Comprehensive User's Guide

XLON® Serial Gateway

Comprehensive User's Guide

Revision 1.11

Comprehensive User's Guide

LIST OF CONTENT

1 DB9 Connector Pinout Description

The signal assignments for the DB9 connector are shown as follows:

Pin	Name	Description
1	DCD	Data Carrier Detect
2	RX-	Receive Data (-)
3	TX-	Transmit Data (-)
4	DTR	Data Terminal Ready
5	GND	Signal Ground
6	DSR	Data Set Ready
7	TX+RTS	Transmit Data (+) or Request to Send
8	RX+CTS	Receive Data (+) or Clear to Send
9	RI	Ring Indicator

2 RS232 Cabling Connections

The DB9 connector of the XLON® LSG has the pinout of a Data Terminal Equipment (DTE) for use with RS232 applications. Therefore you need a null modem cable to connect the XLON® LSG to another DTE.

2.1 XLON® LSG to DTE

To connect the XLON® LSG RS232 ports to another DTE like a PC, we recommend the use of DTE to DTE cable configuration (null modem cable), which is shown as follows:

XLON® LSG	DTE
TX-	RxD
RX-	TxD
RX+CTS	RTS
TX+RTS	CTS
DSR	DTR
DTR	DSR

GND	GND

2.2 EXLON® LSG to DCE (Modem)

To connect the EXLON® LSG RS232 ports to a Data Communications Equipment (DCE) like a modem, we recommend the use of DTE to DCE cable configuration (1-to-1-cable), which is shown as follows:

EXLON® LSG	DCE
TX-	TxD
RX-	RxD
RX+CTS	CTS
TX+RTS	RTS
DSR	DSR
DTR	DTR
DCD	DCD
RI	RI
GND	GND

3 RS422 Cabling Connections

To connect the RS422 ports of the EXLON® LSG to other DTE devices, we recommend the use of the DTE to DTE configuration, which is shown as follows:

EXLON® LSG	DTE
RX-	TX-
TX-	RX-
RX+CTS	TX+
TX+RTS	RX+
GND	GND

4 RS485 Cabling Connections

The RS485 communication (half-duplex) is based on the cable sharing method, which is connected as following :

XLON® LSG	Other 485 device
RX- and TX-	RX- and TX-
RX+CTS and TX+RTS	RX+ and TX+
GND	GND

5 Power Supply

Once the XLON® LSG is physically attached to the desired channel, power must be supplied via one of the two power input connectors. For powering the device one could use the Mini DC Power Jack (Ø 1.3mm Inner Pin Diameter) or supply the device via the USB port.

The power supply should be within 4,75 – 5,25 VDC @ 100 mA.

5.1 Via Mini DC Power Jack

The device can be connected via a T-cable from the keyboard or mouse connector of a PC or Laptop to the Mini DC Power Jack. The Inner Pin Diameter of the jack is 1.3 mm. The center pin is +5 VDC, the outer pin is GND (ground). There is also a schematic symbol on the housing, which shows the polarity of the Mini DC Power Jack.
If you want to supply the device via an external plug-in power supply, we recommend one with regulated 5V ± 3% tolerance maximum. The plug-in power supply must have all the necessary approvals like CE, FCC or UL.

5.2 Via USB port

The device can also be powered via a USB cable which is connected to a PC or Laptop.

6 LON Network Connector

The XLON® LSG is available with an integrated Free Topology transceiver (FTT-10A) or an electrically isolated (1.5kV) RS485 transceiver.

6.1 Free Topology Network

There is a 2-pin Weidmueller clamp in use with the FTT network. Since FTT networks are polarity insensitive, there is no need to take care of correct wiring to the clamp.

6.2 RS485 Network

There is a 8-pin RJ45 connector when using a RS485 network. There are two pin-out versions of the RJ45 connector available:

Ordering Information: DH-199-500-xx-485

Where xx could be: RJ or GS – The standard version is -RJ

RJ45 Pin#	Option –RJ	Option -GS
1	Signal 485 A	Signal 485 A
2	Signal 485 B	Signal 485 B
3	N.C.	N.C.
4	GROUND	GROUND
5	N.C.	N.C.
6	N.C.	N.C.
7	N.C.	GROUND
8	N.C.	N.C.

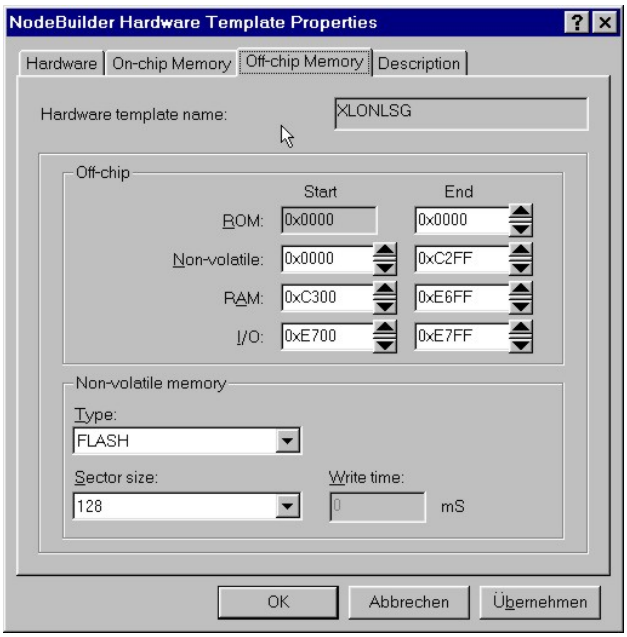
Pin 1 is on the left side of the RJ45 connector, when looking at the front side of the connector.

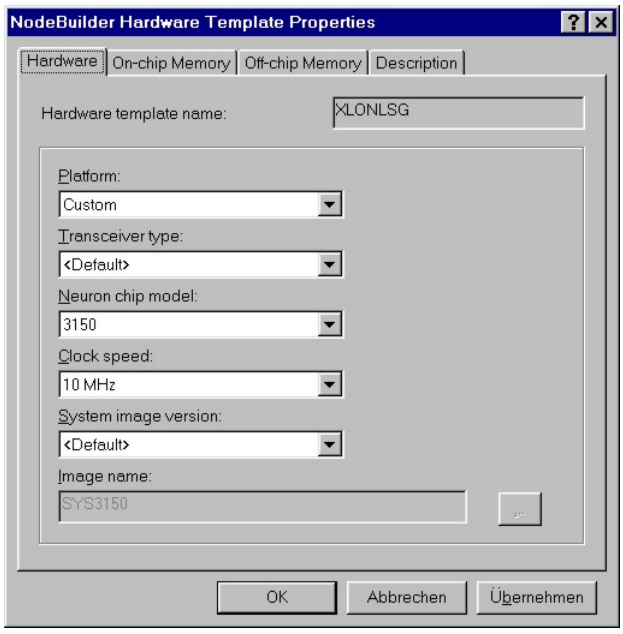
7 Software Programming with NodeBuilder 3 Tool

The XLON® LSG is a programmable serial gateway, which is used to connect non-LonWorks applications or –devices to a LON network. All standard Neuron Chip firmware features described in the *Neuron C Programmer's Guide* are available to the application programmer. For example there is no support for more than 62 bound network variables on such a node. The serial gateway will also not be usable as a network interface for LonMaker or LNS-based applications, although it can still be installed and managed by such an application.

7.1 Hardware Resources

The Hardware Resources of the XLON® LSG have to be defined within the Device Template of the NodeBuilder 3 Development Tool. The Device Template file named *XLONLSG.NbHwt* comes with the XLON® LSG and must be copied into the *Templates* Directory of NodeBuilder 3.





In the above screenshots you can see all the necessary NodeBuilder Hardware Template settings that have to be done for the XLON® LSG.

7.2 Software Resources

The EXLON® LSG software diskette contains the following files:

Location	Filename	Description
Root	Readme.txt	Must read notes
Root	Users Guide for XLONLSG.pdf	This document
Template	XLONLSG.NbHwt	Device hardware template
Source	UART.nc	Neuron C file containing LSG functions
Source	VAR.nc	Neuron C file containing LSG variable definitions
Source	XLONLSG.h	Defines, Typedefinitions, Function prototypes
Source	XLONLSG.nc	Neuron C file containing when( ) clauses

To start software development, please copy all files to the required directories of your NodeBuilder 3 Development Platform. For more detailed informations, please take a look at *NodeBuilder User's Guide*.

7.3 EXLON® LSG Functions

The EXLON® LSG Software includes everything the programmer needs to handle the UART functionality. The file *UART.nc* contains source code of all Neuron C functions and in the file *VAR.c* all necessary variables are defined. In the file *XLONLSG.h* all the appropriate defines, type definitions and function prototypes are defined.
In the file *XLONLSG.nc* all *when ( )* clauses are defined and you can see how to use the UART functions for an application.

7.3.1 UART Initialization

int lsg_init(lsg_format lsgformat, lsg_baud lsgbaud, lsg_intfc lsgintfc, unsigned int timeout_10ms, unsigned int len);

This function initializes the UART. It sets up the frame format, the serial interface bit rate, the type of serial interface, receiver timeout and receiver minimum data length before signalling the Neuron. This function should be called once in the *reset* clause of the application program.

The LSG frame format parameters are listed in *XLONLSG.h* and may be set to:

LSG Frame Format	Data Bits	Parity	Stop Bits
format_8N1	8	No	One
format_8E1	8	Even	One
format_8O1	8	Odd	One
format_7E1	7	Even	One
format_7O1	7	Odd	One

The *lsg_baud* parameter may be set to:

LSG Baud Format	Nominal Rate	Error
baud_57600	57600	< 1%
baud_38400	38400	-2.34% * Note
baud_28800	28800	< 1%
baud_19200	19200	< 1%
baud_9600	9600	< 1%
baud_4800	4800	< 1%
baud_2400	2400	< 1%
baud_1200	1200	< 1%
baud_600	600	< 1%
baud_300	300	< 1%
baud_150	150	< 1%
baud_110	110	< 1%
baud_75	75	< 1%
baud_50	50	< 1%

*** Note:** using rates that are off by 2.3% or more will not work in all systems.

The *lsg_intfc* parameter may be set to:

LSG Interface Format	Description
intfc_3wire	RS232 transceiver with Rx and Tx line support
intfc_8wire	RS232 transceiver with full modem line support
intfc_RS422	RS422 transceiver
intfc_RS485	RS485 transceiver – no echo mode supported

The *timeout_10ms* parameter defines the receiver timeout scalable in steps of 10ms from 0 up to the maxmum of 500ms. If *timeout_10ms* is set to 0, ***no*** timeout is supported. In this case the UART signals the Neuron only if the defined length in *len* is reached.

The *len* parameter defines the minimum number of bytes in the UART receiver FIFO, before signalling the Neuron. This parameter could be set from *1* up to *MAX_UART_RXD_LENGTH*, which is defined in the file *XLONLSG.h*.

The return value of the function *lsg_init* could be one of the following:

Return Value	Description
OKAY	Everything is fine
ERROR_INVALID_LEN	Invalid length for receiver FIFO
ERROR_UART_BUSY	UART is busy – could not send data
ERROR_WRONG_TIMEOUT	Timeout value not supported – maximum is 500 ms
ERROR_UART_RECEIVER_EMPTY	No data from UART available

7.3.2 Transmitting Data

boolean lsg_txrdy (void);

This function returns *TRUE* if the UART is ready to accept a character or string to be transmitted and *FALSE* otherwise.

*int lsg_puts(char *s, unsigned int len);*

This function checks if the UART is ready and then outputs a string to the UART. *S* is the pointer to the string, the length of the string without NUL has to be defined in *len*. The *len*parameter could be set from *1* up to *MAX_UART_TXD_LENGTH*, which is defined in the file *XLONLSG.h*.

The return value of the function *lsg_puts* could be one of the following:

Return Value	Description
OKAY	Everything is fine
ERROR_INVALID_LEN	Invalid length for receiver FIFO
ERROR_UART_BUSY	UART is busy – could not send data

7.3.3 Receiving Data

boolean lsg_rxrdy (void);

This function returns *TRUE* if the UART has one or more characters in its input FIFO buffer and *FALSE* otherwise.

*int lsg_gets(char *s, unsigned int *len);*

This function checks if the UART has data available and if this is the case, the data is stored into the user defined string *s*. The length of the userstring is defined in *len*. The *len* parameter could be set from *1* up to *MAX_UART_RXD_LENGTH*, which is defined in the file *XLONLSG.h*.

The return value of the function *lsg_gets* could be one of the following:

Return Value	Description
OKAY	Everything is fine
ERROR_INVALID_LEN	Invalid length for receiver FIFO
ERROR_UART_RECEIVER_EMPTY	No data from UART available

7.3.4 Get LSG Status Register

*int lsg_get_status (unsigned char *s);*

The return value of the function *lsg_get_status* could be one of the following:

Return Value	Description
OKAY	Everything is fine
ERROR_INVALID_DATA	No valid LSG status information in *s

This function stores the state of the modem status lines into the user defined string *s[0]*. The status is encoded in the following way:

Bit 0 of s[0] = State of CTS line
Bit 1 of s[0] = State of DSR line
Bit 2 of s[0] = State of DCD line
Bit 3 of s[0] = State of RI line
Bit 4..7 of s[0] = Reserved for future use
Bit 0..7 of s[1] = Reserved for future use

IMPORTANT! To use the function lsg_get_status in the right way, the programmer has to set the event mask as required when calling the function lsg_set_control. Therefore the function lsg_set_control has to be called before use of the function lsg_get_status.

7.3.5 Set LSG Control Register

int lsg_set_control (unsigned char control_register, unsigned char eventmask);

The return value of the function *lsg_set_control* could be one of the following:

Return Value	Description
OKAY	Everything is fine
ERROR_UART_BUSY	UART is busy – could not send data

This function handles the LSG modem control lines RTS and DTR as defined in *control_register*. In addition too the programmer can define an event mask for handling the modem status lines when calling the function lsg_get_status.

7.4 Programming Example

7.4.1 UART Initialization

Depending on the type of application, the programmer may initialize the UART of the EXLON® LSG within the Reset-Task of the Neuron like this:

```
when (reset){

  return_value = lsg_init(format_8N1, baud_9600, intfc_3wire, 50, 5);// UART init RS232
//  return_value = lsg_init(format_8N1, baud_9600, intfc_RS422, 50, 5);// UART init RS422

} // end of when (reset) task
```

7.4.2 Transmitting Data via the UART

Sending a test-string via the UART when pressing the Service-Pin of the Neuron could be done according the following example:

```
when (service_pin_state()){                    // SVC-Pin pressed? If yes then send test-string

  return_value = lsg_puts(snd_string, 18);     // Send string

}
```

7.4.3 Receiving Data via the UART

```
when (lsg_rxrdy ()){                           // Test if UART-data is available

  return_value = lsg_gets(rcv_string, &rcv_len);   // If yes - store received bytes in rcv_string buffer

}
```

7.4.4 Monitor Application

```
when (lsg_rxrdy ()){                           // Test if UART-data is available

  return_value = lsg_gets(rcv_string, &rcv_len);   // If yes - store received bytes in rcv_string buffer

  memcpy(nvi_send_text.ascii, rcv_string, sizeof(nvo_rcv_text));   // Copy rcv_string to snd_string
  return_value = lsg_puts(nvi_send_text.ascii,rcv_len);            // Send received string so we do monitoring
}
```

8 Revision History

Version	Date	Description	State	By	Comments
1.00	16/07/02	First draft	Internal	SD	Preliminary
1.01	13/08/02	Extended with Power Jack and Network Connector Description	Internal	SD	Preliminary
1.10	14/08/02	Software Programming	Internal	SD	Preliminary
1.11	15/07/03	Adding new software features	Released	SD	Official release