



# **Communication Processor Driver Documentation**

## **ASCII Serial Driver Module**

Author: Joe Teixeira

Date: 7-Jun-04

Revision: 0.2





## Table of Contents

<b>OVERVIEW</b> .....	<b>1</b>
<b>FUNCTIONALITY</b> .....	<b>1</b>
Communication Interface.....	1
<b>NETWORK CONFIGURATION</b> .....	<b>2</b>
Handshaking .....	2
Header .....	3
Delimiter .....	3
Checksum .....	3
Errors .....	3
<b>MAPPING CONFIGURATION</b> .....	<b>5</b>
Errors .....	5
<b>STATUS COUNTERS</b> .....	<b>7</b>
<b>OTHER MESSAGES</b> .....	<b>8</b>



# Overview

This document details the configuration interface for implementing a Liaison S3 or S5 Generic ASCII Serial (GAS) module driver installation. This document assumes the reader is familiar with the Liaison's S3 and/or S5 communication processor configuration process. For more information refer to the configuration manual available in the support section of the website.

The ASCII serial driver module, referred to as GAS, provides connectivity to any device capable of sending or receiving non-formatted ASCII characters.

# Functionality

The ASCII<sup>i</sup> serial driver module's functionality is confined to transmit and receive ASCII data strings<sup>ii</sup>. A number of standard message formatting options are provided as well as a choice of message validation protocols (LRC, CRC). Parsing<sup>iii</sup> received data is handled using script functions. Numerical data can be extracted from the received string and placed in the Pronghorn<sup>iv</sup> database in a variety of data formats (bit, integer, float, etc...). Output strings are similarly constructed using scripted functions.

Scripting<sup>v</sup>, while designed to be straightforward, does require some familiarity with programming conventions; either software or PLC-type experience. If you do not have the time or inclination to learn the scripting language, Liaison does provide configuration services at a very reasonable rate. On the other hand, a scripting instruction manual and example code can be provided to those parties interested in learning to do it themselves.

# Communication Interface

The use of the ASCII serial driver module requires the availability of a serial port. Either RS-232/422 or 485. Serial communication is relatively primitive compared to Ethernet and/or other modern industrial protocols and may require special attention to jumper setting, termination, isolation and other factors. Please refer to your hardware documentation and consider discussing your application with Liaison technical support.

---

<sup>i</sup> ASCII consists of a set of numerical codes corresponding to the alphabet and various special printed characters. It is a widely used standard. Many industrial devices use ASCII for serial communication of their parameterization and process data

<sup>ii</sup> A 'string' of ASCII codes comprises a message. The message is delimited with special codes to indicate the start and end of the message

<sup>iii</sup> a grammatical evaluation of string data for its numerical values. Thus an alphanumeric character for "64" is parsed or read as the number 64 which can be then used arithmetically.

<sup>iv</sup> The Liaison communication processor has an operating system and then a daemon, or specialized process which actually conducts the communication process. Liaison names this daemon "pronghorn"

<sup>v</sup> a simple method for creating grammatical structures similar to macros in many popular software programs. Scripting allows a user to construct a function which can be called before or after a value is stored in the internal database. Additionally, it can be used in conjunction with various control codes to produce certain behaviors, such as comparing a value with a constant and then triggering queries or updates on one or more network modules

# Network Configuration

The configuration of this module involves at least two files and usually three. The net.csv and map.csv files are used respectively for communication parameters and data mapping. A third file; the script.lua file, is often used to create user-defined functions. Please refer to the configuration manual for more information. The succeeding tables outline the available parameters and their default values. Some of the parameters are standard parameters used to configure the serial interface while others are particular to this module. Not specifying a particular parameter means that its default value will be used. Parameters may be listed in any order. The following table provides the communication parameters used in the net.csv file (network parameterization).

<i>Name</i>	<i>Valid Range</i>	<i>Default</i>	<i>Description</i>
Header	0 - 255, refer to <a href="#">Header</a> section	STX	This character identifies the beginning of a packet transmission.
Delimiter	0 -255, refer to <a href="#">Delimiter</a> section	ETX	This character identifies the end of a packet transmission.
Checksum	refer to <a href="#">Checksum</a>	None	Specify the type of error checking to be used
Retries	0 - 9	0	This is the number for times a transaction will re-transmitted upon the detection of a timeout
Timeout	0 - 360000	1000	The number of milliseconds to wait for a response
Com_Port	1 - 16	1	The serial port interface to be used
Baud	50 - 460800 <sup>vi</sup>	9600	The serial line speed being used.
Stop_Bits	1 or 2	1	The number of stop-bits per transmitted character
Data_Bits	5 - 8	8	The number of data-bits per transmitted character
Parity	None, Even, Odd	None	The type of parity checking performed
Tx_Delay	0 - 65535	0	The number milliseconds to wait before transmitting
Handshake	Yes or No	No	Specifies hardware handshaking (RTS/CTS) is to be used
IRQ	0 - 16 <sup>vii</sup>	0	The IRQ used to access the serial hardware.
IO_Port	0 - 0xffff <sup>viii</sup>	0	The port address used to access the serial hardware.

## Handshaking

Some ASCII protocols feature a couple of different types of 'handshaking' to verify the successful receipt of a transmission, or to signal a failed transmission. The most common types are software acknowledgments where particular codes are exchanged, or a hardware acknowledgement in the form of specific signals in the RS-232/422 or 485 media protocol. Please refer to your documentation to determine which, if any are being employed.

<i>Method</i>	<i>Description</i>
No handshaking	No confirmation response sent
ACK/NAK	A 0x06 character is sent to the device to indicate successful while a 0x15 character is sent to device to indicate unsuccessful.
RTS/CTS	When RTS is turned on the device should transmit. When he RTS is

<sup>vi</sup>Valid values are: 50, 75, 134, 200, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56700, 115200, 230400, 460800. This depends on support of the hardware being used.

<sup>vii</sup>A value of zero specifies to use default values for the given serial port being used. Typically COM1=IRQ3, COM2=IRQ4, COM3=IRQ3, COM4=IRQ4.

<sup>viii</sup>A value of zero specifies to use default values for the given serial port being used. Typically COM1=0x3e0, COM2=0x2e0, COM3=0x3f0, COM4=0x2f0.

## ASCII Serial Driver Module

<i>Method</i>	<i>Description</i>
	turned off the device should not transmit.

## Header

Each transmission can optionally be prefixed with a header byte, to identify the beginning of a transmission. Although, any character can be used the following are typical.

<i>Character</i>	<i>Mnemonic</i>	<i>Description</i>
0x1b	ESC	Escape character
0x02	STX	Start of Transmission

## Delimiter

A delimiter is used to indicate the end of transmission packet and must be present.

<i>Character</i>	<i>Mnemonic</i>	<i>Description</i>
0x0d	CR	Carriage Return
0x02	ETX	End of Transmission

## Checksum

Many ASCII implementation feature an error checksum code added to each message to ascertain uncorrupted transmission. The LRC<sup>ix</sup> checksum is calculated on all the characters between (including) the Header and Delimiter characters. The calculation is accomplished by adding all the binary bytes using exclusive-OR (XOR). The one-byte result is then added to the packet. The placement of the LRC byte varies from implementation to implementation. Users may indicate the positioning consistent with their application.

<i>Mnemonic</i>	<i>Description</i>
None	No error checking done
DLI+LRC	Use LRC coding and place it after the Delimiter character
LRC+DLI	Use LRC coding and place it before the Delimiter character

### Errors

If any of the validation, for the network configuration parameters, fails a critical error message will be placed in the logs. In addition the module driver will not start. The following is a list of critical messages that might be generated.

<i>Msg ID</i>	<i>Mnemonic</i>	<i>Description</i>
3506	CRT_HANDSHAKE	Handshaking method (%s) not valid - The specified handshaking method is not recognized
3507	CRT_HEADER	Header value (%s) not valid - The specified header character is not valid
3508	CRT_DELIMITER	Delimiter value (%s) not valid - The specified delimiter character is not valid
96	CRT_BAD_BAUD	Invalid BAUD rate (%d) - The communication speed specified is not on of the predefined

<sup>ix</sup> Logical Redundancy Check

## ASCII Serial Driver Module

<i>Msg ID</i>	<i>Mnemonic</i>	<i>Description</i>
		values of: 50, 75, 134, 200, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56700, 115200, 230400, 460800
97	CRT_BAD_STOPBITS	Invalid number of stop bits (%d) must be 1 or 2 - The number of stop-bits specified is not one of 1 or 2.
98	CRT_BAD_PARITY	Invalid parity (%s) must be 'None', 'Even' or 'Odd' - The parity check specified is not one of the predefined mnemonics.
99	CRT_BAD_DATABITS	Invalid number of data bits (%d) must be between 5 and 8 - The number of data-bits specified is out of range.
100	CRT_BAD_TIMEOUT	Invalid timeout value (%d) must be less than %d ms - The timeout values specified is too large, it should be less than 360000 milliseconds

## Mapping Configuration

Mappings<sup>x</sup> are found in a CSV<sup>xi</sup> file named maps.csv which may be edited either through the Edit>Config>Map.csv menu selection in the Java application accessible through your browser, or by opening the file directly in Excel or other spreadsheet program. The spreadsheet program has the advantage of allowing a user to annotate a project extensively outside of the delimited areas, which effectively combines the configuration file and its documentation. The parameters listed only pertain to this module. Refer the **Configuration Manual** for a list of the other common parameters used to configure a mapping. Most parameters have a default value that will be used if the parameter does not exist.

<b>Name</b>	<b>Valid Range</b>	<b>Default</b>	<b>Description</b>
Key	0 - 255	0	This a unique key value used to identify different unsolicited packet request. This key is identified/generated in the special script function "Gas_UnWrite()"

Special note on the following supported polling command types.

<b>Command</b>	<b>Description</b>
wu, wc, w	These are write events to the module driver. The preformatted ASCII data will be transmitted out the serial port. Alternately a script function can be used to format the ASCII packet.
un, uw, ur	These are read or write events coming from the network. In this case a ASCII data from an attached device. Ensure that the mapping contains enough bytes to contain the maximum number of characters or use a script function to parse/strip the packet. If different requests of the same packet size are received then the special script function "Gas_UnWrite()" should be used to uniquely identify a particular packet. This function must return a key that will identify which map (using the corresponding "Key" parameter) this packet belongs to.
ra, r	These are solicited read events. This is used for a more sophisticated ASCII protocols that supports a typical master/slave architecture. This type of map needs a second script function to be defined that will create the ASCII request packet (the portion between the Header and Delimiter characters). The response can also be parsed with a script function (this must be the first Script parameter, as data will be written to the database)

## Errors

If any of the validation, for the mapping configuration parameters, fails an error message will be placed in the logs. In addition that particular mapping item will not be added to the internal mapping list. However the module continues with any remaining mappings.

<sup>x</sup> a term referring to the definition of data source or destination within the network module and its corresponding data register location in the pronghorn database. A command that reads ten registers from a remote device and deposits them into internal pronghorn locations is a single mapping. Many mappings can be configured for each active network module

<sup>xi</sup> comma separated values; a text file format common to spreadsheets for listing arrays of values. The delimited is usually a comma, although in some cases it is a semi-colon. Pronghorn automatically determines which by reading the first character of the file which in a properly configured file is always blank and thus stores only the delimiter type being used

## ASCII Serial Driver Module

<i>Msg ID</i>	<i>Mnemonic</i>	<i>Description</i>
66	ERR_BAD_POLLTYPE	Map item %d has invalid/wrong Poll command (%s) - The specified file type is not valid
65	ERR_BAD_DATATYPE	Map item %d has invalid Data Type (%s) - The specified file type is not valid
52	ERR_NOT_FOUND	Did not find '%s' in 'Script List' - The specified script name was not found in scripts.lua
52	ERR_NOT_FOUND	Did not find '%s' in 'Tag List' - The specified tag name was not found in tag database

## Status Counters

This module driver maintains a set of status counter for recording the operational status of the module and the network.

<i>Msg ID</i>	<i>Mnemonic</i>	<i>Description</i>
3509	CNT_MSG_SENT	Number of Messages transmitted - Keeps track of the number of transactions transmitted from this module.
3510	CNT_MSG_RECV	Number of Messages received - Keeps track of the number of transaction received for this module.
3511	CNT_ACK_SENT	Number of ACK's transmitted - Keeps track of the number of good handshakes sent
3512	CNT_NAK_SENT	Number of RTRY's transmitted - Keeps track of the number of error handshakes sent
3513	CNT_ACK_RECV	Number of ACK's received - Keeps track of the number of good handshakes received
3514	CNT_NAK_RECV	Number of NAK's received - Keeps track of the number of error handshakes received
3515	CNT_RETRIES	Number of retries - Keeps track of the number of re-transmits was performed
3516	CRD_CHAR_SENT	Number of characters transmitted - The number of characters sent out the serial port (maintained by the hardware interface).
3519	CRD_CHAR_RECV	Number of characters received - The number of characters received from the serial port (maintained by the hardware interface).

## Other Messages

In the normal operation of this module certain types of message may appear in the logs. The messages specific to this module are listed below. Refer to the ***Pronghorn Users Guide*** for a list of the common messages that can be seen in the logs.

<i><b>Msg ID</b></i>	<i><b>Mnemonic</b></i>	<i><b>Description</b></i>
3503	ERR_BAD_CRC	Response with wrong CRC (actual=0x%02x%02x, expected=0x%02x%02x) - A packet was received with a CRC that differs from the calculated CRC
3504	ERR_UNWRITE	Error while processing unsolicited Write (Len=%d) - An unexpected error occurred while processing the packet
3519	ERR_UNREAD	Error while processing unsolicited Read (Len=%d) - An unexpected error occurred while processing the packet
3505	NTC_RETRY	Retry #%d, due to error %d - Displays the retry count and error the reason code
3520	ERR_NAK_RECV	NAK reply received - A negative acknowledgment was received
75	ERR_WRONG_RESPONSE	Unexpected Response 0x%x, s/b 0x%x - The received and expected delimiters did not match