LEGO USB Tower Interface Reference

____

LEGO Technology Center, LEGO System A/S

.

Table of Contents

1 Introduction

This document describes in detail the communication interface between the LEGO USB Tower (tower) and the host computer (host).

Communication between the tower and the host takes place through the USB bus. The tower firmware is compliant with version 1.1 of the USB specifications. The remainder of this document assumes the reader has a basic knowledge with the USB Specification 1.1, especially with chapter 9.

The USB communication characteristics of the tower are as follows:

- Two interrupt endpoints
- Vendor-specific device class (no HID or other USB class standards)

The tower has two interrupt endpoints: endpoint 1 is used for device-to-host communications, while endpoint 2 is used for host-to-device communications.

2 USB Tower Capabilities

The USB Tower is able to send and receive infrared signals (IR) and send Visual Light Link (VLL) signals.

The communication speed for:
1. VLL is fixed
2. Available IR speeds can be requested to the tower via the GET_CAPS command. It is possible to set different tx and rx speed.

The Tower has four USB configurations:
1. Low power (default)
2. High power
3. Low power, low bandwidth (8 ms polling interval)
4. High power, low bandwith (8 ms polling interval)

The Tower control panel on the Host/Windows system controls the active USB configuration. The Tower will always start in low power configuration and thereby being accepted by both low and high power ports on the PC system. Selecting one of the high-power configurations will allow the Tower to function in long range.

The user will see the green indicator LED switched on when the Tower:
1. Is powered and has been successfully configured from the host (ON for about 1 second)
2. Has received an IR transmission request from the host computer (ON for 1 or more seconds)
3. Is delivering IR-received data to the host computer (ON for 1 or more seconds)
4. Has detected an internal error (ON forever, until a reset occurs)

3 Tower control through USB Vendor requests

The Tower is controlled through specific USB vendor requests, defined by us and explained in detail in the following paragraphs.

All parameters set by vendor requests have default values in the Tower. The tower reverts to its default parameters every time it is reset by a vendor request or by unplugging/plug the Tower. Through the Windows control panel, it is possible to specify some parameter values that the driver automatically sets every time the tower is initialised by the application software.

In general, there is no need to set tower parameters unless there is a need for changing the default values.

All text returned on vendor requests are in Unicode format. The default word format is little-endian (PC/Windows standard). All USB-defined standard requests and any data/descriptors returned from standard requests use the little-endian format instead, no matter what's the data format defined in the tower.

3.1 List of Vendor Requests

All USB requests – standard and vendor – are identified by a setup packet containing:

- A request code (bmRequest)
- A word-sized parameter (wValue)
- A word-sized index (wIndex)
- A word-sized length of the expected reply (wLength)

See chapter 9 of the USB 1.1 specifications.

SET_PARM

Request Code

LTW_REQ_SET_PARM

Description

This command allows setting a number of different tower parameters.

Parameters

LOBYTE(wValue)	Parameter code to be set. It's one of the following codes (the default values are shown in **bold** face): LTW_PARM_MODE — Mode: VLL, **IR** or IRC LTW_PARM_RANGE — Range: Short, **Medium** or Long LTW_PARM_ERRDETECT — Filter spurious data from the IR receiver circuit: **On**/Off LTW_PARM_ERRSTATUS — Current error status of the tower (**no error**) LTW_PARM_ENDIAN — Vendor request word format: **Little Endian** (PC/Windows standard) or Big Endian (Apple/Motorola standard) LTW_PARM_ID_LED_MODE — Id LED control: **Firmware-controlled** or Host controlled LTW_PARM_ERROR_SIGNAL — Signal on internal error: **On**/Off
HIBYTE(wValue)	New value of the parameter
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_SET_PARM_REPLY)

Response

The response is a LTW_REQ_GET_SET_PARM_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bValue	Current value of the parameter specified in the request. If the request succeeded, this should contain the value specified by the host in the request parameters.

GET_PARM

Request Code

LTW_REQ_GET_PARM

Description

This command allows getting a number of different tower parameters.

Parameters

LOBYTE(wValue)	Code of the parameter value to be fetched. It's one of the following codes: LTW_PARM_MODE — Mode: VLL, IR or IRC LTW_PARM_RANGE — Range: Short, Medium or Long LTW_PARM_ERRDETECT — Filter spurious data from the IR receiver circuit: On/Off Remark: Doesn´t work in the IRC mode. LTW_PARM_ERRSTATUS — Current error status of the tower . LTW_PARM_ENDIAN — Vendor request word format: Little Endian (PC/Windows standard) or Big Endian (Apple/Motorola standard) LTW_PARM_ID_LED_MODE — Id LED control: Firmware-controlled or Host controlled LTW_PARM_ERROR_SIGNAL — Signal on internal error: On/Off
HIBYTE(wValue)	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_SET_PARM_REPLY)

Response

The response is a LTW_REQ_GET_SET_PARM_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bValue	Current value of the parameter specified in the request (values defined in VendReq.h)

SET_PARM_IRC

Request Code

LTW_REQ_SET_PARM_IRC

Description

This command allows setting a number of different tower parameters specific to the Infrared Remote Control protocol, a special protocol for LEGO cars.

NB: The parameters listed in the above sections (GET_PARM and SET_PARM) apply to the IRC mode as well if nothing else is stated. This special command only gathers the specific IRC commands.

Parameters

LOBYTE(wValue)	Parameter code to be set. It's one of the following codes (the default values are shown in **bold** face): LTW_PARM_IRC_PACKETSIZE Packet size, in bytes (default=**2**) LTW_PARM_IRC_DELAY_TX Transmit delay time between packets, in ms (**80**)
HIBYTE(wValue)	The parameter to be set
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_SET_PARM_IRC_REPLY)

Response

The response is a LTW_REQ_GET_SET_PARM_IRC_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bValue	Current value of the parameter specified in the request (values defined in VendReq.h)

GET_PARM_IRC

Request Code

LTW_REQ_GET_PARM_IRC

Description

This command allows getting a number of different tower parameters specific to the Infrared Remote Control protocol, a special protocol for LEGO cars.

Parameters

LOBYTE(wValue)	Code of the parameter value to be fetched. It's one of the following codes: LTW_PARM_IRC_PACKETSIZE Packet size, in bytes . LTW_PARM_IRC_DELAY_TX Transmit delay time between packets, in ms.
HIBYTE(wValue)	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_SET_PARM_IRC_REPLY)

Response

The response is a LTW_REQ_GET_SET_PARM_IRC_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bValue	Current value of the parameter specified in the request (values defined in VendReq.h)

FLUSH

Description

This requests flushes communication buffers according to the request parameter. Both firmware and driver buffers (if applicable) are flushed.

Purpose

1. Flush the transmitter buffer: Is used to flush the transmitter buffer and thereby stopping the transmission.
2. Flush the receiver buffer: Used to flush the receiver buffer before communication with the Pbrick starts.
3. Flush all buffers: Used before communication with the Pbrick starts.

Parameters

LOBYTE(wValue)	Flush constants, ORred together: LTW_TX_BUFFER — Flush the Transmit buffer LTW_RX_BUFFER — Flush the receive buffer
HIBYTE(wValue)	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_FLUSH_REPLY)

Response

The response is a LTW_REQ_FLUSH_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bBufferNo	Flushed buffers (same as wValue, above)

RESET

Description

Reset ports and all internal parameters to the default values.

Purpose

Used in case of an internal error to restart the Tower

Parameters

LOBYTE(wValue)	Not used; set to 0
HIBYTE(wValue)	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_RESET_REPLY)

Response

The response is a LTW_REQ_FLUSH_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used

GET_STAT

Description:

Get statistics from the IR receiver since the last RESET_STAT command.

Statistics:
1. Number of received bytes
2. Number of Overrun errors (new byte received before last received byte was read)
3. Number of Noise errors (noise in bit pattern, used to filter spurious 0xFF from the IR receiver)
4. Number of Framing errors

Purpose:

To analyze received signal quality and to filter spurious 0xFF.

Parameters

LOBYTE(wValue)	Not used; set to 0
HIBYTE(wValue)	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_STAT_REPLY)

Response

The response is a LTW_REQ_FLUSH_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used
wNoOfRxBytes	Number of received bytes
wOverrunErrorCount	# of times overrun error has happen
wNoiseCount	# of bytes with wrong bits
wFramingErrorCount	# of bytes with framing errors

RESET_STAT

Description

Reset statistics from the IR receiver (See GET_STAT).

Purpose

Reset statistics after the Host has read the statistics.

Parameters

LOBYTE(wValue)	Not used; set to 0
HIBYTE(wValue)	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_RESET_STATS_REPLY)

Response

The response is a LTW_REQ_RESET_STATS_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used

GET_POWER

Description

Get the Tower power configuration:
1. Low power (100 mA)
2. High power (500 mA)

The same information can be obtained in a standard way by querying the current device configuration (GET_CONFIGURATION standard request) and then retrieveing the configuration descriptor (GET_DESCRIPTOR(CONFIGURATION) standard request) to see the power requirements of the device.

Purpose

To determine which power configuration is active. The Tower is only able to send in short and medium range when the low power configuration is active.

Parameters

LOBYTE(wValue)	Not used; set to 0
HIBYTE(wValue)	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_POWER_REPLY)

Response

The response is a LTW_REQ_GET_POWER_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used
bPowerMode	Current power mode of the tower: LTW_POWER_LOW or LTW_POWER_HIGH

SET_LED

Description

Set the state of the ID or VLL led according to request parameters. In order to set the led state of the ID led, you must have set the led mode to software control (see SET_PARM (LED_MODE) request)

Purpose

Allow software control of the tower leds. This feature is used in the hardware production test to check that the leds are working.

Parameters

LOBYTE(wValue)	Led indicator:	
	LTW_LED_ID	The “green” led (normally used to indicate IR transmission)
	LTW_LED_VLL	The red led, normally used to carry out VLL communications
HIBYTE(wValue)	Color code:	
	LTW_LED_COLOR_ON	Switch on led.
	LTW_LED_COLOR_OFF	Switch off led.
	LTW_LED_COLOR_DEFAULT	Same as COLOR_ON.
	LTW_LED_COLOR_BLACK	Same as COLOR_OFF.
	LTW_LED_COLOR_GREEN LTW_LED_COLOR_YELLOW LTW_LED_COLOR_ORANGE LTW_LED_COLOR_RED	Same as ON on the current firmware. It could indicate a specific color if a multicolor led is used in the future.
wIndex	Not used; set to 0	
wLength	at least sizeof(LTW_REQ_GET_SET_LED_ REPLY)	

Response

The response is a LTW_REQ_GET_SET_LED_ REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used
bLedID	Same as LOBYTE(wValue)
bColor	Same as HIBYTE(wValue)

GET_LED

Description:

Get the current state of the ID or VLL led according to the request parameter:

Purpose:

Used to display the current state of the Tower leds in the Host application and in the hardware production test to check that the leds are working.

Parameters

LOBYTE(wValue)	Led indicator:	
	LTW_LED_ID	The "green" led (normally used to indicate IR transmission)
	LTW_LED_VLL	The red led, normally used to carry out VLL communications
HIBYTE(wValue)	Not used; set to 0	
wIndex	Not used; set to 0	
wLength	at least sizeof(LTW_REQ_GET_SET_LED_ REPLY)	

Response

The response is a LTW_REQ_GET_SET_LED_ REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.	
bErrCode	Error code (0 if success)	
bSpare	Not used	
bLedID	Same as LOBYTE(wValue)	
bColor	Color code:	
	LTW_LED_COLOR_ON	Led is switched on.
	LTW_LED_COLOR_OFF	Led is switched off.
	LTW_LED_COLOR_DEFAULT	Same as COLOR_ON
	LTW_LED_COLOR_BLACK	Same as COLOR_OFF
	LTW_LED_COLOR_GREEN LTW_LED_COLOR_YELLOW LTW_LED_COLOR_ORANGE LTW_LED_COLOR_RED	Same as ON on the current firmware. It could indicate a specific color if a multicolour led is used in the future.

SET_TX_SPEED / SET_RX_SPEED

Description

Set IR transmit/receive speed.

Different receiver speed:
If you want to receive at a different speed than the transmission, you have to send a SET_RX_SPEED after you send the SET_TX_SPEED.

Different carrier frequency:
By default, the frequency is set to 38 kHz when you choose 2400 baud and 76 kHz when you choose 4800 baud. If you want to send at a different carrier frequency than the default you have to send a SET_TX_CARRIER_FREQUENCY.

IF YOU NEED TO SET NON STANDARD CARRIER FREQUENCY/DUTY CYCLE VALUES, YOU WANT TO DO IT **AFTER** HAVING SET THE BAUD RATE, SINCE SETTING THE BAUD RATE AUTOMATICALLY REVERTS TO THE DEFAULT FREQUENCY/DUTY CYCLE VALUES FOR THAT SPEED.

Different carrier duty cycle:
By default the duty cycle is set according to the table in SET_TX_CARRIER_DUTY_CYCLE. If you want to send with a different carrier duty cycle than the default you have to send a SET_TX_CARRIER_DUTY_CYCLE.

Purpose

Used to switch between IR transmission baud rates: 2400, 4800 baud and others depending on the capability of the Tower.

Parameters

wValue	Speed indicator. One of the following constants: SPEED_COMM_BAUD_1200 SPEED_COMM_BAUD_2400 SPEED_COMM_BAUD_4800 SPEED_COMM_BAUD_9600 SPEED_COMM_BAUD_19200
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_SET_SPEED_REPLY)

Response

The response is a LTW_REQ_GET_SET_TX_SPEED_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used
wTxSpeed	Same as wValue

GET_TX_SPEED / GET_RX_SPEED

Description

Get current IR transmit/receive speed.

Purpose

Used to check the current IR transmission speed

Parameters

wValue	Not Used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_SET_SPEED_REPLY)

Response

The response is a LTW_REQ_GET_SET_TX_SPEED_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used
wTxSpeed	Speed indicator. One of the following constants: SPEED_COMM_BAUD_1200 SPEED_COMM_BAUD_2400 SPEED_COMM_BAUD_4800 SPEED_COMM_BAUD_9600 SPEED_COMM_BAUD_19200

GET_TX_STATE

Description

Get actual state of the transmitter:
1. Ready (transmitter buffer is empty, ready to receive more data to send)
2. Busy (busy sending)

Purpose

Used for communication flow control

Parameters

LOBYTE(wValue)	Not used; set to 0
HIBYTE(wValue)	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_TX_STATE_REPLY)

Response

The response is a LTW_REQ_GET_TX_STATE_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used
bTxState	CAPS_TX_STATE_READY or CAPS_TX_STATE_BUSY
bSpare2	Not used

SET_TX_CARRIER_FREQUENCY

Description

There are two default carrier frequencies connected to the transmission speed:
1. 38 kHz at 2400 baud
2. 76 kHz at 4800 baud

It is possible to get the supported frequency range for the device by asking for its capabilities (see the GET_CAPS request).

Purpose

Prepared for future changes

Parameters

LOBYTE(wValue)	New frequency, in kHz
HIBYTE(wValue)	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_SET_CARRIER_FREQUENCY_REPLY)

Response

The response is a LTW_REQ_GET_SET_CARRIER_FREQUENCY_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure
bErrCode	Error code (0 if success)
bSpare	Not used
bTxFrequency	Same as LOBYTE(wValue) above
bSpare2	Not used

GET_TX_CARRIER_FREQUENCY

Description

Get current transmission carrier frequency.

Purpose

Prepared for future changes.

Parameters

LOBYTE(wValue)	Not used; set to 0
HIBYTE(wValue)	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_SET_CARRIER_FREQUENCY_REPLY)

Response

The response is a LTW_REQ_GET_SET_CARRIER_FREQUENCY_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure
bErrCode	Error code (0 if success)
bSpare	Not used
bTxFrequency	Current transmission carrier frequency, in kHz
bSpare2	Not used

SET_TX_CARRIER_DUTY_CYCLE

Description

Set transmission carrier duty cycle for the specified transmission range. The firmware is able to set a different duty cycle for each transmission range (in fact, the duty cycle values **affects** the actual range of the device).

The carrier duty cycle will by default be set to the following values if you are using the default carrier frequencies 38 and 76 kHz:

Frequency/Range	Short	Medium	Long
38 kHz	8,00 µs	2,50 µs	8,00 µs
76 kHz	4,00 µs	1,25 µs	4,00 µs

<u>Be careful </u>in playing with these values, as they heavily affects the amount of power drawn from the USB bus – therefore wrong values could send the tower out-of-specs.

Purpose

Prepared for future changes.

Parameters

wValue	New duty cycle, in 1/100s of microsecond (µs)
wIndex	Range to which the new duty cycle is to be applied. One of the following: LTW_RANGE_SHORT LTW_RANGE_MEDIUM LTW_RANGE_LONG
wLength	at least sizeof(LTW_REQ_GET_SET_CARRIER_DUTY_CYCLE_REPLY)

Response

The response is a LTW_REQ_GET_SET_CARRIER_DUTY_CYCLE_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used
bTxRange	Range (same as wIndex, above)
bSpare2	Not used
wTxDutyCycle	New duty cycle value (same as wValue, above)

GET_TX_CARRIER_DUTY_CYCLE

Description

Get transmission carrier duty cycle for the given range

Purpose

Prepared for future changes.

Parameters

wValue	Not used; set to 0
wIndex	Range for which we are querying the duty cycle. One of the following: LTW_RANGE_SHORT LTW_RANGE_MEDIUM LTW_RANGE_LONG
wLength	at least sizeof(LTW_REQ_GET_SET_CARRIER_DUTY_CYCLE_REPLY)

Response

The response is a LTW_REQ_GET_SET_CARRIER_DUTY_CYCLE_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used
bTxRange	Range (same as wIndex, above)
bSpare2	Not used
wTxDutyCycle	Current duty cycle value (same as wValue, above)

GET_CAPS

Description

Get a list of the Tower capabilities according to the requested link type.

Purpose:

Used by the Host driver to get the capabilities of the Tower. In that way the Tower can be compatible with future versions of the Host driver that supports new Towers and other communication devices.

Parameters

wValue	Link type for which capabilities are requested. One of the following: LTW_CAPS_IR LTW_CAPS_VLL LTW_CAPS_IRC LTW_CAPS_RADIO (not supported currently)
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_CAPS_REPLY)

Response

The response is a LTW_REQ_GET_CAPS_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used
bCommDirections	CAPS_COMM_DIRECTIONS_TRANSMIT and/or CAPS_COMM_DIRECTIONS_RECEIVE (they can be ORred together)
bCommRange	Ranges supported (can be ORred together): LTW_CAPS_RANGE_SHORT LTW_CAPS_RANGE_MEDIUM LTW_CAPS_RANGE_LONG
wCommTransmitRate	Supported Transmit speeds: CAPS_COMM_BAUD_1200 CAPS_COMM_BAUD_2400 CAPS_COMM_BAUD_4800 CAPS_COMM_BAUD_9600 CAPS_COMM_BAUD_19200
wCommReceiveRate	Supported receive speed (same constants as above)
bCommTransmitMinFrequency bCommTransmitMaxFrequency	Minimum and maximum carrier frequency
wCommTransmitMinDutyCycle wCommTransmitMaxDutyCycle	Minimum and maximum duty cycle values (applicable to all ranges)
bNoOfBytesInUartTxBuffer	Maximum size of the internal UART Transmit buffer
bNoOfBytesInUartRxBuffer	Maximum size of the internal UART Transmit buffer

GET_VERSION

Description

Get firmware version information.

Purpose

Version control.

Parameters

wValue	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_VERSION_REPLY)

Response

The response is a LTW_REQ_GET_VERSION_REPLY structure:

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used
bMajorVersion	Major version number
bMinorVersion	Minor version number
wBuildNo	Build number

GET_COPYRIGHT

Description

Get copyright information.

Purpose

Used by the Host driver to identify the firmware as LEGO firmware.

Parameters

wValue	Not used; set to 0
wIndex	Not used; set to 0
wLength	at least sizeof(LTW_REQ_GET_ COPYRIGHT _REPLY) **plus** a suitable size for the copyright string.

Response

The response is a LTW_REQ_GET_COPYRIGHT_REPLY structure, followed by a non-NULL terminated UNICODE string containing copyright information.
If the size specified in the request is not enough to hold the whole copyright string, the firmware just replies with the requested amount of data (so a truncated string is returned).

wNoOfBytes	No. of bytes contained in the structure.
bErrCode	Error code (0 if success)
bSpare	Not used

3.2 Firmware Vendor Request Reply Error Codes

The following table lists all the error that the firmware can return in the bErrCode field of the various vendor requests:

Internal Error Code	**Description**
LTW_REQERR_SUCCESS	Request succeded
LTW_REQERR_BADPARM	Bad vendparameter and/or value
LTW_REQERR_BUSY	Tower is busy
LTW_REQERR_NOPOWER	Not enough power to carry out the requested operation
LTW_REQERR_WRONGMODE	Not in the right mode to execute this request
LTW_INTERNAL_ERROR	Internal error in the Tower
LTW_REQERR_BADREQUEST	Bad request

3.3 Firmware Internal Error Codes

The following table lists all the internal errors that the firmware can return on the GET_PARM (LTW_PARM_ERRSTATUS) vendor requests. The internal error code is only set when a serious error occur.

Error Code	**Description**
LTW_NO_ERROR	No error
Function / parameter error	
LTW_NO_SUPPORT	No support for this function
LTW_ERR_VEND_REQ	Invalid vendor request
USB errors	
LTW_ERR_USB	Unspecified USB problem
LTW_ERR_USB_FUNC	Call to wrong USB function
LTW_ERR_USB_DATA	No or invalid data received
LTW_ERR_USB_ENDPOINT	Wrong endpoint
LTW_ERR_USB_PACKETSIZE	Invalid packetsize
LTW_ERR_USB_TX_EP0	Error on sending on endpoint 0
LTW_ERR_USB_EP0_PACKETSIZE	Tried to send more than MAXPACKETSIZE on endpoint 0
LTW_ERR_USB_EP1_PACKETSIZE	Tried to send more than MAXPACKETSIZE on endpoint 1
LTW_ERR_USB_VEND_BUFF_FULL	Answer buffer full
UART errors	
LTW_ERR_UART	SW UART problems
LTW_ERR_UART_MODE	Invalid UART mode
LTW_ERR_UART_TX_STATE	Invalid TX state
Tower errors	
LTW_ERR_TOWER	Tower out of order 0
LTW_ERR_UNDEFINED	Undefined error

3.4 Constant Definitions

This section contains the numeric definitions of the constants mentioned in this document. This information is included here for completeness only. For actual coding, you should refer to the *VendReq.h* header file (provided by LEGO), which also contains types and structure definitions.

```
////////////////////////////////////////////////////////////////////////////////////////////
//                        VENDOR REQUESTS TYPES
////////////////////////////////////////////////////////////////////////////////////////////

// USB Vendor requests
//
#define LTW_REQ_GET_PARM                        0x01 // Get parameter for standard IR mode
#define LTW_REQ_SET_PARM                        0x02 // Set parameter for standard IR mode
#define LTW_REQ_FLUSH                           0x03 // Flush UART buffers
#define LTW_REQ_RESET                           0x04 // Performing a user reset
#define LTW_REQ_GET_STAT                        0x05 // Get statistics on IR receiver
#define LTW_REQ_GET_POWER                       0x06 // Get current powermode
#define LTW_REQ_GET_LED                         0x08 // Get led state and color
#define LTW_REQ_SET_LED                         0x09 // Switch on/off led + color of led
#define LTW_REQ_RESET_STAT                      0x10 // Reset statictics on IR receiver
#define LTW_REQ_GET_PARM_IRC                    0x11 // Get parameter for IRC mode
#define LTW_REQ_SET_PARM_IRC                    0x12 // Set parameter for IRC mode
#define LTW_REQ_GET_TX_SPEED                    0xEE // Get IR transmission speed
#define LTW_REQ_SET_TX_SPEED                    0xEF // Set IR transmission speed
#define LTW_REQ_GET_RX_SPEED                    0xF0 // Get IR receiving speed
#define LTW_REQ_SET_RX_SPEED                    0xF1 // Set IR receiving speed
#define LTW_REQ_GET_TX_STATE                    0xF2 // Get state of the transmitter
#define LTW_REQ_GET_TX_CARRIER_FREQUENCY        0xF3 // Set transmission carrier frequency
#define LTW_REQ_SET_TX_CARRIER_FREQUENCY        0xF4 // Set transmission carrier frequency
#define LTW_REQ_GET_TX_CARRIER_DUTY_CYCLE       0xF5 // Set transmission carrier dutycycle
#define LTW_REQ_SET_TX_CARRIER_DUTY_CYCLE       0xF6 // Set transmission carrier dutycycle
#define LTW_REQ_GET_CAPS                        0xFC // Get capabilities
#define LTW_REQ_GET_VERSION                     0xFD // Get version information
#define LTW_REQ_GET_COPYRIGHT                   0xFE // Get copyright information
#define LTW_REQ_GET_CREDITS                     0xFF // Credits list

//       Request return codes (bLtwErrCode in the reply format)
//
#define LTW_REQERR_SUCCESS                      0x00 // Request succeded
#define LTW_REQERR_BADPARM                      0x01 // Bad vendparameter and/or value
#define LTW_REQERR_BUSY                         0x02 // Tower is busy
#define LTW_REQERR_NOPOWER                      0x03 // Not enough power (current)
#define LTW_REQERR_WRONGMODE                    0x04 // Not in the right mode
#define LTW_INTERNAL_ERROR                      0xFE // Internal error in the Tower
#define LTW_REQERR_BADREQUEST                   0xFF // Bad request

// Bad request reply format
//
typedef struct LTW_REQ_BAD_REPLY
        { Word wNoOfBytes;                      // Number of bytes in the reply
          Byte bErrCode;                        // Request return code
          Byte bSpare;                          // Structure padding byte
} LTW_REQ_BAD_REPLY;


////////////////////////////////////////////////////////////////////////////////////////////
//                        VENDOR REQUEST: GET_PARM and SET_PARM
////////////////////////////////////////////////////////////////////////////////////////////

//       -------------------------------------------------------------
//       Parameter (1 byte) and value (1 byte) for GET_PARM and SET_PARM request
//
//       The parameter code goes in the low-order byte of wvalue. the value (argument)
//       for the request goes into the high-order byte. windex is reserved for future
//       use.
//

#define LTW_PARM_MODE          0x01    // Tower mode:
        #define LTW_MODE_VLL           0x01    // - Send VLL
        #define LTW_MODE_IR            0x02    // - Send/Receive IR
        #define LTW_MODE_IRC           0x04    // - Send/Receive IR (Technic protocol)
        #define LTW_MODE_RADIO         0x08    // Send/Receive Radio signals

#define LTW_PARM_RANGE         0x02    // Transmission range:
        #define LTW_RANGE_SHORT        0x01    // - Short
        #define LTW_RANGE_MEDIUM       0x02    // - Medium
        #define LTW_RANGE_LONG          0x03    // - Long

#define LTW_PARM_ERRDETECT      0x03    // Error detection on IR receiver:
        #define LTW_ERRDETECT_ON        0x01    // - on
        #define LTW_ERRDETECT_OFF       0x02    // - off

#define LTW_PARM_ERRSTATUS      0x04    // Current internal errorcode:
        #define LTW_NO_ERROR            0       // - No error
        #define LTW_NO_SUPPORT          1       // - No support for this function
        #define LTW_ERR_VEND_REQ        10      // - Invalid vendor request

        // USB errors
        #define LTW_ERR_USB                     100     // - Unspecified USB problem
        #define LTW_ERR_USB_FUNC                101     // - Call to wrong USB function
        #define LTW_ERR_USB_DATA                102     // - No or invalid data received
        #define LTW_ERR_USB_ENDPOINT            103     // - Wrong endpoint
        #define LTW_ERR_USB_PACKETSIZE          104     // - Invalid packetsize
        #define LTW_ERR_USB_TX_EP0              105     // - Error on sending on endpoint 0
        #define LTW_ERR_USB_EP0_PACKETSIZE      106     // - Wrong packet size on EP0
        #define LTW_ERR_USB_EP1_PACKETSIZE      107     // - Wrong packet size on EP1
        #define LTW_ERR_USB_VEND_BUFF_FULL      108     // - Vendor answer buffer full
        #define LTW_ERR_USB_TX_EP1              109     // - Send error on endpoint 1

        // UART errors
        #define LTW_ERR_UART                    200     // - UART problems
        #define LTW_ERR_UART_MODE               201     // - Invalid UART mode
        #define LTW_ERR_UART_TX_STATE           210     // - Invalid TX state
        #define LTW_ERR_UART_LOW_POWER          211     // - Tried to send in long range when
                                                        // configured as a low power device

        // Tower errors
        #define LTW_ERR_TOWER                   240     // - Tower out of order
        #define LTW_ERR_UNDEFINED               255     // - undefined error


#define LTW_PARM_ENDIAN                         0x97    // Endian (word-format):
        #define LTW_ENDIAN_LITTLE               0x01    // - Little endian
        #define LTW_ENDIAN_BIG                  0x02    // - Big endian

#define LTW_PARM_ID_LED_MODE  0x98              // Indicator LED control mode:
        #define LTW_ID_LED_HW_CTRL              0x01    // - LED controlled by firmware
        #define LTW_ID_LED_SW_CTRL              0x02    // - LED controlled by the host

#define LTW_PARM_ERROR_SIGNAL 0x99              // Signal on internal error:
        #define LTW_ID_LED_ON                   0x01    // - Indicator LED will be switched
                                                        // on when a serious error occurs
        #define LTW_ID_LED_OFF                  0x02    // - Indicator LED will NOT be
                                                        // on when a serious error occurs

// ----------------------------------------------------------------
// Reply format
//
//      All vendor request replies from the tower start with the following
//      four bytes:
//

typedef struct LTW_REQ_REPLY_HEADER
        { Word wNoOfBytes;                              // Number of bytes in the reply
          Byte bErrCode;                                // Request return code
          Byte bValue;                                  // Request return value
} LTW_REQ_REPLY_HEADER;

typedef LTW_REQ_REPLY_HEADER LTW_REQ_GET_SET_PARM_REPLY;


//////////////////////////////////////////////////////////////////////////////////////////
//                      VENDOR REQUEST: SET_PARM_IRC and GET_PARM_IRC
//////////////////////////////////////////////////////////////////////////////////////////

#define LTW_PARM_IRC_PACKETSIZE          0x01   // Packet size, in bytes (default = 2)
#define LTW_PARM_IRC_TIMEFRAME          0x02    // Time frame for packet sync, in ms(def=80)
```

```
typedef LTW_REQ_REPLY_HEADER LTW_REQ_GET_SET_PARM_IRC_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                       VENDOR REQUEST: FLUSH
/////////////////////////////////////////////////////////////////////////////////////////

//      -------------------------------------------------------------
//      Combined parameter and value in 2 bytes for FLUSH request
//
//  Format: 1. byte = Buffer no. as they are defined under SET_LED request, 2. byte = 00
//      -------------------------------------------------------------
//
//      Parameter: Buffer no.  This goes into the low-order byte of the wvalue field
//      of the request's setup packet
//
#define LTW_TX_BUFFER         0x01           // Transmission buffer
#define LTW_RX_BUFFER         0x02           // Receiver buffer
#define LTW_ALL_BUFFERS       0x03           // All buffers


// -----------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_FLUSH_REPLY
       { Word wNoOfBytes;                           // Number of bytes in the reply
         Byte bErrCode;                             // Request return code
         Byte bBufferno;                            // Flushed bufferno.
} LTW_REQ_FLUSH_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                       VENDOR REQUEST: RESET
/////////////////////////////////////////////////////////////////////////////////////////

//      No request parameter or value
//

// -----------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_RESET_REPLY
       { Word wNoOfBytes;            // Number of bytes in the reply
         Byte bErrCode;              // Request return code
         Byte bSpare;                // Spare byte to obtain an even number of bytes
} LTW_REQ_RESET_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                       VENDOR REQUEST: GET_STAT
/////////////////////////////////////////////////////////////////////////////////////////

//      No request parameter or value
//

// -----------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_STAT_REPLY
       { Word  wNoOfBytes;           // Number of bytes in the reply
         Byte  bErrCode;             // Request return code
         Byte  bSpare;               // Spare byte to obtain an even number of bytes

         Word  wNoOfRxBytes;         // Number of received bytes
         Word  wOverrunErrorCount;   // # of times overrun error has happen
         Word  wNoiseCount;          // # of bytes with wrong bits
         Word  wFramingErrorCount;   // # of bytes with framing errors
} LTW_REQ_GET_STAT_REPLY;
```

```
/////////////////////////////////////////////////////////////////////////////////////////
//                      VENDOR REQUESTS: GET_POWER
/////////////////////////////////////////////////////////////////////////////////////////

//      No request parameter or value
//

// ---------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_POWER_REPLY
        { Word wNoOfBytes;     // Number of bytes in the reply
          Byte bErrCode;       // Request return code
          Byte bSpare;         // Spare byte to obtain an even number of bytes

          Byte bPower;         // Powermode (POWER_LOW or POWER_HIGH)
#define LTW_POWER_LOW          0x01    // - Tower is set to low power device
#define LTW_POWER_HIGH         0x02    // - Tower is set to high power device
          Byte bSpare2;

} LTW_REQ_GET_POWER_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                      VENDOR REQUEST: SET_LED and GET_LED
/////////////////////////////////////////////////////////////////////////////////////////

// ---------------------------------------------------------------
// Parameter (1 byte) and value (1 byte) for SET_LED request
//
// Parameter: Led no.  This goes into the low-order byte of the wvalue field of the
//              request's setup packet
//
#define LTW_LED_ID                                    0x01 // ID LED
#define LTW_LED_VLL                                   0x02 // VLL LED

//      Value:     Led color   THis goes into the high-order byte of the wvalue field of
//                             the request's setup packet
//
//      COLOR identifiers (low order bytes in the value field of the request)
//      if a color not supported by the hardware is used, the firmware approximates
//      to the nearest one. The current USB Tower version supports only Green color.
//
#define LTW_LED_COLOR_BLACK           0x00 // Black = same as off
#define LTW_LED_COLOR_GREEN           0x01 // Green
#define LTW_LED_COLOR_YELLOW          0x02 // Yellow
#define LTW_LED_COLOR_ORANGE          0x04 // Orange
#define LTW_LED_COLOR_RED             0x08 // Red
//
//      more color codes can be added here...
//
#define LTW_LED_COLOR_DEFAULT         0xFF           // Whatever color is the default

#define LTW_LED_COLOR_ON              LTW_LED_COLOR_DEFAULT         // generic ON
#define LTW_LED_COLOR_OFF             LTW_LED_COLOR_BLACK           // generic OFF


// ---------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_SET_LED_REPLY
        { Word wNoOfBytes;            // Number of bytes in the reply
          Byte bErrCode;              // Request return code
          Byte bSpare;                // Spare byte to obtain an even number of bytes

          Byte bLedId;                // See parameter
          Byte bColor;                // See value
} LTW_REQ_GET_SET_LED_REPLY;
```

```
/////////////////////////////////////////////////////////////////////////////////////////
//                      VENDOR REQUEST: RESET_STAT
/////////////////////////////////////////////////////////////////////////////////////////

//      No request parameter or value
//

//      -------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_RESET_STAT_REPLY
        { Word wNoOfBytes;     // Number of bytes in the reply
          Byte bErrCode;       // Request return code
          Byte bSpare;         // Used only for having a even number of bytes in the reply
} LTW_REQ_RESET_STAT_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                      VENDOR REQUEST: GET_PARM_IRC and SET_PARM_IRC
/////////////////////////////////////////////////////////////////////////////////////////

// -------------------------------------------------------------
// Parameter (1 byte) and value (1 byte) for GET_PARM_IRC and SET_PARM_IRC request
//
// The parameter code goes in the low-order byte of wvalue. the value (argument)
// for the request goes into the high-order byte. windex is reserved for future
// use.
//

#define LTW_PARM_IRC_PACKETSIZE 0x01  // IRC packetsize
// Value: Packetsize in bytes

#define LTW_PARM_IRC_DELAY_TX 0x02    // Delay between packets to send
// Value: Transmit delay time between packets in ms.

// --------------------------------------------------------------
// Reply format
//
typedef LTW_REQ_REPLY_HEADER LTW_REQ_GET_SET_PARM_IRC_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                      VENDOR REQUESTS: GET and SET_TX_SPEED
/////////////////////////////////////////////////////////////////////////////////////////

// -------------------------------------------------------------
// Combined parameter and value in 2 bytes for GET and SET_TX_SPEED request
//
#define SPEED_COMM_BAUD_1200                 CAPS_COMM_BAUD_1200
#define SPEED_COMM_BAUD_2400                 CAPS_COMM_BAUD_2400
#define SPEED_COMM_BAUD_4800                 CAPS_COMM_BAUD_4800
#define SPEED_COMM_BAUD_9600                 CAPS_COMM_BAUD_9600
#define SPEED_COMM_BAUD_19200                CAPS_COMM_BAUD_19200

// -------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_SET_TX_SPEED_REPLY
        { Word wNoOfBytes;     // Number of bytes in the reply
          Byte bErrCode;       // Request return code
          Byte bSpare;         // Spare byte to obtain an even number of bytes

          Word wTxSpeed;       // Transmission speed
} LTW_REQ_GET_SET_TX_SPEED_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                      VENDOR REQUESTS: GET and SET_RX_SPEED
/////////////////////////////////////////////////////////////////////////////////////////

//      -------------------------------------------------------------
//      Combined parameter and value in 2 bytes for GET and SET_RX_SPEED request
//
//  Same definitions as SET_TX_SPEED !

// -------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_SET_RX_SPEED_REPLY
        { Word wNoOfBytes;     // Number of bytes in the reply
          Byte bErrCode;       // Request return code
          Byte bSpare;         // Spare byte to obtain an even number of bytes

          Word wRxSpeed;       // Receiving speed
} LTW_REQ_GET_SET_RX_SPEED_REPLY;

/////////////////////////////////////////////////////////////////////////////////////////
//                       VENDOR REQUESTS: GET_TX_STATE
/////////////////////////////////////////////////////////////////////////////////////////

//      No request parameter or value
//

// -------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_TX_STATE_REPLY
        { Word wNoOfBytes;     // Number of bytes in the reply
          Byte bErrCode;       // Request return code
          Byte bSpare;         // Spare byte to obtain an even number of bytes

          Byte bTxState;       // Transmitter states:
#define CAPS_TX_STATE_READY           0x01    // - Transmitter buffer empty, Ready to
                                              //   receive more data to send
#define CAPS_TX_STATE_BUSY            0x02    // - Busy sending data

          Byte bSpare2;               // Spare byte to obtain an even number of bytes
} LTW_REQ_GET_TX_STATE_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                       VENDOR REQUESTS: SET_TX_CARRIER_FREQUENCY
/////////////////////////////////////////////////////////////////////////////////////////

//      -------------------------------------------------------------
//      Combined parameter and value in 2 bytes for SET_TX_CARRIER_FREQUENCY request
//
//  Parameter (1 byte):  frequency e.g.: 38 kHz = 38
//  Value     (1 bytes): 00

//      The parameter code goes in the low-order byte of wvalue. the value (argument)
//      for the request goes into the high-order byte. windex is reserved for future
//      use.

//      -------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_SET_TX_CARRIER_FREQUENCY_REPLY
        { Word wNoOfBytes;     // Number of bytes in the reply
          Byte bErrCode;       // Request return code
          Byte bSpare;         // Spare byte to obtain an even number of bytes

          Byte bTxFrequency;   // Transmission frequency
          Byte bSpare2;        // Spare byte to obtain an even number of bytes
} LTW_REQ_GET_SET_TX_CARRIER_FREQUENCY_REPLY;

/////////////////////////////////////////////////////////////////////////////////////////
//                       VENDOR REQUESTS: SET_TX_CARRIER_DUTY_CYCLE
/////////////////////////////////////////////////////////////////////////////////////////

//      -------------------------------------------------------------
//      Parameter (1 byte) and value (1 byte) for SET_TX_CARRIER_DUTY_CYCLE request
```

```
//
//      Parameter: Range This goes into the low-order byte of the wvalue field of the
//                       request's setup packet
//
//     See LTW_PARM_RANGE for definitions.
//

//      Value:     Duty cycle  This goes into the high-order byte of the wvalue field of
//                  the request's setup packet
//
//        Duty cycle in us multiplied by 10. E.g. 1,5 us = 15

// -------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_SET_TX_CARRIER_DUTY_CYCLE_REPLY
        { Word wNoOfBytes;            // Number of bytes in the reply
          Byte bErrCode;              // Request return code
          Byte bSpare;                // Spare byte to obtain an even number of bytes

          Byte bTxRange;              // Transmission range
          Byte bSpare2;

          Word wTxDutyCycle;   // Transmission dutycycle in this range in 1/10 of a ms.
} LTW_REQ_GET_SET_TX_CARRIER_DUTY_CYCLE_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                      VENDOR REQUEST: GET_CAPS
/////////////////////////////////////////////////////////////////////////////////////////

//      -------------------------------------------------------------
//      Combined parameter and value in 2 bytes for GET_CAPS request
//

//      Supported communication modes
#define LTW_CAPS_VLL                            LTW_MODE_VLL
#define LTW_CAPS_IR                             LTW_MODE_IR
#define LTW_CAPS_IRC                            LTW_MODE_IRC
#define LTW_CAPS_RADIO                          LTW_MODE_RADIO

// -------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_CAPS_REPLY
        { Word wNoOfBytes;            // Number of bytes in the reply
          Byte bErrCode;              // Request return code
          Byte bSpare;                // Spare byte to obtain even number of bytes

          Byte bCommDirections;       // Communication directions:
#define CAPS_COMM_DIRECTION_TRANSMIT  0x01    // - Transmit
#define CAPS_COMM_DIRECTION_RECEIVE           0x02    // - Receive
#define CAPS_COMM_DIRECTION_BOTH              0x03    // - Both transmit and receive
          Byte bCommRange;            // Communication range:
#define CAPS_COMM_RANGE_SHORT                 0x01    // - Short
#define CAPS_COMM_RANGE_MEDIUM                0x02    // - Medium
#define CAPS_COMM_RANGE_LONG                  0x04    // - Long
#define CAPS_COMM_RANGE_ALL                   0x07    // - short, medium and long range
          Word wCommTransmitRate;     // Communication transmit rate
                                      // (same definition as receive rate)
          Word wCommReceiveRate;      // Communication receive rate:
#define CAPS_COMM_BAUD_1200                   0x0004  // - 1200 baud
#define CAPS_COMM_BAUD_2400                   0x0008  // - 2400 baud
#define CAPS_COMM_BAUD_4800                   0x0010  // - 4800 baud
#define CAPS_COMM_BAUD_9600                   0x0020  // - 9600 baud
#define CAPS_COMM_BAUD_19200                  0x0040  // - 19200 baud
          Byte bCommTransmitMinFrequency;     // Minimum frequency between 30 and 99 kHz
                                              // E.g. 30 kHz = 30
          Byte bCommTransmitMaxFrequency;     // Maximum frequency between 30 and 99 kHz.
                                              // E.g. 30 kHz = 30
          Word wCommTransmitMinDutyCycle;     // Minimum duty cycle in us multiplied by
```

```
                                             // 100. E.g.  1,25 us = 125
         Word wCommTransmitMaxDutyCycle;     // Maximum duty cycle in us multiplied by
                                             // 100. E.g. 18,9  us = 1890
         Byte bNoOfBytesInUartTxBuffer;      // Number of bytes defined in the
                                             // transmitter buffer
         Byte bNoOfBytesInUartRxBuffer;      // Number of bytes defined in the
                                             // receiver buffer
} LTW_REQ_GET_CAPS_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                      VENDOR REQUEST: GET_VERSION
/////////////////////////////////////////////////////////////////////////////////////////

//      No request parameter or value
//

// -------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_VERSION_REPLY
       { Word wNoOfBytes;            // Number of bytes in the reply
         Byte bErrCode;              // Request return code
         Byte bSpare;                // Spare byte to obtain an even number of bytes

         Byte bMajorVersion;         // Major version e.g. 1
         Byte bMinorVersion;         // Minor version e.g. 01
         Word wBuildNo;              // Build no. e.g. 0112
} LTW_REQ_GET_VERSION_REPLY;


/////////////////////////////////////////////////////////////////////////////////////////
//                      VENDOR REQUEST: GET_COPYRIGHT
/////////////////////////////////////////////////////////////////////////////////////////

//      No request parameter or value
//

//      -------------------------------------------------------------
// Reply format
//
typedef struct LTW_REQ_GET_COPYRIGHT_REPLY
       { Word wNoOfBytes;            // Number of bytes in the reply
         Byte bErrCode;              // Request return code
         Byte bSpare;                // Spare byte to obtain an even number of bytes

         // Copyright string: placed in the Tower ROM -- NOT null-terminated
} LTW_REQ_GET_COPYRIGHT_REPLY;

#endif
```