000uS(5.0mS) can be specified in steps of 500uS This refresh time must be set fast enough to avoid display flicker.

```
# Example:
Refresh = 1500          # make refresh time 1.5mS per column.
```

4.4 LEDMAP definition section

The purpose of the ledmap is to associate some arrangement of lamps/leds in a particular application of the XLuminate board, an arrangement and light resource numbering scheme that is easy for the user to work with, with the underlying hardware driving signals of rows, columns and display planes. Whether the actual application has its lamps/leds arranged in the form of a row/column array or not is of no significance provided the light resources themselves are labelled in a systematic way.

A ledmap consists of a series of entry lines where each entry describes a light resource or 'logical-cell'. Each light source may be a lamp or monochrome led, or may be a 2-colour or 3-colour RGB led. The collection of these logical-cells describes the full display.

In the case of a monochrome led, the ledmap will need to define which hardware row and column drive signals are connected to the led and in which display plane it is connected.

Example of ledmap entries for monochrome leds (monochrome-mode):

#	Logical cell	Display_plane	Row(Source)	Column(Sink)
M	0	0	5	2
M	1	0	5	3

(# means a comment line, M means a monochrome resource)

Provided lamps or monochrome leds are used through the XLuminate application, then the XLuminate's monochrome display mode should be selected. Consequences of selecting monochrome mode are:

- The XLuminate board on its own can control up to 256 monochrome leds arranged as a 16 x 16 matrix of rows and columns in a single display plane (display-0).
- Each additional Multiplex Expansion Board added to configuration (limited to a maximum of 2 boards) can drive an additional display plane of 256 monochrome leds (display-1 and display-2).
- Display pattern data to be shown on any display will consist of a single byte of brightness data per logical cell defined in the ledmap.

When any 2-colour or 3-colour RGB leds are used in the application, the format of led map entry changes to convey this information:

Example of ledmap entries for RGB leds mixed with bi-colour and monochrome leds (RGB-mode):

#	Logical cell	Disp	Row	Col	Disp	Row	Col	Disp	Row	Col
R	0	0	7	0	0	7	1	0	7	2
R	1	0	7	3	0	7	4	0	7	5
B	2	1	12	2	1	13	6			
M	3	0	7	6						

(# means a comment line, R = RGB resource B = bi-colour resource M = mono resource)

From this example it can be seen that the ledmap format allows mixing of any types of light resource (mono, bi-colour or RGB leds) in any order desired. However to operate with one or more RGB or bi-colour leds, the XLuminate board must be switched into its RGB display mode. Consequences of switching to RGB mode are:

- The 3 display planes available in monochrome mode (with two Multiplex Expansion boards fitted) are collapsed into a single display plane in RGB mode, using all three display planes to drive up to 256 RGB leds driven as a single display.
- Pattern data per logical cell now needs to contain 3 bytes of brightness data, one byte each for Red, Green and Blue brightness levels.
- 3 bytes of brightness data per logical-cell must still be provided even when using monochrome or bi-colour leds in RGB mode: for monochrome leds the Red value controls the led brightness - Green and Blue values are ignored; for bi-colour leds the Red value controls the brightness of the first element, the Green value the second element, the Blue value is ignored.

The example ledmaps above in each case started with logical-cell zero and went up incrementally from there. This is probably the most logical but is not a requirement: the cells can be defined in any order desired, however if a ledmap is defined starting from, say, logical cell-20 upwards, pattern data will still need to be supplied for blank cells 0-19 in addition to brightness data for cells 20 and above.

Note also that XLuminate software will detect and report a ledmap entry that duplicates the definition of a logical-cell but will not detect the connection of a particular matrix row and column to more than one led.

4.5 Pattern definition section

Pattern definitions begin with a line containing the 'P' keyword followed by the number of the pattern being defined (with or without 'white-space' in between) and two further optional parameters may then follow the pattern number on this first line. The two optional parameters are used when working with partial patterns:

- When only part of a pattern is being replaced/modified, the address of the first pattern cell to be modified. The amount of data supplied determines the number of pattern cells modified. The default start address is zero.
- The second parameter can only be included provided a first pattern cell value has been specified above (however zero is also a perfectly acceptable first pattern cell address). When replacing part of a pattern, this parameter defines whether the rest of the pattern data, apart from the data supplied with this keyword, should stay as it is, or should the rest of the pattern be cleared first before the new cell data are merged in.

The start of the image data portion of a pattern definition is signalled by the CR (new line) character at the end of the first line containing the 'P' keyword. Bytes of pattern data then follow, one byte of brightness data per cell in monochrome mode, 3 bytes (one each for the brightness of red, green and blue components respectively) for each RGB cell until:

- a new pattern file keyword is encountered in the file,

- end-of-file is reached,
- sufficient bytes have been received to fill the remainder of the pattern from the (optional) starting cell specified with the 'P' keyword, to the end of the pattern based on the maximum pattern size and the current display mode (mono/RGB).

Pattern brightness values can range from 0-255 where 0 indicates an led that is switched off, while 255 represents maximum brightness.

```
# Example of pattern definition:
P2 15 1
0x00 0x7F 0xFF
```

The first line identifies Pattern-2 as the target, that the first logical-cell to be modified is cell-15 and the '1' indicates any data previously stored in pattern-2 should be erased before the new pattern data is written.

The interpretation of the three pattern bytes supplied on the second line depends on which display mode is active at the time:

- Monochrome mode: the pattern bytes define the brightness of cells-15 off, 16 at half intensity and 17 at full intensity.
- RGB mode: cell-15 will have no red component, half intensity green and full intensity blue.

The end result of defining patterns in a pattern file is that when the pattern file is loaded the patterns are stored on the XLuminate board ready for display, or can be included in pattern sequences. Processing the pattern file does not in itself cause any patterns to be displayed: the user program will need to issue an API call to SelectPattern or a StartSequence for a sequence that includes the pattern.

4.6 Sequence definition section

Sequences are instructions to the XLuminate to tell it which of the display patterns already stored in its memory should be displayed, in what order, and for how much time per pattern.

Up to 16 sequences can be stored, each sequence can consist of up to 127 entries, each entry identifying the pattern number and the duration in time (in mili-seconds) that the specific pattern should be displayed for. Individual pattern times, although entered in mS can range from 10mS to 2550mS (42 seconds) in 10mS increments. The sum of all pattern times for a sequence defines the overall sequence duration.

In monochrome mode each display plane can be executing separate sequences independently of the other display planes. In RGB mode by contrast, only one pattern sequence can be running at any time.

```
# Example sequence:
S4 10 40 11 100 6 35
```

This example defines sequence-4 as consisting of: pattern-10 displayed for 40mS, pattern-11 for 100mS and pattern-6 for 35mS.

This or any other sequence is not activated by the loading of the pattern file which instead simply defines the sequence(s). The user program will need to issue an API StartSequence function for sequence-4 and associate it with one of the display planes before any pattern data will be displayed. This API call can also be used to cause this sequence to repeat one or more times, or to repeat indefinitely.

4.7 Troubleshooting pattern files:

The XLuminate software package in general can report on a wide range of error conditions as shown by the error codes listed under "API Return Codes" in Chapter 3 of this document.

In order to avoid an excessively long list of error codes covering every possible error condition, similar error conditions are reported with the same error code, so the user has to work out, for example, which of perhaps 3 values supplied in an API function call is triggering an "OUT_OF_RANGE" error report.

Error checking in the area of pattern file definition and loading has been restricted in the interests of making the product as flexible as possible. The result may be that some errors in the pattern file go unreported, or are reported as appearing to be caused by something unrelated.

The following suggestions and tips on tracing errors involving the driving of display leds and the loading of pattern files may be found helpful:

Use the TestDisplays API function to establish that all leds connect to at least one pair of row/column matrix driving lines. This helps isolate software causes from hardware led wiring faults. Next implement the ledmap and check that every individual led can be selected one at a time and confirm that it appears at the correct location in the overall display. RGB and bi-colour leds should be tested one colour at a time.

If an error is reported by the LoadPatternFile API function, the GetErrorLineNumber API function will provide extra information on the location of the error in most cases:

- Where the words `_PATTFILE_` or `_CONFIG_` occur in the error message, the value returned by GetErrorLineNumber will be the line number in the pattern file that triggered the error.
- Alternatively where the word `_LEDMAP_` occurs in the error message, the GetErrorLineNumber value will be the number of the logical-cell in the ledmap causing the error.

Errors with `_PATTFILE_` in the description are frequently typing or syntax errors in the text of the pattern file itself at the indicated line in the file.

Errors with `_CONFIG_` in the description are being reported by the SetDisplayConfig function being called as part of the LoadPatternFile process. Such errors can be because values for numbers of displays, rows or columns exceed permitted maxima, incorrectly specified 7-segment digits, etc. passed through from the pattern file to the SetDisplayConfig() function.

Errors with `_LEDMAP_` in the description try to identify errors in the definition of logical cells in the LEDMAP portion of the pattern file, such as illegal display, row or column values, duplicate logical-cell definitions, etc.

Where a keyword and its associated parameters extend over two or more lines of the pattern file, eg. pattern definitions, errors report the line number on which the keyword appeared (the 'P' in the case of a pattern definition) although the actual error may be in the data on the lines following the keyword.

The number reported by GetErrorLineNumber sometimes needs to be treated with caution. One reason for this is that the LoadPatternFile function accumulates any changes of configuration settings and LEDMAP changes encountered while processing the pattern file up until the first pattern or sequence definition is encountered in the pattern file. At this point all accumulated configuration changes are put into effect and the LEDMAP generated before processing of pattern and sequence definitions begins. Thus an error may be reported as caused by the line defining the first pattern or sequence yet the cause can be errors in either configuration settings or LEDMAP.

For these situations, and on any occasion that pattern file errors are proving difficult to track down, a useful approach is to greatly simplify the pattern file by using the '#' character to comment out lines of the pattern file, then re-introducing the commented-out lines of the pattern file until the point of error is discovered.

4.8 Sample Application - Analog Clock:

4.8.1 Overview:

Although trivial as an application, this sample does illustrate how effectively the programmer/user can be shielded from the complexities of matrix display wiring and driving, and how quickly the basics of an application can be put together.

This illustrates the programmer's or user's view of the application at its simplest, numbering the 12 logical cell light resources with labels H1 (hour-1) through H12. This is a very easy and logical representation for the programmer to work with and hides the complexity of matrix row and column lines.

4.8.2 Programmers view:

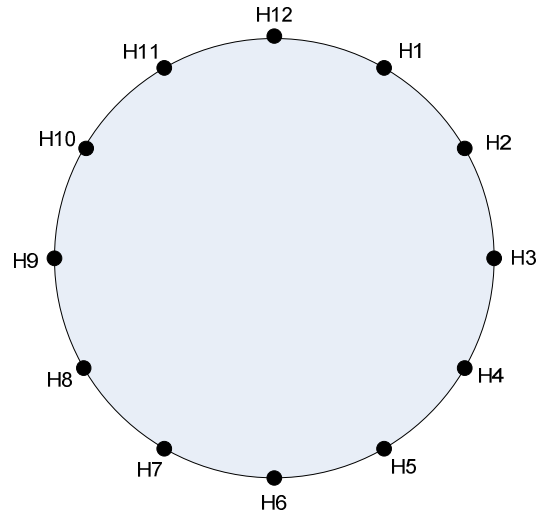


Figure 4.1 Clock face using RGB leds.

4.8.3 Sample Pattern File:

Setting up the display configuration:

```
#
# Configuration - default values are acceptable for all other parameters
RGBMode = 1           # RGB mode selected
```

Defining the LEDMAP:

```
#
# LEDMAP
# Logical cell  Disp  Row  Col  Disp  Row  Col  Disp  Row  Col  # Symbol
R      0        0    0    0    0    0    1    0    0    2    # Symbol H1
R      1        0    1    0    0    1    1    0    1    2    # Symbol H2
R      2        0    2    0    0    2    1    0    2    2    # Symbol H3
R      3        0    3    0    0    3    1    0    3    2    # Symbol H4
R      4        0    4    0    0    4    1    0    4    2    # Symbol H5
R      5        0    5    0    0    5    1    0    5    2    # Symbol H6
R      6        0    6    0    0    6    1    0    6    2    # Symbol H7
R      7        0    7    0    0    7    1    0    7    2    # Symbol H8
R      8        0    8    0    0    8    1    0    8    2    # Symbol H9
R      9        0    9    0    0    9    1    0    9    2    # Symbol H10
R     10        0   10    0    0   10    1    0   10    2    # Symbol H11
R     11        0   11    0    0   11    1    0   11    2    # Symbol H12
```

(# means a comment, R = RGB resource B = bi-colour resource M = mono resource)

Defining some display patterns:

```
#
P0 0 1          # easy way to clear display
0x00 0x00 0x00

#
# Hours 1 to 12 displaying white on the RGB leds:
P1 0 1          # one-o-clock - just one logical cell and clearing all others
0xFF 0xFF 0xFF

P2 1 1          # two-o-clock
0xFF 0xFF 0xFF

P3 2 1          # three-o-clock
0xFF 0xFF 0xFF

P4 3 1          # four-o-clock
0xFF 0xFF 0xFF

P5 4 1          # five-o-clock
0xFF 0xFF 0xFF

P6 5 1          # six-o-clock
0xFF 0xFF 0xFF

P7 6 1          # seven-o-clock
0xFF 0xFF 0xFF

P8 7 1          # eight-o-clock
0xFF 0xFF 0xFF

P9 8 1          # nine-o-clock
0xFF 0xFF 0xFF

P10 9 1         # ten-o-clock
0xFF 0xFF 0xFF

P11 10 1        # eleven-o-clock
0xFF 0xFF 0xFF

P12 11 1        # twelve-o-clock
0xFF 0xFF 0xFF
```

Defining a pattern sequence:

```
#
# Example Clock Sequence as sequence-0
# Idea here is to operate display like a sweeping 'seconds' display on a clock, making a complete sweep in 60
# seconds. Pattern-0 for 2-seconds must be repeated twice as the maximum time for any single pattern is
# 2.5 seconds.
#
S0 1 1000 0 2000 0 2000      # Turn on one-o-clock led for 1 second then blank display for 4 seconds
2 1000 0 2000 0 2000      # Turn on two-o-clock led
3 1000 0 2000 0 2000      # Turn on three-o-clock led
4 1000 0 2000 0 2000      # Turn on four-o-clock led
5 1000 0 2000 0 2000      # Turn on five-o-clock led
6 1000 0 2000 0 2000      # Turn on six-o-clock led
7 1000 0 2000 0 2000      # Turn on seven-o-clock led
8 1000 0 2000 0 2000      # Turn on eight-o-clock led
9 1000 0 2000 0 2000      # Turn on nine-o-clock led
10 1000 0 2000 0 2000      # Turn on ten-o-clock led
11 1000 0 2000 0 2000      # Turn on eleven-o-clock
12 1000 0 2000 0 2000      # Turn on twelve-o-clock led

( # means a comment)
```