# LTS-20 LonTalk® Serial Adapter and PSG-20 User's Guide

Version 2

ECHELON®

Corporation

078-0181-01C

Echelon Corporation
www.echelon.com

# Preface

This document describes how to use an LTS-20 LonTalk Serial Adapter and a host processor with an EIA-232 (formerly RS-232) serial interface with a LONWORKS® network.

## Audience

This user's guide provides specifications and instructions for LTS-20 users.

## Content

This manual provides detailed information about the hardware and software for the LTS-20 and PSG-20.

- Chapter 1 introduces the LTS-20.
- Chapter 2 provides an overview of the LTS-20.
- Chapter 3 describes product development with the LTS-20 module.
- Chapter 4 describes EMI and ESD issues for LTS-20 and PSG-20 modules.
- Chapter 5 describes the LTS-20 software.
- Chapter 6 discusses creating SLTA network driver.
- Chapter 7 discusses using the DOS network driver.
- Chapter 8 discusses using the UNIX network driver.
- Chapter 9 discusses using the LTS-NSI mode.
- Chapter 10 discusses the LTS-20 MIP mode software.
- Chapter 11 describes using the drivers and link manager with LTS-20 NSI mode.
- Chapter 12 discusses using the DOS driver with LTS-20 MIP mode.
- Chapter 13 explains how to create an LTS-20 MIP mode driver.
- Chapter 14 discusses initialization and installation.
- Chapter 15 explains how to use the LST-20 with a modem.
- Chapter 16 discusses using the host connect utility with the LTS-20 MIP mode.
- Chapter 17 details using a programmable serial gateway.
- Chapter 18 details modem troubleshooting.
- Appendix A lists the default communications parameters for LTS-20-based products.
- Appendix B describes the Windows DLL files supplied with the LTS-20.
- Appendix C includes a copy of the software license agreements.

## Related Manuals

The following Echelon documents are suggested reading for more information:

- The *LNS™ DDE Server User's Guide* is a manual for developers on how to create user interface, monitoring, and control applications that communicate with LONWORKS networks from computers running Microsoft Windows.
- The *LonMaker® for Windows Integration Tool User's Guide* is a manual for users on how to install networks using the integration tool.
- The *LonBuilder® User's Guide* describes how to develop LONWORKS applications with the LonBuilder Developer's Workbench.
- The *NodeBuilder™ User's Guide* describes how to develop LONWORKS applications with the NodeBuilder Development Tool.
- The *LONWORKS Host Application Programmer's Guide* describes how to write a host application that can be used with a serial adapter.
- The *Neuron® Chip Data Book* describes the LonTalk message formats that can be used with a serial adapter. It also describes the network management and network diagnostic messages that can be sent with such an adapter.

## Web Access

Engineering bulletins and data sheets supporting this product are available on the Echelon Web site. General information regarding Echelon, its business, and its products also are located on the site at http://www.echelon.com. The Developer's Toolbox located at the Web site includes drivers for the LTS-20.

# Contents

1

# LTS-20 Introduction

The LTS-20 LonTalk Serial Adapter Module is a network interface that enables any host processor with an EIA-232 serial interface to connect to a LONWORKS network. A replacement for the previous generation LTS-10 Core Module, the LTS-20 is supplied with both Network Services Interface (NSI) firmware to support LNS – the standard LONWORKS® network operating system – as well as Microprocessor Interface Program (MIP) firmware to support older API-based tools. The LTS-20 extends the reach of LONWORKS technology to a variety of hosts, including desktop, laptop, and palmtop PCs, workstations, embedded microprocessors, and microcontrollers.

The LTS-20 enables the attached host to act as an application node on a LONWORKS network. When used with a PC host and the LonMaker for Windows Integration Tool (or an older generation tool using the LonManager API), or LNS DDE Server, the LTS-20 can also be used to build sophisticated network management, monitoring, and control tools for LONWORKS networks.

The LTS-20 is a direct replacement for Echelon's model 65200-100 LTS-10 module. The two modules are pin-for-pin compatible and also have identical physical dimensions. The LTS-20 is equipped with an NSI to enable it to be used in conjunction with LNS – the standard operating system for LONWORKS control networks. By default the LTS-20 is shipped with the NSI mode enabled. A jumper is provided which, when cut by the customer, disables the NSI and enables the MIP mode for emulating the behavior of the LTS-10.

Intended to be embedded within an OEM's product, the LTS-20 and its associated interface logic can be used to connect to a host through a pair of modems and the telephone network. This allows the monitoring, control, or network management application computer to be remote from the network. A node using the LTS-20, associated logic, and modems can initiate a telephone call to a remote host computer, and can be set up to answer incoming calls from a remote host.

**A Connectivity Starter Kit (Model 58030-01) should be ordered for initial development with an LTS-20.** The kit includes both software and documentation. The software includes network drivers for Windows® 95, 98, and NT. Supplied as a single in-line module (SIM) form-factor building block, the LTS-20 can be used to create custom serial interfaces to a wide range of network media.

# 2

# LTS-20 Overview

This chapter provides an overview of the model 65202 LTS-20 LonTalk Serial Adapter Module.

## Mechanical Description

The LTS-20 Module consists of a 67mm by 28mm by 7mm (2.65" by 1.1" by 0.3") module with the core electronics and firmware required to implement a serial LonTalk Adapter. The module is attached to a motherboard using a 40-position, 0.050-inch spacing SIM socket. Two compatible sockets are available:

- Molex 15-82-1175 SIM Vertical Connector with metal latches, 0.050 Centerline Single Row Connector - 40 position.
- AMP 4-382487-0, SIM II Right Angle Connector, 0.050 Centerline Single Row Connector - 40 position.

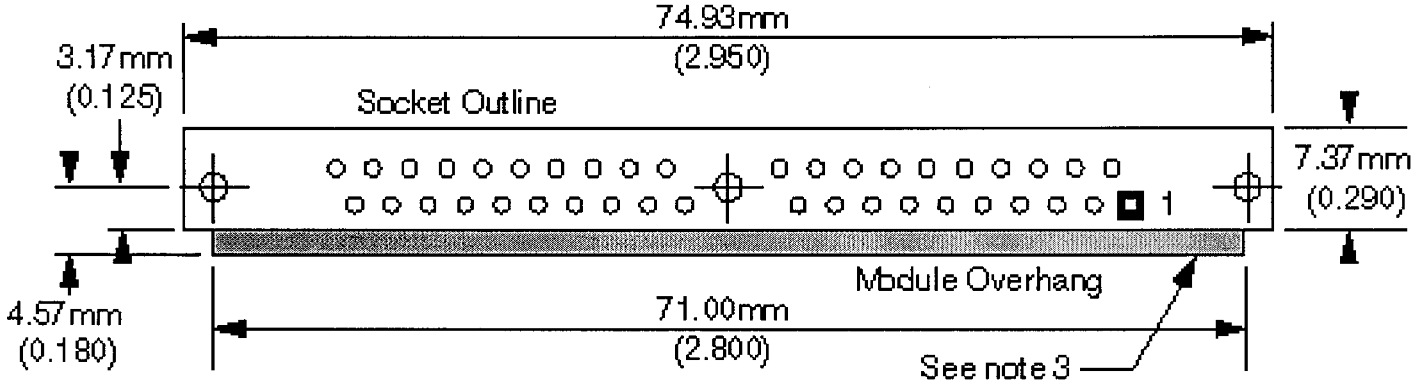
For information about Molex parts call +1-708-969-4550 or fax +1-708-969-1352.

Within North America, AMP drawings can be obtained via FAX using the free AMP FAX service. Call 1-800-522-6752 from a touchtone phone and order customer prints using the AMP part number. Additional information on the connectors is available in AMP application note number AMP 114-1060 and reliability information is available in AMP product specification 108-1297.

Figure 2.1 illustrates the mechanical footprint for the module and vertically mounted socket. Figure 2.2 shows the recommended PCB pad layout for the vertically mounted socket. Figures 2.3 and 2.4 provide the same information for the right-angle socket.

Decisions about component placement on the motherboard must consider electromagnetic interference (EMI) and electrostatic discharge (ESD) issues discussed in Chapter 4 of this document.

**LTS-20 Footprint when using Molex Part Number 15-82-1175 (Component Side, Vertical SIM Mounting)**

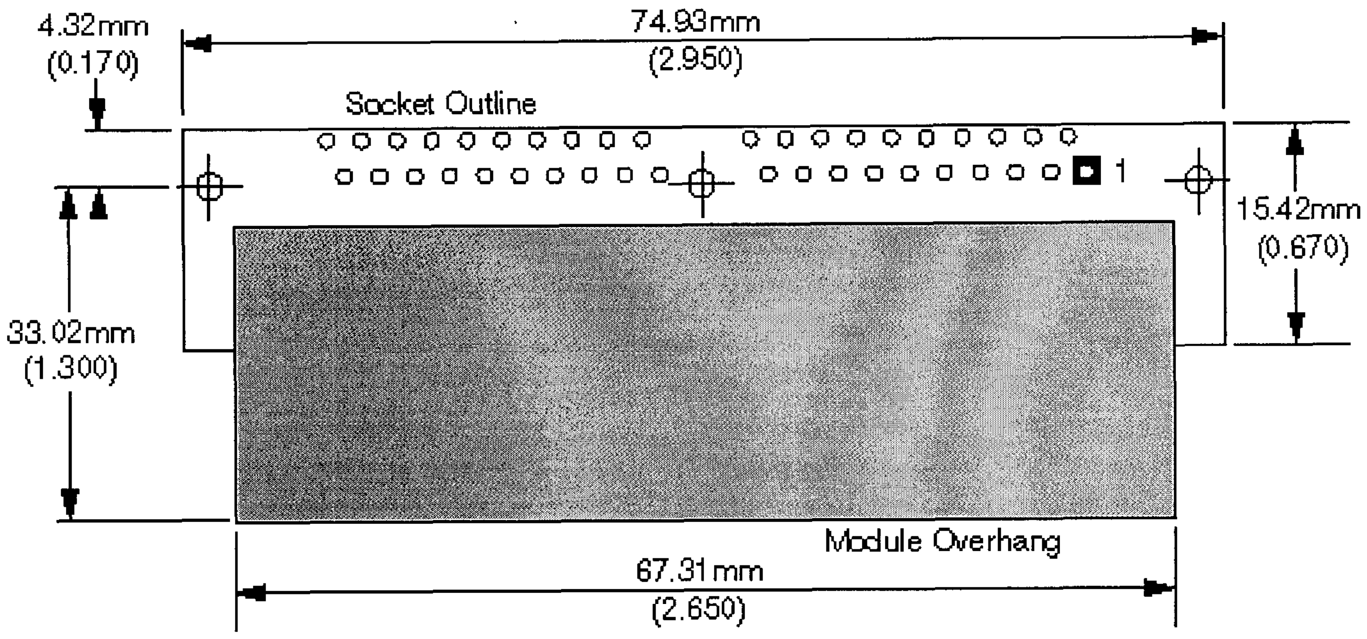


Notes:

1. Dimensions in mm (inches).
2. Tolerances ± .13mm (0.0005)
3. Components standing higher that 3.81mm(.15) should not be closer than 12.4mm (0.5) to this edge of the socket to allow clearance to insert the module.
4. Allow 33.02mm (1.3) clearance above PCB over the footprint area. Additional clearance required to insert the module.
5. Socket dimensions are subject to change. Contact Molex for the most current information.
6. Alternate AMP part is 822021-1.

**Figure 2.1** LTS-20 Vertical Socket Mechanical Footprint

**LTS-20 PCB Footprint when using AMP Part Number 4-382487-0**
**(Component Side, Horizontal Mounting)**



Notes:

1. Dimensions in mm (inches).
2. Tolerances ± .13mm (0.0005)
3. Do not position components in the overhang region.
4. Allow 12.7mm (0.5) clearance above PCB over the entire footprint area. Additional clearance required during assembly to insert the module.
5. Socket dimensions are subject to change. Contact AMP for the most current information.

**Figure 2.2** LTS-20 Horizontal Socket Pad Layout

## Power Requirements

The modules require a +5VDC ±10% power source with a minimum of 150mA of current capacity.

### *Power Supply Decoupling and Filtering*

The design for the module power supply must consider filtering and decoupling requirements of the module. The power supply filter must prevent noise generated by the core module from conducting onto external wires. Switching power supply designs must also consider the effects of radiated EMI.

The modules require a clean power supply to prevent RF noise from conducting onto the network through active drive circuits. Power supply noise near the network transmission frequency may degrade network performance.

The modules include 2.2$\mu$F and 0.1$\mu$F power supply bypass capacitors close to pins 1, 9, and 31. In general, high-frequency decoupling capacitors valued at 0.1$\mu$F or 0.01$\mu$F placed near pins 1, 9, and 31 on the motherboard are necessary to reduce EMI.

### *Low Voltage Protection*

It is necessary to include a low voltage indicator (LVI) circuit on the module motherboard to drive the ~RESET line of the core module. See the *Neuron Chip Data Book* for details. Failure to include such protection may cause data corruption to configuration data maintained in EEPROM on the module's Neuron Chip. In the sample circuit of figure 3.1, protection is provided via a Motorola MC33164.

## Electrical Interface

The pinout of the modules is shown in table 2.1.

**Table 2.1** Pinout of the LTS-20

| ***Name*** | ***Function*** | ***Pin #*** |
|---|---|---|
| AUTOBAUD | Automatic serial bit rate detect enable input | 19 |
| BAUD0 | Serial bit rate 0 input (LSB) | 16 |
| BAUD1 | Serial bit rate 1 input | 15 |
| BAUD2 | Serial bit rate 2 input (MSB) | 21 |
| CFG0 | EIA-232 interface option (1 = 8 wire, 0 = 3 wire) | 17 |
| CFG1 | Network Disable (1 = disable after reset) | 14 |
| CFG2 | Modem Support (1 = remote host; 0 = local host) | 20 |
| CFG3 | Interface link protocol (1 = Buffered; 0 = ALERT/ACK) | 18 |
| CLK OUT | Neuron Chip CLK2 output | 11 |
| CP0 | Network communication port 0 | 6 |
| CP1 | Network communication port 1 | 5 |
| CP2 | Network communication port 2 | 4 |
| CP3 | Network communication port 3 | 7 |
| CP4 | Network communication port 4 | 3 |
| ~CTS | EIA-232 clear to send output from UART | 36 |
| ~DCD IN | EIA-232 serial data carrier detect input (DTE only) | 35 |
| ~DCD OUT | EIA-232 serial data carrier detect output (DCE only) | 40 |
| DCE | Indicates whether connected as DCE | 22 |
| ~DSR | EIA-232 data set ready output from UART | 38 |
| ~DTR | EIA-232 data terminal ready input to UART | 37 |
| PKT | Packet transmitted output | 13 |
| ~RESET | Neuron Chip reset input and output | 8 |
| ~RI IN | EIA-232 ring indicator input (DTE only) | 33 |
| ~RI OUT | EIA-232 ring indicator output (DCE only) | 34 |
| ~RTS | EIA-232 request to send input to UART | 39 |
| SERIAL IN | EIA-232 serial data input to UART | 26 |
| SERIAL OUT | EIA-232 serial data output to UART | 27 |
| ~SERVICE | Neuron Chip service pin input and output | 12 |
| ~TEST | Manufacturing test pin, tie to Vcc in final product | 30 |
| XID0 | Transceiver ID 0 input (LSB) | 25 |
| XID1 | Transceiver ID 1 input | 23 |
| XID2 | Transceiver ID 2 input | 29 |
| XID3 | Transceiver ID 3 input | 28 |
| XID4 | Transceiver ID 4 input | 24 |
| $V_{CC}$ | +5VDC input | 1, 9, 31 |
| GND | Ground | 2, 10, 32 |

## NSI/MIP MODE JUMPER R2

The NSI/MIP jumper R2 (a 220□ resistor) determines the start-up mode of the module. The module is shipped in the NSI mode, with the jumper intact. Cutting jumper R2 disables the NSI mode and enables the MIP mode (for emulating the LTS-10 module). DO NOT cut the jumper while the module is powered – only cut the jumper with the module unpowered. Observe appropriate ESD protection suitable for CMOS devices when handling the module or cutting jumper R2. To ensure reliable operation, <u>R2 should be removed in its entirety and not simply cut at one end.</u>

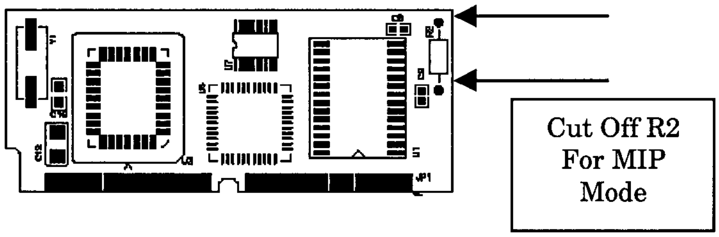


**Figure 2.3** R2 Jumper

## AUTOBAUD

The AUTOBAUD input signal enables automatic baud rate detection on the LTS-20 as described in Chapter 7. The input is a floating CMOS input and must be asserted high to enable automatic baud rate detection or be asserted low to disable automatic baud rate detection.

## BAUD[2..0]

The BAUD[2..0] input signals set the EIA-232 serial bit rate on the LTS-20 module as described in Chapter 7 and summarized in Table 2.2. The inputs are not used when AUTOBAUD is enabled. The inputs are floating CMOS inputs and must be asserted high to select a "1" and asserted low to select a "0".

**Table 2.2** LTS-20 Baud Rate Inputs

| ***BAUD[2..0]*** | ***Serial Bit Rate*** |
|---|---|
| 0 0 0 | 14,400 bps |
| 0 0 1 | 1,200 bps |
| 0 1 0 | 2,400 bps |
| 0 1 1 | 9,600 bps |
| 1 0 0 | 19,200 bps |
| 1 0 1 | 38,400 bps |
| 1 1 0 | 57,600 bps |
| 1 1 1 | 115,200 bps |

## CFG0

The CFG0 input signal selects a full 8-wire interface or 3-wire interface for the LTS-20 module as described in Chapter 7. The input is a floating CMOS input and must be asserted high to select a full 8-wire interface or asserted low to select a 3-wire interface.

## CFG1

The CFG1 input signal enables or disables network communications after reset for the LTS-20 module as described in Chapter 7. The input is a floating CMOS input and must be asserted high to disable network communications after reset or asserted low to enable network communications after reset.

## CFG2

The CFG2 input signal controls the use of the LTS-20 module with a modem as described in Chapter 12. The input is a floating CMOS input and must be asserted high to enable modem support for a remote host or asserted low to enable local host support.

## CFG3

The CFG3 input signal controls the network interface link protocol used between the LTS-20 module and a local host as described in Chapter 11. The input is a floating CMOS input and must be asserted high to select the buffered link protocol or asserted low to select the ALERT/ACK link protocol.

## CLK OUT

The CLK OUT output signal is driven by the CLK2 pin of the core module Neuron Chip. It can drive one HCMOS load, and can be used to interface to the FTT-10A Free Topology Transceiver or the LPT-10 Link Power Transceiver.

## CP(4..0)

The CP[4..0] signals are connected to the CP[4..0] pins of the core module Neuron Chip. The function of these pins is described in the *Neuron Chip Data Book.*

## ~CTS

EIA-232 clear to send output when the LTS-20 is connected as a DCE device. This output should be used as the EIA-232 request to send (~RTS) output when the LTS-20 is connected as a DTE device. The output is driven by the core module UART and must be connected to an EIA-232 driver if EIA-232 voltage levels are required, or can be ignored for a 3-wire serial interface.

## ~DCD IN

EIA-232 data carrier detect input when the LTS-20 is connected as a DTE device. This input is not used when the LTS-20 is connected as a DCE device. This input is not used by the firmware when connected to a local host, and is used to detect incoming calls when used with a remote host (i.e. when CFG2 is set to the *Remote Host* state). The input is connected to the core module UART and must be externally connected to an EIA-232 receiver if EIA-232 voltage levels are used, or should be connected to ground for a 3-wire serial interface.

## ~DCD OUT

EIA-232 data carrier detect output when the LTS-20 is connected as a DCE device. This output is not used when the LTS-20 is connected as a DTE device. This output is always asserted high by the firmware. The output is driven by the core module UART and must be connected to an EIA-232 driver if EIA-232 voltage levels are required, or can be ignored for a 3-wire serial interface.

## DCE

The input is a floating CMOS input. It is not used by the firmware. It should be pulled high or low by the motherboard.

## ~DSR

EIA-232 data set ready output when the LTS-20 is connected as a DCE device. This output should be used as the EIA-232 data terminal ready (~DTR) output when the LTS-20 is connected as a DTE device. This output is always asserted high by the firmware when used with a local host and is asserted low for 500ms to hang up the modem when used with a remote host (i.e., when CFG2 is set to the *Remote Host* state). The output is driven by the core module UART and must be connected to an EIA-232 driver if EIA-232 voltage levels are required, or can be ignored for a 3-wire serial interface.

## ~DTR

EIA-232 data terminal ready input when the LTS-20 is connected as a DCE device. This input should be used as the EIA-232 data set ready (~DSR) input when the LTS-20 is connected as a DTE device. The input is connected to the core module UART and must be externally connected to an EIA-232 receiver if EIA-232 voltage levels are used, or should be externally connected to ground for a 3-wire serial interface.

## PKT

The PKT output signal is asserted high when interface buffers are passed from the host to LTS-20 module. PKT can be used to drive an activity LED, as in the example circuit shown in figure 3.1. The output is controlled by writing to the memory mapped I/O at location 0xE7E0—write 0x01 to drive the signal high, and 0x00 to drive the signal low. PKT can source 2mA with $V_{OH} \geq 2.4V$, and it can sink 8mA with $V_{OL} \leq 0.45V$.

## ~RESET

The ~RESET signal is connected to the ~RESET pin of the core module Neuron Chip. The function of the ~RESET pin is described in the *Neuron Chip Data Book.* The core modules include a reset circuit as shown in figure 2.5.

The ~RESET signal should be driven (open collector or open drain only) by a low voltage protection circuit (LVI) on the core module motherboard as described under *Low Voltage Protection* earlier in this chapter. The use of an LVI is critical for reliable operation of the LTS-20.

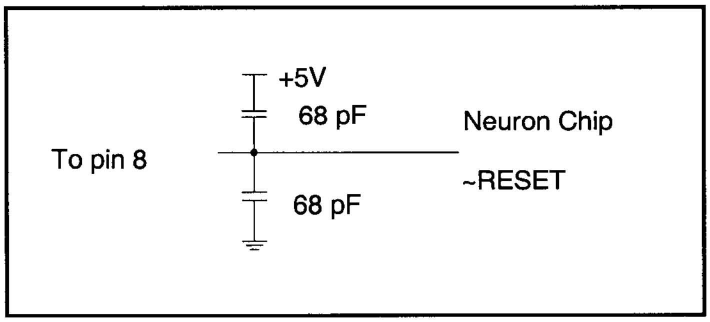


**Figure 2.4** LTS-20 Reset Circuit

## ~RI IN

EIA-232 ring indicator input when the LTS-20 is connected as a DTE device. This input is not used when the LTS-20 is connected as a DCE device. The output is connected to the core module UART and must be externally connected to an EIA-232 receiver if EIA-232 voltage levels are used, or should be externally connected to ground for a 3-wire serial interface.

## ~RI OUT

EIA-232 ring indicator output when the LTS-20 is connected as a DCE device. This output is not used when the LTS-20 is connected as a DTE device. This output is always set to the inactive state by the firmware. The output is driven by the core module UART and must be connected to an EIA-232 driver if EIA-232 voltage levels are required, or can be ignored for a 3-wire serial interface.

## ~RTS

EIA-232 request to send input when the LTS-20 is connected as a DCE device. This input should be used as the EIA-232 clear to send (~CTS) output when the LTS-20 is connected as a DTE device. The input is connected to the core module UART and must be externally connected to an EIA-232 receiver if EIA-232 voltage levels are used, or should be externally connected to ground for a 3-wire serial interface.

## SERIAL IN

EIA-232 received data (RXD) input. The input is connected to the core module UART and must be externally connected to an EIA-232 receiver if EIA-232 voltage levels are used.

## SERIAL OUT

EIA-232 transmitted data (TXD) output. The output is driven by the core module UART and must be connected to an EIA-232 driver if EIA-232 voltage levels are required.

## ~SERVICE

The ~SERVICE signal is connected to the ~SERVICE pin of the module's Neuron Chip. The function of the ~SERVICE pin is described in the *Neuron Chip Data Book.* The internal pullup resistor for the service pin is enabled. A service LED will reflect the firmware status: *blinking* means that the module is unconfigured, *off* means that it is configured, and *on steadily* means that it is applicationless. If the service LED is on steadily, a critical error has been detected by the firmware. A push-button connected to this pin may be used during installation to broadcast the 48-bit Neuron ID on the network.

Typical applications do not require debounce conditioning of momentary push buttons attached to the ~SERVICE pin. The software response time associated with this input is long enough to effectively provide a software debounce for switches with a contact bounce settling time as long as 20 milliseconds.

## ~TEST

The ~TEST input signal is used to put an LTS-20 module in test mode during manufacturing test. Use of this signal is described in the LTSMFT.NC Neuron C file in the manufacturing test directory (MFT) of the software. This signal should be tied high in a shipping, production-level node.

## XID(4..0)

The LTS-20 comes preconfigured with many common LONWORKS transceiver parameters. The XID[4..0] input signals specify a transceiver identification (ID) to select the appropriate transceiver type.

The transceiver ID inputs eliminate a manufacturing step by automatically configuring the LTS-20 for most transceivers. A special transceiver ID is reserved for programming any custom transceiver type. This value causes the communication port pins to be configured as all inputs so that no line will be driven by both the transceiver and LTS-20 Neuron Chip before the chip can be properly configured.

The LTS-20 firmware reads the transceiver ID inputs on both power-up and on reset. If it is being powered-up for the first time, or if the transceiver ID is different from the last time it was powered-up, the parameters specified in table 2.3 are loaded. If it is being re-powered-up, and the transceiver ID is not 30, the LTS-20 firmware compares the network bit rate and input clock for the specified transceiver to the current transceiver parameters. If these parameters don't match, then all transceiver parameters are reinitialized. This allows a network management tool to change parameters, such as the number of priority slots, without the new values being overwritten by the LTS-20 firmware.

**Table 2.3** LTS-20 Transceiver IDs

| *ID* | *XID [4..0]* | *Name* | *Media* | *Bit Rate (bps)* |
|---|---|---|---|---|
| 01 | 00001 | TP/XF-78 | Isolated Twisted Pair | 78k |
| 03 | 00011 | TP/XF-1250 | Isolated Twisted Pair | 1.25M |
| 04 | 00100 | FT-10 | Free Topology, Link Power | 78k |
| 05 | 00101 | TP/RS485-39 | RS-485 Twisted Pair | 39k |
| 09 | 01001 | PL-10 | Power Line (FCC-band) | 10k |
| 10 | 01010 | TP/RS485-625 | RS-485 Twisted Pair | 625k |
| 11 | 01011 | TP/RS485-1250 | RS-485 Twisted Pair | 1.25M |
| 12 | 01100 | TP/RS485-78 | RS-485 Twisted Pair | 78k |
| 15 | 01111 | PL-20A | Power Line (narrow band A-band) | 3.6k |
| 16 | 10000 | PL-20C | Power Line (C-band - CENELEC) | 5k |
| 17 | 10001 | PL-20N | Power Line (C-band – non-CENELEC) | 5k |
| 18 | 10010 | PL-30 | Power Line (A-band) | 2k |
| 24 | 11000 | FO-10 | Fiber Optic | 1.25M |
| 27 | 11011 | DC-78 | Direct Connect | 78k |
| 28 | 11100 | DC-625 | Direct Connect | 625k |
| 29 | 11101 | DC-1250 | Direct Connect | 1.25M |
| 30 * | 11110 | Custom | Custom | N/A |

Notes: Type 30 can be used for any transceiver type; the communications port is initially defined as all inputs to prevent circuit conflicts. When using type 30, the transceiver parameters must be reprogrammed by establishing communication over the serial port, as described in the next chapter.

See Appendix A for a listing of the communications parameters for each transceiver type.

## LTS-20 Software Configuration Options

The types of messages passed between the host and the LTS-20 are determined by EEPROM configuration options. These options are described under *Network Interface Configuration Options* in Chapter 3 of the *LONWORKS Host Application Programmer's Guide*. The *Network Disable Option* affects whether or not the LTS-20 can send and receive application messages. This option is described in Chapter 7 under *Initializing an SLTA.*

The buffer configuration parameters can be changed at any time by sending *Write Memory* network management messages to the LTS-20, either from a host (using local network management messages) or over the network from a network management tool. See the *Neuron Chip Data Book*, Appendix A, for details of the data structures within the Neuron Chip that control the partitioning of RAM for buffers.

The following table summarizes the memory usage of the default configuration. The table also lists the maximum size of the buffer memory pool. If the LTS-20 is configured to use more bytes than are available in the pool, it will most likely crash or behave erratically since the remaining RAM is used by the system firmware.

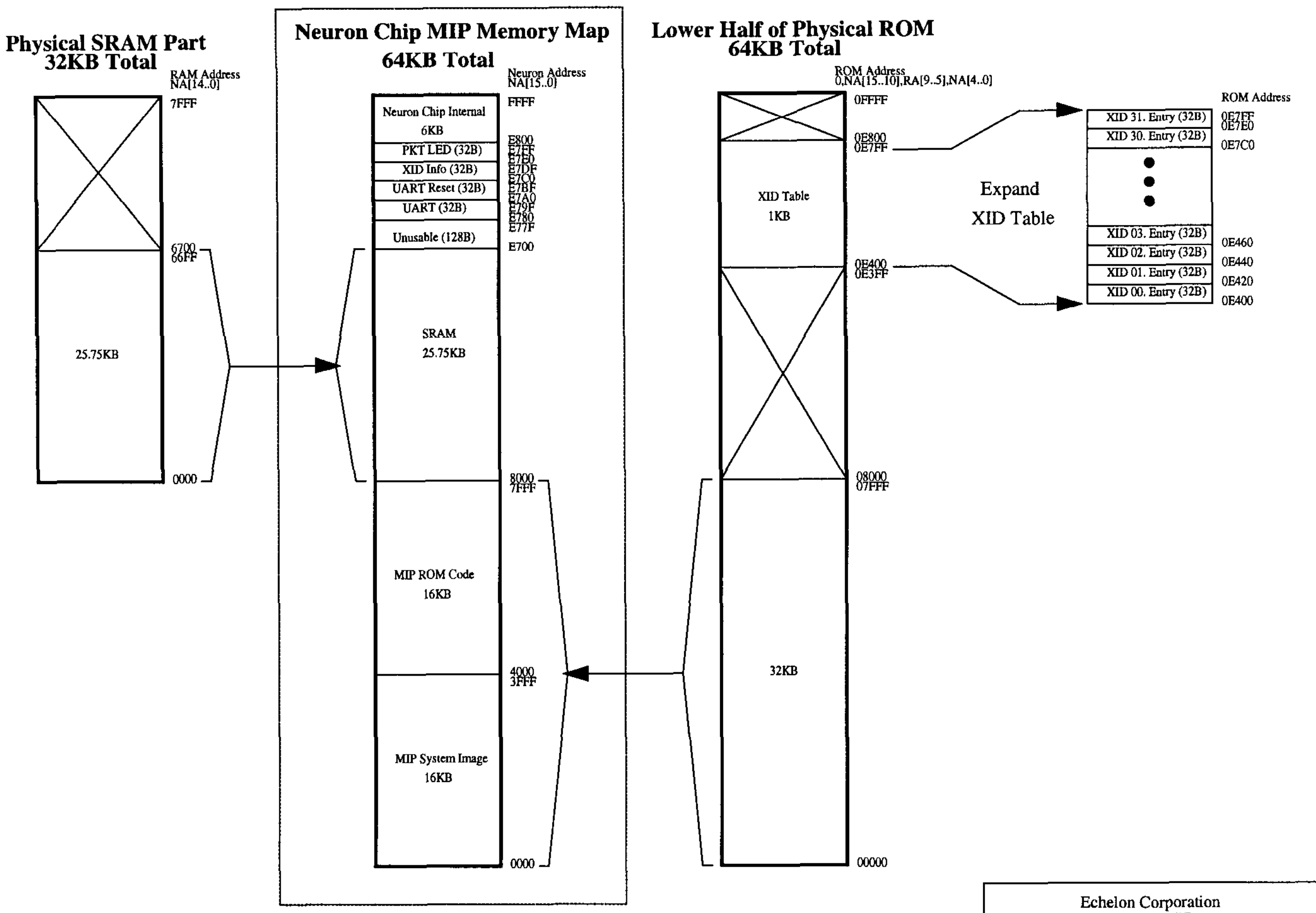
The default MIP mode EEPROM configuration settings for the LTS-20 are as follows:

| ***Configuration Parameters*** | ***Default Setting*** |
|---|---|
| Initial State | Unconfigured |
| Explicit addressing | Enabled[1] |
| Network variable processing | Host Selection[1] |
| Program ID string | "SLTA" |

| ***Buffer Parameter*** | ***Default Count*** | ***Default Size*** | ***Default Total*** |
|---|---|---|---|
| Receive transaction buffers | 8 | 13[2] | 104 |
| Transmit transaction buffers | 2 | 28[1,2] | 56[1] |
| Application input buffers | 15 | 255 | 3825 |
| Application output buffers | 7 | 255 | 1785 |
| Network input buffers | 31 | 255 | 7905 |
| Network output buffers | 2 | 255 | 510 |
| Priority app. output buffers | 5 | 255 | 1275 |
| Priority net. output buffers | 2 | 255 | 510 |
| **Total bytes used for buffers** | | | **15,970** |

[1]These values apply to LTS-20 Neuron Chip application version 7 only
[2]These values are fixed and cannot be modified

The amount of RAM memory available for buffers in the MIP mode is 25.75 bytes. This total includes both on-chip and off-chip RAM. When calculating the total RAM requirement for a given configuration, remember that there will be a fragmentation boundary when going to the off-chip RAM as buffers are built. This fragmentation may be up to a single buffer size in unusable RAM.



parent = .\lts-20_C.dsn\Info_Pages\Info_Pages.sch
this page = .\lts-20_C.dsn\MIP_Memory_Map\MIP_Memory_Map.sch
last page update = (0621)30Jun1999mb

Echelon Corporation
4015 MIRANDA AVE
PALO ALTO, CA 94304
PH: (650) 855-7400
FAX: (650) 856-6153

| | | |
|---|---|---|
| Title | LTS-20 / PSG-20: MIP Mode Memory Map | |
| Size | Document Number 012-1193-01 | Rev C |
| Date: | Wednesday, June 30, 1999 | Sheet 4 of 10 |

**Figure 2.5** LTS-20 MIP Mode Memory Map

The default NSI mode EEPROM configuration settings for the LTS-20 are as follows:

| ***Configuration Parameters*** | ***Default Setting*** |
|---|---|
| Initial State | Unconfigured |
| Explicit addressing | Enabled[1] |
| Network variable processing | Host Selection[1] |
| Program ID string | "SLTA" |

| ***Buffer Parameter*** | ***Default Count*** | ***Default Size*** | ***Default Total*** |
|---|---|---|---|
| Receive transaction buffers | 16 | 13[2] | 208 |
| Transmit transaction buffers | 2 | 28[1,2] | 56[1] |
| Application input buffers | 3 | 255 | 765 |
| Application output buffers | 3 | 255 | 765 |
| Network input buffers | 2 | 66 | 132 |
| Network output buffers | 2 | 66 | 132 |
| Priority app. output buffers | 3 | 255 | 765 |
| Priority net. output buffers | 2 | 66 | 132 |
| **Total bytes used for buffers** | | | **2,955** |

[1]These values apply to LTS-20 Neuron Chip application version 7 only
[2]These values are fixed and cannot be modified

The amount of RAM memory available for buffers in the NSI mode is 2,955 bytes. This total includes both on-chip and off-chip RAM. When calculating the total RAM requirement for a given configuration, remember that there will be a fragmentation boundary when going to the off-chip RAM as buffers are built. This fragmentation may be up to a single buffer size in unusable RAM.

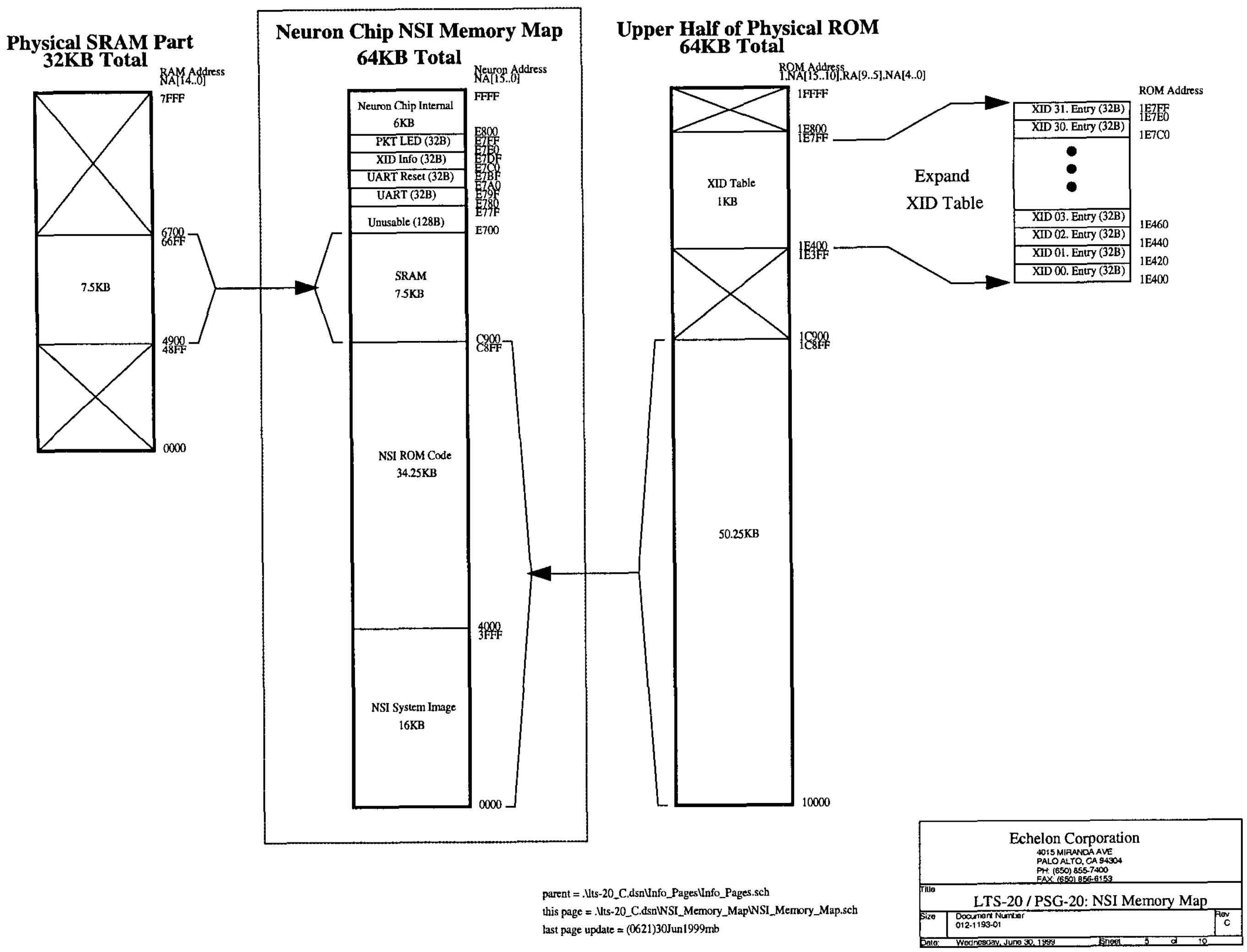


**Figure 2.6** LTS-20 NSI Memory Map

The NODEUTIL node utility application available from the Developer's Toolbox on the Echelon web site (www.echelon.com) can be used to modify the buffer configuration from a PC host. See the README.TXT file included with NODEUTIL for details.

# 3

# Developing an SLTA with the LTS-20 module

This chapter describes the process of developing a Serial LonTalk Adapter based on the LTS-20 Module.

## Overview

To create a complete serial interface (SLTA), with functions similar to Echelon's SLTA-10 Serial LonTalk Adapter, based on the LTS-20 Module, follow these steps:

1. Build an SLTA motherboard according to the specifications described in Chapter 2 and the guidelines described in Chapter 4. The motherboard may be part of custom application hardware, or may be a standalone board. Figure 3.1 is a sample motherboard schematic for an SLTA based on the use of the SMX™ transceivers. Additional transceiver interfaces are described in the rest of this chapter.
2. Ensure that the communications parameters in the LTS-20 are compatible with the transceiver. The transceivers listed in table 2.3 are supported directly by the LTS-20 as predefined types. Set the transceiver ID lines to select the proper transceiver type. For custom transceivers, modify the communications parameters as described under *Using Custom Transceivers* in this chapter.
3. Install the SLTA on a network as described in Chapter 7. The network may be a development network for initial testing, a manufacturing network for configuration during manufacture, or a production network for field installation.

## Using Predefined Transceivers

The LTS-20 includes pre-defined transceiver parameters for the transceivers listed in table 2.3. When using any of these transceivers, the communications parameters are automatically programmed as described in Chapter 2.

The following sections describe the hardware interface for standard LONWORKS transceivers available from Echelon for twisted pair, link power, and power line communications. The user's guide for each transceiver contains documentation on the interface requirements. The following sections provide additional information on using these transceivers with the LTS-20.

### *TPT/XF-78 and TPT/XF-1250 Twisted Pair Transceivers*

The TPT/XF-78 and TPT/XF-1250 Twisted Pair Transceiver Modules support transformer-isolated communications over a twisted pair cable. The transceiver ID should be set to 1 for the TPT/XF-78, and to 3 for the TPT/XF-1250.

See the *LONWORKS TPT Twisted Pair Transceiver Module User's Guide* for details on these channel types.

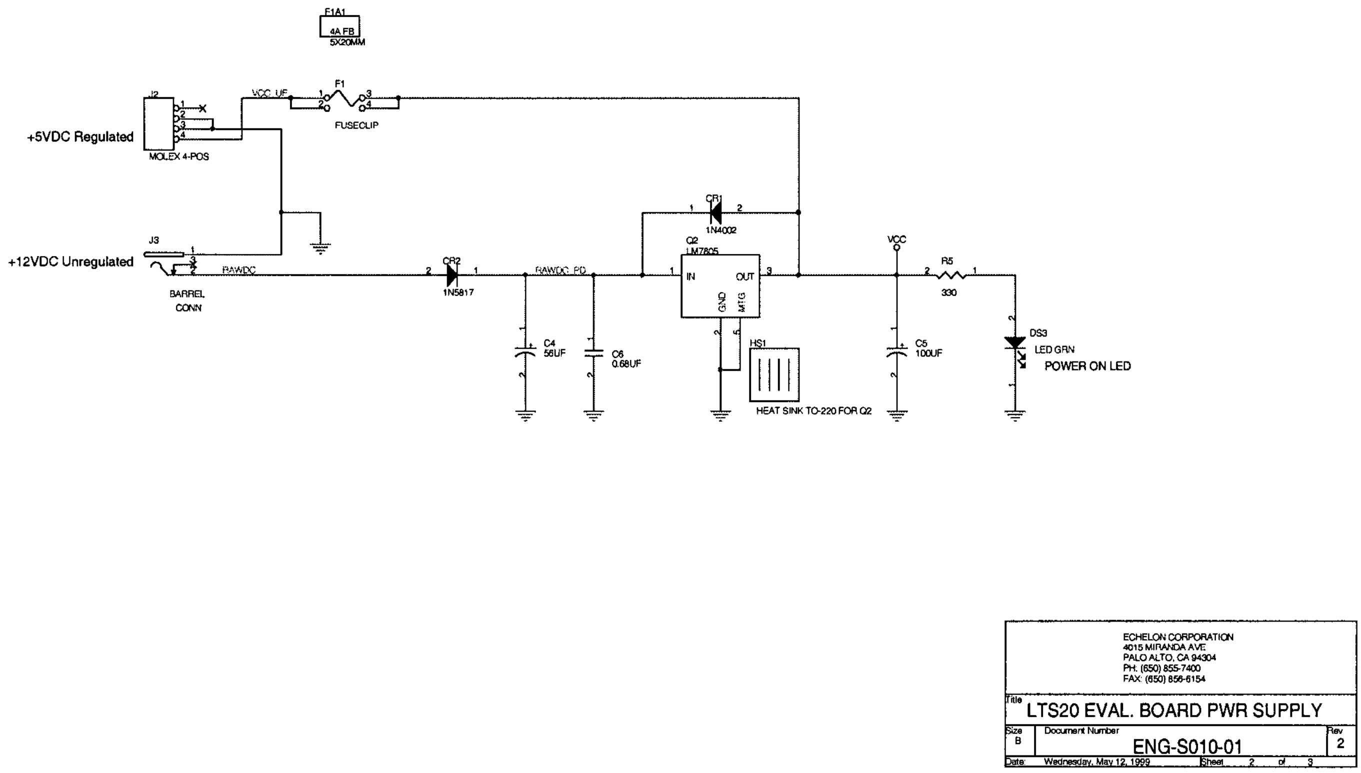
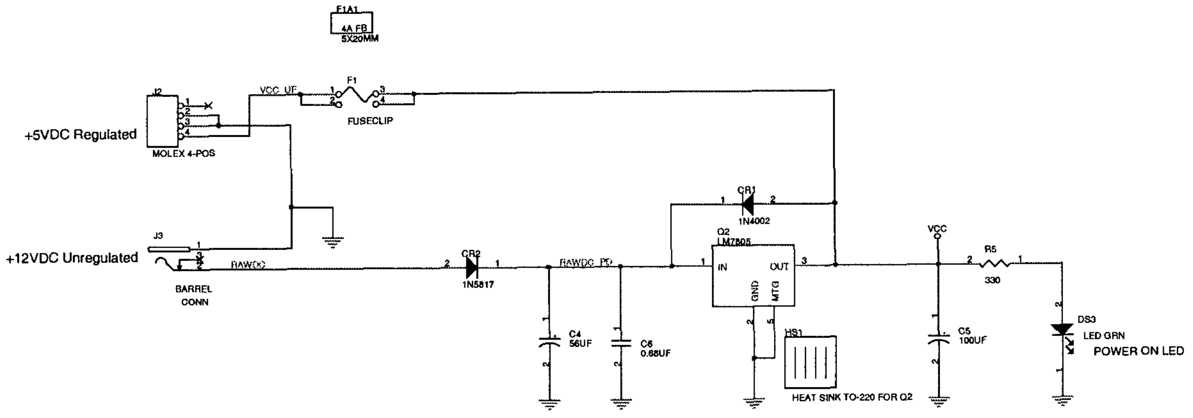


**Figure 3.1** LTS-20 Evaluation Board



| | | |
|---|---|---|
| ECHELON CORPORATION<br>4015 MIRANDA AVE<br>PALO ALTO, CA 94304<br>PH: (650) 855-7400<br>FAX: (650) 856-6154 | | |
| Title LTS20 EVAL. BOARD PWR SUPPLY | | |
| Size B | Document Number ENG-S010-01 | Rev 2 |
| Date: Wednesday, May 12, 1999 | Sheet 2 of 3 | |

**Figure 3.2** LTS-20 Evaluation Board Power Supply

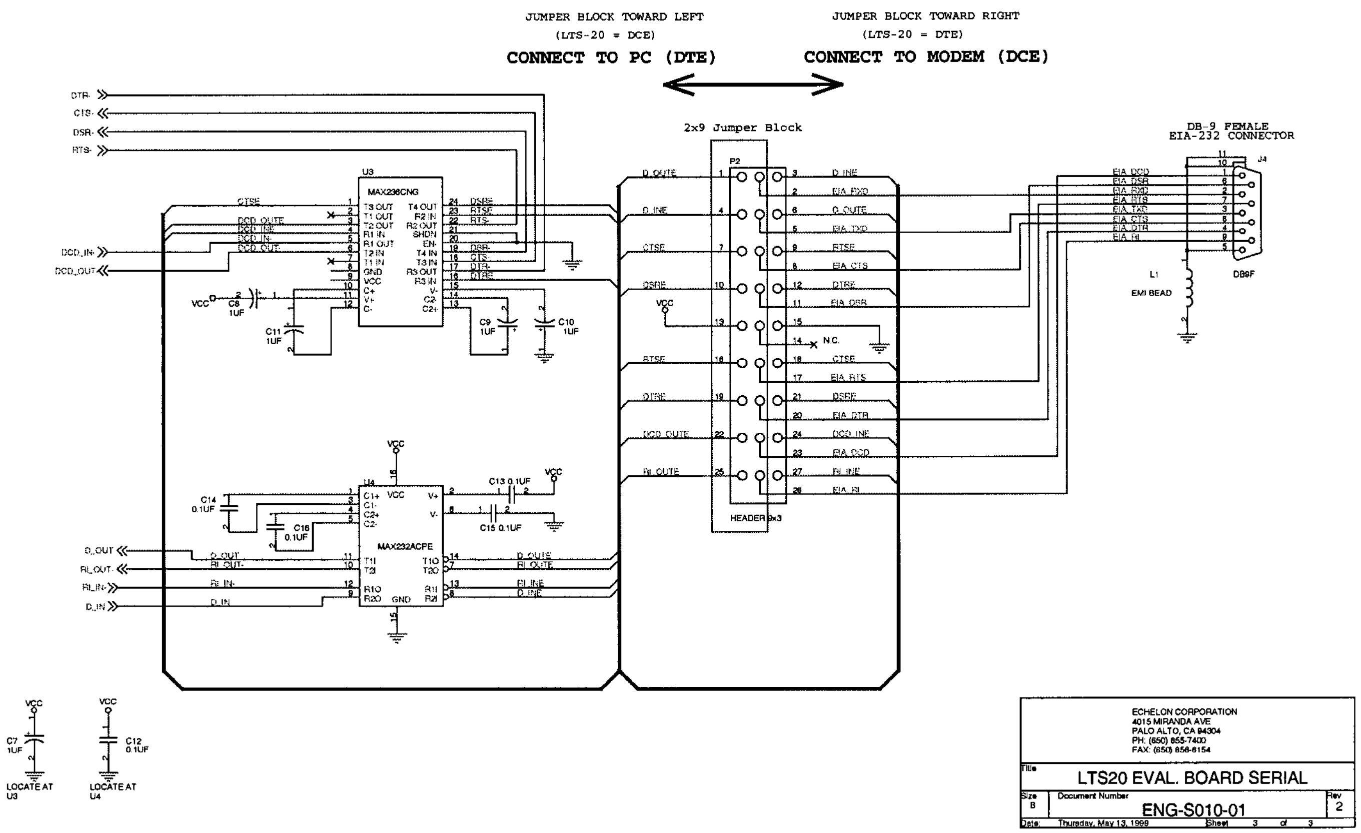


**Figure 3.3** LTS-20 Evaluation Board Serial

## FTT-10A Free Topology and LPT-10 Link Power Transceivers

The FTT-10A Free Topology Transceiver provides 78kbps signaling without regard for cabling topology, and is by far the most popular twisted pair medium for LONWORKS networks. The LPT-10 Link Power Transceiver Module supports free topology communications over the same twisted pair cable that carries power for application nodes. Power is supplied from a 48VDC power supply and is coupled to the network via an LPI-10 Link Power Interface Module. Both a power supply and an LPI-10 module are required to operate LPT-10 transceivers. The LPT-10 transceiver does not provide sufficient power for the LTS-20, which must be locally powered and optically isolated from the LPT-10 transceiver. The transceiver ID input must be set to 4 to support the LPT-10 and FTT-10A transceivers.

Note that an FTT-10A transceiver equipped with decoupling capacitors can operate on a link power segment, but an LPT-10 transceiver cannot operate on an unpowered FTT-10A segment.

## PLT Power Line Transceiver

A PLT Power Line Transceiver Module supports communications over AC or DC power mains. It may be connected to the LTS-20 module and a coupling circuit as shown in figure 3.2. The transceiver ID input must be set to support the correct PLT transceiver. See the pertinent LONWORKS PLT power line transceiver module user's guide for additional information, including a description of the coupling circuits.

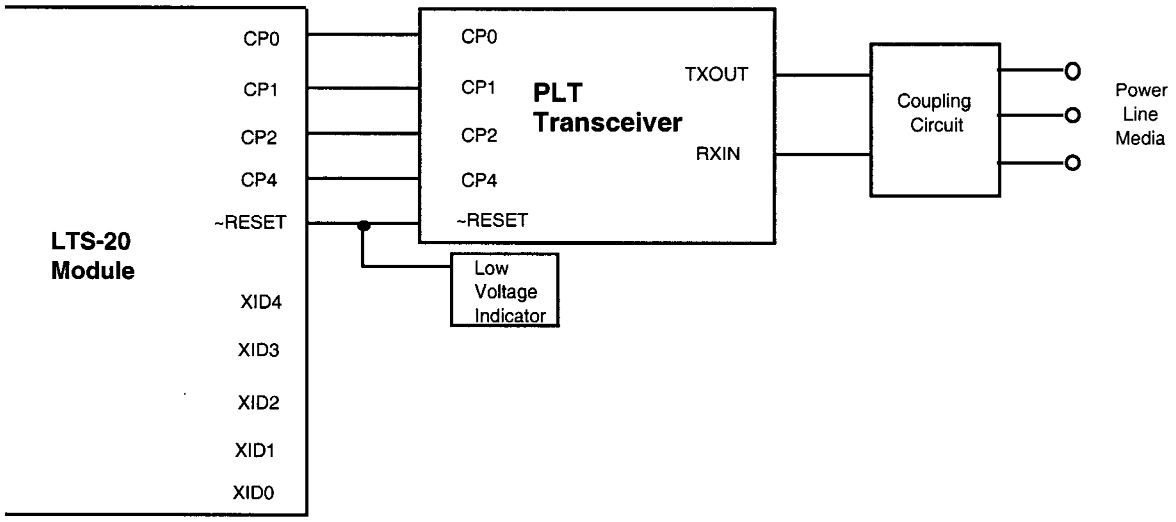


**Figure 3.4** Sample PLT Power Line Transceiver Interface

## Using Custom Transceivers

The LTS-20 module can be used with transceivers not listed in table 2.3 as long as the communications parameters are programmed to match the custom transceiver. Since network communication is not possible before these parameters are set, they must be programmed by the host over the EIA-232 link. The steps for programming a custom transceiver type are:

1 Determine the appropriate transceiver parameters for your channel. A discussion of transceiver modes and parameters may be found in Chapter 6 and Appendix A, section 6 of the *Neuron Chip Data Book*. Transceiver parameters may be modeled and fine-tuned using LonBuilder.

2 Select a transceiver ID of 30 (custom) on the LTS-20 transceiver ID inputs. The pins should remain set to this value in the production SLTA.

3 Install the transceiver parameters using a network management tool such as the LonMaker for Windows Integration Tool. The transceiver parameters are programmed into non-volatile EEPROM so the module will retain the new parameters after power is removed.

# 4

# LTS-20 Design Issues

This chapter examines a number of design issues, including a discussion of electromagnetic interference (EMI) and electrostatic discharge (ESD). These issues should be considered when designing hardware based on the LTS-20 module.

## EMI Design Issues

The high-speed digital signals associated with microcontroller designs can generate unintentional Electromagnetic Interference (EMI). High-speed voltage changes generate RF currents that can cause radiation from a product with a length of wire or piece of metal that can serve as an antenna.

Products that use the LTS-20 module will generally need to demonstrate compliance with EMI limits enforced by various regulatory agencies. In the USA, the FCC requires that unintentional radiators comply with Part 15 level "A" for industrial products, and level "B" for products that can be used in residential environments. Similar regulations are imposed in most countries throughout the world.

Echelon has designed the LTS-20 module with low enough RF noise levels for design into level "B" products. This section describes design considerations to enable products based on the core modules to meet EMI regulations.

## Designing Systems for EMC (Electromagnetic Compatibility)

The LTS-20 module has been designed so that products using them should be able to meet both FCC and, based on radiated emissions, EN55022 level "B" limits. Careful system design is important to ensure that a product based on the core modules will achieve the desired EMC. Information on designing products for EMC is available in several forms including books, seminars, and consulting services. This section provides useful design tips for EMC.

### *EMC Design Tips*

- Most of the EMI will be radiated by the network cable and the power cable.
- Filtering is generally necessary to keep RF noise from getting out on the power cable.
- EMI radiators should be kept away from the LTS-20 module to prevent internal RF noise from coupling onto the radiators.
- The LTS-20 module must be well grounded.
- Early EMI testing of prototypes at a certified outdoor range is an extremely important step in the design of level "B" products. This testing ensures that grounding and enclosure design questions are addressed early enough to avoid most last-minute changes.

## ESD Design Issues

Electrostatic Discharge (ESD) is encountered frequently in industrial and commercial use of electronic systems. Reliable system designs must consider the effects of ESD and take steps to protect sensitive components. Static discharges occur frequently in low-humidity environments when operators touch electronic equipment. The static voltages generated by humans can easily exceed 10kV. Keyboards, connectors, and enclosures provide paths for static discharges to reach ESD sensitive components such as the Neuron Chip. This section describes techniques to design ESD immunity into products based on the LTS-20 modules.

## Designing Systems for ESD Immunity

ESD hardening includes the following techniques:

- Provide adequate creepage and clearance distances to prevent ESD hits from reaching sensitive circuitry;
- Provide low impedance paths for ESD hits to ground;
- Use diode clamps or transient voltage suppression devices for accessible, sensitive circuits

The best protection from ESD damage is circuit inaccessibility. If all circuit components are positioned away from package seams, the static discharges can be prevented from reaching ESD sensitive components. There are two measures of "distance" to consider for inaccessibility: creepage and clearance. *Creepage* is the shortest distance between two points along the contours of a surface. *Clearance* is the shortest distance between two points through the air. An ESD hit generally arcs farther along a surface than it will when passing straight through the air. For example, a 20 kV discharge will arc about 0.4 inches (10 mm) through dry air, but the same discharge can travel over 0.8 inches (20mm) along a clean surface. Dirty surfaces can allow arcing over even longer creepage distances.

When ESD hits to circuitry cannot be avoided through creepage, clearance, and ground guarding techniques, i.e., at external connector pins, explicit clamping of the exposed lines is required to shunt the ESD current. Consult *Protection of Electronic Circuits from Overvoltages*, by Ronald B. Standler, for advice about ESD and transient protection for exposed circuit lines. In general, exposed lines require diode clamps to the power supply rails or zener clamps to chassis ground in order to shunt the ESD current to ground while clamping the voltage low enough to prevent circuit damage. The Neuron Chip's communications port lines are connected directly to the LTS-20 edge connector without any ESD protection beyond that provided by the chip itself. If these lines will be exposed to ESD in a custom SLTA, protection must be added to the motherboard.

# 5

# The LTS-20 Software

This chapter describes the LTS-20 software that is shipped with the Connectivity Starter Kit.

## Software Overview

The LTS-20 software includes ANSI C source code for HA, a sample host application for MS-DOS that can be used as a basis for a user-developed host application on other host platforms. This application provides examples of sending and receiving network variable messages, as well as allowing a node based on an LTS-20 to be installed and bound by a network management tool such as the LonManager LonMaker for Windows Integration Tool or the LonBuilder network manager.

Two network drivers (Windows 95/98 and Windows NT) are included so that an LTS-20 may be immediately used with LNS applications. Source code for DOS and UNIX network drivers is also provided as a basis for a user-developed network driver for other hosts or operating systems using the MIP. DLL software is provided to make it easier to use the network driver under the Microsoft® Windows operating system.

An executable program and source code is also provided for a Host Connection Utility (HCU), which may be used to initiate and terminate the host to serial connection when the LTS-20 is used with a remote host. An example written in Neuron C is also provided as a basis for user-developed nodes on a LONWORKS network that need to initiate outgoing calls to a remote host.

The LTS-20 includes NSI firmware that moves the upper layers of the LonTalk Protocol off the Neuron Chip within a node onto a host processor. This firmware allows the LTS-20 to be used by a host application to send and receive LonTalk messages. The host application may be a custom application as described in the *LNS for Windows Developer's Kit or LNS DDE Server User's Guide*. When using the LTS-20 in the MIP mode, the host application may also be a network management application based on tools using the now-discontinued LonManager API. The firmware in an LTS-20 is fixed in ROM and need not be reprogrammed to use any of the module's capabilities.

## Installing LTS-20 Software

The LTS-20 software is supplied on a diskette, together with an installation program. To install the LTS-20 software, follow these steps:

1. Place the diskette in one of the disk drives of your PC. This will typically be the `A:` or B: drive.

2. Start the automatic installation procedure by entering:

```
A:INSTALL [ENTER]
```

Substitute your disk drive name for the A: if you are using a different drive.

3. You will be asked to enter the name of your LONWORKS installation directory. The default is:

```
C:\ECHELON
```

If you have other Echelon software products installed in the \LONWORKS directory, rather than the \ECHELON directory, enter \LONWORKS in place of the default directory name.

The LTS-20 software will be installed in the LTS-20 sub-directory of your LONWORKS directory, with the exception of the DOS network driver LDVSLTA.SYS. This file will be installed in the BIN sub-directory of your LONWORKS directory. To install the DOS network driver into your CONFIG.SYS file, follow the instructions in Chapter 9.

The SLTA directory will contain the following files:

- **Read-Me File**. The README.TXT file includes a list of all the files on the distribution disk, and also includes any updates to the documentation that occurred since the documentation was printed.
- **DOS Network Driver Sources**. The DOS network driver source code is contained in the LDVSLTA directory. These files can be used as the basis for creating drivers for hosts other than PCs running DOS (see also the UNIX network driver sources). See the README.TXT file for a description of the driver files. See Chapter 8 for a description of the DOS network driver and Chapter 7 for a description of how to write a network driver for other hosts. See Chapter 4 of the *LONWORKS Host Application Programmer's Guide* for a description of the services that must be supplied by a LONWORKS network driver.

  The source files to build the DOS driver are:

| | |
|---|---|
| LDVSLTA.CFG | Configuration file for Borland C. |
| MAKEFILE | Make file script for Borland C. |
| MDV_TIME.C | Code to manage the PC timer. |
| MDV_TIME.H | External interface definitions for the timer handler. |
| MSD_DEFS.H | Data structure and literal definitions. |
| MSD_DIFC.C | DOS driver interface functions. |
| MSD_DRVR.H | DOS driver interface and literal definitions. |
| MSD_EXEC.C | Main open, close, read, and write processing. |
| MSD_FRST.C | Module to be linked first in the network driver. |
| MSD_IRQC.ASM | Serial I/O interrupt procedure. |
| MSD_LAST.C | Module to be linked last in the network driver. |
| MSD_RAW.C | Direct serial I/O (modem) processing. |
| MSD_SEGD.ASM | Defines data segment register for driver. |
| MSD_SIO.C | PC/AT UART interface processing. |
| MSD_TXRX.C | Single byte link layer processing. |
| MSD_UART.H | Defines PC/AT UART registers. |

- **UNIX Network Driver Sources**. The UNIX network driver source code is contained in the UNIX directory. These files can be used as the basis for creating drivers for any UNIX host, and can also be used as the basis for developing drivers for other hosts. See Chapter 10 for a description of the UNIX network driver and Chapter 8 for a description of how to write a network driver for other hosts. See Chapter 4 of the *LONWORKS Host Application Programmer's Guide* for a description of the services that must be supplied by a LONWORKS network driver. The source files to build the UNIX driver are:

| | |
|---|---|
| `LDVSLTA.C` | UNIX driver functions. |
| `LDVSLTA.H` | UNIX driver declarations. |

- **External Interface Files**. External interface files included for use by network management tools are contained in the `LTS-20` directories. Fifteen external interface files are included for the standard transceiver types that are directly supported by the LTS-20. See *Binding to a Host Node* in Chapter 3 of the *LONWORKS Host Application Programmer's Guide* for a description of how to use these files to bind to an SLTA node. Appendix B of the *LONWORKS Host Application Programmer's Guide* provides a detailed description of how to modify these files to incorporate network variables and message tags. These interface files are provided in version 3 formats; version 2 formats are available by running the utility `XIF3TO2.EXE` (available from Echelon's ftp site) on the version 3 XIF files. Version 3 external interface files are compatible with the latest releases of all Echelon software products. External interface files in version 3 format are contained in the `SLTA2\XIF_V3` and `LTS-20\XIF_V3` directories.

  Each SLTA/2 directory contains the following files:

| | |
|---|---|
| `NSLTA125.XIF` | For SLTA/2 with a TP/XF-1250 transceiver. |
| `NSLTA78K.XIF` | For SLTA/2 with a TP/XF-78 transceiver. |
| `NSLTA485.XIF` | For SLTA/2 with a TP-RS485-39 transceiver. |
| `NSLTAFT1.XIF` | For SLTA/2 with a TP/FT-10 transceiver. |

  Each LTS-20 directory contains the following files:

| | |
|---|---|
| `LTS1250.XIF` | For LTS-20s with a TP/XF-1250 transceiver. |
| `LTS78K.XIF` | For LTS-20s with a TP/XF-78 transceiver. |
| `LTS485A.XIF` | For LTS-20s with a TP-RS485-39 transceiver. |
| `LTS485B.XIF` | For LTS-20s with a TP-RS485-78 transceiver. |
| `LTS485C.XIF` | For LTS-20s with a TP-RS485-625 transceiver. |
| `LTS485D.XIF` | For LTS-20s with a TP-RS485-1250 transceiver. |
| `LTSFT10.XIF` | For LTS-20s with a FT-10 or LPT-10 transceiver. |
| `LTSPL10.XIF` | For LTS-20s with a PL-10 transceiver. |
| `LTSPL20A.XIF` | For LTS-20s with a PL-22 transceiver (A-band). |
| `LTSPL20C.XIF` | For LTS-20s with a PL-21/PL-22 transceiver (CENELEC protocol on). |
| `LTSPL20N.XIF` | For LTS-20s with a PL-21/PL-22 transceiver (CENELEC protocol off). |
| `LTSPL30.XIF` | For LTS-20s with a PL-30 transceiver. |
| `LTSFO10.XIF` | For LTS-20s with a FO-10 transceiver. |

| | |
|---|---|
| `LTSDC78.XIF` | For LTS-20s using Direct Connect at 78kbps. |
| `LTSDC625.XIF` | For LTS-20s using Direct Connect at 625kbps. |
| `LTSDC125.XIF` | For LTS-20s using Direct Connect at 1250kbps. |

- **Sample Host Application**. A sample host application is contained in the HA directory. See Appendix A of the *LONWORKS Host Application Programmer's Guide* for a description of the example. The following files are included:

| | |
|---|---|
| `README.TXT` | A description of the sample host application. |
| `HA.EXE` | An executable version of the sample host application for DOS. The SLTA DOS network driver must be installed to run this application. |
| `HA.C` | The main program for the example. |
| `NI_MSG.C` | A general purpose network interface library that can be used with any host application. |
| `APPLCMDS.C` | Functions to handle application layer network variable commands |
| `NI_CALLB.C` | The host-bound network management dispatcher. |
| `APPLMSG.H` | Application message handler function prototypes. |
| `HA_COMN.H` | The HA common declarations. |
| `NI_CALLB.H` | The definitions for the network management dispatcher. |
| `APPLMSG.C` | Functions to handle application network variable and explicit messages. |
| `HAUIF.C` | Command-line user interface for the example. |
| `IOCTL.C` | I/O control function for Microsoft C. |
| `LDVINTFC.C` | Device interface driver. |
| `LDVINTFC.H` | Include file for device driver interface. |
| `NI_MSG.H` | Definitions for network interface message structures. |
| `NI_MGMT.H` | Definitions for network management message structures used by the example. |
| `HAUIF.H` | Definitions for the host application example user interface. |
| `MAKEFILE` | A make file script for Borland C. |
| `MSOFT.MAK` | A make file script for Microsoft C. |
| `HA_V3.XIF` | An external interface file which may be used to bind the example with LonBuilder. |
| `HA_TEST.NC` | A Neuron C program which may be loaded into a Neuron emulator and bound to the sample host application for testing. |

| | |
|---|---|
| `DISPLAY.H` | A Neuron C include file to drive the Gizmo 2 I/O module for the test example. |

- **Host Connect Utility**. A sample host connection utility is contained in the `HCU` directory, with source code. See Chapter 12 for details. The files supplied are:

| | |
|---|---|
| `HCU.EXE` | Executable file for the Host Connection Utility. |
| `HCU_MAIN.C` | The main C source program. |
| `HCU.CFG` | Configuration file for Borland C. |
| `MAKEFILE` | Make file script for Borland C. |
| `MSD_DRVR.H` | Driver definition include file. |

- **Neuron C Connection Example**. A sample Neuron C program is contained in the `NC_APPS` directory. This program shows how a node on a network connected to the SLTA can dial out and connect to a remote host computer. See Chapter 11 for details. The files supplied are:

| | |
|---|---|
| `DIALOUT.NC` | Neuron C source program to dial out with the SLTA. |
| `GIZSETUP.NC` | An example Neuron C program for configuring the SLTA. Configures the EEPROM directories of an SLTA using the Gizmo 2 I/O module as the user interface. |
| `SLTA_ANM.H` | Definitions of SLTA-specific network management messages. |

- **Manufacturing Test Files**. The files supplied in the `LTS-20\MFT` directory provide a Neuron C application example which can be used as a manufacturing test aid for products based on the LTS-20. They are:

| | |
|---|---|
| `LTSMFT.NC` | Neuron C source file, including full documentation. |
| `LTSMFT.H` | Include file. |

This application is designed to aid in the testing of circuitry that is external to the LTS-20 module, such as EIA-232 interface drivers and connectors. It may be programmed into a LONWORKS device which then communicates with the LTS-20 module via the network. The LTS-20 circuitry is tested with some of its signals connected in a loopback manner. The assertion of the TEST input (pin 30) will cause the LTS-20 firmware to come up in the test mode.

## *Installing the Windows DLL Software*

A second diskette contains the Windows Dynamic Link Library (DLL) files. These files may be used when developing a host application to run under Microsoft Windows. The file `WLDV.DLL` should be copied to your Windows directory (typically `C:\WINDOWS`). The files `LDV.H` and `LON.H` should be copied to a directory in the include file search path of your C compiler. The file `WLDV.LIB` should be copied to a directory in the library search path of your application linker. See Appendix B for information on using the Windows DLL.

# 6
# Creating an LTS-20 MIP Mode Network Driver

This chapter describes the process of building a network driver for a host that is to be connected to an LTS-20 operating in MIP mode. The example network drivers for DOS, Windows, and UNIX are described. Similar logic can be used on other host processors and operating systems. This chapter also includes a description of the network interface protocol for the LTS-20 operating in MIP mode. The network interface protocol defines the format of the data passed across the EIA-232 interface, and varies depending on the configuration of the LTS-20 and the network driver. If a LONWORKS standard network driver is used, the format of the data passed between the driver and the application is defined by the network driver protocol and is independent of the network interface protocol; the driver is responsible for providing the necessary translations. This chapter will therefore be of interest only to those needing to develop a network driver for a host other than DOS, Windows, or UNIX.

*If you are using a DOS, Windows, or Unix host, you can skip this chapter and instead read Chapters7 or 8, which describe the DOS and UNIX network drivers.*

## Purpose of the Network Driver

The network driver provides a hardware-independent interface between the host application and the network interface. By using network drivers with consistent calling conventions, host applications can be transparently moved between different network interfaces. For example, the standard LTS-20 MIP mode DOS driver, together with the Windows DLL software, allows DOS and Windows applications, such as those based on the LonManager API, to be debugged using the network driver for the LonBuilder Development Station. These applications can later be used with the network driver for the SLTA-10 operating in MIP mode, without modifying the host application.

**For the purposes of this chapter, the term "SLTA" refers to the LTS-20 module operating in MIP mode.**

A LONWORKS standard network driver must supply the functions defined under *Network Driver Services* in Chapter 4 of the *LONWORKS Host Application Programmer's Guide.* The Windows DLL software is described in Appendix B.

## Example Network Drivers

The SLTA is delivered with source code for example network drivers for DOS, Windows, and UNIX. The DOS driver is used for both DOS and Windows applications. See the comments in the source code of the network drivers for an explanation of how the network drivers work. These drivers can be used as templates for a LONWORKS standard network driver. The DOS network driver is compatible with the LonManager APIs for DOS and Windows, LonMaker, and the LonManager DDE Server. A sample host application for DOS is also supplied. The functions `ldv_open()`, `ldv_read()`, `ldv_write()`, and `ldv_close()` form a suitable operating-system independent definition for the network driver. These functions support multiple network interfaces, and hide the DOS-specific aspects of the DOS network driver.

The UNIX network driver is a source library that uses the UNIX serial device driver. It also supports the `ldv_open()`, `ldv_read()`, `ldv_write()`, and `ldv_close()` functions.

## Implementing an SLTA Network Driver

The network driver manages the physical interface with the SLTA, implements the network interface protocol, performs flow control, manages input and output buffers, and provides a read/write interface to the host application.

Figure 6.1 illustrates how the network driver fits into the host application architecture.

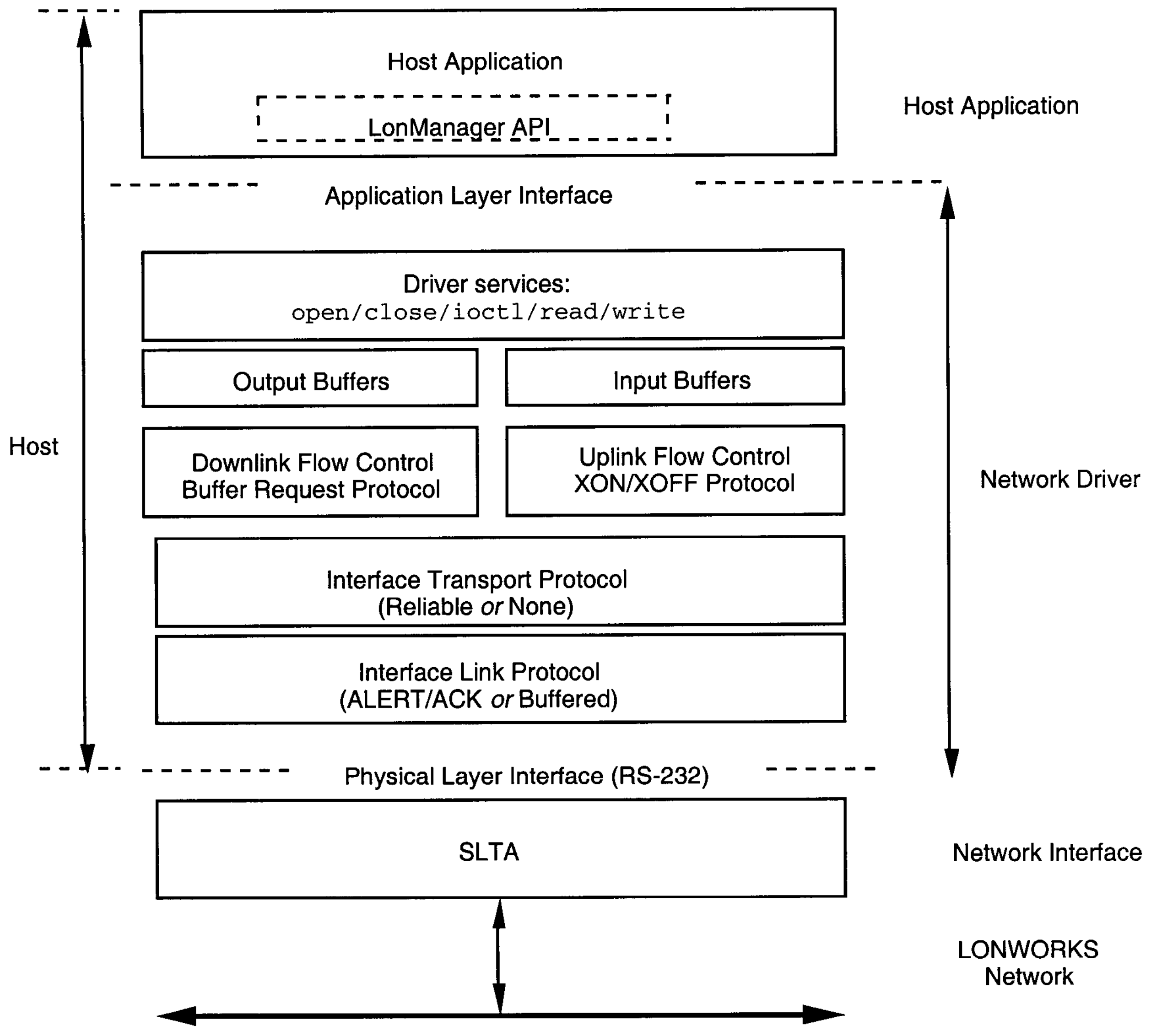


**Figure 6.1** Host Application Architecture

To implement an SLTA network driver for a host other than DOS, Windows, or UNIX, follow these steps:

1 Implement and test low-level serial I/O. Serial I/O may be performed directly to the host's UART as is done in the DOS network driver, or may be performed by a serial I/O driver on the host as is done by the UNIX network driver. Serial I/O should be interrupt driven for better performance.

The UNIX network driver uses the UNIX serial port driver for all low-level serial I/O and interrupt support. This simplifies the driver and also simplifies porting between different versions of UNIX. The serial device is opened by the `ldv_open()` function and closed by the `ldv_close()` function. Data is read

from and written to the serial device using the UNIX `read()` and `write()` system calls.

The UNIX network driver includes a `ldv_post_events()` function that should be called periodically from the client application in order to assure that the SLTA traffic is being processed.

The DOS network driver serial I/O functions are implemented by `MSD_SIO.C`, `MSD_UART.H`, and `MSD_IRQC.ASM`. These files may all be replaced as long as the required serial I/O functions in `MSD_SIO.C` are provided. The definitions of the UART registers are in `MSD_UART.H`. The DOS serial I/O interrupt service routines are in `MSD_IRQC.ASM`.

The DOS network driver uses the DOS system timer tick interrupt (vector 0x1C) and the serial I/O device interrupt for the relevant COM port to perform background processing of the serial network interface. The driver hooks into these interrupt vectors and executes driver code whenever the LON(n) device is opened. Flags internal to the driver prevent the interrupt code thread from interfering with the normal application foreground execution of functions within the driver.

The `smip_int_main()` function in the DOS network driver services the serial port connected to the network interface. The function `tick_int_main()` services the timer tick interrupt every 55 msec.

Both network drivers are fully buffered for both outgoing and incoming messaging. Read and write functions work with circular buffers within the driver. The host interrupt service routine handles the other ends of these buffer queues.

Both network drivers only support a single set of output buffers. An elaboration on this design could implement a set of priority output buffers. The write function could determine into which of the two buffer sets to place messages, and the driver service function could service the priority buffers first.

2 Implement and test timer support functions. Timer support may be provided by a hardware timer as is done in the DOS network driver, by a system service as is done in the UNIX network driver, or by implementing a background software task. The UNIX network driver uses a once per second signal that is handled by the `second_service()` function. The DOS timer functions are implemented by `MDV_TIME.C` and `MDV_TIME.H`.

3 Implement and test the host side of the network interface protocol. The network interface protocol is implemented by the `rx_process()` and `tx_process()` functions in the UNIX driver, and by the functions in `MSD_TXRX.C` for the DOS network driver.

4 Implement and test raw modem I/O if you need to support a modem interface. Raw I/O manages the serial interface to the modem when the modem is not connected to a host and is used for modem initialization and control. The raw I/O interface is implemented in `MSD_RAW.C` for the DOS network driver, and is not implemented in the UNIX network driver.

5 Implement and test the buffer request states, buffer management, and read/write interfaces. These functions are implemented by `MSD_EXEC.C` for the DOS

network driver. The read/write interface is implemented in the `ldv_read()` and `ldv_write()` functions for the UNIX network driver

The following files are unique to a DOS driver and would probably not be used in a port to another host: `MSD_DRVR.H`, `MSD_DIFC.C`, `MSD_FRST.C`, `MSD_LAST.C`, `MSD_SEGD.ASM`.

## Network Interface Protocol

The network driver implements the host side of the network interface protocol, providing an easy-to-use and interface-independent read/write interface to the host application. The network interface protocol is a layered protocol that includes the following layers:

- **Presentation Layer**. Defines packet formats for network variables and explicit messages. This is the only layer visible to the host application. The remaining layers are managed by the network driver.
- **Session Layer**. Manages flow control, buffer requests, and grants.
- **Transport Layer**. Ensures end-to-end reliability between the host and the SLTA.
- **Link Layer**. Controls access to the serial link.
- **Physical Layer**. EIA-232 interface.

The physical layer is defined by the EIA RS-232 standard. The link, transport, session, and presentation layers are described in the following sections.

## Link Layer Protocol

The default interface link layer protocol is the *ALERT/ACK protocol*. This protocol may be used when the host is a microcontroller or microprocessor such as a PC running DOS or Windows. The alternative interface link protocol is the *buffered protocol*. This protocol is used with computer hosts that can asynchronously buffer an entire packet. All data are transmitted using 1 start bit, 8 data bits, no parity bits, and 1 stop bit.

### *ALERT/ACK Link Protocol*

The DOS network driver uses the ALERT/ACK link protocol by default (i.e. the `/N` option is not specified). See Chapter 8 for a description of the network driver options. The UNIX network driver uses the ALERT/ACK link protocol if the `alert_ack_prtcl` variable is set to TRUE in the source code (this is not the default). The CFG3 input of the SLTA, as described in Chapter 6, must be in the *ALERT/ACK* state.

When using this protocol, all transfers between the SLTA and the host consist of serial data streams that start off with the link-layer header sequence described in figure 6.2. Whenever one device, either the SLTA or the host, needs to send a command or message, the sender starts the sequence by transmitting the ALERT byte (value 01 hex). When this byte is received by the receiver, that device responds by transmitting the ALERT ACK byte (value FE hex). This low level handshaking process prevents the sender from transmitting the rest of the sequence before the receiving device is ready. Once the ALERT ACK byte is received by the sender it sends the rest of the message without any other interactions.

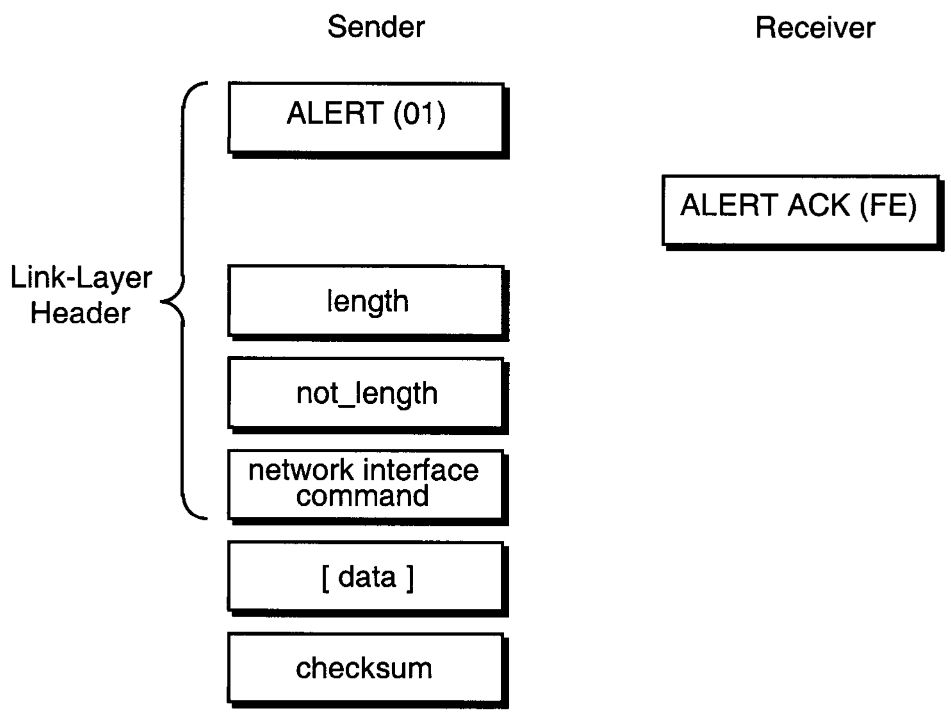


**Figure 6.2** SLTA ALERT/ACK Link Protocol

The link-layer header contains a length byte followed by a one's complement of the length byte. These values are always validated by the receiver before accepting the rest of the message. Following the length bytes is the network interface command. See Appendix D of the *Host Application Programmer's Guide* for a description of the command byte structure. If the message contains a data field it follows the command byte. Finally, a checksum terminates the sequence.

The length byte value describes the length of the network interface command byte plus the length of the data field. This value will always be at least 1. The checksum is a two's complement of the sum of the command byte and all of the bytes in the data field, if it exists. Checksum errors detected by the host will cause an error to be reported to the application, and checksum errors detected by the SLTA will cause the message to be ignored.

The SLTA places the following requirements on the rate of the received serial data stream. When receiving, the maximum wait period for the length byte following the transmission of the ALERT ACK byte is 100ms (or 1 second when attached to a modem). All subsequent bytes received must occur within 100ms after the previous byte, otherwise the SLTA receive process will abort. Likewise, the SLTA uses a wait period of 100ms (or 1 second when attached to a modem) before aborting for the reception of the ALERT ACK when transmitting a message. If the ALERT ACK is not received in time, the SLTA repeats the process by transmitting another ALERT byte.

The SLTA cannot support a full duplex communications process between it and the host. The network driver included with the SLTA takes this into account. Data frames transmitted to the SLTA while it is in the process of sending uplink messages will be lost if more than 16 bytes are sent to the SLTA.

## Buffered Link Protocol

The DOS network driver uses the buffered link protocol when the /N option is specified. See Chapter 7 for a description of this option. The UNIX network driver uses the buffered link protocol if the `alert_ack_prtcl` variable is set to FALSE in the source code (this is the default). The CFG3 input of the SLTA, as described in Chapter 6, must be in the buffered protocol state.

When using this protocol, the link-layer header contains a length byte followed by a one's complement of the length byte. These values are always validated by the receiver before accepting the rest of the message. Following the length bytes is the network interface command. See Appendix D of the *Host Application Programmer's Guide* for a description of the command byte structure. If the message contains a data field it follows the command byte. Finally, a checksum terminates the sequence.

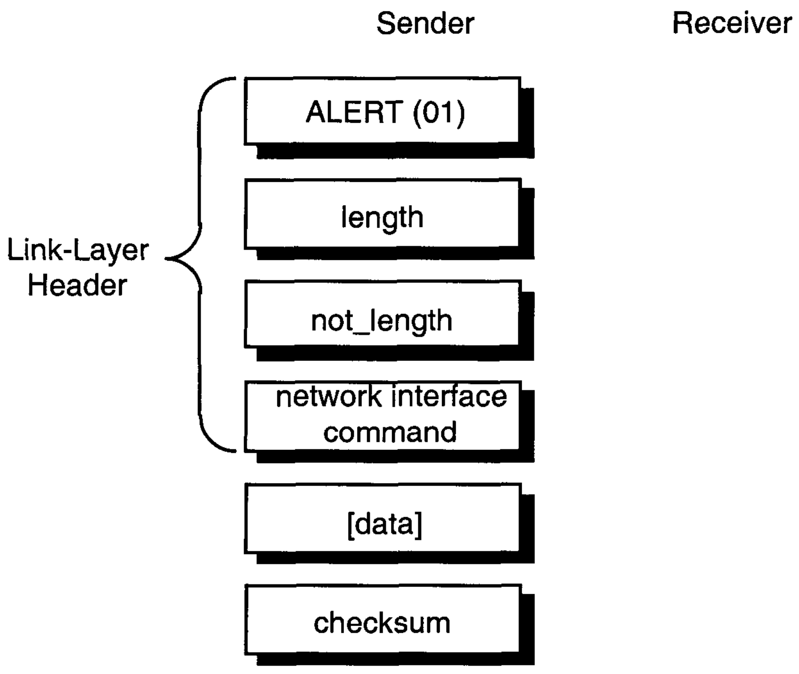


**Figure 6.3** SLTA Buffered Link Protocol

The length byte value describes the length of the network interface command byte plus the length of the data field. This value will always be at least 1. The checksum is a two's complement of the sum of the command byte and all of the bytes in the data field, if it exists. Checksum errors detected by the host will cause an error to be reported to the application, and checksum errors detected by the SLTA will cause the message to be ignored.

This protocol is used when the host is capable of accepting asynchronously occurring input data without losing characters. The host is also relieved of the obligation of responding to an ALERT character within 50 ms. This protocol may therefore be used by an application-level handler calling an interrupt-driven buffered serial device driver. Drivers with these characteristics are typically provided with real time operating systems such as VRTX or time-sharing operating systems such as UNIX or VMS. In this case, these drivers should be set up for binary data communications without software flow control.

The buffered link protocol should not be used when the SLTA is attached to a modem.

The buffered link protocol can only be used on multitasking operating systems such as UNIX if the host application executes often enough to empty any incoming buffers. For example, if the SLTA is receiving 70 packets per second, and each packet is 25 bytes, the host will receive 1750 bytes per second. If the host has a serial input buffer of 256 bytes, the buffer will fill within 150 milliseconds if the host application is preempted. If the host application is preempted for longer than 150 milliseconds, incoming data will be lost due to lack of serial buffer space. In this case, the ALERT/ACK protocol should be used, or the buffer space increased to handle the worst case traffic during the maximum preemption period.

## Transport Layer Protocol

When used with a local host, the SLTA assumes a reliable connection and does not use a transport layer protocol. When used with a remote host, the SLTA assumes that the link may not be reliable and enables the reliable transport protocol. The reliable transport protocol adds an ACK/NACK transport protocol to the network interface protocol. A sequence number is also added to the link-layer header. This protocol can therefore recover from checksum errors on the host to SLTA link.

The reliable transport protocol is enabled on the SLTA with the *Remote Host* option selected by the CFG2 input as described in Chapter 12. The reliable transport protocol is enabled on the DOS network driver with the `/M` option as described in Chapter 7. The reliable transport protocol is not supported by the UNIX network driver.

The link-layer header contains an ALERT (0x01) byte, a sequence number, and a length byte followed by a one's complement of the length byte. These values are always validated by the receiver before accepting the rest of the message. Following the length bytes is the network interface command. See Appendix D of the *Host Application Programmer's Guide* for a description of the command byte structure. If the message contains a data field it follows the command byte. Finally, a checksum terminates the sequence.

The ALERT/ACK link protocol should be used with remote hosts. With this protocol, the sender will start the sequence by transmitting the ALERT byte. When this byte is received by the receiver, that device responds by transmitting the ALERT ACK byte (value FE hex). This low level handshaking process prevents the sender from transmitting the rest of the sequence before the receiving device is ready. Once the ALERT ACK byte is received by the sender it sends the rest of the message without any other interactions.

The length byte value describes the length of the network interface command byte plus the length of the data field. This value will always be at least 1. The checksum is a two's complement of the sum of the command byte and all of the bytes in the data field, if it exists. If the receiver receives a message in sequence, with a valid checksum, it responds with an ACK (0x06). Otherwise it responds with a NACK (0x15), requesting a re-transmission.

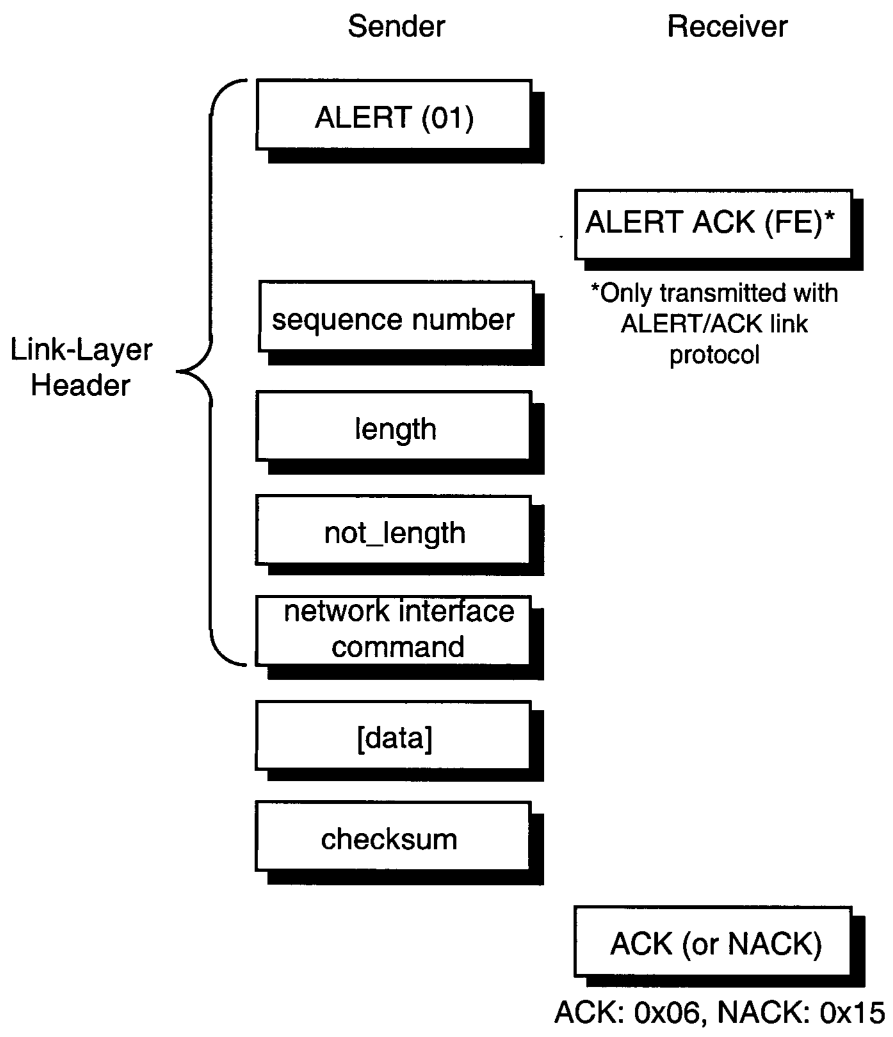


**Figure 6.4** SLTA Reliable Transport Protocol

## SLTA Timing Data

Certain aspects of the SLTA link and transport layer protocols implement fail-safe timeouts in order to control the time spent waiting for protocol states to change when errors occur. These timeouts are kept constant with either a 10MHz or 5MHz Neuron input clock.

### Downlink Byte-to-Byte Receive Timeout

The downlink byte-to-byte receive timeout is the maximum allowable period between the end of a single byte data frame sent downlink to the SLTA, to the end of the next single byte data frame sent downlink to the SLTA. This period is 100ms in local host mode and 1 second in remote host mode. When this timeout occurs, the SLTA discards the downlink buffer and returns to the NORMAL state. If the reliable transport protocol is enabled, the SLTA also sends a NACK byte after this timeout.

### Uplink Message Life

The uplink message life is the maximum allowable period between the SLTA sending an ALERT byte to the host and the host responding with an ALERT ACK byte. This period is 100ms in local host mode and 1 second in remote host mode. When this timeout occurs, the SLTA will resend the ALERT byte. This process is repeated until 3 seconds have elapsed, after which the uplink message is discarded. This timeout only applies to the ALERT/ACK link protocol and is not used for the buffered link protocol.

### ACK/NACK Receive Timeout

When using the reliable transport protocol, the SLTA will wait for the ACK or NACK byte to be sent downlink following the end of the uplink transmission of a message. This period is 1 second, after which the SLTA will re-send the uplink message.

### Uplink Timeout Message Retry Count

When using the reliable transport protocol the SLTA will re-send uplink messages whenever the ACK/NACK timeout period has elapsed. This retry process is limited to 5 retries, after which the uplink message is discarded. There is no retry limit applied to re-sends due to the reception of the NACK byte.

## Session Layer Protocol

The network interface link and transport protocols described above are used for all host-to-SLTA communications. Layered on top of these protocols is a downlink buffer request protocol and an uplink flow control protocol.

### Downlink Buffer Request Protocol

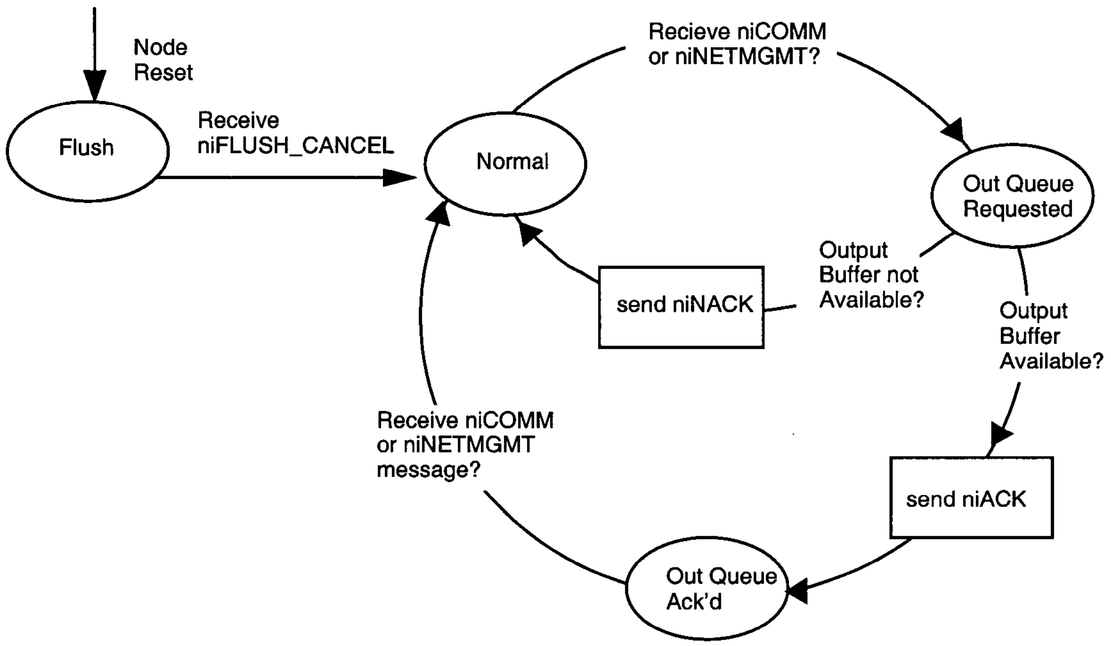
The network driver receives application buffers from the host application, translates them to interface buffers, and passes the interface buffers to the SLTA. There are two types of downlink commands from the host to the SLTA – commands that can be executed directly by the SLTA, and commands that need to be buffered in the SLTA.

Downlink commands that are executed directly by the SLTA are:

niRESET, niFLUSH_CANCEL, niONLINE, niOFFLINE, niFLUSH, niFLUSH_IGN, niPUPXOFF, niPUPXON, niSLEEP, and niSSTATUS.

See the *Host Application Programmer's Guide*, Appendix D, for a description of these commands.

Downlink commands that are buffered in the SLTA are niNETMGMT (for network management commands to be executed by the SLTA itself) and niCOMM (for messages to be sent out on the network, including network variables, explicit messages, and network management messages addressed to other nodes). For these two commands, a buffer request protocol is used to ensure that the SLTA has a free application buffer for the data. The network driver must first request an output buffer before sending the interface buffer. The network driver must hold the buffers in an output queue until the SLTA is ready to receive them. The network driver takes the SLTA through 3 states to request a buffer and send the interface buffer. Figure 6.5 summarizes the downlink state transitions.



*Note: niNETMGMT commands are allowed in the Flush state.*

**Figure 6.5** SLTA Downlink Flow Control States

Following is the sequence of events for transferring an niCOMM or niNETMGMT command downlink to the SLTA:

**1** The SLTA is initially in the NORMAL state.

**2** The network driver requests an output buffer by sending a link-layer header (see figures 6.2 and 6.3) with a niCOMM or niNETMGMT command and the appropriate queue value (niTQ, niTQ_P, niNTQ, niNTQ_P). The data portion of the interface buffer is not sent with the buffer request. This puts the SLTA in the OUTPUT QUEUE REQUESTED state.

3 If an output buffer is not available, the SLTA responds with a `niNACK` (0xC1) command. The SLTA returns to the `NORMAL` state, and the driver starts again at step 2.

4 When an output buffer is available, the SLTA responds with a `niACK` (0xC0) command. The SLTA is now in the `OUTPUT QUEUE ACKNOWLEDGED` state. While in this state, the network driver can only transfer downlink LonTalk messages, uplink source quench commands (`niPUPXOFF`), uplink source resume commands (`niPUPXON`), or reset commands (`niRESET`) since the SLTA is waiting for a message in this state. All other network interface commands sent downlink will be ignored, and will return the SLTA to the `NORMAL` state.

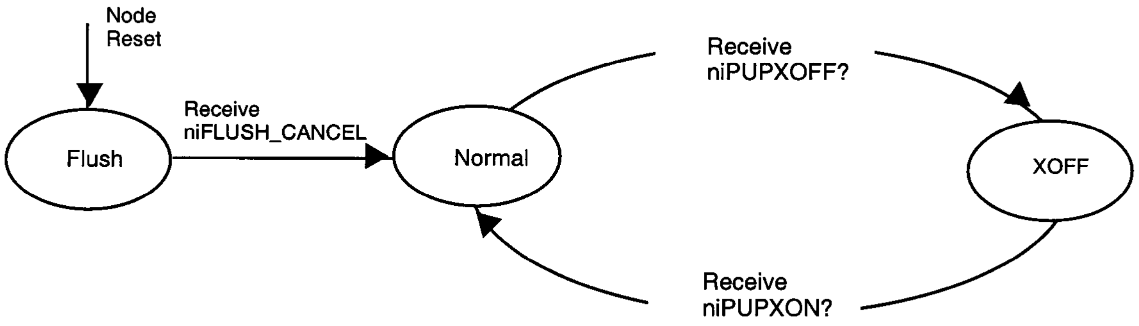
5 Upon receiving the `niACK` acknowledgment, the network driver transfers the entire interface buffer to the SLTA. This buffer has the same command and queue value sent in step 2, and also contains the data and checksum. Upon completion of this transfer, the SLTA returns to the NORMAL state.

The network driver must preserve the continuity of the type of buffer request and the type of message sent downlink. For example, if the network driver sends the `niCOMM+niTQ_P` command requesting a priority output buffer, and follows this with a message transfer with the non-priority `niCOMM+niTQ` command, the SLTA will incorrectly store the message in a priority output buffer, the type originally requested.

## Uplink Flow Control Protocol

Uplink traffic may be incoming LonTalk messages, output buffer request acknowledgments, completion events, or local commands. The network driver translates the interface buffers to application buffer format and stores the buffers in a queue until the host application is ready to read them.

There is no buffer request protocol for uplink traffic. The network driver is normally assumed to have sufficient buffers. The network driver can suspend or resume uplink traffic when no network driver input buffers are available by sending the *Uplink Source Quench* (`niPUPXOFF`) command to the SLTA. This prevents the STLA from sending any LonTalk messages uplink. When the network driver senses that network driver input buffers are available, it sends the *Uplink Source Resume* (`niPUPXON`) command to resume uplink transfers. Figure 6.6 summarizes the uplink state transitions.



*Note: Responses to niNETMGMT and niSSTATUS commands are allowed in the Flush state.*

**Figure 6.6** SLTA Uplink Flow Control States

Note that for SLTA firmware versions 7 or higher, the host may chose to sidestep the downlink buffer request protocol. In this case, the complete message is sent downlink without any buffer request step. If the SLTA has a free output buffer, then the message will be transferred into the SLTA successfully. If not, there will be no indication and the message will be lost. The exception to this case is when using the transport layer protocol, in which case the SLTA will send the NACK to the host, which should force the host to re-send the message. Otherwise, in order to use this feature successfully, the host driver must manage the number of available output buffers within the SLTA. This feature is included in the DOS driver for the SLTA.

## Presentation Layer Protocol

The network driver exchanges LonTalk packets with the host application at the presentation layer. The LonTalk packet enclosed in a command of type `niCOMM` or `niNETMGMT` is described in detail in the *Host Application Programmer's Guide*. It is summarized here for convenience.

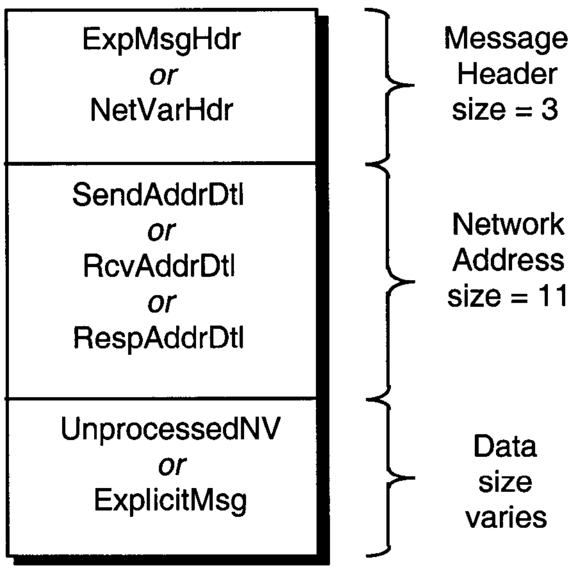


**Figure 6.7** Application Buffer Format

The SLTA firmware is configured with explicit addressing enabled, and therefore the 11-byte network address field is always present in an uplink or downlink buffer. The firmware is also configured with host selection enabled, so the data field of the buffer is either an unprocessed network variable or an explicit message. The processed network variable option is not available with the SLTA.

# 7

# Using the DOS Network Driver

This chapter describes the DOS network driver supplied with the Connectivity Starter Kit for use with the LTS-20 operating in MIP mode. The DOS network driver provides a device-independent interface between a DOS or Windows host application and the LTS-20. The driver is configurable to use one of four PC/AT serial ports, COM1 through COM4, at one of eight serial bit rates.

## Installing the SLTA Network Driver for DOS

**For the purposes of this chapter, the term "SLTA" refers to the LTS-20 module operating in MIP mode.**

*The DOS driver is supplied on the floppy diskette included with the Connectivity Starter Kit. The latest version of this driver may be obtained from the Echelon web site. See the Preface of this manual for ftp site access information.*

The SLTA network driver is installed by adding a `DEVICE` command to the DOS `CONFIG.SYS` file. Edit the `CONFIG.SYS` file to include the line:

```
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS [options]
```

Substitute your drive and directory name if other than `C:\LONWORKS\BIN`. Reboot the PC after adding this line to load the driver. For example, the following command would be used with a locally attached SLTA installed on COM2 as device LON1 running at 38,400 bps with autobaud enabled (this is the factory default):

```
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS /P2 /D1 /B38400 /A
```

See *Example Configurations* in Chapter 11 for examples.

*Warning! The `/A` option must be present in the `CONFIG.SYS` entry if Autobaud is in enable, or the adapter will not function correctly. The /A option also may be left in the `CONFIG.SYS` entry if Autobaud is disabled.*

The available options for the DOS network driver are described in the following sections.

### Buffer Options

`/O`*nn* Sets the number of output (downlink) buffers within the driver to <*nn*>. The buffer sizes within the driver are pre-set to accommodate 255 byte packets. The SLTA has application output buffers in the RAM of the Neuron Chip which, by default, are smaller than this, but may be increased to as large as 255 bytes. The default is eight buffers in the network driver. There must be at least 2 buffers and the maximum allowed number for <*nn*> is limited by the size of the buffer (258) times the total number of input and output buffers within the driver. The entire buffer space plus the driver code itself cannot exceed 64Kbytes. The size of the driver code itself is 9Kbytes. The number of output buffers required is determined by the characteristics of the host application. If the host application always waits for an outgoing message completion before sending another message, then only two buffers are required. If the host application is set up to overlap transactions, more buffers may be required. In this case greater parallelism may be achieved at the expense of host application code complexity.

`/I`*nn* Sets the number of input (uplink) buffers within the driver to <*nn*>. The buffer sizes within the driver are pre-set to accommodate 255 byte packets. The SLTA has application input buffers in the RAM of the Neuron Chip which, by default, are smaller than this, but may be increased to as large as 255 bytes. The default is eight buffers in the network driver. There must be at least 2 buffers and the maximum allowed number for <*nn*> is limited by the size of the buffer (258) times the total number of input and output buffers within the driver. The entire buffer space plus the driver code itself cannot exceed 64Kbytes. The number of input buffers required is determined by the expected incoming traffic and the capability of the host application to process it. If the incoming traffic is bursty, more input buffers are required. If the application cannot process incoming traffic fast enough, the input buffer pool will fill up with unprocessed packets. In that case, the SLTA will not be able to pass any new data to the host, and the input application buffers in the SLTA will start to fill up. Once that occurs, messages will be lost, possibly causing incoming LonTalk transactions to be retried, and eventually causing the sender of the message to receive a failure indication.

## Serial Bit Rate Options

**/B***nnnnnn*

Sets the serial bit rate to <*nnnnnn*>. The available serial bit rates are listed below. The default is 38,400 bps.

Available serial bit rates are:

**1200, 2400, 9600, 14400, 19200, 38400, 57600, 115200.**

This rate represents the serial bit rate between the PC and the SLTA when using a direct serial connection, and between the PC and the modem when using a remote connection. *The 115,200 bps rate is only available on the TP/XF-1250 SLTA/2 and SLTAs based on the LTS-10 module (SLTAs with a 10 MHz input clock).* For remote connections, the PC to modem serial bit rate, telephone line speed (i.e. modem to modem serial bit rate), and the modem to SLTA serial bit rate may be different. The PC to modem serial bit rate is controlled by the network driver on the PC using the /B option; the telephone line speed is selected by the modems based on modem configuration; and the modem to SLTA serial bit rate is controlled by the hardware configuration of the SLTA as described in Chapter 2 (autobaud cannot be used in this configuration).

For local connections with the SLTA Autobaud option disabled, the serial bit rate specified by this driver option must match the rate specified by the Baud Rate inputs to the SLTA.

**/A**

***Warning**: If you are using the default hardware configuration (autobaud enabled), the autobaud option (/A)* ***must*** *be enabled or the SLTA will not function properly.*

Enables the autobaud feature. This provides the autobaud sequence whenever the driver is opened. The default setting for the driver is autobaud disabled. If the Autobaud input on the SLTA hardware is enabled, then this option must be specified This option may not be used with the modem support (/M) option.

## DOS Device Options

**/P*n*** Sets the serial port to <*n*> where <*n*> is 1-4 for COM1 - COM4. The default is COM1.

**/D*n*** Defines the device unit number as <*n*>, where <*n*> is between 1 and 9, so that the DOS device name is "LON1" through "LON9". The default is 1 for "LON1". This option can be used to support multiple network interfaces on a single PC. For example, this device name is passed as a parameter to `lxt_open()` when using the LonManager API. When invoking the sample host application `HA`, the device may be specified with the `-D` option, for example:

```
HA -DLON2
```

**/U*n*** Sets the serial port interrupt request number (IRQ) to a non-standard value <*n*>, where <*n*> is between 1 and 7. If the serial port in use is COM3 or COM4, you may want to use a unique, unused IRQ for that port. Many serial ports and internal modems allow the selection of a non-standard IRQ such as IRQ2 or IRQ5.

**/C** Enables communications interrupt chaining. Some PCs may incorporate up to four serial ports. If supported by the serial hardware, COM1 and COM3 may share the same interrupt (as do COM2 and COM4). This option may enable the driver to support the shared interrupt by "chaining" to the interrupt vector that was in place when the driver was loaded. This option is not necessary if your system does not use COM3 or COM4, or if COM3 or COM4 use a different interrupt request number. When installing two SLTA network drivers on a system on COM1 and COM3 (or COM2 and COM4 with the same interrupt request number), the last installation of the driver should use this option. Here is an example of a `CONFIG.SYS` file entry for such a system.

```
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS /B38400 /A /P1
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS /B38400 /A /P3 /C
```

Standard hardware configurations are used for the COM1 - COM4 settings:

| ***Device*** | ***I/O Address*** | ***Interrupt (*)*** |
|---|---|---|
| COM1 | 0x3F8 | 4 |
| COM2 | 0x2F8 | 3 |
| COM3 | 0x3E8 | 4 |
| COM4 | 0x2E8 | 3 |

(*) May be changed with the /U option

## Timing Options

`/Rnn` Defines the flush/retry count in 55ms intervals. This value is used in error states for re-transmitting requests and for terminating receive flushes when input errors occur. See *Troubleshooting* in Chapter 17. Normally, this option should not be specified.

`/Wnnn` Includes a delay of <*nnn*> microseconds when transmitting downlink. This parameter can be used to pace the rate at which bytes are transmitted downlink to the SLTA, and may be required for high-performance network management tools such as LonMaker. The delay is executed following the transfer of each data byte to the host's UART, and only after the first 15 bytes have been sent. Since the SLTA employs a 16-byte deep FIFO buffer in its UART, the first group of bytes sent do not need to be paced. The pacing delay will have no effect unless it is greater than the actual period it takes to transmit a single byte at the given serial bit rate. The time taken to transmit a byte is 173 $\mu$s at 57,600 bps, and 86 $\mu$s at 115,200 bps. For a 10 MHz SLTA (TP/XF-1250 SLTA/2 or LTS-10), this option should be used at 115,200 bps if messages greater than 16 bytes are to be transmitted. A value of `/W120` is suggested. For a 5 MHz SLTA (TP/XF-78, TP/FT-10, or TP-RS485 SLTA/2), this option should be used at 57,600 bps, again if messages greater than 16 bytes are to be transmitted. A value of `/W240` is suggested. This option is not required at the default serial bit rate of 38,400bps, or at slower serial bit rates.

`/Z` By default, the SLTA firmware disables network communications after a reset by entering a FLUSH state. This state causes the SLTA to ignore all incoming messages and prevents all outgoing messages, even service pin messages. The DOS network driver for the SLTA automatically enables network communications when the SLTA is opened and when it receives an uplink message from the SLTA indicating that it has been reset. However, the host application itself must explicitly enable network communications if the /Z option is specified and the CFG1 input is set to *Network Disabled*. See Chapter 8 for more information. If CFG1 is set to *Network Enabled*, the SLTA will go directly to the `NORMAL` state, thus allowing communications.

The following table summarizes these options:

| ***Network Disable Input*** | ***DOS Driver /Z Option*** | ***When SLTA Enables Network Communications*** |
|---|---|---|
| Disabled | Specified | Host application command |
| Disabled | Not specified | Opening network driver |
| Enabled | Don't care | Immediately after reset |

Host applications which need to configure the SLTA prior to enabling network communications should use this option. This option should not be used with the LonManager API, LonManager LonMaker, or the 16-bit LonManager DDE Server. More information about the `niFLUSH_CANCEL` message is provided in the *LONWORKS Host Application Programmer's Guide.*

## Network Interface Protocol Options

**/F**

Enables the full interrupt mode of the driver. If this option is not specified, the driver will disable interrupts for the duration of each link-layer transfer. This ensures that no data will be lost due to other system interrupts, and is acceptable at high serial bit rates. The driver will use interrupts for the first byte of each uplink interface buffer. When the uplink interrupt is received, the driver reads the rest of the message without interrupts via polled I/O. Interrupts are disabled during the uplink transfer. This assures that no other system interrupts will occur resulting in lost uplink data frames. Downlink transmissions are sent directly via polled I/O of the serial port from the write function call. The host write functions will not return until the message has been sent downlink. When using the ALERT/ACK link protocol, interrupt latency is not a problem, since the SLTA-to-host protocol includes an acknowledgment of the start of the message. The driver employs timeouts in order to prevent lockout of the write function, and timeouts for clearing various states of the transmitter/receiver when line errors occur.

When operating at lower serial bit rates, it becomes less desirable to disable interrupts for long periods. The trade-off with using the full interrupt mode is that other system interrupts may cause loss of data in the serial port's UART. If the `/F` option is specified, the driver uses interrupts for every uplink and downlink byte transferred. Downlink messages are buffered from the device write function and are sent downlink under interrupt control. Uplink messages are received under interrupt control and are buffered also. This option should be used for serial bit rates of 9,600 bps or slower. Do not use this option with the HP 95LX.

**/M**

Enables modem support and the reliable transport protocol. This option must be specified if the host is to communicate with the SLTA via a modem connection. The SLTA must be configured with CFG2 input in the *Remote Host* setting. In this mode the driver relies on the state of the DCD signal from the modem to determine if it is connected to an SLTA through a modem connection or not. When connected, the selected SLTA <-> Host network interface link protocol is in effect. When disconnected the only allowable link layer traffic is of the 'modem direct' type, where ASCII strings are being exchanged between the host and the modem, for example, AT commands to dial out. Any other network interface traffic is not allowed when disconnected from the SLTA. Calls to the read function will result in no network interface data messages (`LDV_NO_MSG_AVAIL`), and any call to a write function that needs to communicate with the SLTA via the modems will result in a No Output Buffers Available error (`LDV_NO_BUFF_AVAIL`). Once the connection is made, normal network interface traffic may resume.

This option also enables the reliable transport protocol. This protocol includes the addition of a message sequence number and the end of message ACK/NACK code. See Chapter 8 for a description of this protocol.

| | |
|---|---|
| **/N** | Disables the ALERT/ACK network interface link protocol, and enables the buffered network interface link protocol. Network interface messages are sent without a wait for the ALERT ACK response. Both sides of the interface (the SLTA and the driver) must agree on this setting. This option should not be used with the */M* option. |
| **/Q** | Allows modem responses to be sent uplink to the host. When the telephone link is disconnected, these messages are ASCII strings with the network interface command type `niDRIVER` (0xF0). If /Q is specified, the host application must be able to handle messages, such as `NO CARRIER`, that might come from the modem itself if problems occur in the connection. |
| **/X** | Disables the the buffer request protocol. This option only works with SLTA application versions 7 or later. When this option is enabled, the driver requests the buffer count from the SLTA using the `niSBUFC (0xE7)` command whenever the interface is opened, or when the interface is reset, and reports an `niRESET` to the host. The driver keeps track of the number of available output buffers in the SLTA by examining both uplink and downlink messages. This option prevents the use of one message type: A local network management command not using a request/response service. Normally this type of message is not used. One exception could be the `Set Node Mode: Reset` command, which would result in the node resetting and the buffer management recovering on its own anyway. Otherwise, if this type of message is used, no uplink response would occur and the driver could not track the fact that a new output buffer has been made available. |

The following table summarizes the relationship between the CFG jumpers of the SLTA and the driver options that control the network interface protocol.

| *Input* | *Input State* | *Driver Option* |
|---|---|---|
| CFG 2 | Local Host; No Transport Protocol | /M not specified |
| CFG 2 | Remote Host; Reliable Transport Protocol | /M specified |
| CFG 3 | ALERT/ACK Link Protocol | /N not specified |
| CFG 3 | Buffered Link Protocol | /N specified |

## Calling the Network Driver from a Host Application

The DOS network driver supports the `open`, `close`, `read`, `write`, and `ioctl` DOS calls. All LONWORKS standard network drivers for DOS support these calls. See Chapter 4 of the *Host Application Programmer's Guide* for more details.

When the DOS SLTA network driver is loaded during execution of the `CONFIG.SYS` file, it does not attempt to communicate with the SLTA.

When the network driver is opened with the DOS open call, it establishes communications with the SLTA. The network driver returns an error if this fails, for example, if the SLTA is disconnected, powered down, or configured incorrectly. If the open call succeeds, the driver enables network communications by clearing the SLTA `FLUSH` state, if configured to do so.

The DOS read call is defined to return the number of bytes read from the device. Some LONWORKS standard network drivers return 0 if there are no uplink messages available. DOS reports this as an end-of-file condition and prevents further reads from succeeding. However, the SLTA driver returns a length of 2, and sets the first byte of the caller's buffer (the cmd/queue byte) to 0 to indicate that there is no uplink message available.

Normally, the DOS `read` and `write` calls are not used with LONWORKS standard network drivers. This is because any error from the network interface will display the familiar `Abort`, `Retry`, `Fail`? error message from DOS, unless the caller has installed a critical error device handler. Therefore, DOS applications using a network device typically call direct entry points into the driver. This also allows more detailed error status to be returned to the application. The addresses of these entry points are obtained by calling the `ioctl()` function of the driver.

This function call is used as follows:

```
int ioctl(int handle, int func, void far *argdx, int argcx);
```

- `handle` is an integer returned by an earlier successful call to `open()`, specifying the LONWORKS network driver `LONn` to be opened.

- `func` is the value 2, meaning that the application is reading information from the driver. For LONWORKS standard DOS network drivers, the information returned is the network interface direct call structure.
- `argdx` is a pointer to a caller-declared structure that will contain the direct entry points into the driver. See the structure `direct_calls` in the file `NI_MSG.C` in the supplied example host application for usage.
- `argcx` is the size of the structure.

Function code 2 is supported by all LONWORKS network drivers for DOS to return three direct entry points into the driver code. The network driver for the SLTA supports an additional option to function code 2, as well as function code 3, which is used to manage the modem control state of the driver. These options are not used when the SLTA is connected directly to a host. They are provided primarily for use while establishing communications with a remote host. For example, the host connect utility (HCU) described in Chapter 12 of this manual uses these functions. Host applications that only communicate to the SLTA via an already-established telephone connection do not need to concern themselves with these functions. If you wish to establish or take down telephone connections during the execution of your host application, use the source code of HCU as a guide.

When function code 2 is used, argdx points to the `direct_calls` structure defined for all LONWORKS standard network drivers for DOS. If `argcx` is 13, the size of the standard direct calls structure, then three direct entry point addresses are returned as usual. If `argcx` is 4 (the size of the structure `ioctl_get_dcd_s`), then the state of the modem's DCD line is returned as a TRUE or FALSE value. Note that the status field is 16 bits in this structure, but 8 bits in the direct calls structure.

```
struct ioctl_get_dcd_s {
      unsigned ioctl_stat;   // 16 bit status
      unsigned dcd_state;    // Data Carrier Detect (TRUE or FALSE)
}
```

Function code 3 is used when the application wishes to write information to the driver. For the SLTA driver, argdx points to the following structure, and `argcx` is its size:

```
struct ioctl_o_info_s {
    unsigned  ioctl_stat;      // 16 bit status
    unsigned  sub_command;     // use enum sub_command
    unsigned  mode;
    unsigned  mode_aux;
}
enum sub_command {
    SUBC_set_opt     = 1,      // set driver options
    SUBC_set_DTR     = 2,      // set DTR line
    SUBC_set_baud    = 3,      // set serial bit rate
};
```

There are three sub-commands, used to set the various modes of the driver, the state of the DTR (Data Terminal Ready) line to the modem, and to set the serial bit rate of the serial interface.

When sub-command 1 is used, the `mode` field in the structure is a bit mask defining which of the driver modes is to be changed, and the `mode_aux` field specifies bits defining the new states of those modes. It is possible to set more than one of the modes by OR'ing the following bit-masks together:

| | |
|---|---|
| 0x0001 | Enables modem support. |
| 0x0002 | Allows modem responses to host - same as the /Q option. |
| 0x0004 | Forces direct modem mode. In this mode, the network driver is communicating directly with the modem. |
| 0x0010 | Enables the buffered link protocol and disables the ALERT/ACK link protocol - same as the /N option. |
| 0x0020 | Enables the reliable transport protocol. |

The `/M` option corresponds to 0x0021.

Sub-command 2 is used to set the state of the DTR line. In this case, the DTR signal is enabled (on) if the mode field is true.

Sub-command 3 is used to set the serial bit rate of the serial interface. The mode field determines the new bit rate as follows: 0:14,400; 1:1,200; 2:2,400; 3:9,600; 4:19,200; 5:38,400; 6:57,600; 7:115,200.

## Using the SLTA Driver under Microsoft Windows

In order to use the direct entry points to a LONWORKS standard network driver for DOS under Microsoft Windows, an interface based on the DOS Protected Mode Interface (DPMI) must be provided. This type of interface, in the form of Windows DLL software, is supplied with the Connectivity Starter Kit, as well as with the 16-bit LonManager DDE Server. See Appendix B for information on using the Windows DLL directly.

# 8 Using the UNIX Network Driver

This chapter describes the UNIX network driver supplied with the Connectivity Starter Kit for use with the LTS-20 operating in MIP mode. The UNIX network driver provides a device-independent interface between a UNIX host application and the LTS-20.

## Installing the SLTA Network Driver for UNIX

**For the purposes of this chapter, the term "SLTA" refers to the LTS-20 module operating in MIP mode.**

The SLTA network driver for UNIX is not a UNIX device driver. It is instead a source library that provides an interface to an existing UNIX serial port driver. The UNIX network driver handles the SLTA network interface protocol and runs on top of the UNIX serial port driver, which in turn handles the interrupt processing and buffering of uplink and downlink serial data. This greatly simplifies the SLTA network driver and makes it more portable to different versions of UNIX, as well as other operating systems.

The `LDVSLTA.C` and `LDVSLTA.H` files contain the C source code for the UNIX network driver. This code has been tested with Sun UNIX (SunOS Release 4.1) and SCO UNIX (Release 4.2.0). Changes may be required for other versions of UNIX. The source code must be compiled and linked with your application. The serial device driver names may have to be changed for different versions of UNIX. For example, Sun UNIX uses `/dev/ttya` and `/dev/ttyb`, and SCO UNIX uses `/dev/tty1a` and `/dev/tty2a`.

The UNIX network driver is implemented with the following defaults (these defaults may be changed by modifying the source code):

- **Link Layer Protocol**. The buffered link protocol is used if the `alert_ack_prtcl` variable is set to `FALSE` in the source code (this is the default). The ALERT/ACK link protocol is used if the `alert_ack_prtcl` variable is set to `TRUE`. See *Buffered Link Protocol* in Chapter 8 for warnings on use of the buffered link protocol with UNIX. The buffered link protocol should not be used with modems.
- **Transport Layer Protocol**. The reliable transport protocol is not supported.
- **Physical Layer Protocol**. The call to `ioctl()` in `ldv_open()` initializes the serial link to 19,200bps. The code for the `ioctl()` call may have to be modified for different versions of UNIX and serial port configurations.
- **Modem Support**. The UNIX network driver does not include modem support for remote hosts.
- **Buffers**. The number of input and output buffers is controlled by the `ldv_buffers` variable and defaults to 10 each.

## Calling the Network Driver from a Host Application

The functions provided by the UNIX network driver are the same as those listed under *Standard Network Driver Services* in chapter 4 of the *LONWORKS Host Application Programmer's Guide*, with the addition of a new service, `ldv_post_events()`.

## ldv_open()

```
typedef int LNI;
LNI handle = ldv_open(char *serial_device_name);
```

Initializes the SLTA and returns a handle for accessing it. Opens the serial device and enables network communications (i.e. the `FLUSH` state is canceled). The `serial_device_name` parameter must be the name of an installed serial device such as `/dev/ttya`. If the `ldv_open()` call succeeds, the SLTA will send an niRESET command uplink to the host. Only one device may be open at a time.

## ldv_read()

```
LDVCode error = ldv_read(LNI handle, void *msg_p, unsigned length);
```

Reads an application buffer from the SLTA. The `msg_p` argument is a pointer to an application buffer as defined in chapter 3 of the *LONWORKS Host Application Programmer's Guide*. If a buffer is not available, the `LDV_NO_MSG_AVAIL` error code is returned.

## ldv_write()

```
LDVCode error = ldv_write(LNI handle, void *msg_p, unsigned length);
```

Writes an application buffer to the SLTA. The `msg_p` argument is a pointer to an application buffer as defined in chapter 3 of the *LONWORKS Host Application Programmer's Guide*. If a buffer is not available within the SLTA to accept the application buffer, the `LDV_NO_BUFF_AVAIL` error code is returned.

## ldv_post_events()

```
void ldv_post_events(void);
```

This is the network interface background service function. Its purpose is to process any uplink or downlink SLTA traffic. This function should be called periodically from the host application in order to assure that the SLTA traffic is being processed. This processing includes off-loading the UNIX serial port driver's input buffers, and moving downlink any messages that are buffered in the network driver's output buffers.

This function is also called from within the interface library's `ldv_read()` and `ldv_write()` functions. The host application does not have to explicitly call the `ldv_post_events()` function if it is periodically polling the interface using the `ldv_read()` function.

# LTS-20 LonTalk® Serial Adapter and PSG-20 User's Guide

Version 2

ECHELON®

Corporation

078-0181-01C

Echelon Corporation
www.echelon.com

# Preface

This document describes how to use an LTS-20 LonTalk Serial Adapter and a host processor with an EIA-232 (formerly RS-232) serial interface with a LONWORKS® network.

## Audience

This user's guide provides specifications and instructions for LTS-20 users.

## Content

This manual provides detailed information about the hardware and software for the LTS-20 and PSG-20.

- Chapter 1 introduces the LTS-20.
- Chapter 2 provides an overview of the LTS-20.
- Chapter 3 describes product development with the LTS-20 module.
- Chapter 4 describes EMI and ESD issues for LTS-20 and PSG-20 modules.
- Chapter 5 describes the LTS-20 software.
- Chapter 6 discusses creating SLTA network driver.
- Chapter 7 discusses using the DOS network driver.
- Chapter 8 discusses using the UNIX network driver.
- Chapter 9 discusses using the LTS-NSI mode.
- Chapter 10 discusses the LTS-20 MIP mode software.
- Chapter 11 describes using the drivers and link manager with LTS-20 NSI mode.
- Chapter 12 discusses using the DOS driver with LTS-20 MIP mode.
- Chapter 13 explains how to create an LTS-20 MIP mode driver.
- Chapter 14 discusses initialization and installation.
- Chapter 15 explains how to use the LST-20 with a modem.
- Chapter 16 discusses using the host connect utility with the LTS-20 MIP mode.
- Chapter 17 details using a programmable serial gateway.
- Chapter 18 details modem troubleshooting.
- Appendix A lists the default communications parameters for LTS-20-based products.
- Appendix B describes the Windows DLL files supplied with the LTS-20.
- Appendix C includes a copy of the software license agreements.

## Related Manuals

The following Echelon documents are suggested reading for more information:

- The *LNS™ DDE Server User's Guide* is a manual for developers on how to create user interface, monitoring, and control applications that communicate with LONWORKS networks from computers running Microsoft Windows.
- The *LonMaker® for Windows Integration Tool User's Guide* is a manual for users on how to install networks using the integration tool.
- The *LonBuilder® User's Guide* describes how to develop LONWORKS applications with the LonBuilder Developer's Workbench.
- The *NodeBuilder™ User's Guide* describes how to develop LONWORKS applications with the NodeBuilder Development Tool.
- The *LONWORKS Host Application Programmer's Guide* describes how to write a host application that can be used with a serial adapter.
- The *Neuron® Chip Data Book* describes the LonTalk message formats that can be used with a serial adapter. It also describes the network management and network diagnostic messages that can be sent with such an adapter.

## Web Access

Engineering bulletins and data sheets supporting this product are available on the Echelon Web site. General information regarding Echelon, its business, and its products also are located on the site at http://www.echelon.com. The Developer's Toolbox located at the Web site includes drivers for the LTS-20.

# Contents

# 1

# LTS-20 Introduction

The LTS-20 LonTalk Serial Adapter Module is a network interface that enables any host processor with an EIA-232 serial interface to connect to a LONWORKS network. A replacement for the previous generation LTS-10 Core Module, the LTS-20 is supplied with both Network Services Interface (NSI) firmware to support LNS – the standard LONWORKS® network operating system – as well as Microprocessor Interface Program (MIP) firmware to support older API-based tools. The LTS-20 extends the reach of LONWORKS technology to a variety of hosts, including desktop, laptop, and palmtop PCs, workstations, embedded microprocessors, and microcontrollers.

The LTS-20 enables the attached host to act as an application node on a LONWORKS network. When used with a PC host and the LonMaker for Windows Integration Tool (or an older generation tool using the LonManager API), or LNS DDE Server, the LTS-20 can also be used to build sophisticated network management, monitoring, and control tools for LONWORKS networks.

The LTS-20 is a direct replacement for Echelon's model 65200-100 LTS-10 module. The two modules are pin-for-pin compatible and also have identical physical dimensions. The LTS-20 is equipped with an NSI to enable it to be used in conjunction with LNS – the standard operating system for LONWORKS control networks. By default the LTS-20 is shipped with the NSI mode enabled. A jumper is provided which, when cut by the customer, disables the NSI and enables the MIP mode for emulating the behavior of the LTS-10.

Intended to be embedded within an OEM's product, the LTS-20 and its associated interface logic can be used to connect to a host through a pair of modems and the telephone network. This allows the monitoring, control, or network management application computer to be remote from the network. A node using the LTS-20, associated logic, and modems can initiate a telephone call to a remote host computer, and can be set up to answer incoming calls from a remote host.

**A Connectivity Starter Kit (Model 58030-01) should be ordered for initial development with an LTS-20.** The kit includes both software and documentation. The software includes network drivers for Windows® 95, 98, and NT. Supplied as a single in-line module (SIM) form-factor building block, the LTS-20 can be used to create custom serial interfaces to a wide range of network media.

# 2

# LTS-20 Overview

This chapter provides an overview of the model 65202 LTS-20 LonTalk Serial Adapter Module.

## Mechanical Description

The LTS-20 Module consists of a 67mm by 28mm by 7mm (2.65" by 1.1" by 0.3") module with the core electronics and firmware required to implement a serial LonTalk Adapter. The module is attached to a motherboard using a 40-position, 0.050-inch spacing SIM socket. Two compatible sockets are available:

- Molex 15-82-1175 SIM Vertical Connector with metal latches, 0.050 Centerline Single Row Connector - 40 position.
- AMP 4-382487-0, SIM II Right Angle Connector, 0.050 Centerline Single Row Connector - 40 position.

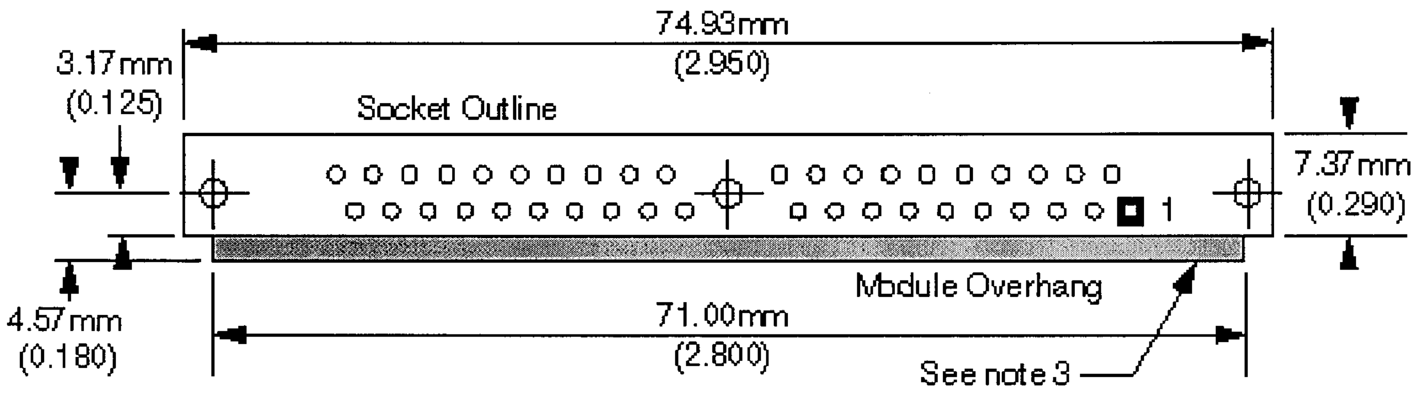
For information about Molex parts call +1-708-969-4550 or fax +1-708-969-1352.

Within North America, AMP drawings can be obtained via FAX using the free AMP FAX service. Call 1-800-522-6752 from a touchtone phone and order customer prints using the AMP part number. Additional information on the connectors is available in AMP application note number AMP 114-1060 and reliability information is available in AMP product specification 108-1297.

Figure 2.1 illustrates the mechanical footprint for the module and vertically mounted socket. Figure 2.2 shows the recommended PCB pad layout for the vertically mounted socket. Figures 2.3 and 2.4 provide the same information for the right-angle socket.

Decisions about component placement on the motherboard must consider electromagnetic interference (EMI) and electrostatic discharge (ESD) issues discussed in Chapter 4 of this document.

**LTS-20 Footprint when using Molex Part Number 15-82-1175 (Component Side, Vertical SIM Mounting)**

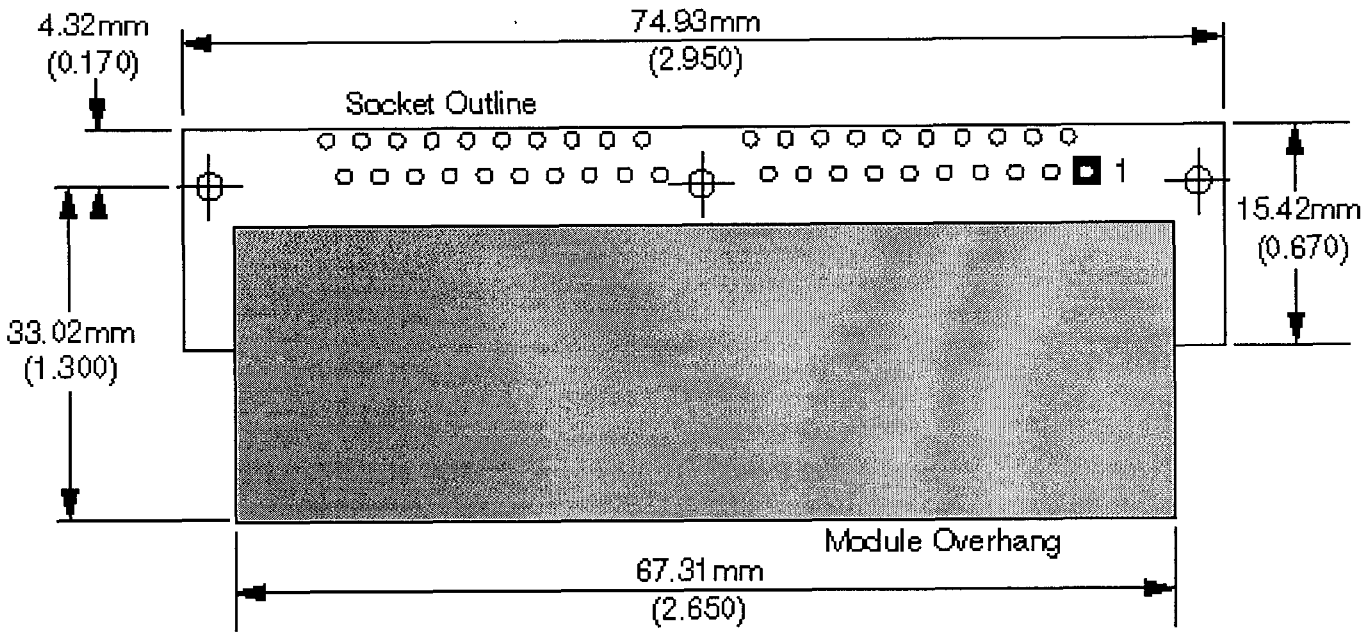


Notes:

1. Dimensions in mm (inches).
2. Tolerances ± .13mm (0.0005)
3. Components standing higher that 3.81mm(.15) should not be closer than 12.4mm (0.5) to this edge of the socket to allow clearance to insert the module.
4. Allow 33.02mm (1.3) clearance above PCB over the footprint area. Additional clearance required to insert the module.
5. Socket dimensions are subject to change. Contact Molex for the most current information.
6. Alternate AMP part is 822021-1.

**Figure 2.1** LTS-20 Vertical Socket Mechanical Footprint

**LTS-20 PCB Footprint when using AMP Part Number 4-382487-0**
**(Component Side, Horizontal Mounting)**



Notes:

1. Dimensions in mm (inches).
2. Tolerances ± .13mm (0.0005)
3. Do not position components in the overhang region.
4. Allow 12.7mm (0.5) clearance above PCB over the entire footprint area. Additional clearance required during assembly to insert the module.
5. Socket dimensions are subject to change. Contact AMP for the most current information.

**Figure 2.2** LTS-20 Horizontal Socket Pad Layout

## Power Requirements

The modules require a +5VDC ±10% power source with a minimum of 150mA of current capacity.

### *Power Supply Decoupling and Filtering*

The design for the module power supply must consider filtering and decoupling requirements of the module. The power supply filter must prevent noise generated by the core module from conducting onto external wires. Switching power supply designs must also consider the effects of radiated EMI.

The modules require a clean power supply to prevent RF noise from conducting onto the network through active drive circuits. Power supply noise near the network transmission frequency may degrade network performance.

The modules include 2.2$\mu$F and 0.1$\mu$F power supply bypass capacitors close to pins 1, 9, and 31. In general, high-frequency decoupling capacitors valued at 0.1$\mu$F or 0.01$\mu$F placed near pins 1, 9, and 31 on the motherboard are necessary to reduce EMI.

### *Low Voltage Protection*

It is necessary to include a low voltage indicator (LVI) circuit on the module motherboard to drive the ~RESET line of the core module. See the *Neuron Chip Data Book* for details. Failure to include such protection may cause data corruption to configuration data maintained in EEPROM on the module's Neuron Chip. In the sample circuit of figure 3.1, protection is provided via a Motorola MC33164.

## Electrical Interface

The pinout of the modules is shown in table 2.1.

**Table 2.1** Pinout of the LTS-20

| ***Name*** | ***Function*** | ***Pin #*** |
|---|---|---|
| AUTOBAUD | Automatic serial bit rate detect enable input | 19 |
| BAUD0 | Serial bit rate 0 input (LSB) | 16 |
| BAUD1 | Serial bit rate 1 input | 15 |
| BAUD2 | Serial bit rate 2 input (MSB) | 21 |
| CFG0 | EIA-232 interface option (1 = 8 wire, 0 = 3 wire) | 17 |
| CFG1 | Network Disable (1 = disable after reset) | 14 |
| CFG2 | Modem Support (1 = remote host; 0 = local host) | 20 |
| CFG3 | Interface link protocol (1 = Buffered; 0 = ALERT/ACK) | 18 |
| CLK OUT | Neuron Chip CLK2 output | 11 |
| CP0 | Network communication port 0 | 6 |
| CP1 | Network communication port 1 | 5 |
| CP2 | Network communication port 2 | 4 |
| CP3 | Network communication port 3 | 7 |
| CP4 | Network communication port 4 | 3 |
| ~CTS | EIA-232 clear to send output from UART | 36 |
| ~DCD IN | EIA-232 serial data carrier detect input (DTE only) | 35 |
| ~DCD OUT | EIA-232 serial data carrier detect output (DCE only) | 40 |
| DCE | Indicates whether connected as DCE | 22 |
| ~DSR | EIA-232 data set ready output from UART | 38 |
| ~DTR | EIA-232 data terminal ready input to UART | 37 |
| PKT | Packet transmitted output | 13 |
| ~RESET | Neuron Chip reset input and output | 8 |
| ~RI IN | EIA-232 ring indicator input (DTE only) | 33 |
| ~RI OUT | EIA-232 ring indicator output (DCE only) | 34 |
| ~RTS | EIA-232 request to send input to UART | 39 |
| SERIAL IN | EIA-232 serial data input to UART | 26 |
| SERIAL OUT | EIA-232 serial data output to UART | 27 |
| ~SERVICE | Neuron Chip service pin input and output | 12 |
| ~TEST | Manufacturing test pin, tie to Vcc in final product | 30 |
| XID0 | Transceiver ID 0 input (LSB) | 25 |
| XID1 | Transceiver ID 1 input | 23 |
| XID2 | Transceiver ID 2 input | 29 |
| XID3 | Transceiver ID 3 input | 28 |
| XID4 | Transceiver ID 4 input | 24 |
| $V_{CC}$ | +5VDC input | 1, 9, 31 |
| GND | Ground | 2, 10, 32 |

## NSI/MIP MODE JUMPER R2

The NSI/MIP jumper R2 (a 220□ resistor) determines the start-up mode of the module. The module is shipped in the NSI mode, with the jumper intact. Cutting jumper R2 disables the NSI mode and enables the MIP mode (for emulating the LTS-10 module). DO NOT cut the jumper while the module is powered – only cut the jumper with the module unpowered. Observe appropriate ESD protection suitable for CMOS devices when handling the module or cutting jumper R2. To ensure reliable operation, R2 should be removed in its entirety and not simply cut at one end.

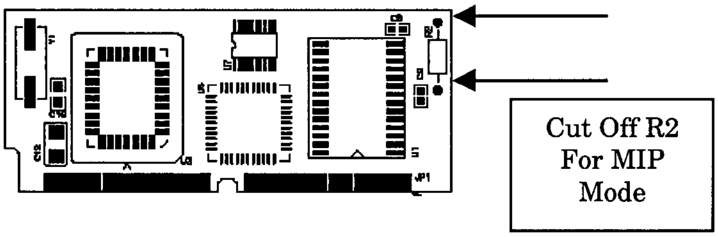


**Figure 2.3** R2 Jumper

## AUTOBAUD

The AUTOBAUD input signal enables automatic baud rate detection on the LTS-20 as described in Chapter 7. The input is a floating CMOS input and must be asserted high to enable automatic baud rate detection or be asserted low to disable automatic baud rate detection.

## BAUD[2..0]

The BAUD[2..0] input signals set the EIA-232 serial bit rate on the LTS-20 module as described in Chapter 7 and summarized in Table 2.2. The inputs are not used when AUTOBAUD is enabled. The inputs are floating CMOS inputs and must be asserted high to select a "1" and asserted low to select a "0".

**Table 2.2** LTS-20 Baud Rate Inputs

| *BAUD[2..0]* | *Serial Bit Rate* |
|---|---|
| 0 0 0 | 14,400 bps |
| 0 0 1 | 1,200 bps |
| 0 1 0 | 2,400 bps |
| 0 1 1 | 9,600 bps |
| 1 0 0 | 19,200 bps |
| 1 0 1 | 38,400 bps |
| 1 1 0 | 57,600 bps |
| 1 1 1 | 115,200 bps |

## CFG0

The CFG0 input signal selects a full 8-wire interface or 3-wire interface for the LTS-20 module as described in Chapter 7. The input is a floating CMOS input and must be asserted high to select a full 8-wire interface or asserted low to select a 3-wire interface.

## CFG1

The CFG1 input signal enables or disables network communications after reset for the LTS-20 module as described in Chapter 7. The input is a floating CMOS input and must be asserted high to disable network communications after reset or asserted low to enable network communications after reset.

## CFG2

The CFG2 input signal controls the use of the LTS-20 module with a modem as described in Chapter 12. The input is a floating CMOS input and must be asserted high to enable modem support for a remote host or asserted low to enable local host support.

## CFG3

The CFG3 input signal controls the network interface link protocol used between the LTS-20 module and a local host as described in Chapter 11. The input is a floating CMOS input and must be asserted high to select the buffered link protocol or asserted low to select the ALERT/ACK link protocol.

## CLK OUT

The CLK OUT output signal is driven by the CLK2 pin of the core module Neuron Chip. It can drive one HCMOS load, and can be used to interface to the FTT-10A Free Topology Transceiver or the LPT-10 Link Power Transceiver.

## CP(4..0)

The CP[4..0] signals are connected to the CP[4..0] pins of the core module Neuron Chip. The function of these pins is described in the *Neuron Chip Data Book.*

## ~CTS

EIA-232 clear to send output when the LTS-20 is connected as a DCE device. This output should be used as the EIA-232 request to send (~RTS) output when the LTS-20 is connected as a DTE device. The output is driven by the core module UART and must be connected to an EIA-232 driver if EIA-232 voltage levels are required, or can be ignored for a 3-wire serial interface.

## ~DCD IN

EIA-232 data carrier detect input when the LTS-20 is connected as a DTE device. This input is not used when the LTS-20 is connected as a DCE device. This input is not used by the firmware when connected to a local host, and is used to detect incoming calls when used with a remote host (i.e. when CFG2 is set to the *Remote Host* state). The input is connected to the core module UART and must be externally connected to an EIA-232 receiver if EIA-232 voltage levels are used, or should be connected to ground for a 3-wire serial interface.

## ~DCD OUT

EIA-232 data carrier detect output when the LTS-20 is connected as a DCE device. This output is not used when the LTS-20 is connected as a DTE device. This output is always asserted high by the firmware. The output is driven by the core module UART and must be connected to an EIA-232 driver if EIA-232 voltage levels are required, or can be ignored for a 3-wire serial interface.

## DCE

The input is a floating CMOS input. It is not used by the firmware. It should be pulled high or low by the motherboard.

## ~DSR

EIA-232 data set ready output when the LTS-20 is connected as a DCE device. This output should be used as the EIA-232 data terminal ready (~DTR) output when the LTS-20 is connected as a DTE device. This output is always asserted high by the firmware when used with a local host and is asserted low for 500ms to hang up the modem when used with a remote host (i.e., when CFG2 is set to the *Remote Host* state). The output is driven by the core module UART and must be connected to an EIA-232 driver if EIA-232 voltage levels are required, or can be ignored for a 3-wire serial interface.

## ~DTR

EIA-232 data terminal ready input when the LTS-20 is connected as a DCE device. This input should be used as the EIA-232 data set ready (~DSR) input when the LTS-20 is connected as a DTE device. The input is connected to the core module UART and must be externally connected to an EIA-232 receiver if EIA-232 voltage levels are used, or should be externally connected to ground for a 3-wire serial interface.

## PKT

The PKT output signal is asserted high when interface buffers are passed from the host to LTS-20 module. PKT can be used to drive an activity LED, as in the example circuit shown in figure 3.1. The output is controlled by writing to the memory mapped I/O at location 0xE7E0—write 0x01 to drive the signal high, and 0x00 to drive the signal low. PKT can source 2mA with $V_{OH} \geq 2.4V$, and it can sink 8mA with $V_{OL} \leq 0.45V$.

## ~RESET

The ~RESET signal is connected to the ~RESET pin of the core module Neuron Chip. The function of the ~RESET pin is described in the *Neuron Chip Data Book.* The core modules include a reset circuit as shown in figure 2.5.

The ~RESET signal should be driven (open collector or open drain only) by a low voltage protection circuit (LVI) on the core module motherboard as described under *Low Voltage Protection* earlier in this chapter. The use of an LVI is critical for reliable operation of the LTS-20.

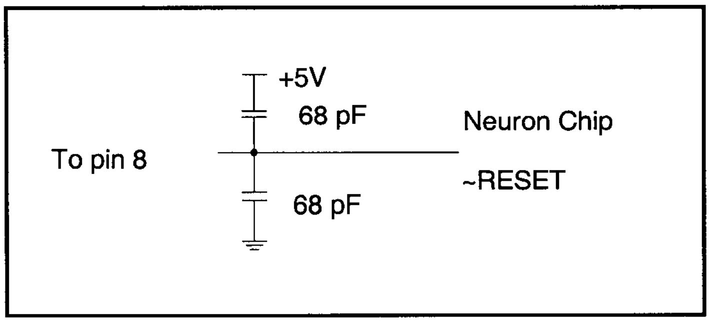


**Figure 2.4** LTS-20 Reset Circuit

## ~RI IN

EIA-232 ring indicator input when the LTS-20 is connected as a DTE device. This input is not used when the LTS-20 is connected as a DCE device. The output is connected to the core module UART and must be externally connected to an EIA-232 receiver if EIA-232 voltage levels are used, or should be externally connected to ground for a 3-wire serial interface.

## ~RI OUT

EIA-232 ring indicator output when the LTS-20 is connected as a DCE device. This output is not used when the LTS-20 is connected as a DTE device. This output is always set to the inactive state by the firmware. The output is driven by the core module UART and must be connected to an EIA-232 driver if EIA-232 voltage levels are required, or can be ignored for a 3-wire serial interface.

## ~RTS

EIA-232 request to send input when the LTS-20 is connected as a DCE device. This input should be used as the EIA-232 clear to send (~CTS) output when the LTS-20 is connected as a DTE device. The input is connected to the core module UART and must be externally connected to an EIA-232 receiver if EIA-232 voltage levels are used, or should be externally connected to ground for a 3-wire serial interface.

## SERIAL IN

EIA-232 received data (RXD) input. The input is connected to the core module UART and must be externally connected to an EIA-232 receiver if EIA-232 voltage levels are used.

## SERIAL OUT

EIA-232 transmitted data (TXD) output. The output is driven by the core module UART and must be connected to an EIA-232 driver if EIA-232 voltage levels are required.

**Table 2.3** LTS-20 Transceiver IDs

| ID | XID [4..0] | Name | Media | Bit Rate (bps) |
|---|---|---|---|---|
| 01 | 00001 | TP/XF-78 | Isolated Twisted Pair | 78k |
| 03 | 00011 | TP/XF-1250 | Isolated Twisted Pair | 1.25M |
| 04 | 00100 | FT-10 | Free Topology, Link Power | 78k |
| 05 | 00101 | TP/RS485-39 | RS-485 Twisted Pair | 39k |
| 09 | 01001 | PL-10 | Power Line (FCC-band) | 10k |
| 10 | 01010 | TP/RS485-625 | RS-485 Twisted Pair | 625k |
| 11 | 01011 | TP/RS485-1250 | RS-485 Twisted Pair | 1.25M |
| 12 | 01100 | TP/RS485-78 | RS-485 Twisted Pair | 78k |
| 15 | 01111 | PL-20A | Power Line (narrow band A-band) | 3.6k |
| 16 | 10000 | PL-20C | Power Line (C-band - CENELEC) | 5k |
| 17 | 10001 | PL-20N | Power Line (C-band – non-CENELEC) | 5k |
| 18 | 10010 | PL-30 | Power Line (A-band) | 2k |
| 24 | 11000 | FO-10 | Fiber Optic | 1.25M |
| 27 | 11011 | DC-78 | Direct Connect | 78k |
| 28 | 11100 | DC-625 | Direct Connect | 625k |
| 29 | 11101 | DC-1250 | Direct Connect | 1.25M |
| 30 * | 11110 | Custom | Custom | N/A |

Notes: Type 30 can be used for any transceiver type; the communications port is initially defined as all inputs to prevent circuit conflicts. When using type 30, the transceiver parameters must be reprogrammed by establishing communication over the serial port, as described in the next chapter.

See Appendix A for a listing of the communications parameters for each transceiver type.

## LTS-20 Software Configuration Options

The types of messages passed between the host and the LTS-20 are determined by EEPROM configuration options. These options are described under *Network Interface Configuration Options* in Chapter 3 of the *LONWORKS Host Application Programmer's Guide*. The *Network Disable Option* affects whether or not the LTS-20 can send and receive application messages. This option is described in Chapter 7 under *Initializing an SLTA.*

The buffer configuration parameters can be changed at any time by sending *Write Memory* network management messages to the LTS-20, either from a host (using local network management messages) or over the network from a network management tool. See the *Neuron Chip Data Book*, Appendix A, for details of the data structures within the Neuron Chip that control the partitioning of RAM for buffers.

The following table summarizes the memory usage of the default configuration. The table also lists the maximum size of the buffer memory pool. If the LTS-20 is configured to use more bytes than are available in the pool, it will most likely crash or behave erratically since the remaining RAM is used by the system firmware.

The default MIP mode EEPROM configuration settings for the LTS-20 are as follows:

| ***Configuration Parameters*** | ***Default Setting*** |
|---|---|
| Initial State | Unconfigured |
| Explicit addressing | Enabled[1] |
| Network variable processing | Host Selection[1] |
| Program ID string | "SLTA" |

| ***Buffer Parameter*** | ***Default Count*** | ***Default Size*** | ***Default Total*** |
|---|---|---|---|
| Receive transaction buffers | 8 | 13[2] | 104 |
| Transmit transaction buffers | 2 | 28[1,2] | 56[1] |
| Application input buffers | 15 | 255 | 3825 |
| Application output buffers | 7 | 255 | 1785 |
| Network input buffers | 31 | 255 | 7905 |
| Network output buffers | 2 | 255 | 510 |
| Priority app. output buffers | 5 | 255 | 1275 |
| Priority net. output buffers | 2 | 255 | 510 |
| **Total bytes used for buffers** | | | **15,970** |

[1]These values apply to LTS-20 Neuron Chip application version 7 only
[2]These values are fixed and cannot be modified

The amount of RAM memory available for buffers in the MIP mode is 25.75 bytes. This total includes both on-chip and off-chip RAM. When calculating the total RAM requirement for a given configuration, remember that there will be a fragmentation boundary when going to the off-chip RAM as buffers are built. This fragmentation may be up to a single buffer size in unusable RAM.

## ~SERVICE

The ~SERVICE signal is connected to the ~SERVICE pin of the module's Neuron Chip. The function of the ~SERVICE pin is described in the *Neuron Chip Data Book.* The internal pullup resistor for the service pin is enabled. A service LED will reflect the firmware status: *blinking* means that the module is unconfigured, *off* means that it is configured, and *on steadily* means that it is applicationless. If the service LED is on steadily, a critical error has been detected by the firmware. A push-button connected to this pin may be used during installation to broadcast the 48-bit Neuron ID on the network.

Typical applications do not require debounce conditioning of momentary push buttons attached to the ~SERVICE pin. The software response time associated with this input is long enough to effectively provide a software debounce for switches with a contact bounce settling time as long as 20 milliseconds.

## ~TEST

The ~TEST input signal is used to put an LTS-20 module in test mode during manufacturing test. Use of this signal is described in the LTSMFT.NC Neuron C file in the manufacturing test directory (MFT) of the software. This signal should be tied high in a shipping, production-level node.

## XID(4..0)

The LTS-20 comes preconfigured with many common LONWORKS transceiver parameters. The XID[4..0] input signals specify a transceiver identification (ID) to select the appropriate transceiver type.

The transceiver ID inputs eliminate a manufacturing step by automatically configuring the LTS-20 for most transceivers. A special transceiver ID is reserved for programming any custom transceiver type. This value causes the communication port pins to be configured as all inputs so that no line will be driven by both the transceiver and LTS-20 Neuron Chip before the chip can be properly configured.

The LTS-20 firmware reads the transceiver ID inputs on both power-up and on reset. If it is being powered-up for the first time, or if the transceiver ID is different from the last time it was powered-up, the parameters specified in table 2.3 are loaded. If it is being re-powered-up, and the transceiver ID is not 30, the LTS-20 firmware compares the network bit rate and input clock for the specified transceiver to the current transceiver parameters. If these parameters don't match, then all transceiver parameters are reinitialized. This allows a network management tool to change parameters, such as the number of priority slots, without the new values being overwritten by the LTS-20 firmware.

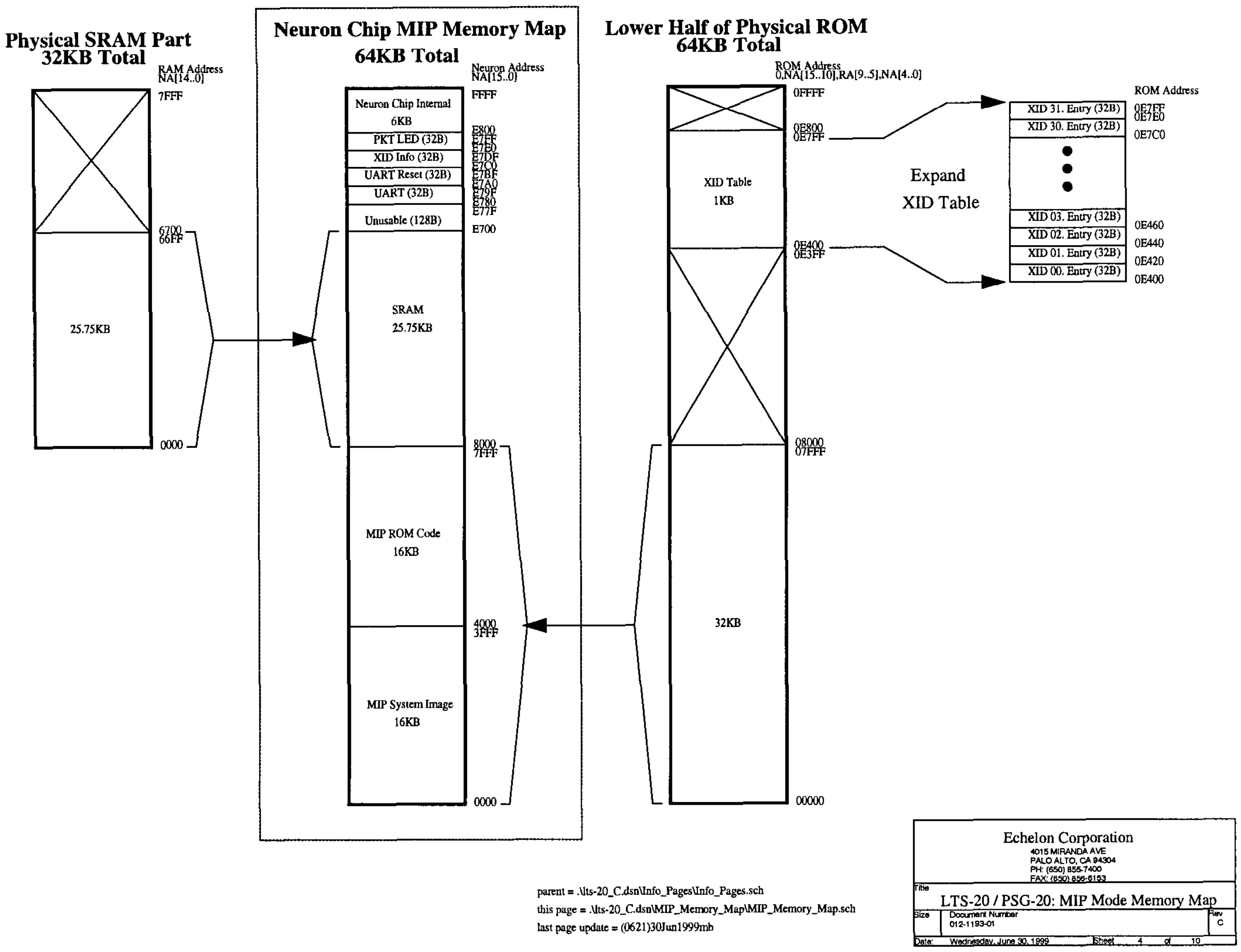


**Figure 2.5** LTS-20 MIP Mode Memory Map

The default NSI mode EEPROM configuration settings for the LTS-20 are as follows:

| *Configuration Parameters* | *Default Setting* |
|---|---|
| Initial State | Unconfigured |
| Explicit addressing | Enabled[1] |
| Network variable processing | Host Selection[1] |
| Program ID string | "SLTA" |

| *Buffer Parameter* | *Default Count* | *Default Size* | *Default Total* |
|---|---|---|---|
| Receive transaction buffers | 16 | 13[2] | 208 |
| Transmit transaction buffers | 2 | 28[1,2] | 56[1] |
| Application input buffers | 3 | 255 | 765 |
| Application output buffers | 3 | 255 | 765 |
| Network input buffers | 2 | 66 | 132 |
| Network output buffers | 2 | 66 | 132 |
| Priority app. output buffers | 3 | 255 | 765 |
| Priority net. output buffers | 2 | 66 | 132 |
| **Total bytes used for buffers** | | | **2,955** |

[1]These values apply to LTS-20 Neuron Chip application version 7 only
[2]These values are fixed and cannot be modified

The amount of RAM memory available for buffers in the NSI mode is 2,955 bytes. This total includes both on-chip and off-chip RAM. When calculating the total RAM requirement for a given configuration, remember that there will be a fragmentation boundary when going to the off-chip RAM as buffers are built. This fragmentation may be up to a single buffer size in unusable RAM.

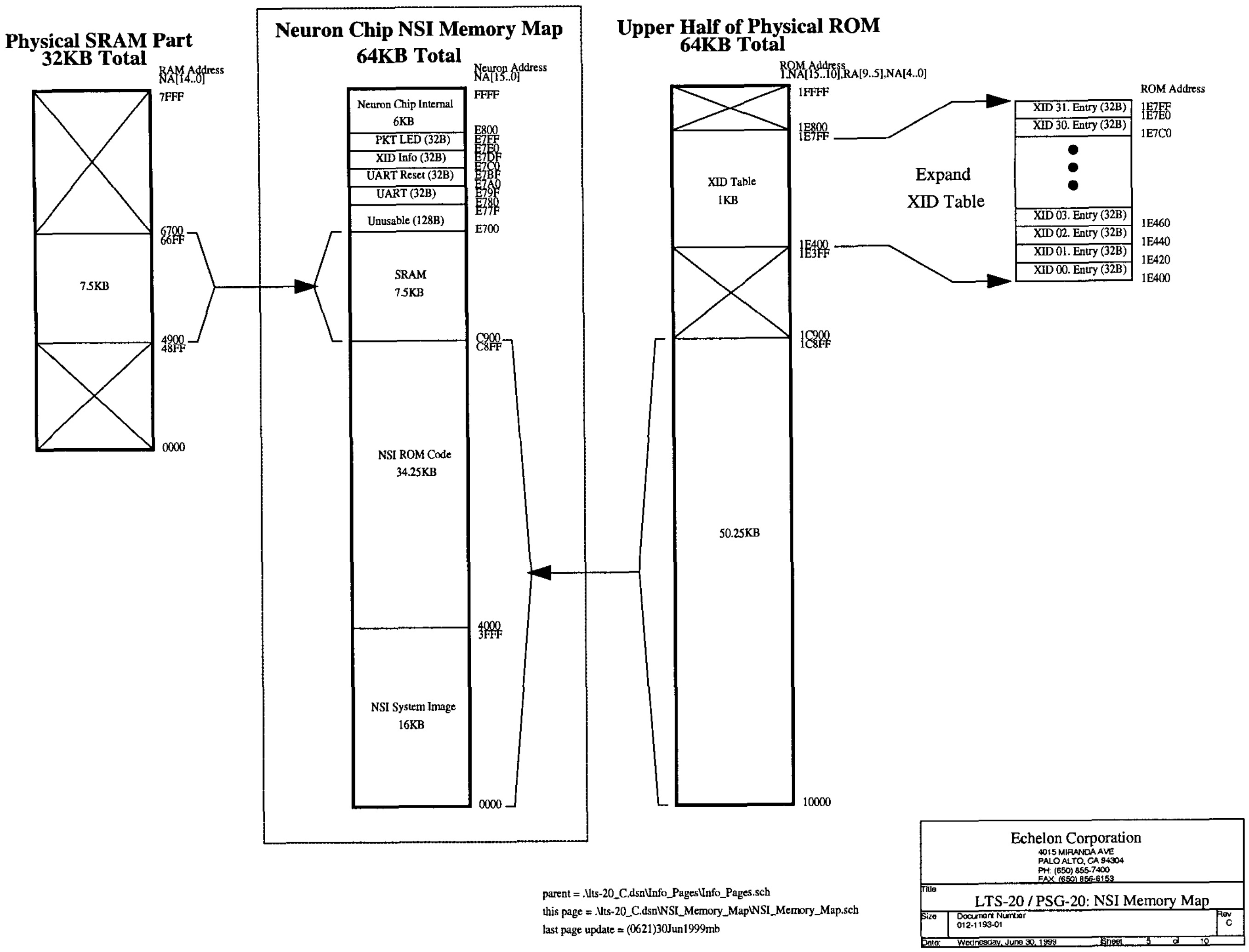


**Figure 2.6** LTS-20 NSI Memory Map

The NODEUTIL node utility application available from the Developer's Toolbox on the Echelon web site (www.echelon.com) can be used to modify the buffer configuration from a PC host. See the README.TXT file included with NODEUTIL for details.

# 3

# Developing an SLTA with the LTS-20 module

This chapter describes the process of developing a Serial LonTalk Adapter based on the LTS-20 Module.

## Overview

To create a complete serial interface (SLTA), with functions similar to Echelon's SLTA-10 Serial LonTalk Adapter, based on the LTS-20 Module, follow these steps:

1. Build an SLTA motherboard according to the specifications described in Chapter 2 and the guidelines described in Chapter 4. The motherboard may be part of custom application hardware, or may be a standalone board. Figure 3.1 is a sample motherboard schematic for an SLTA based on the use of the SMX™ transceivers. Additional transceiver interfaces are described in the rest of this chapter.
2. Ensure that the communications parameters in the LTS-20 are compatible with the transceiver. The transceivers listed in table 2.3 are supported directly by the LTS-20 as predefined types. Set the transceiver ID lines to select the proper transceiver type. For custom transceivers, modify the communications parameters as described under *Using Custom Transceivers* in this chapter.
3. Install the SLTA on a network as described in Chapter 7. The network may be a development network for initial testing, a manufacturing network for configuration during manufacture, or a production network for field installation.

## Using Predefined Transceivers

The LTS-20 includes pre-defined transceiver parameters for the transceivers listed in table 2.3. When using any of these transceivers, the communications parameters are automatically programmed as described in Chapter 2.

The following sections describe the hardware interface for standard LONWORKS transceivers available from Echelon for twisted pair, link power, and power line communications. The user's guide for each transceiver contains documentation on the interface requirements. The following sections provide additional information on using these transceivers with the LTS-20.

### *TPT/XF-78 and TPT/XF-1250 Twisted Pair Transceivers*

The TPT/XF-78 and TPT/XF-1250 Twisted Pair Transceiver Modules support transformer-isolated communications over a twisted pair cable. The transceiver ID should be set to 1 for the TPT/XF-78, and to 3 for the TPT/XF-1250.

See the *LONWORKS TPT Twisted Pair Transceiver Module User's Guide* for details on these channel types.

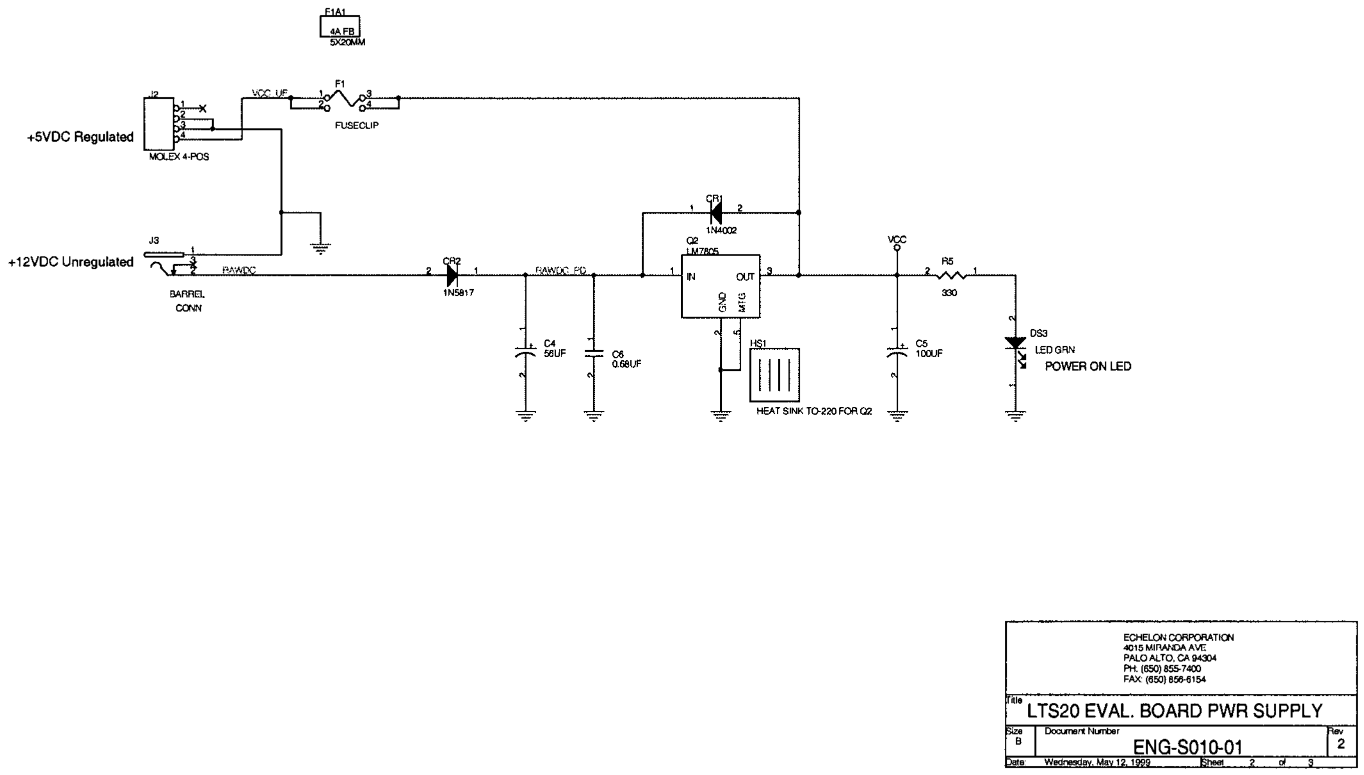


**Figure 3.1** LTS-20 Evaluation Board

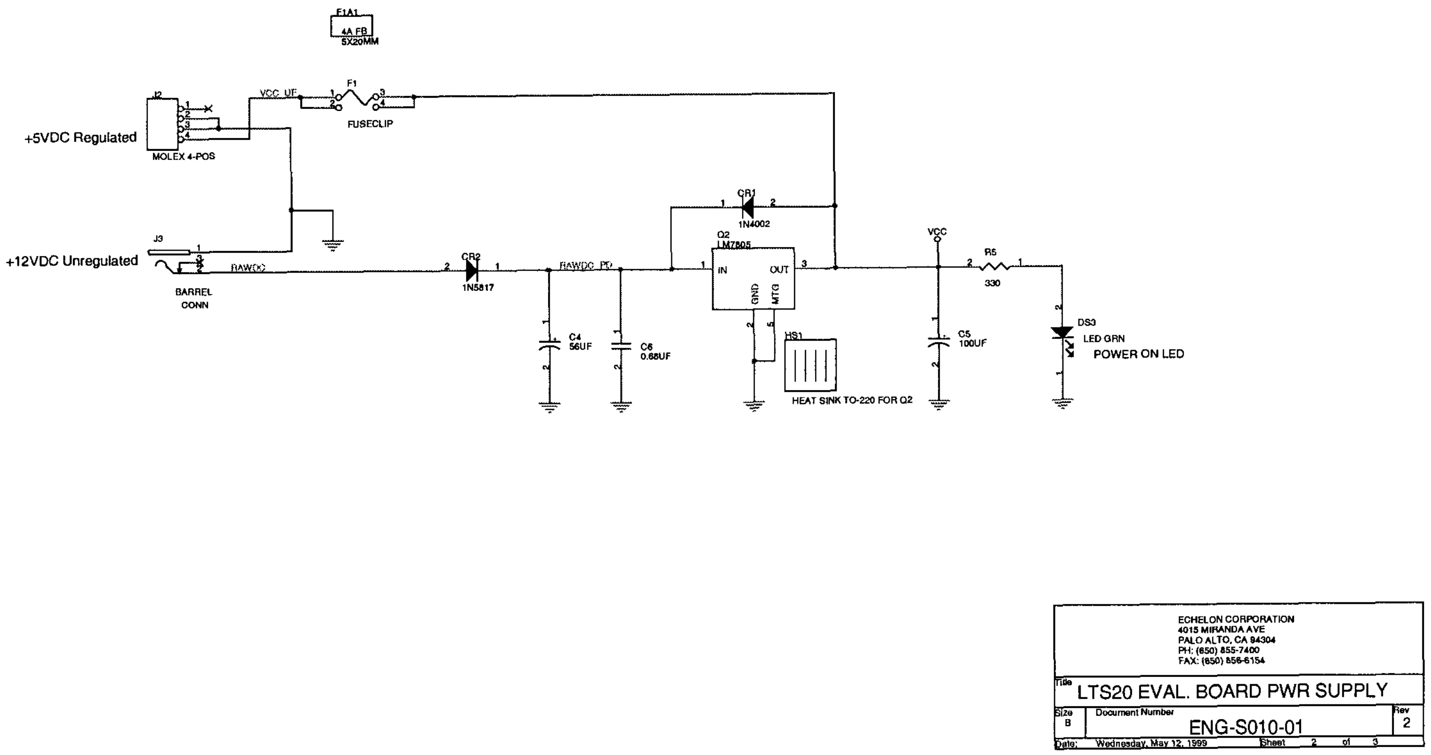


**Figure 3.2** LTS-20 Evaluation Board Power Supply

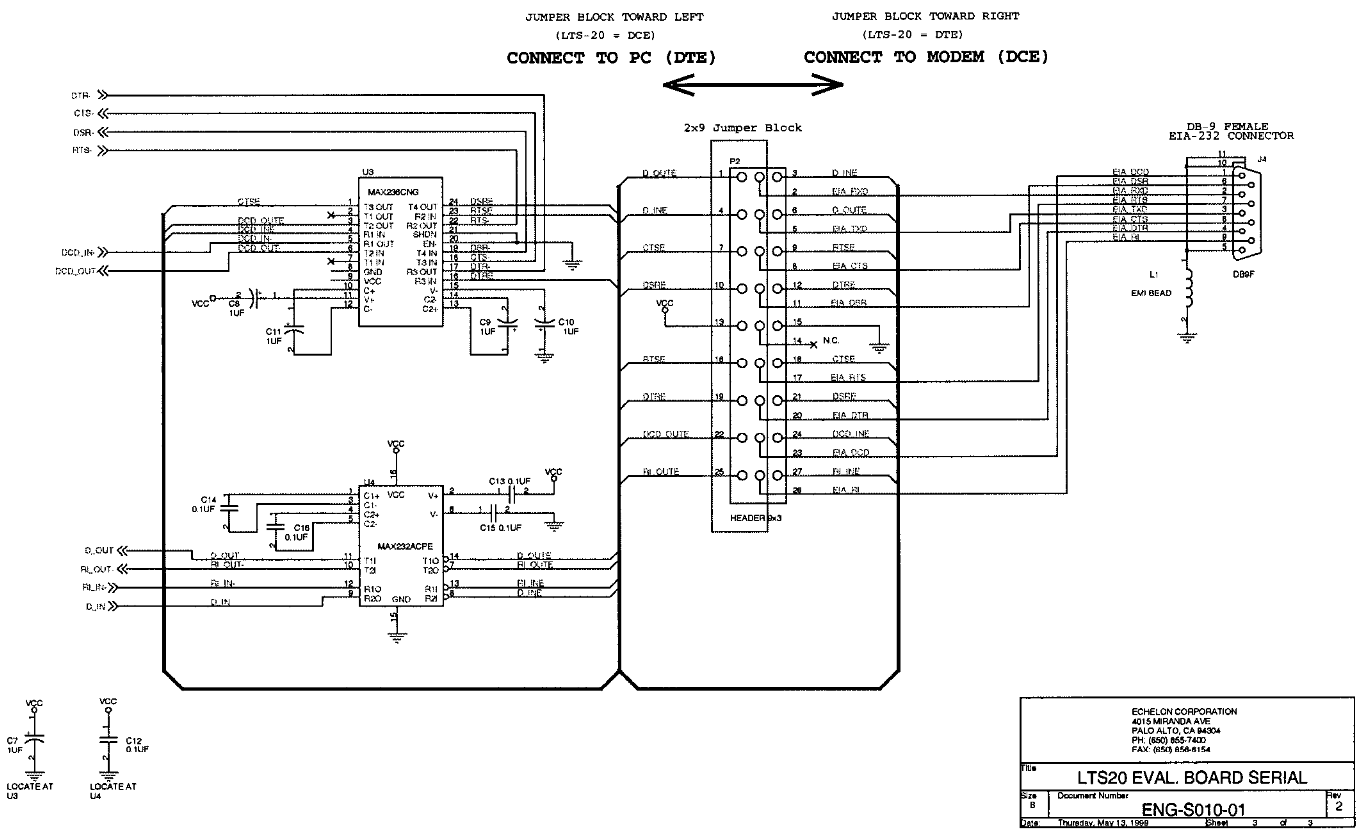


**Figure 3.3** LTS-20 Evaluation Board Serial

## FTT-10A Free Topology and LPT-10 Link Power Transceivers

The FTT-10A Free Topology Transceiver provides 78kbps signaling without regard for cabling topology, and is by far the most popular twisted pair medium for LONWORKS networks. The LPT-10 Link Power Transceiver Module supports free topology communications over the same twisted pair cable that carries power for application nodes. Power is supplied from a 48VDC power supply and is coupled to the network via an LPI-10 Link Power Interface Module. Both a power supply and an LPI-10 module are required to operate LPT-10 transceivers. The LPT-10 transceiver does not provide sufficient power for the LTS-20, which must be locally powered and optically isolated from the LPT-10 transceiver. The transceiver ID input must be set to 4 to support the LPT-10 and FTT-10A transceivers.

Note that an FTT-10A transceiver equipped with decoupling capacitors can operate on a link power segment, but an LPT-10 transceiver cannot operate on an unpowered FTT-10A segment.

## PLT Power Line Transceiver

A PLT Power Line Transceiver Module supports communications over AC or DC power mains. It may be connected to the LTS-20 module and a coupling circuit as shown in figure 3.2. The transceiver ID input must be set to support the correct PLT transceiver. See the pertinent LONWORKS PLT power line transceiver module user's guide for additional information, including a description of the coupling circuits.

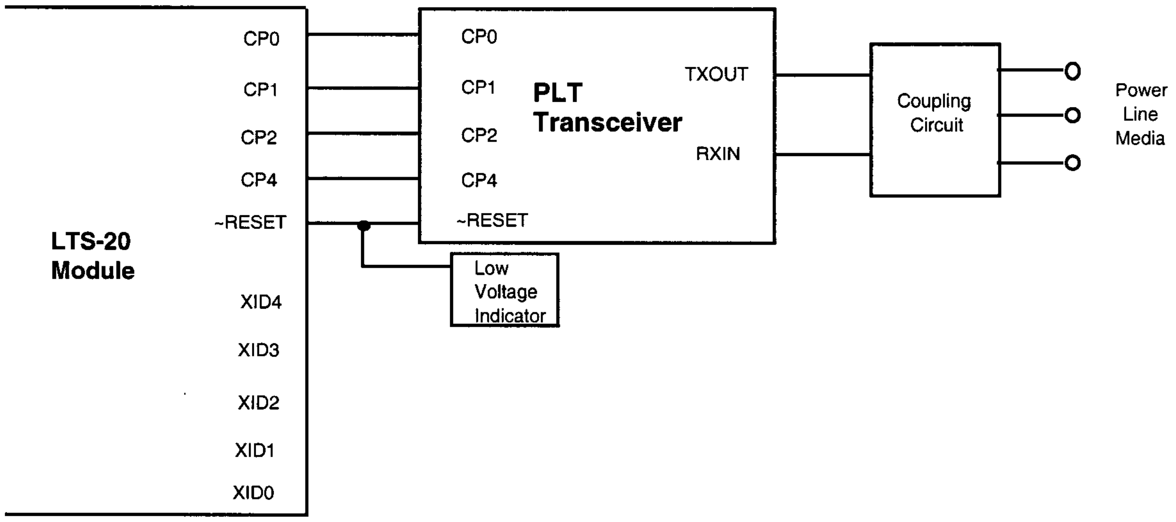


**Figure 3.4** Sample PLT Power Line Transceiver Interface

## Using Custom Transceivers

The LTS-20 module can be used with transceivers not listed in table 2.3 as long as the communications parameters are programmed to match the custom transceiver. Since network communication is not possible before these parameters are set, they must be programmed by the host over the EIA-232 link. The steps for programming a custom transceiver type are:

1 Determine the appropriate transceiver parameters for your channel. A discussion of transceiver modes and parameters may be found in Chapter 6 and Appendix A, section 6 of the *Neuron Chip Data Book*. Transceiver parameters may be modeled and fine-tuned using LonBuilder.

2 Select a transceiver ID of 30 (custom) on the LTS-20 transceiver ID inputs. The pins should remain set to this value in the production SLTA.

3 Install the transceiver parameters using a network management tool such as the LonMaker for Windows Integration Tool. The transceiver parameters are programmed into non-volatile EEPROM so the module will retain the new parameters after power is removed.

# 4

# LTS-20 Design Issues

This chapter examines a number of design issues, including a discussion of electromagnetic interference (EMI) and electrostatic discharge (ESD). These issues should be considered when designing hardware based on the LTS-20 module.

## EMI Design Issues

The high-speed digital signals associated with microcontroller designs can generate unintentional Electromagnetic Interference (EMI). High-speed voltage changes generate RF currents that can cause radiation from a product with a length of wire or piece of metal that can serve as an antenna.

Products that use the LTS-20 module will generally need to demonstrate compliance with EMI limits enforced by various regulatory agencies. In the USA, the FCC requires that unintentional radiators comply with Part 15 level "A" for industrial products, and level "B" for products that can be used in residential environments. Similar regulations are imposed in most countries throughout the world.

Echelon has designed the LTS-20 module with low enough RF noise levels for design into level "B" products. This section describes design considerations to enable products based on the core modules to meet EMI regulations.

## Designing Systems for EMC (Electromagnetic Compatibility)

The LTS-20 module has been designed so that products using them should be able to meet both FCC and, based on radiated emissions, EN55022 level "B" limits. Careful system design is important to ensure that a product based on the core modules will achieve the desired EMC. Information on designing products for EMC is available in several forms including books, seminars, and consulting services. This section provides useful design tips for EMC.

### *EMC Design Tips*

- Most of the EMI will be radiated by the network cable and the power cable.
- Filtering is generally necessary to keep RF noise from getting out on the power cable.
- EMI radiators should be kept away from the LTS-20 module to prevent internal RF noise from coupling onto the radiators.
- The LTS-20 module must be well grounded.
- Early EMI testing of prototypes at a certified outdoor range is an extremely important step in the design of level "B" products. This testing ensures that grounding and enclosure design questions are addressed early enough to avoid most last-minute changes.

## ESD Design Issues

Electrostatic Discharge (ESD) is encountered frequently in industrial and commercial use of electronic systems. Reliable system designs must consider the effects of ESD and take steps to protect sensitive components. Static discharges occur frequently in low-humidity environments when operators touch electronic equipment. The static voltages generated by humans can easily exceed 10kV. Keyboards, connectors, and enclosures provide paths for static discharges to reach ESD sensitive components such as the Neuron Chip. This section describes techniques to design ESD immunity into products based on the LTS-20 modules.

## *Designing Systems for ESD Immunity*

ESD hardening includes the following techniques:

- Provide adequate creepage and clearance distances to prevent ESD hits from reaching sensitive circuitry;
- Provide low impedance paths for ESD hits to ground;
- Use diode clamps or transient voltage suppression devices for accessible, sensitive circuits

The best protection from ESD damage is circuit inaccessibility. If all circuit components are positioned away from package seams, the static discharges can be prevented from reaching ESD sensitive components. There are two measures of "distance" to consider for inaccessibility: creepage and clearance. *Creepage* is the shortest distance between two points along the contours of a surface. *Clearance* is the shortest distance between two points through the air. An ESD hit generally arcs farther along a surface than it will when passing straight through the air. For example, a 20 kV discharge will arc about 0.4 inches (10 mm) through dry air, but the same discharge can travel over 0.8 inches (20mm) along a clean surface. Dirty surfaces can allow arcing over even longer creepage distances.

When ESD hits to circuitry cannot be avoided through creepage, clearance, and ground guarding techniques, i.e., at external connector pins, explicit clamping of the exposed lines is required to shunt the ESD current. Consult *Protection of Electronic Circuits from Overvoltages*, by Ronald B. Standler, for advice about ESD and transient protection for exposed circuit lines. In general, exposed lines require diode clamps to the power supply rails or zener clamps to chassis ground in order to shunt the ESD current to ground while clamping the voltage low enough to prevent circuit damage. The Neuron Chip's communications port lines are connected directly to the LTS-20 edge connector without any ESD protection beyond that provided by the chip itself. If these lines will be exposed to ESD in a custom SLTA, protection must be added to the motherboard.

# 5

# The LTS-20 Software

This chapter describes the LTS-20 software that is shipped with the Connectivity Starter Kit.

## Software Overview

The LTS-20 software includes ANSI C source code for HA, a sample host application for MS-DOS that can be used as a basis for a user-developed host application on other host platforms. This application provides examples of sending and receiving network variable messages, as well as allowing a node based on an LTS-20 to be installed and bound by a network management tool such as the LonManager LonMaker for Windows Integration Tool or the LonBuilder network manager.

Two network drivers (Windows 95/98 and Windows NT) are included so that an LTS-20 may be immediately used with LNS applications. Source code for DOS and UNIX network drivers is also provided as a basis for a user-developed network driver for other hosts or operating systems using the MIP. DLL software is provided to make it easier to use the network driver under the Microsoft® Windows operating system.

An executable program and source code is also provided for a Host Connection Utility (HCU), which may be used to initiate and terminate the host to serial connection when the LTS-20 is used with a remote host. An example written in Neuron C is also provided as a basis for user-developed nodes on a LONWORKS network that need to initiate outgoing calls to a remote host.

The LTS-20 includes NSI firmware that moves the upper layers of the LonTalk Protocol off the Neuron Chip within a node onto a host processor. This firmware allows the LTS-20 to be used by a host application to send and receive LonTalk messages. The host application may be a custom application as described in the *LNS for Windows Developer's Kit or LNS DDE Server User's Guide*. When using the LTS-20 in the MIP mode, the host application may also be a network management application based on tools using the now-discontinued LonManager API. The firmware in an LTS-20 is fixed in ROM and need not be reprogrammed to use any of the module's capabilities.

## Installing LTS-20 Software

The LTS-20 software is supplied on a diskette, together with an installation program. To install the LTS-20 software, follow these steps:

1. Place the diskette in one of the disk drives of your PC. This will typically be the A: or B: drive.

2. Start the automatic installation procedure by entering:

```
A:INSTALL [ENTER]
```

Substitute your disk drive name for the A: if you are using a different drive.

3. You will be asked to enter the name of your LONWORKS installation directory. The default is:

```
C:\ECHELON
```

If you have other Echelon software products installed in the \LONWORKS directory, rather than the \ECHELON directory, enter \LONWORKS in place of the default directory name.

The LTS-20 software will be installed in the LTS-20 sub-directory of your LONWORKS directory, with the exception of the DOS network driver LDVSLTA.SYS. This file will be installed in the BIN sub-directory of your LONWORKS directory. To install the DOS network driver into your CONFIG.SYS file, follow the instructions in Chapter 9.

The SLTA directory will contain the following files:

- **Read-Me File**. The README.TXT file includes a list of all the files on the distribution disk, and also includes any updates to the documentation that occurred since the documentation was printed.
- **DOS Network Driver Sources**. The DOS network driver source code is contained in the LDVSLTA directory. These files can be used as the basis for creating drivers for hosts other than PCs running DOS (see also the UNIX network driver sources). See the README.TXT file for a description of the driver files. See Chapter 8 for a description of the DOS network driver and Chapter 7 for a description of how to write a network driver for other hosts. See Chapter 4 of the *LONWORKS Host Application Programmer's Guide* for a description of the services that must be supplied by a LONWORKS network driver.

  The source files to build the DOS driver are:

  | | |
  |---|---|
  | LDVSLTA.CFG | Configuration file for Borland C. |
  | MAKEFILE | Make file script for Borland C. |
  | MDV_TIME.C | Code to manage the PC timer. |
  | MDV_TIME.H | External interface definitions for the timer handler. |
  | MSD_DEFS.H | Data structure and literal definitions. |
  | MSD_DIFC.C | DOS driver interface functions. |
  | MSD_DRVR.H | DOS driver interface and literal definitions. |
  | MSD_EXEC.C | Main open, close, read, and write processing. |
  | MSD_FRST.C | Module to be linked first in the network driver. |
  | MSD_IRQC.ASM | Serial I/O interrupt procedure. |
  | MSD_LAST.C | Module to be linked last in the network driver. |
  | MSD_RAW.C | Direct serial I/O (modem) processing. |
  | MSD_SEGD.ASM | Defines data segment register for driver. |
  | MSD_SIO.C | PC/AT UART interface processing. |
  | MSD_TXRX.C | Single byte link layer processing. |
  | MSD_UART.H | Defines PC/AT UART registers. |

- **UNIX Network Driver Sources**. The UNIX network driver source code is contained in the UNIX directory. These files can be used as the basis for creating drivers for any UNIX host, and can also be used as the basis for developing drivers for other hosts. See Chapter 10 for a description of the UNIX network driver and Chapter 8 for a description of how to write a network driver for other hosts. See Chapter 4 of the *LONWORKS Host Application Programmer's Guide* for a description of the services that must be supplied by a LONWORKS network driver. The source files to build the UNIX driver are:

| | |
|---|---|
| `LDVSLTA.C` | UNIX driver functions. |
| `LDVSLTA.H` | UNIX driver declarations. |

- **External Interface Files**. External interface files included for use by network management tools are contained in the `LTS-20` directories. Fifteen external interface files are included for the standard transceiver types that are directly supported by the LTS-20. See *Binding to a Host Node* in Chapter 3 of the *LONWORKS Host Application Programmer's Guide* for a description of how to use these files to bind to an SLTA node. Appendix B of the *LONWORKS Host Application Programmer's Guide* provides a detailed description of how to modify these files to incorporate network variables and message tags. These interface files are provided in version 3 formats; version 2 formats are available by running the utility `XIF3TO2.EXE` (available from Echelon's ftp site) on the version 3 XIF files. Version 3 external interface files are compatible with the latest releases of all Echelon software products. External interface files in version 3 format are contained in the `SLTA2\XIF_V3` and `LTS-20\XIF_V3` directories.

Each SLTA/2 directory contains the following files:

| | |
|---|---|
| `NSLTA125.XIF` | For SLTA/2 with a TP/XF-1250 transceiver. |
| `NSLTA78K.XIF` | For SLTA/2 with a TP/XF-78 transceiver. |
| `NSLTA485.XIF` | For SLTA/2 with a TP-RS485-39 transceiver. |
| `NSLTAFT1.XIF` | For SLTA/2 with a TP/FT-10 transceiver. |

Each LTS-20 directory contains the following files:

| | |
|---|---|
| `LTS1250.XIF` | For LTS-20s with a TP/XF-1250 transceiver. |
| `LTS78K.XIF` | For LTS-20s with a TP/XF-78 transceiver. |
| `LTS485A.XIF` | For LTS-20s with a TP-RS485-39 transceiver. |
| `LTS485B.XIF` | For LTS-20s with a TP-RS485-78 transceiver. |
| `LTS485C.XIF` | For LTS-20s with a TP-RS485-625 transceiver. |
| `LTS485D.XIF` | For LTS-20s with a TP-RS485-1250 transceiver. |
| `LTSFT10.XIF` | For LTS-20s with a FT-10 or LPT-10 transceiver. |
| `LTSPL10.XIF` | For LTS-20s with a PL-10 transceiver. |
| `LTSPL20A.XIF` | For LTS-20s with a PL-22 transceiver (A-band). |
| `LTSPL20C.XIF` | For LTS-20s with a PL-21/PL-22 transceiver (CENELEC protocol on). |
| `LTSPL20N.XIF` | For LTS-20s with a PL-21/PL-22 transceiver (CENELEC protocol off). |
| `LTSPL30.XIF` | For LTS-20s with a PL-30 transceiver. |
| `LTSFO10.XIF` | For LTS-20s with a FO-10 transceiver. |

| | |
|---|---|
| `LTSDC78.XIF` | For LTS-20s using Direct Connect at 78kbps. |
| `LTSDC625.XIF` | For LTS-20s using Direct Connect at 625kbps. |
| `LTSDC125.XIF` | For LTS-20s using Direct Connect at 1250kbps. |

- **Sample Host Application**. A sample host application is contained in the HA directory. See Appendix A of the *LONWORKS Host Application Programmer's Guide* for a description of the example. The following files are included:

| | |
|---|---|
| `README.TXT` | A description of the sample host application. |
| `HA.EXE` | An executable version of the sample host application for DOS. The SLTA DOS network driver must be installed to run this application. |
| `HA.C` | The main program for the example. |
| `NI_MSG.C` | A general purpose network interface library that can be used with any host application. |
| `APPLCMDS.C` | Functions to handle application layer network variable commands |
| `NI_CALLB.C` | The host-bound network management dispatcher. |
| `APPLMSG.H` | Application message handler function prototypes. |
| `HA_COMN.H` | The HA common declarations. |
| `NI_CALLB.H` | The definitions for the network management dispatcher. |
| `APPLMSG.C` | Functions to handle application network variable and explicit messages. |
| `HAUIF.C` | Command-line user interface for the example. |
| `IOCTL.C` | I/O control function for Microsoft C. |
| `LDVINTFC.C` | Device interface driver. |
| `LDVINTFC.H` | Include file for device driver interface. |
| `NI_MSG.H` | Definitions for network interface message structures. |
| `NI_MGMT.H` | Definitions for network management message structures used by the example. |
| `HAUIF.H` | Definitions for the host application example user interface. |
| `MAKEFILE` | A make file script for Borland C. |
| `MSOFT.MAK` | A make file script for Microsoft C. |
| `HA_V3.XIF` | An external interface file which may be used to bind the example with LonBuilder. |
| `HA_TEST.NC` | A Neuron C program which may be loaded into a Neuron emulator and bound to the sample host application for testing. |

| | |
|---|---|
| `DISPLAY.H` | A Neuron C include file to drive the Gizmo 2 I/O module for the test example. |

- **Host Connect Utility**. A sample host connection utility is contained in the `HCU` directory, with source code. See Chapter 12 for details. The files supplied are:

| | |
|---|---|
| `HCU.EXE` | Executable file for the Host Connection Utility. |
| `HCU_MAIN.C` | The main C source program. |
| `HCU.CFG` | Configuration file for Borland C. |
| `MAKEFILE` | Make file script for Borland C. |
| `MSD_DRVR.H` | Driver definition include file. |

- **Neuron C Connection Example**. A sample Neuron C program is contained in the `NC_APPS` directory. This program shows how a node on a network connected to the SLTA can dial out and connect to a remote host computer. See Chapter 11 for details. The files supplied are:

| | |
|---|---|
| `DIALOUT.NC` | Neuron C source program to dial out with the SLTA. |
| `GIZSETUP.NC` | An example Neuron C program for configuring the SLTA. Configures the EEPROM directories of an SLTA using the Gizmo 2 I/O module as the user interface. |
| `SLTA_ANM.H` | Definitions of SLTA-specific network management messages. |

- **Manufacturing Test Files**. The files supplied in the `LTS-20\MFT` directory provide a Neuron C application example which can be used as a manufacturing test aid for products based on the LTS-20. They are:

| | |
|---|---|
| `LTSMFT.NC` | Neuron C source file, including full documentation. |
| `LTSMFT.H` | Include file. |

This application is designed to aid in the testing of circuitry that is external to the LTS-20 module, such as EIA-232 interface drivers and connectors. It may be programmed into a LONWORKS device which then communicates with the LTS-20 module via the network. The LTS-20 circuitry is tested with some of its signals connected in a loopback manner. The assertion of the TEST input (pin 30) will cause the LTS-20 firmware to come up in the test mode.

## Installing the Windows DLL Software

A second diskette contains the Windows Dynamic Link Library (DLL) files. These files may be used when developing a host application to run under Microsoft Windows. The file `WLDV.DLL` should be copied to your Windows directory (typically `C:\WINDOWS`). The files `LDV.H` and `LON.H` should be copied to a directory in the include file search path of your C compiler. The file `WLDV.LIB` should be copied to a directory in the library search path of your application linker. See Appendix B for information on using the Windows DLL.

# 6 Creating an LTS-20 MIP Mode Network Driver

This chapter describes the process of building a network driver for a host that is to be connected to an LTS-20 operating in MIP mode. The example network drivers for DOS, Windows, and UNIX are described. Similar logic can be used on other host processors and operating systems. This chapter also includes a description of the network interface protocol for the LTS-20 operating in MIP mode. The network interface protocol defines the format of the data passed across the EIA-232 interface, and varies depending on the configuration of the LTS-20 and the network driver. If a LONWORKS standard network driver is used, the format of the data passed between the driver and the application is defined by the network driver protocol and is independent of the network interface protocol; the driver is responsible for providing the necessary translations. This chapter will therefore be of interest only to those needing to develop a network driver for a host other than DOS, Windows, or UNIX.

*If you are using a DOS, Windows, or Unix host, you can skip this chapter and instead read Chapters7 or 8, which describe the DOS and UNIX network drivers.*

## Purpose of the Network Driver

The network driver provides a hardware-independent interface between the host application and the network interface. By using network drivers with consistent calling conventions, host applications can be transparently moved between different network interfaces. For example, the standard LTS-20 MIP mode DOS driver, together with the Windows DLL software, allows DOS and Windows applications, such as those based on the LonManager API, to be debugged using the network driver for the LonBuilder Development Station. These applications can later be used with the network driver for the SLTA-10 operating in MIP mode, without modifying the host application.

**For the purposes of this chapter, the term "SLTA" refers to the LTS-20 module operating in MIP mode.**

A LONWORKS standard network driver must supply the functions defined under *Network Driver Services* in Chapter 4 of the *LONWORKS Host Application Programmer's Guide*. The Windows DLL software is described in Appendix B.

## Example Network Drivers

The SLTA is delivered with source code for example network drivers for DOS, Windows, and UNIX. The DOS driver is used for both DOS and Windows applications. See the comments in the source code of the network drivers for an explanation of how the network drivers work. These drivers can be used as templates for a LONWORKS standard network driver. The DOS network driver is compatible with the LonManager APIs for DOS and Windows, LonMaker, and the LonManager DDE Server. A sample host application for DOS is also supplied. The functions `ldv_open()`, `ldv_read()`, `ldv_write()`, and `ldv_close()` form a suitable operating-system independent definition for the network driver. These functions support multiple network interfaces, and hide the DOS-specific aspects of the DOS network driver.

The UNIX network driver is a source library that uses the UNIX serial device driver. It also supports the `ldv_open()`, `ldv_read()`, `ldv_write()`, and `ldv_close()` functions.

## Implementing an SLTA Network Driver

The network driver manages the physical interface with the SLTA, implements the network interface protocol, performs flow control, manages input and output buffers, and provides a read/write interface to the host application.

Figure 6.1 illustrates how the network driver fits into the host application architecture.

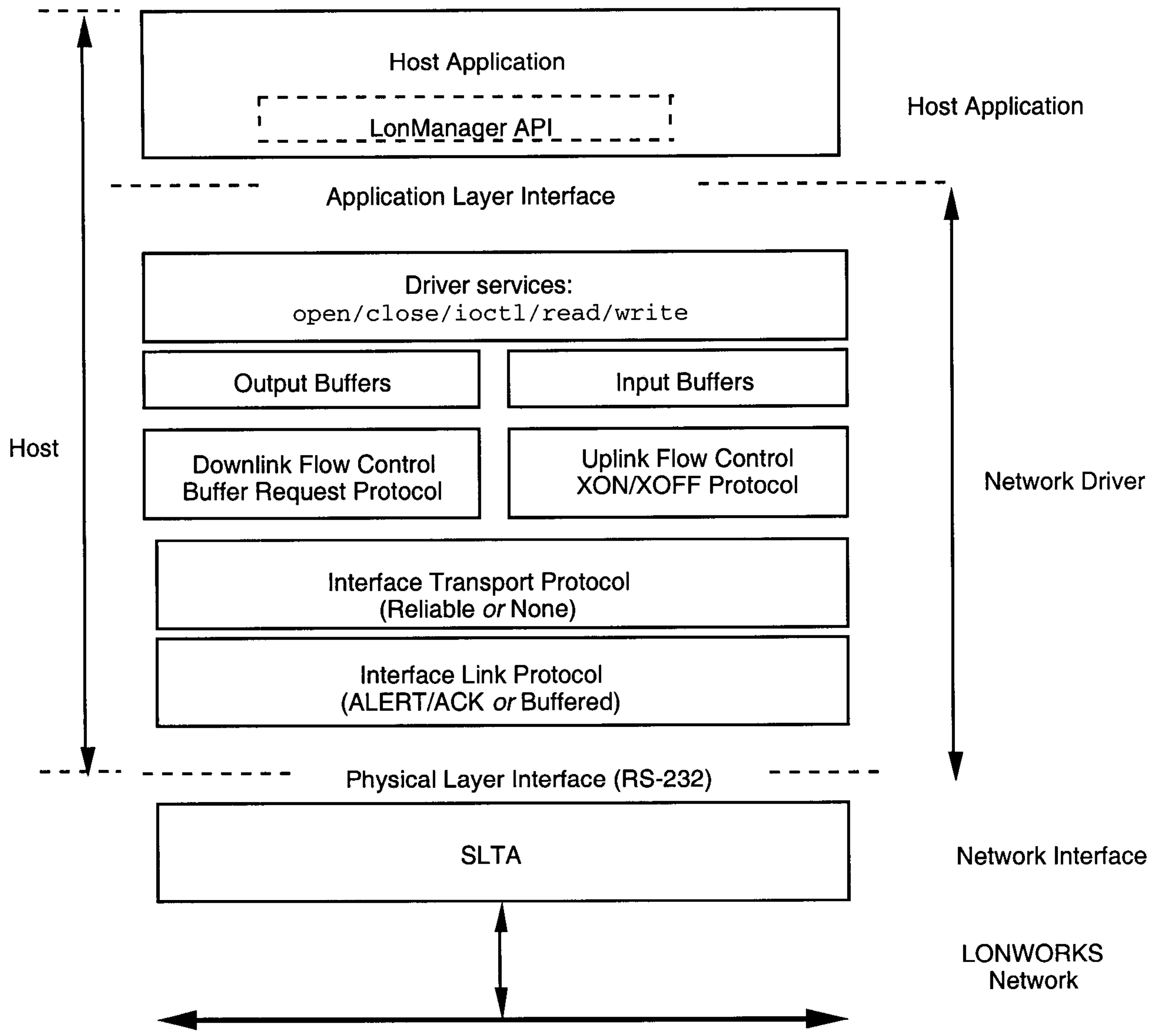


**Figure 6.1** Host Application Architecture

To implement an SLTA network driver for a host other than DOS, Windows, or UNIX, follow these steps:

**1** Implement and test low-level serial I/O. Serial I/O may be performed directly to the host's UART as is done in the DOS network driver, or may be performed by a serial I/O driver on the host as is done by the UNIX network driver. Serial I/O should be interrupt driven for better performance.

The UNIX network driver uses the UNIX serial port driver for all low-level serial I/O and interrupt support. This simplifies the driver and also simplifies porting between different versions of UNIX. The serial device is opened by the `ldv_open()` function and closed by the `ldv_close()` function. Data is read

from and written to the serial device using the UNIX `read()` and `write()` system calls.

The UNIX network driver includes a `ldv_post_events()` function that should be called periodically from the client application in order to assure that the SLTA traffic is being processed.

The DOS network driver serial I/O functions are implemented by `MSD_SIO.C`, `MSD_UART.H`, and `MSD_IRQC.ASM`. These files may all be replaced as long as the required serial I/O functions in `MSD_SIO.C` are provided. The definitions of the UART registers are in `MSD_UART.H`. The DOS serial I/O interrupt service routines are in `MSD_IRQC.ASM`.

The DOS network driver uses the DOS system timer tick interrupt (vector 0x1C) and the serial I/O device interrupt for the relevant COM port to perform background processing of the serial network interface. The driver hooks into these interrupt vectors and executes driver code whenever the LON(n) device is opened. Flags internal to the driver prevent the interrupt code thread from interfering with the normal application foreground execution of functions within the driver.

The `smip_int_main()` function in the DOS network driver services the serial port connected to the network interface. The function `tick_int_main()` services the timer tick interrupt every 55 msec.

Both network drivers are fully buffered for both outgoing and incoming messaging. Read and write functions work with circular buffers within the driver. The host interrupt service routine handles the other ends of these buffer queues.

Both network drivers only support a single set of output buffers. An elaboration on this design could implement a set of priority output buffers. The write function could determine into which of the two buffer sets to place messages, and the driver service function could service the priority buffers first.

**2** Implement and test timer support functions. Timer support may be provided by a hardware timer as is done in the DOS network driver, by a system service as is done in the UNIX network driver, or by implementing a background software task. The UNIX network driver uses a once per second signal that is handled by the `second_service()` function. The DOS timer functions are implemented by `MDV_TIME.C` and `MDV_TIME.H`.

**3** Implement and test the host side of the network interface protocol. The network interface protocol is implemented by the `rx_process()` and `tx_process()` functions in the UNIX driver, and by the functions in `MSD_TXRX.C` for the DOS network driver.

**4** Implement and test raw modem I/O if you need to support a modem interface. Raw I/O manages the serial interface to the modem when the modem is not connected to a host and is used for modem initialization and control. The raw I/O interface is implemented in `MSD_RAW.C` for the DOS network driver, and is not implemented in the UNIX network driver.

**5** Implement and test the buffer request states, buffer management, and read/write interfaces. These functions are implemented by `MSD_EXEC.C` for the DOS

network driver. The read/write interface is implemented in the `ldv_read()` and `ldv_write()` functions for the UNIX network driver

The following files are unique to a DOS driver and would probably not be used in a port to another host: `MSD_DRVR.H`, `MSD_DIFC.C`, `MSD_FRST.C`, `MSD_LAST.C`, `MSD_SEGD.ASM`.

## Network Interface Protocol

The network driver implements the host side of the network interface protocol, providing an easy-to-use and interface-independent read/write interface to the host application. The network interface protocol is a layered protocol that includes the following layers:

- **Presentation Layer**. Defines packet formats for network variables and explicit messages. This is the only layer visible to the host application. The remaining layers are managed by the network driver.
- **Session Layer**. Manages flow control, buffer requests, and grants.
- **Transport Layer**. Ensures end-to-end reliability between the host and the SLTA.
- **Link Layer**. Controls access to the serial link.
- **Physical Layer**. EIA-232 interface.

The physical layer is defined by the EIA RS-232 standard. The link, transport, session, and presentation layers are described in the following sections.

## Link Layer Protocol

The default interface link layer protocol is the *ALERT/ACK protocol*. This protocol may be used when the host is a microcontroller or microprocessor such as a PC running DOS or Windows. The alternative interface link protocol is the *buffered protocol*. This protocol is used with computer hosts that can asynchronously buffer an entire packet. All data are transmitted using 1 start bit, 8 data bits, no parity bits, and 1 stop bit.

### *ALERT/ACK Link Protocol*

The DOS network driver uses the ALERT/ACK link protocol by default (i.e. the `/N` option is not specified). See Chapter 8 for a description of the network driver options. The UNIX network driver uses the ALERT/ACK link protocol if the `alert_ack_prtcl` variable is set to TRUE in the source code (this is not the default). The CFG3 input of the SLTA, as described in Chapter 6, must be in the *ALERT/ACK* state.

When using this protocol, all transfers between the SLTA and the host consist of serial data streams that start off with the link-layer header sequence described in figure 6.2. Whenever one device, either the SLTA or the host, needs to send a command or message, the sender starts the sequence by transmitting the ALERT byte (value 01 hex). When this byte is received by the receiver, that device responds by transmitting the ALERT ACK byte (value FE hex). This low level handshaking process prevents the sender from transmitting the rest of the sequence before the receiving device is ready. Once the ALERT ACK byte is received by the sender it sends the rest of the message without any other interactions.

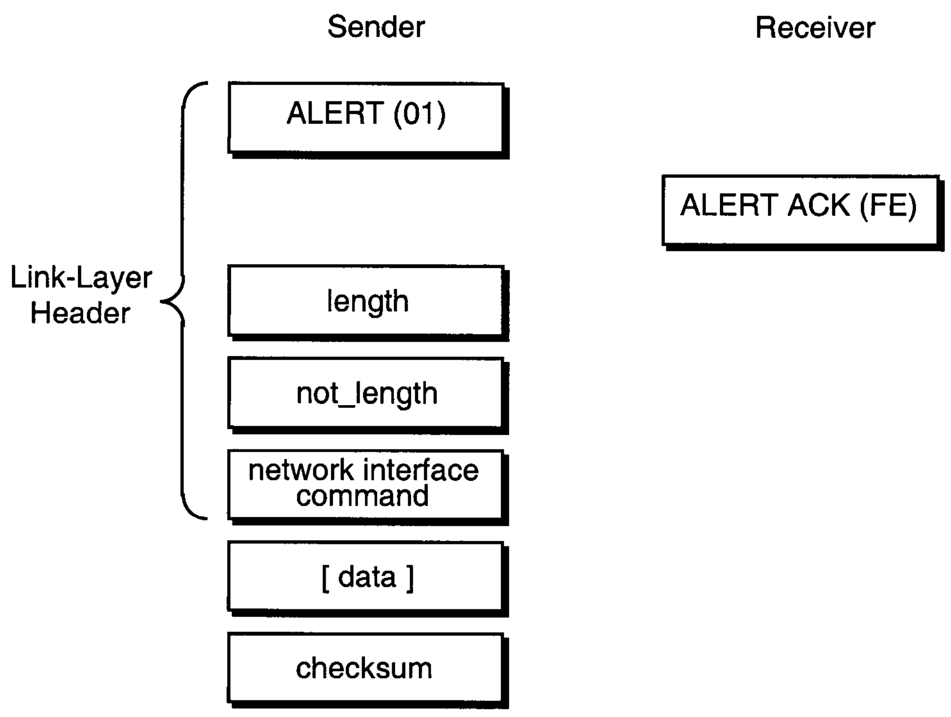


**Figure 6.2** SLTA ALERT/ACK Link Protocol

The link-layer header contains a length byte followed by a one's complement of the length byte. These values are always validated by the receiver before accepting the rest of the message. Following the length bytes is the network interface command. See Appendix D of the *Host Application Programmer's Guide* for a description of the command byte structure. If the message contains a data field it follows the command byte. Finally, a checksum terminates the sequence.

The length byte value describes the length of the network interface command byte plus the length of the data field. This value will always be at least 1. The checksum is a two's complement of the sum of the command byte and all of the bytes in the data field, if it exists. Checksum errors detected by the host will cause an error to be reported to the application, and checksum errors detected by the SLTA will cause the message to be ignored.

The SLTA places the following requirements on the rate of the received serial data stream. When receiving, the maximum wait period for the length byte following the transmission of the ALERT ACK byte is 100ms (or 1 second when attached to a modem). All subsequent bytes received must occur within 100ms after the previous byte, otherwise the SLTA receive process will abort. Likewise, the SLTA uses a wait period of 100ms (or 1 second when attached to a modem) before aborting for the reception of the ALERT ACK when transmitting a message. If the ALERT ACK is not received in time, the SLTA repeats the process by transmitting another ALERT byte.

The SLTA cannot support a full duplex communications process between it and the host. The network driver included with the SLTA takes this into account. Data frames transmitted to the SLTA while it is in the process of sending uplink messages will be lost if more than 16 bytes are sent to the SLTA.

## Buffered Link Protocol

The DOS network driver uses the buffered link protocol when the /N option is specified. See Chapter 7 for a description of this option. The UNIX network driver uses the buffered link protocol if the `alert_ack_prtcl` variable is set to FALSE in the source code (this is the default). The CFG3 input of the SLTA, as described in Chapter 6, must be in the buffered protocol state.

When using this protocol, the link-layer header contains a length byte followed by a one's complement of the length byte. These values are always validated by the receiver before accepting the rest of the message. Following the length bytes is the network interface command. See Appendix D of the *Host Application Programmer's Guide* for a description of the command byte structure. If the message contains a data field it follows the command byte. Finally, a checksum terminates the sequence.

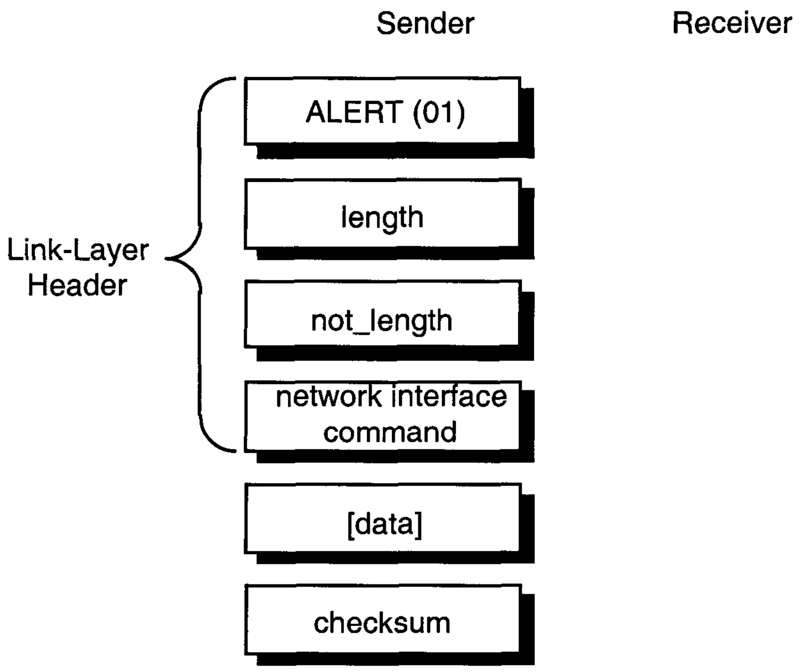


**Figure 6.3** SLTA Buffered Link Protocol

The length byte value describes the length of the network interface command byte plus the length of the data field. This value will always be at least 1. The checksum is a two's complement of the sum of the command byte and all of the bytes in the data field, if it exists. Checksum errors detected by the host will cause an error to be reported to the application, and checksum errors detected by the SLTA will cause the message to be ignored.

This protocol is used when the host is capable of accepting asynchronously occurring input data without losing characters. The host is also relieved of the obligation of responding to an ALERT character within 50 ms. This protocol may therefore be used by an application-level handler calling an interrupt-driven buffered serial device driver. Drivers with these characteristics are typically provided with real time operating systems such as VRTX or time-sharing operating systems such as UNIX or VMS. In this case, these drivers should be set up for binary data communications without software flow control.

The buffered link protocol should not be used when the SLTA is attached to a modem.

The buffered link protocol can only be used on multitasking operating systems such as UNIX if the host application executes often enough to empty any incoming buffers. For example, if the SLTA is receiving 70 packets per second, and each packet is 25 bytes, the host will receive 1750 bytes per second. If the host has a serial input buffer of 256 bytes, the buffer will fill within 150 milliseconds if the host application is preempted. If the host application is preempted for longer than 150 milliseconds, incoming data will be lost due to lack of serial buffer space. In this case, the ALERT/ACK protocol should be used, or the buffer space increased to handle the worst case traffic during the maximum preemption period.

## Transport Layer Protocol

When used with a local host, the SLTA assumes a reliable connection and does not use a transport layer protocol. When used with a remote host, the SLTA assumes that the link may not be reliable and enables the reliable transport protocol. The reliable transport protocol adds an ACK/NACK transport protocol to the network interface protocol. A sequence number is also added to the link-layer header. This protocol can therefore recover from checksum errors on the host to SLTA link.

The reliable transport protocol is enabled on the SLTA with the *Remote Host* option selected by the CFG2 input as described in Chapter 12. The reliable transport protocol is enabled on the DOS network driver with the `/M` option as described in Chapter 7. The reliable transport protocol is not supported by the UNIX network driver.

The link-layer header contains an ALERT (0x01) byte, a sequence number, and a length byte followed by a one's complement of the length byte. These values are always validated by the receiver before accepting the rest of the message. Following the length bytes is the network interface command. See Appendix D of the *Host Application Programmer's Guide* for a description of the command byte structure. If the message contains a data field it follows the command byte. Finally, a checksum terminates the sequence.

The ALERT/ACK link protocol should be used with remote hosts. With this protocol, the sender will start the sequence by transmitting the ALERT byte. When this byte is received by the receiver, that device responds by transmitting the ALERT ACK byte (value FE hex). This low level handshaking process prevents the sender from transmitting the rest of the sequence before the receiving device is ready. Once the ALERT ACK byte is received by the sender it sends the rest of the message without any other interactions.

The length byte value describes the length of the network interface command byte plus the length of the data field. This value will always be at least 1. The checksum is a two's complement of the sum of the command byte and all of the bytes in the data field, if it exists. If the receiver receives a message in sequence, with a valid checksum, it responds with an ACK (0x06). Otherwise it responds with a NACK (0x15), requesting a re-transmission.

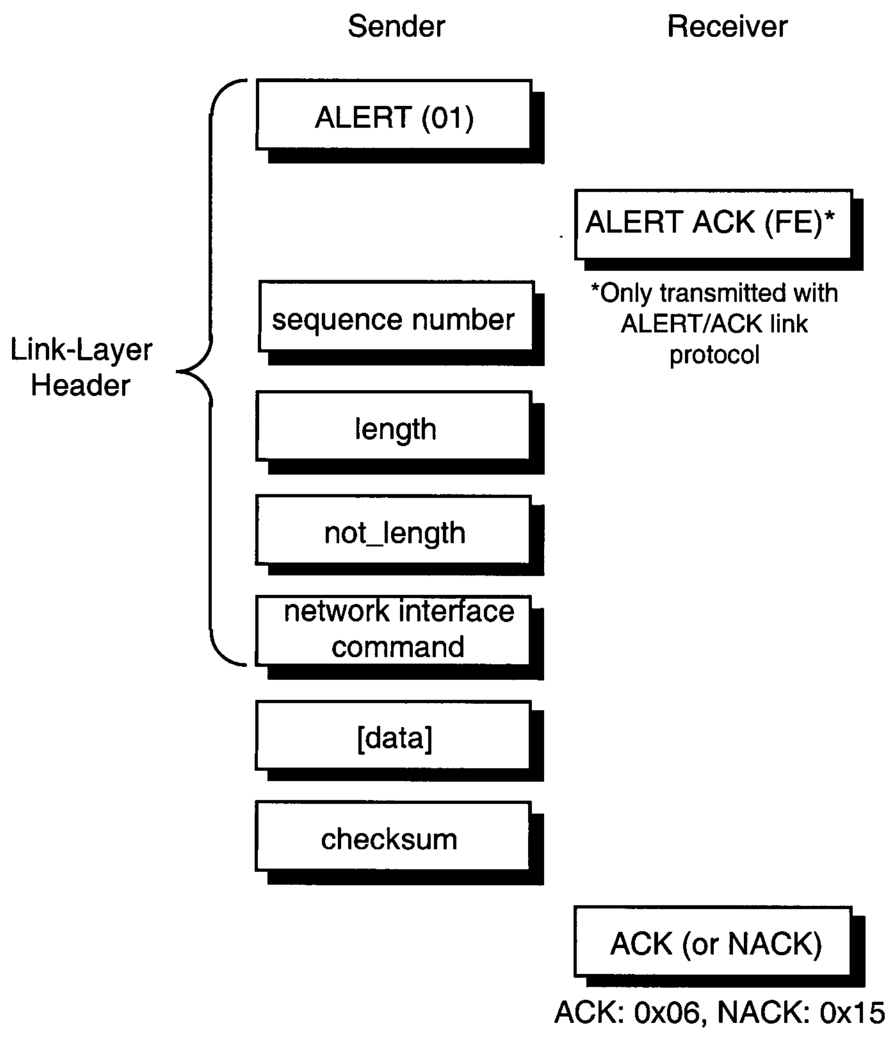


**Figure 6.4** SLTA Reliable Transport Protocol

## SLTA Timing Data

Certain aspects of the SLTA link and transport layer protocols implement fail-safe timeouts in order to control the time spent waiting for protocol states to change when errors occur. These timeouts are kept constant with either a 10MHz or 5MHz Neuron input clock.

### Downlink Byte-to-Byte Receive Timeout

The downlink byte-to-byte receive timeout is the maximum allowable period between the end of a single byte data frame sent downlink to the SLTA, to the end of the next single byte data frame sent downlink to the SLTA. This period is 100ms in local host mode and 1 second in remote host mode. When this timeout occurs, the SLTA discards the downlink buffer and returns to the NORMAL state. If the reliable transport protocol is enabled, the SLTA also sends a NACK byte after this timeout.

### Uplink Message Life

The uplink message life is the maximum allowable period between the SLTA sending an ALERT byte to the host and the host responding with an ALERT ACK byte. This period is 100ms in local host mode and 1 second in remote host mode. When this timeout occurs, the SLTA will resend the ALERT byte. This process is repeated until 3 seconds have elapsed, after which the uplink message is discarded. This timeout only applies to the ALERT/ACK link protocol and is not used for the buffered link protocol.

### ACK/NACK Receive Timeout

When using the reliable transport protocol, the SLTA will wait for the ACK or NACK byte to be sent downlink following the end of the uplink transmission of a message. This period is 1 second, after which the SLTA will re-send the uplink message.

### Uplink Timeout Message Retry Count

When using the reliable transport protocol the SLTA will re-send uplink messages whenever the ACK/NACK timeout period has elapsed. This retry process is limited to 5 retries, after which the uplink message is discarded. There is no retry limit applied to re-sends due to the reception of the NACK byte.

## Session Layer Protocol

The network interface link and transport protocols described above are used for all host-to-SLTA communications. Layered on top of these protocols is a downlink buffer request protocol and an uplink flow control protocol.

### Downlink Buffer Request Protocol

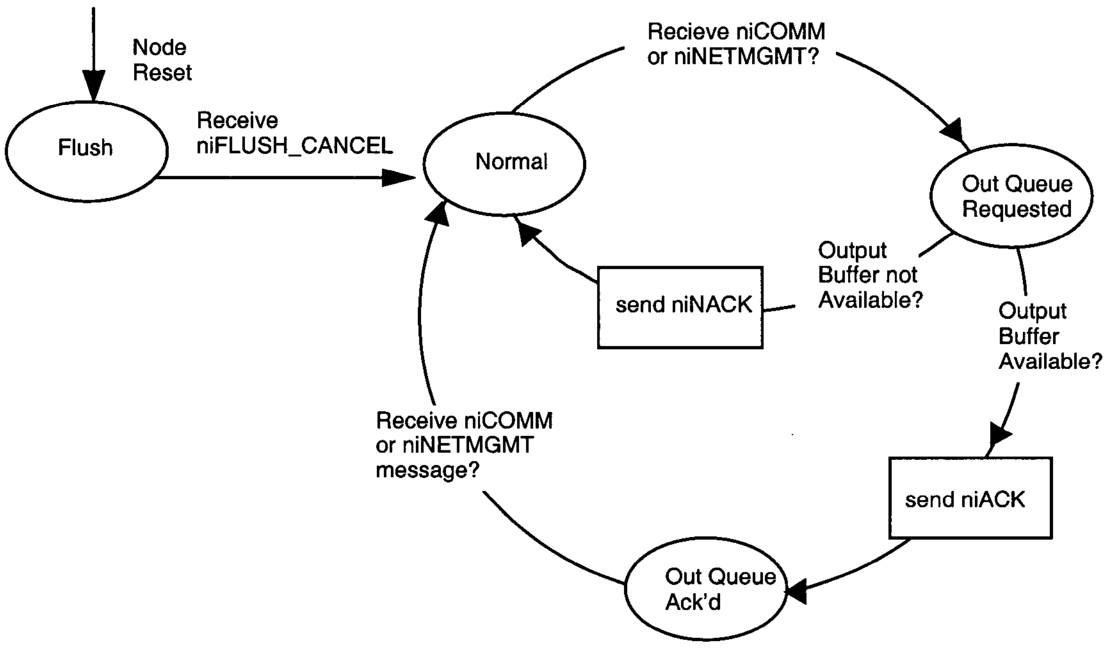
The network driver receives application buffers from the host application, translates them to interface buffers, and passes the interface buffers to the SLTA. There are two types of downlink commands from the host to the SLTA – commands that can be executed directly by the SLTA, and commands that need to be buffered in the SLTA.

Downlink commands that are executed directly by the SLTA are:

niRESET, niFLUSH_CANCEL, niONLINE, niOFFLINE, niFLUSH, niFLUSH_IGN, niPUPXOFF, niPUPXON, niSLEEP, and niSSTATUS.

See the *Host Application Programmer's Guide*, Appendix D, for a description of these commands.

Downlink commands that are buffered in the SLTA are niNETMGMT (for network management commands to be executed by the SLTA itself) and niCOMM (for messages to be sent out on the network, including network variables, explicit messages, and network management messages addressed to other nodes). For these two commands, a buffer request protocol is used to ensure that the SLTA has a free application buffer for the data. The network driver must first request an output buffer before sending the interface buffer. The network driver must hold the buffers in an output queue until the SLTA is ready to receive them. The network driver takes the SLTA through 3 states to request a buffer and send the interface buffer. Figure 6.5 summarizes the downlink state transitions.



*Note: niNETMGMT commands are allowed in the Flush state.*

**Figure 6.5** SLTA Downlink Flow Control States

Following is the sequence of events for transferring an niCOMM or niNETMGMT command downlink to the SLTA:

1 The SLTA is initially in the NORMAL state.

2 The network driver requests an output buffer by sending a link-layer header (see figures 6.2 and 6.3) with a niCOMM or niNETMGMT command and the appropriate queue value (niTQ, niTQ_P, niNTQ, niNTQ_P). The data portion of the interface buffer is not sent with the buffer request. This puts the SLTA in the OUTPUT QUEUE REQUESTED state.

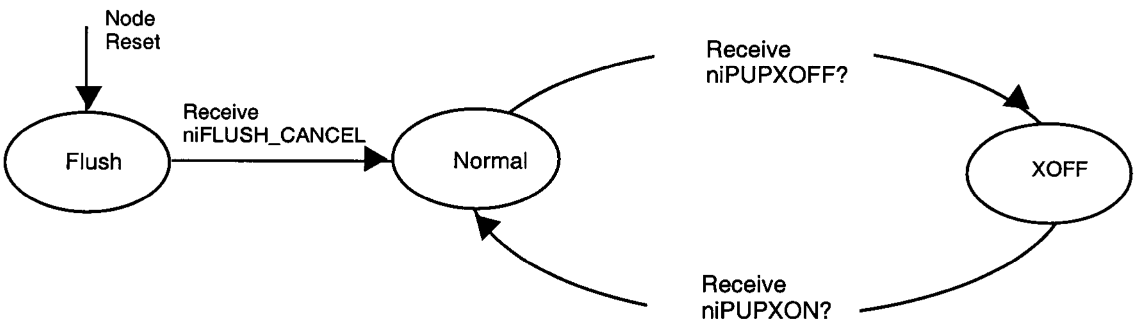
3. If an output buffer is not available, the SLTA responds with a niNACK (0xC1) command. The SLTA returns to the NORMAL state, and the driver starts again at step 2.
4. When an output buffer is available, the SLTA responds with a niACK (0xC0) command. The SLTA is now in the OUTPUT QUEUE ACKNOWLEDGED state. While in this state, the network driver can only transfer downlink LonTalk messages, uplink source quench commands (niPUPXOFF), uplink source resume commands (niPUPXON), or reset commands (niRESET) since the SLTA is waiting for a message in this state. All other network interface commands sent downlink will be ignored, and will return the SLTA to the NORMAL state.
5. Upon receiving the niACK acknowledgment, the network driver transfers the entire interface buffer to the SLTA. This buffer has the same command and queue value sent in step 2, and also contains the data and checksum. Upon completion of this transfer, the SLTA returns to the NORMAL state.

The network driver must preserve the continuity of the type of buffer request and the type of message sent downlink. For example, if the network driver sends the niCOMM+niTQ_P command requesting a priority output buffer, and follows this with a message transfer with the non-priority niCOMM+niTQ command, the SLTA will incorrectly store the message in a priority output buffer, the type originally requested.

## Uplink Flow Control Protocol

Uplink traffic may be incoming LonTalk messages, output buffer request acknowledgments, completion events, or local commands. The network driver translates the interface buffers to application buffer format and stores the buffers in a queue until the host application is ready to read them.

There is no buffer request protocol for uplink traffic. The network driver is normally assumed to have sufficient buffers. The network driver can suspend or resume uplink traffic when no network driver input buffers are available by sending the *Uplink Source Quench* (niPUPXOFF) command to the SLTA. This prevents the STLA from sending any LonTalk messages uplink. When the network driver senses that network driver input buffers are available, it sends the *Uplink Source Resume* (niPUPXON) command to resume uplink transfers. Figure 6.6 summarizes the uplink state transitions.



*Note: Responses to niNETMGMT and niSSTATUS commands are allowed in the Flush state.*

**Figure 6.6** SLTA Uplink Flow Control States

Note that for SLTA firmware versions 7 or higher, the host may chose to sidestep the downlink buffer request protocol. In this case, the complete message is sent downlink without any buffer request step. If the SLTA has a free output buffer, then the message will be transferred into the SLTA successfully. If not, there will be no indication and the message will be lost. The exception to this case is when using the transport layer protocol, in which case the SLTA will send the NACK to the host, which should force the host to re-send the message. Otherwise, in order to use this feature successfully, the host driver must manage the number of available output buffers within the SLTA. This feature is included in the DOS driver for the SLTA.

## Presentation Layer Protocol

The network driver exchanges LonTalk packets with the host application at the presentation layer. The LonTalk packet enclosed in a command of type `niCOMM` or `niNETMGMT` is described in detail in the *Host Application Programmer's Guide*. It is summarized here for convenience.

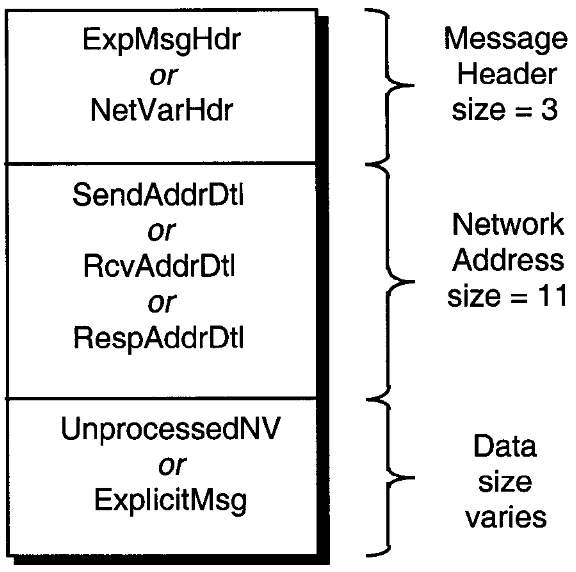


**Figure 6.7** Application Buffer Format

The SLTA firmware is configured with explicit addressing enabled, and therefore the 11-byte network address field is always present in an uplink or downlink buffer. The firmware is also configured with host selection enabled, so the data field of the buffer is either an unprocessed network variable or an explicit message. The processed network variable option is not available with the SLTA.

# 7

# Using the DOS Network Driver

This chapter describes the DOS network driver supplied with the Connectivity Starter Kit for use with the LTS-20 operating in MIP mode. The DOS network driver provides a device-independent interface between a DOS or Windows host application and the LTS-20. The driver is configurable to use one of four PC/AT serial ports, COM1 through COM4, at one of eight serial bit rates.

## Installing the SLTA Network Driver for DOS

**For the purposes of this chapter, the term "SLTA" refers to the LTS-20 module operating in MIP mode.**

*The DOS driver is supplied on the floppy diskette included with the Connectivity Starter Kit. The latest version of this driver may be obtained from the Echelon web site. See the Preface of this manual for ftp site access information.*

The SLTA network driver is installed by adding a `DEVICE` command to the DOS `CONFIG.SYS` file. Edit the `CONFIG.SYS` file to include the line:

```
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS [options]
```

Substitute your drive and directory name if other than `C:\LONWORKS\BIN`. Reboot the PC after adding this line to load the driver. For example, the following command would be used with a locally attached SLTA installed on COM2 as device LON1 running at 38,400 bps with autobaud enabled (this is the factory default):

```
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS /P2 /D1 /B38400 /A
```

See *Example Configurations* in Chapter 11 for examples.

*Warning! The `/A` option must be present in the `CONFIG.SYS` entry if Autobaud is in enable, or the adapter will not function correctly. The /A option also may be left in the `CONFIG.SYS` entry if Autobaud is disabled.*

The available options for the DOS network driver are described in the following sections.

### Buffer Options

| | |
|---|---|
| `/O`*nn* | Sets the number of output (downlink) buffers within the driver to <*nn*>. The buffer sizes within the driver are pre-set to accommodate 255 byte packets. The SLTA has application output buffers in the RAM of the Neuron Chip which, by default, are smaller than this, but may be increased to as large as 255 bytes. The default is eight buffers in the network driver. There must be at least 2 buffers and the maximum allowed number for <*nn*> is limited by the size of the buffer (258) times the total number of input and output buffers within the driver. The entire buffer space plus the driver code itself cannot exceed 64Kbytes. The size of the driver code itself is 9Kbytes. The number of output buffers required is determined by the characteristics of the host application. If the host application always waits for an outgoing message completion before sending another message, then only two buffers are required. If the host application is set up to overlap transactions, more buffers may be required. In this case greater parallelism may be achieved at the expense of host application code complexity. |
| `/I`*nn* | Sets the number of input (uplink) buffers within the driver to <*nn*>. The buffer sizes within the driver are pre-set to accommodate 255 byte packets. The SLTA has application input buffers in the RAM of the Neuron Chip which, by default, are smaller than this, but may be increased to as large as 255 bytes. The default is eight buffers in the |

network driver. There must be at least 2 buffers and the maximum allowed number for <*nn*> is limited by the size of the buffer (258) times the total number of input and output buffers within the driver. The entire buffer space plus the driver code itself cannot exceed 64Kbytes. The number of input buffers required is determined by the expected incoming traffic and the capability of the host application to process it. If the incoming traffic is bursty, more input buffers are required. If the application cannot process incoming traffic fast enough, the input buffer pool will fill up with unprocessed packets. In that case, the SLTA will not be able to pass any new data to the host, and the input application buffers in the SLTA will start to fill up. Once that occurs, messages will be lost, possibly causing incoming LonTalk transactions to be retried, and eventually causing the sender of the message to receive a failure indication.

## Serial Bit Rate Options

**/B***nnnnnn*

Sets the serial bit rate to <*nnnnnn*>. The available serial bit rates are listed below. The default is 38,400 bps.

Available serial bit rates are:

**1200, 2400, 9600, 14400, 19200, 38400, 57600, 115200.**

This rate represents the serial bit rate between the PC and the SLTA when using a direct serial connection, and between the PC and the modem when using a remote connection. *The 115,200 bps rate is only available on the TP/XF-1250 SLTA/2 and SLTAs based on the LTS-10 module (SLTAs with a 10 MHz input clock).* For remote connections, the PC to modem serial bit rate, telephone line speed (i.e. modem to modem serial bit rate), and the modem to SLTA serial bit rate may be different. The PC to modem serial bit rate is controlled by the network driver on the PC using the /B option; the telephone line speed is selected by the modems based on modem configuration; and the modem to SLTA serial bit rate is controlled by the hardware configuration of the SLTA as described in Chapter 2 (autobaud cannot be used in this configuration).

For local connections with the SLTA Autobaud option disabled, the serial bit rate specified by this driver option must match the rate specified by the Baud Rate inputs to the SLTA.

**/A**

*Warning*: *If you are using the default hardware configuration (autobaud enabled), the autobaud option (/A) **must** be enabled or the SLTA will not function properly.*

Enables the autobaud feature. This provides the autobaud sequence whenever the driver is opened. The default setting for the driver is autobaud disabled. If the Autobaud input on the SLTA hardware is enabled, then this option must be specified This option may not be used with the modem support (/M) option.

## DOS Device Options

| | |
|---|---|
| `/Pn` | Sets the serial port to <*n*> where <*n*> is 1-4 for COM1 - COM4. The default is COM1. |
| `/Dn` | Defines the device unit number as <*n*>, where <*n*> is between 1 and 9, so that the DOS device name is "LON1" through "LON9". The default is 1 for "LON1". This option can be used to support multiple network interfaces on a single PC. For example, this device name is passed as a parameter to `lxt_open()` when using the LonManager API. When invoking the sample host application `HA`, the device may be specified with the `-D` option, for example: |

```
HA -DLON2
```

| | |
|---|---|
| `/Un` | Sets the serial port interrupt request number (IRQ) to a non-standard value <*n*>, where <*n*> is between 1 and 7. If the serial port in use is COM3 or COM4, you may want to use a unique, unused IRQ for that port. Many serial ports and internal modems allow the selection of a non-standard IRQ such as IRQ2 or IRQ5. |
| /C | Enables communications interrupt chaining. Some PCs may incorporate up to four serial ports. If supported by the serial hardware, COM1 and COM3 may share the same interrupt (as do COM2 and COM4). This option may enable the driver to support the shared interrupt by "chaining" to the interrupt vector that was in place when the driver was loaded. This option is not necessary if your system does not use COM3 or COM4, or if COM3 or COM4 use a different interrupt request number. When installing two SLTA network drivers on a system on COM1 and COM3 (or COM2 and COM4 with the same interrupt request number), the last installation of the driver should use this option. Here is an example of a `CONFIG.SYS` file entry for such a system. |

```
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS /B38400 /A /P1
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS /B38400 /A /P3 /C
```

Standard hardware configurations are used for the COM1 - COM4 settings:

| ***Device*** | ***I/O Address*** | ***Interrupt (*)*** |
|---|---|---|
| COM1 | 0x3F8 | 4 |
| COM2 | 0x2F8 | 3 |
| COM3 | 0x3E8 | 4 |
| COM4 | 0x2E8 | 3 |

(*) May be changed with the /U option

## Timing Options

| | |
|---|---|
| `/R`*nn* | Defines the flush/retry count in 55ms intervals. This value is used in error states for re-transmitting requests and for terminating receive flushes when input errors occur. See *Troubleshooting* in Chapter 17. Normally, this option should not be specified. |
| `/W`*nnn* | Includes a delay of <*nnn*> microseconds when transmitting downlink. This parameter can be used to pace the rate at which bytes are transmitted downlink to the SLTA, and may be required for high-performance network management tools such as LonMaker. The delay is executed following the transfer of each data byte to the host's UART, and only after the first 15 bytes have been sent. Since the SLTA employs a 16-byte deep FIFO buffer in its UART, the first group of bytes sent do not need to be paced. The pacing delay will have no effect unless it is greater than the actual period it takes to transmit a single byte at the given serial bit rate. The time taken to transmit a byte is 173 $\mu$s at 57,600 bps, and 86 $\mu$s at 115,200 bps. For a 10 MHz SLTA (TP/XF-1250 SLTA/2 or LTS-10), this option should be used at 115,200 bps if messages greater than 16 bytes are to be transmitted. A value of `/W120` is suggested. For a 5 MHz SLTA (TP/XF-78, TP/FT-10, or TP-RS485 SLTA/2), this option should be used at 57,600 bps, again if messages greater than 16 bytes are to be transmitted. A value of `/W240` is suggested. This option is not required at the default serial bit rate of 38,400bps, or at slower serial bit rates. |
| `/Z` | By default, the SLTA firmware disables network communications after a reset by entering a FLUSH state. This state causes the SLTA to ignore all incoming messages and prevents all outgoing messages, even service pin messages. The DOS network driver for the SLTA automatically enables network communications when the SLTA is opened and when it receives an uplink message from the SLTA indicating that it has been reset. However, the host application itself must explicitly enable network communications if the /Z option is specified and the CFG1 input is set to *Network Disabled*. See Chapter 8 for more information. If CFG1 is set to *Network Enabled*, the SLTA will go directly to the `NORMAL` state, thus allowing communications. |

The following table summarizes these options:

| ***Network Disable Input*** | ***DOS Driver /Z Option*** | ***When SLTA Enables Network Communications*** |
|---|---|---|
| Disabled | Specified | Host application command |
| Disabled | Not specified | Opening network driver |
| Enabled | Don't care | Immediately after reset |

Host applications which need to configure the SLTA prior to enabling network communications should use this option. This option should not be used with the LonManager API, LonManager LonMaker, or the 16-bit LonManager DDE Server. More information about the `niFLUSH_CANCEL` message is provided in the *LONWORKS Host Application Programmer's Guide.*

## Network Interface Protocol Options

**/F** Enables the full interrupt mode of the driver. If this option is not specified, the driver will disable interrupts for the duration of each link-layer transfer. This ensures that no data will be lost due to other system interrupts, and is acceptable at high serial bit rates. The driver will use interrupts for the first byte of each uplink interface buffer. When the uplink interrupt is received, the driver reads the rest of the message without interrupts via polled I/O. Interrupts are disabled during the uplink transfer. This assures that no other system interrupts will occur resulting in lost uplink data frames. Downlink transmissions are sent directly via polled I/O of the serial port from the write function call. The host write functions will not return until the message has been sent downlink. When using the ALERT/ACK link protocol, interrupt latency is not a problem, since the SLTA-to-host protocol includes an acknowledgment of the start of the message. The driver employs timeouts in order to prevent lockout of the write function, and timeouts for clearing various states of the transmitter/receiver when line errors occur.

When operating at lower serial bit rates, it becomes less desirable to disable interrupts for long periods. The trade-off with using the full interrupt mode is that other system interrupts may cause loss of data in the serial port's UART. If the `/F` option is specified, the driver uses interrupts for every uplink and downlink byte transferred. Downlink messages are buffered from the device write function and are sent downlink under interrupt control. Uplink messages are received under interrupt control and are buffered also. This option should be used for serial bit rates of 9,600 bps or slower. Do not use this option with the HP 95LX.

**/M** Enables modem support and the reliable transport protocol. This option must be specified if the host is to communicate with the SLTA via a modem connection. The SLTA must be configured with CFG2 input in the *Remote Host* setting. In this mode the driver relies on the state of the DCD signal from the modem to determine if it is connected to an SLTA through a modem connection or not. When connected, the selected SLTA <-> Host network interface link protocol is in effect. When disconnected the only allowable link layer traffic is of the 'modem direct' type, where ASCII strings are being exchanged between the host and the modem, for example, AT commands to dial out. Any other network interface traffic is not allowed when disconnected from the SLTA. Calls to the read function will result in no network interface data messages (`LDV_NO_MSG_AVAIL`), and any call to a write function that needs to communicate with the SLTA via the modems will result in a No Output Buffers Available error (LDV_NO_BUFF_AVAIL). Once the connection is made, normal network interface traffic may resume.

This option also enables the reliable transport protocol. This protocol includes the addition of a message sequence number and the end of message ACK/NACK code. See Chapter 8 for a description of this protocol.

**/N** Disables the ALERT/ACK network interface link protocol, and enables the buffered network interface link protocol. Network interface messages are sent without a wait for the ALERT ACK response. Both sides of the interface (the SLTA and the driver) must agree on this setting. This option should not be used with the */M* option.

**/Q** Allows modem responses to be sent uplink to the host. When the telephone link is disconnected, these messages are ASCII strings with the network interface command type niDRIVER (0xF0). If /Q is specified, the host application must be able to handle messages, such as NO CARRIER, that might come from the modem itself if problems occur in the connection.

**/X** Disables the the buffer request protocol. This option only works with SLTA application versions 7 or later. When this option is enabled, the driver requests the buffer count from the SLTA using the niSBUFC (0xE7) command whenever the interface is opened, or when the interface is reset, and reports an niRESET to the host. The driver keeps track of the number of available output buffers in the SLTA by examining both uplink and downlink messages. This option prevents the use of one message type: A local network management command not using a request/response service. Normally this type of message is not used. One exception could be the Set Node Mode: Reset command, which would result in the node resetting and the buffer management recovering on its own anyway. Otherwise, if this type of message is used, no uplink response would occur and the driver could not track the fact that a new output buffer has been made available.

The following table summarizes the relationship between the CFG jumpers of the SLTA and the driver options that control the network interface protocol.

| *Input* | *Input State* | *Driver Option* |
|---|---|---|
| CFG 2 | Local Host; No Transport Protocol | /M not specified |
| CFG 2 | Remote Host; Reliable Transport Protocol | /M specified |
| CFG 3 | ALERT/ACK Link Protocol | /N not specified |
| CFG 3 | Buffered Link Protocol | /N specified |

## Calling the Network Driver from a Host Application

The DOS network driver supports the `open`, `close`, `read`, `write`, and `ioctl` DOS calls. All LONWORKS standard network drivers for DOS support these calls. See Chapter 4 of the *Host Application Programmer's Guide* for more details.

When the DOS SLTA network driver is loaded during execution of the `CONFIG.SYS` file, it does not attempt to communicate with the SLTA.

When the network driver is opened with the DOS open call, it establishes communications with the SLTA. The network driver returns an error if this fails, for example, if the SLTA is disconnected, powered down, or configured incorrectly. If the open call succeeds, the driver enables network communications by clearing the SLTA `FLUSH` state, if configured to do so.

The DOS read call is defined to return the number of bytes read from the device. Some LONWORKS standard network drivers return 0 if there are no uplink messages available. DOS reports this as an end-of-file condition and prevents further reads from succeeding. However, the SLTA driver returns a length of 2, and sets the first byte of the caller's buffer (the cmd/queue byte) to 0 to indicate that there is no uplink message available.

Normally, the DOS `read` and `write` calls are not used with LONWORKS standard network drivers. This is because any error from the network interface will display the familiar `Abort`, `Retry`, `Fail`? error message from DOS, unless the caller has installed a critical error device handler. Therefore, DOS applications using a network device typically call direct entry points into the driver. This also allows more detailed error status to be returned to the application. The addresses of these entry points are obtained by calling the `ioctl()` function of the driver.

This function call is used as follows:

```
int ioctl(int handle, int func, void far *argdx, int argcx);
```

- `handle` is an integer returned by an earlier successful call to `open()`, specifying the LONWORKS network driver `LONn` to be opened.

- `func` is the value 2, meaning that the application is reading information from the driver. For LONWORKS standard DOS network drivers, the information returned is the network interface direct call structure.
- `argdx` is a pointer to a caller-declared structure that will contain the direct entry points into the driver. See the structure `direct_calls` in the file `NI_MSG.C` in the supplied example host application for usage.
- `argcx` is the size of the structure.

Function code 2 is supported by all LONWORKS network drivers for DOS to return three direct entry points into the driver code. The network driver for the SLTA supports an additional option to function code 2, as well as function code 3, which is used to manage the modem control state of the driver. These options are not used when the SLTA is connected directly to a host. They are provided primarily for use while establishing communications with a remote host. For example, the host connect utility (HCU) described in Chapter 12 of this manual uses these functions. Host applications that only communicate to the SLTA via an already-established telephone connection do not need to concern themselves with these functions. If you wish to establish or take down telephone connections during the execution of your host application, use the source code of HCU as a guide.

When function code 2 is used, argdx points to the `direct_calls` structure defined for all LONWORKS standard network drivers for DOS. If `argcx` is 13, the size of the standard direct calls structure, then three direct entry point addresses are returned as usual. If `argcx` is 4 (the size of the structure `ioctl_get_dcd_s`), then the state of the modem's DCD line is returned as a TRUE or FALSE value. Note that the status field is 16 bits in this structure, but 8 bits in the direct calls structure.

```
struct ioctl_get_dcd_s {
      unsigned ioctl_stat;   // 16 bit status
      unsigned dcd_state;    // Data Carrier Detect (TRUE or FALSE)
}
```

Function code 3 is used when the application wishes to write information to the driver. For the SLTA driver, argdx points to the following structure, and `argcx` is its size:

```
struct ioctl_o_info_s {
    unsigned  ioctl_stat;      // 16 bit status
    unsigned  sub_command;     // use enum sub_command
    unsigned  mode;
    unsigned  mode_aux;
}
enum sub_command {
    SUBC_set_opt     = 1,      // set driver options
    SUBC_set_DTR     = 2,      // set DTR line
    SUBC_set_baud    = 3,      // set serial bit rate
};
```

There are three sub-commands, used to set the various modes of the driver, the state of the DTR (Data Terminal Ready) line to the modem, and to set the serial bit rate of the serial interface.

When sub-command 1 is used, the `mode` field in the structure is a bit mask defining which of the driver modes is to be changed, and the `mode_aux` field specifies bits defining the new states of those modes. It is possible to set more than one of the modes by OR'ing the following bit-masks together:

| | |
|---|---|
| 0x0001 | Enables modem support. |
| 0x0002 | Allows modem responses to host - same as the /Q option. |
| 0x0004 | Forces direct modem mode. In this mode, the network driver is communicating directly with the modem. |
| 0x0010 | Enables the buffered link protocol and disables the ALERT/ACK link protocol - same as the /N option. |
| 0x0020 | Enables the reliable transport protocol. |

The `/M` option corresponds to 0x0021.

Sub-command 2 is used to set the state of the DTR line. In this case, the DTR signal is enabled (on) if the mode field is true.

Sub-command 3 is used to set the serial bit rate of the serial interface. The mode field determines the new bit rate as follows: 0:14,400; 1:1,200; 2:2,400; 3:9,600; 4:19,200; 5:38,400; 6:57,600; 7:115,200.

## Using the SLTA Driver under Microsoft Windows

In order to use the direct entry points to a LONWORKS standard network driver for DOS under Microsoft Windows, an interface based on the DOS Protected Mode Interface (DPMI) must be provided. This type of interface, in the form of Windows DLL software, is supplied with the Connectivity Starter Kit, as well as with the 16-bit LonManager DDE Server. See Appendix B for information on using the Windows DLL directly.

# 8
# Using the UNIX Network Driver

This chapter describes the UNIX network driver supplied with the Connectivity Starter Kit for use with the LTS-20 operating in MIP mode. The UNIX network driver provides a device-independent interface between a UNIX host application and the LTS-20.

## Installing the SLTA Network Driver for UNIX

**For the purposes of this chapter, the term "SLTA" refers to the LTS-20 module operating in MIP mode.**

The SLTA network driver for UNIX is not a UNIX device driver. It is instead a source library that provides an interface to an existing UNIX serial port driver. The UNIX network driver handles the SLTA network interface protocol and runs on top of the UNIX serial port driver, which in turn handles the interrupt processing and buffering of uplink and downlink serial data. This greatly simplifies the SLTA network driver and makes it more portable to different versions of UNIX, as well as other operating systems.

The `LDVSLTA.C` and `LDVSLTA.H` files contain the C source code for the UNIX network driver. This code has been tested with Sun UNIX (SunOS Release 4.1) and SCO UNIX (Release 4.2.0). Changes may be required for other versions of UNIX. The source code must be compiled and linked with your application. The serial device driver names may have to be changed for different versions of UNIX. For example, Sun UNIX uses `/dev/ttya` and `/dev/ttyb`, and SCO UNIX uses `/dev/tty1a` and `/dev/tty2a`.

The UNIX network driver is implemented with the following defaults (these defaults may be changed by modifying the source code):

- **Link Layer Protocol**. The buffered link protocol is used if the `alert_ack_prtcl` variable is set to `FALSE` in the source code (this is the default). The ALERT/ACK link protocol is used if the `alert_ack_prtcl` variable is set to `TRUE`. See *Buffered Link Protocol* in Chapter 8 for warnings on use of the buffered link protocol with UNIX. The buffered link protocol should not be used with modems.
- **Transport Layer Protocol**. The reliable transport protocol is not supported.
- **Physical Layer Protocol**. The call to `ioctl()` in `ldv_open()` initializes the serial link to 19,200bps. The code for the `ioctl()` call may have to be modified for different versions of UNIX and serial port configurations.
- **Modem Support**. The UNIX network driver does not include modem support for remote hosts.
- **Buffers**. The number of input and output buffers is controlled by the `ldv_buffers` variable and defaults to 10 each.

## Calling the Network Driver from a Host Application

The functions provided by the UNIX network driver are the same as those listed under *Standard Network Driver Services* in chapter 4 of the *LONWORKS Host Application Programmer's Guide*, with the addition of a new service, `ldv_post_events()`.

## ldv_open()

```
typedef int LNI;
LNI handle = ldv_open(char *serial_device_name);
```

Initializes the SLTA and returns a handle for accessing it. Opens the serial device and enables network communications (i.e. the `FLUSH` state is canceled). The `serial_device_name` parameter must be the name of an installed serial device such as `/dev/ttya`. If the `ldv_open()` call succeeds, the SLTA will send an niRESET command uplink to the host. Only one device may be open at a time.

## ldv_read()

```
LDVCode error = ldv_read(LNI handle, void *msg_p, unsigned length);
```

Reads an application buffer from the SLTA. The `msg_p` argument is a pointer to an application buffer as defined in chapter 3 of the *LONWORKS Host Application Programmer's Guide*. If a buffer is not available, the `LDV_NO_MSG_AVAIL` error code is returned.

## ldv_write()

```
LDVCode error = ldv_write(LNI handle, void *msg_p, unsigned length);
```

Writes an application buffer to the SLTA. The `msg_p` argument is a pointer to an application buffer as defined in chapter 3 of the *LONWORKS Host Application Programmer's Guide*. If a buffer is not available within the SLTA to accept the application buffer, the `LDV_NO_BUFF_AVAIL` error code is returned.

## ldv_post_events()

```
void ldv_post_events(void);
```

This is the network interface background service function. Its purpose is to process any uplink or downlink SLTA traffic. This function should be called periodically from the host application in order to assure that the SLTA traffic is being processed. This processing includes off-loading the UNIX serial port driver's input buffers, and moving downlink any messages that are buffered in the network driver's output buffers.

This function is also called from within the interface library's `ldv_read()` and `ldv_write()` functions. The host application does not have to explicitly call the `ldv_post_events()` function if it is periodically polling the interface using the `ldv_read()` function.

## ldv_close()

```
LDVCode error = ldv_close(LNI handle);
```

Closes the UNIX serial device, and frees the allocated buffer memory back to the system.

# 9

# The LTS-20 NSI Mode Software

This chapter describes the Windows 95, 98, or NT software used with the LTS-20 NSI mode. This software is available with the LonMaker for Windows Integration Tool (Model 37000), in the Connectivity Starter Kit (Model 58030-01), as part of the LNS Developer's Kit for Windows (Model 34303), and on the Echelon web site (www.echelon.com).

**Skip this Chapter if you are using the LTS-20 MIP mode.**

## LTS-20 NSI Mode Software Overview

The LTS-20 is not shipped with software drivers. The Windows NT driver and SLTALink Manager software are available with the LonMaker for Windows Integration Tool (Model 37000), in the Connectivity Starter Kit (Model 58030-01), as part of the LNS Developer's Kit for Windows (Model 34303) and on the Echelon web site (www.echelon.com).

The LTS-20 NSI mode set-up installs three pieces of software:

- the LTS-20 NSI mode Windows 95 or NT Driver,
- a "stub" driver to run legacy DOS and Windows 3.1x applications, and
- the SLTALink Manager software.

The LTS-20 includes firmware that moves the upper layers of the LonTalk Protocol from the Neuron Chip within an LTS-20 and onto a host processor. This firmware allows theLTS-20 to be used by a host application to send and receive LonTalk messages. The firmware in the LTS-20 is loaded in ROM and cannot be reprogrammed.

Using the LTS-20 NSI mode, the host application may be one of two types. The first type of host application is an LNS-based application, developed with the LNS Developer's Kit for Windows. The second type of application is a legacy DOS or Windows 3.1x application. Under Windows NT, these applications make use of the "stub" driver declared in the `config.sys.os` files, which in turn accesses the Windows NT driver. Under Windows 95, Windows 3.1x applications should use the DOS driver in conjunction with WLDV.DLL. Echelon only supports 32-bit Windows applications based on LNS software accessing the Windows 95 or NT drivers.

### *Windows 95 and Windows NT Software Installation Procedure*

Prior to installation, ensure that the computer is running the Windows 95 or NT Operating System (Windows NT version 3.51 or higher for a direct connect interface; Windows NT version 4.0 or higher for use with modems). The LTS-20 software cannot be installed from DOS, or a DOS shell, nor can it be installed on Windows 3.1 or Windows 3.11.

**The LTS-20 software is shared in common with the SLTA-10 Serial LonTalk Adapter. Any references to the SLTA-10 NSI, including file names, apply to the LTS-20.**

1. Before installing the software, make sure that you have logged in as Administrator (for Windows NT only).

2. Close all open programs.

3. Insert the installation diskette into the PC.

4. Click the Start button on the Windows task bar and select the run command. (If using with Windows NT 3.51: Within Program Manager, choose the Run command from the File menu.)

5. When prompted for a program name, enter the following:

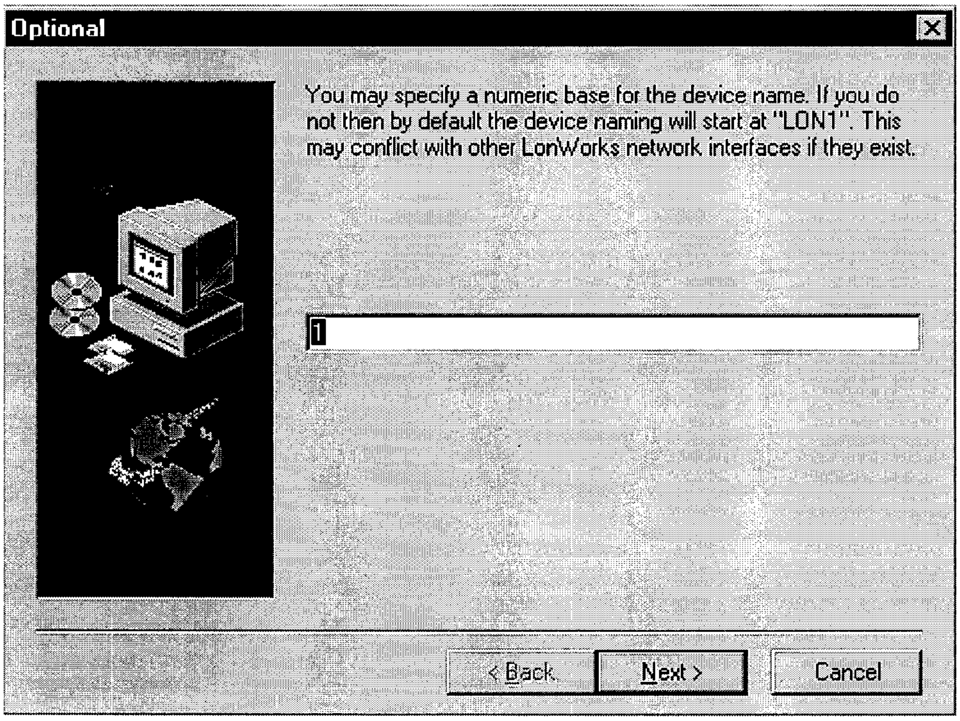
```
A:\SETUP.EXE
```

If necessary, replace `A:` with the drive letter which corresponds to the drive containing the SLTA-10 NSI mode installation diskette.

6. When prompted click the button marked "Next >".

7. When prompted for a destination directory, enter the desired installation directory. By default this directory is `c:\lonworks`, unless previous LONWORKS products have been installed and have registered a different path in the Windows Registry. The path may be modified using the "Browse" button.

8. The next screen presented is shown in figure 9.1. This will determine the LONWORKS naming convention used for the LTS-20.



You may specify a numeric base for the device name. If you do not then by default the device naming will start at "LON1". This may conflict with other LonWorks network interfaces if they exist.

1

**Figure 9.1** LONWORKS Device Naming Convention

9. Clicking the "Next" button concludes installation. At the prompt to restart the computer, remove the SLTA-10 NSI mode installation diskette and restart the computer. **Note that the Windows operating system will not recognize a node using the LTS-20 until the computer is restarted.**

## Windows 95, 98, and NT Software Installation Results

The Windows 95, 98, and NT installation software loads a selection of new files and updated Echelon files to different locations on the PC's hard drive. The function and location of these files can be found in readme.txt.

# 10

# The LTS-20 MIP Mode Software

This chapter describes the LTS-20 MIP mode software shipped with the Connectivity Starter Kit (Model 58030-01) and on the Echelon web site at www.echelon.com. This software is an updated version of the SLTA/2 and LTS-10 adapter software.

Echelon does not provide a 32-bit Windows driver for the LTS-20 MIP mode.

**Skip this Chapter if you are using the LTS-20 NSI mode.**

## LTS-20 MIP Mode Software Overview

The LTS-20 is not shipped with software drivers. The LTS-20 MIP mode software and drivers are supplied in the Connectivity Starter Kit and must be ordered separately. The software includes ANSI C source code for HA, a sample host application for MS-DOS that can be used as a basis for a user-developed host application on other host platforms. This application provides examples of sending and receiving network variable messages, as well as allowing a node based on an LTS-20 to be installed and bound by a network management tool such as the LonManager LonMaker Installation Tool or the LonBuilder network manager.

A network driver for DOS permits the LTS-20 (with its jumper cut to enable the MIP mode) to be used with DOS applications. Source code for a DOS network driver is provided as a basis for a user-developed network driver for other hosts or operating systems. On a separate diskette, DLL software is provided to make it easier to use the network driver under the Microsoft® Windows 3.1x operating system.

An executable program and source code is also provided for a Host Connection Utility (HCU), which may be used to initiate and terminate the host to LTS-20 connection when the LTS-20 is used with a remote host. An example written in Neuron C is also provided as a basis for user-developed nodes on a LONWORKS network that need to initiate outgoing calls to a remote host.

The LTS-20 includes firmware that moves the upper layers of the LonTalk Protocol off the Neuron Chip within an LTS-20 onto a host processor. This firmware allows the LTS-20 to be used by a host application to send and receive LonTalk messages. The host application may be a custom application as described in the *LONWORKS Host Application Programmer's Guide*. The host application may also be a network management or monitoring application based on the LonManager API, LonManager LonMaker installation tool, or LonManager DDE Server. The firmware in an LTS-20 is fixed in ROM and cannot be reprogrammed.

**The LTS-20 software is shared in common with the SLTA-10 Serial LonTalk Adapter. Any references to the SLTA-10 MIP, including file names, apply to the LTS-20.**

## Installing the LTS-20 MIP Mode Adapter Software

The LTS-20 software is supplied in the Connectivity Starter Kit as a diskette. The LTS-20 DOS driver (referred to as the SLTA-10 DOS driver) operates under DOS, Windows 3.1x, and Windows 95 operating systems. To install the LTS-20 software, follow these steps:

1. Place the diskette in one of the disk drives of your PC. This will typically be the `A:` or `B:` drive. Under the Windows 95 operating system, open a DOS console.

2. Start the automatic installation procedure by entering:

```
A:INSTALL [ENTER]
```

Substitute your disk drive name for the `A:` if you are using a different drive.

3. You will be asked to enter the name of your LONWORKS installation directory. `C:\LONWORKS` is the default.

The software will be installed in the SLTA sub-directory of your LONWORKS directory, with the exception of the DOS network driver `LDVSLTA.SYS`. This file will be installed in the BIN sub-directory of your LONWORKS directory. To install the DOS network driver into your `CONFIG.SYS` file, follow the instructions in Chapter 8.

The `SLTA` directory will contain the following files:

- **Read-Me File**. The `README.TXT` file includes a list of all the files on the distribution disk, and also includes any updates to the documentation that occurred since the LTS-20/SLTA-10 Adapter documentation was printed.
- **DOS Network Driver Sources**. The SLTA-10 Adapter DOS network driver source code is contained in the `LDVSLTA` directory. These files can be used as the basis for creating drivers for hosts other than PCs running DOS (see also the UNIX network driver sources). See Chapter 8 for a description of the SLTA-10 Adapter DOS network driver and Chapter 9 for a description of how to write an SLTA-10 Adapter network driver for other hosts. See Chapter 4 of the *LONWORKS Host Application Programmer's Guide* for a description of the services that must be supplied by a LONWORKS network driver.

  The source files to build the DOS driver are:

| | |
|---|---|
| `LDVSLTA.CFG` | Configuration file for Borland C. |
| `MAKEFILE` | Make file script for Borland C. |
| `MDV_TIME.C` | Code to manage the PC timer. |
| `MDV_TIME.H` | External interface definitions for the timer handler. |
| `MSD_DEFS.H` | Data structure and literal definitions. |
| `MSD_DIFC.C` | DOS driver interface functions. |
| `MSD_DRVR.H` | DOS driver interface and literal definitions. |
| `MSD_EXEC.C` | Main open, close, read, and write processing. |
| `MSD_FRST.C` | Module to be linked first in the network driver. |
| `MSD_IRQC.ASM` | Serial I/O interrupt procedure. |
| `MSD_LAST.C` | Module to be linked last in the network driver. |
| `MSD_RAW.C` | Direct serial I/O (modem) processing. |
| `MSD_SEGD.ASM` | Defines data segment register for driver. |
| `MSD_SIO.C` | PC/AT UART interface processing. |
| `MSD_TXRX.C` | Single byte link layer processing. |
| `MSD_UART.H` | Defines PC/AT UART registers. |

- **External Interface Files**. External interface files included for use by network management tools are contained in the `SLTA` directory. External interface files are included for the transceivers available for the LTS-20. See *Binding to a Host Node* in Chapter 3 of the *LONWORKS Host Application Programmer's Guide* for a description of how to use these files to bind to an LTS-20/SLTA-10 Adapter node. Appendix B of the *LONWORKS Host Application Programmer's Guide* provides a detailed description of how to modify these files to incorporate network variables and message tags. These interface files are provided in version 3 formats. External interface files in version 3 format are contained in the `SLTA2\XIF_V3` directory.

The SLTA directories contain at least the following files:

| | |
|---|---|
| `NSLTA125.XIF` | For SLTA-10 Adapter with a TP/XF-1250 transceiver. |
| `NSLTA78K.XIF` | For SLTA-10 Adapter with a TP/XF-78 transceiver. |
| `NSLTAFT1.XIF` | For SLTA-10 Adapter with a TP/FT-10 transceiver. |

- **Sample Host Application**. A sample host application is contained in the `HA` directory. See Appendix A of the *LONWORKS Host Application Programmer's Guide* for a description of the example. The following files are included:

| | |
|---|---|
| `README.TXT` | A description of the sample host application. |
| `HA.EXE` | An executable version of the sample host application for DOS. The SLTA-10 Adapter DOS network driver must be installed to run this application. |
| `HA.C` | The main program for the example. |
| `NI_MSG.C` | A general purpose network interface library that can be used with any host application. |
| `APPLCMDS.C` | Functions to handle application layer network variable commands. |
| `NI_CALLB.C` | The host-bound network management dispatcher. |
| `APPLMSG.H` | Application message handler function prototypes. |
| `HA_COMN.H` | The `HA` common declarations. |
| `NI_CALLB.H` | The definitions for the network management dispatcher. |
| `APPLMSG.C` | Functions to handle application network variable and explicit messages. |
| `HAUIF.C` | Command-line user interface for the example. |
| `IOCTL.C` | I/O control function for Microsoft C. |
| `LDVINTFC.C` | Device interface driver. |
| `LDVINTFC.H` | Include file for device driver interface. |
| `NI_MSG.H` | Definitions for network interface message structures. |
| `NI_MGMT.H` | Definitions for network management message structures used by the example. |
| `HAUIF.H` | Definitions for the host application example user interface. |
| `MAKEFILE` | A make file script for Borland C. |
| `MSOFT.MAK` | A make file script for Microsoft C. |

|  |  |
|---|---|
| HA_V3.XIF | An external interface file which may be used to bind the example with LonBuilder. |
| HA_TEST.NC | A Neuron C program which may be loaded into a Neuron emulator and bound to the sample host application for testing. |
| DISPLAY.H | A Neuron C include file to drive the Gizmo 2 I/O module for the test example. |

- **Host Connect Utility**. A sample host connection utility is contained in the HCU directory, with source code. See Chapter 12 for details. The files supplied are:

|  |  |
|---|---|
| HCU.EXE | Executable file for the Host Connection Utility. |
| HCU_MAIN.C | The main C source program. |
| HCU.CFG | Configuration file for Borland C. |
| MAKEFILE | Make file script for Borland C. |
| MSD_DRVR.H | Driver definition include file. |

- **Neuron C Connection Example**. A sample Neuron C program is contained in the NC_APPS directory. This program shows how a node on a network connected to the LTS-20 can dial out and connect to a remote host computer. The files supplied are:

|  |  |
|---|---|
| DIALOUT.NC | Neuron C source program to dial out with the LTS-20. |
| GIZSETUP.NC | An example Neuron C program for configuring the LTS-20. Configures the EEPROM directories of an LTS-20 using the Gizmo 2 I/O module as the user interface. |
| SLTA_ANM.H | Definitions of SLTA-specific network management messages. |

## *Installing the Windows 3.1x DLL Software*

A second diskette, labeled "LONWORKS Network Driver Interface for Windows 3.1x", contains the 16-bit Windows Dynamic Link Library (DLL) files. These files may be used when developing a host application to run under Microsoft Windows 3.1x. The file WLDV.DLL should be copied to your Windows directory (typically C:\WINDOWS). The files LDV.H and LON.H should be copied to a directory in the include file search path of your C compiler. The file WLDV.LIB should be copied to a directory in the library search path of your application linker. See Appendix A for information on using the Windows DLL.

## *Other Drivers*

A UNIX network driver and source code for the LTS-20 MIP mode is available on the Echelon web site (http://www.echelon.com).

Chapter 9 discusses creating an LTS-20 mode driver for any host.

# 11

# Using the Windows 95 or NT Driver and SLTALink Manager with LTS-20 NSI Mode

This chapter describes the SLTALink Manager software, which establishes and configures local and remote links from the host PC to the LTS-20 in NSI mode. A local link requires a direct cable connection from the host PC to the LTS-20-based node. A remote link requires a pair of modems: one attached to the LTS-20-based nodes and the other attached to the host PC. The SLTALink Manager software controls a remote LTS-20 via a pair of modems through Windows' Telephony Application Programming Interface (TAPI) services under Windows 95 and NT 4.0 or later.

The SLTALink Manager determines when a standard driver open call in a host application requires dialing and handles these cases. Thus, the host application does not need to know if the network services interface is a local LTS-20-based node or a remote LTS-20-based node.

**NOTE: Remote LTS-20-based nodes cannot be used with Windows NT 3.51 because Windows NT 3.51 does not include the 32-bit TAPI services used by the SLTALink Manager software.**

**Skip this Chapter if you are using the LTS-20 MIP mode.**

## Software Overview

**The LTS-20 software is shared in common with the SLTA-10 Serial LonTalk Adapter. Any references to the SLTA-10 NSI, including file names, apply to the LTS-20.**

The SLTALink Manager is a standalone application that can monitor a modem line, answer an incoming phone call, associate the incoming call'sLTS-20-based node (and hence its network) with a LON device, and then launch a pre-determined application for that particular network or LTS-20. Combined with properly designed LNS host application, the SLTALink Manager lets a LONWORKS network establish a connection to a remote PC through a pair of modems based on an event that occurs locally to the network.

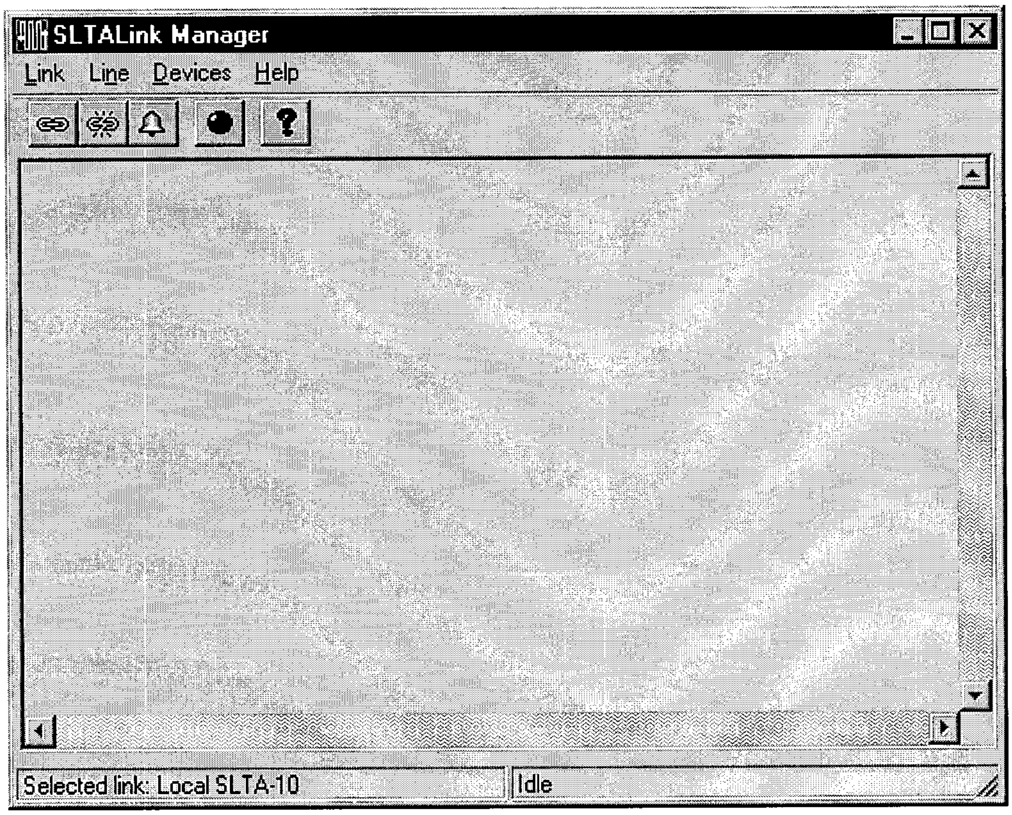
The SLTALink Manager provides a graphical user interface for creating, editing, and diagnosing "links. " Each link represents a particular LTS-20-based node and its network. A link identifies several important aspects of the set-up, including the type of connection (a remote connection via modems or a local, direct connection), the COM port, the Remote Identifier (see below), the baud rate of the serial port on the LTS-20-based node, and the dial-in password, if any. In addition, the link indicates if a security callback is required and may be associated with a host application. The link information is stored in a .s10 file, located by default in the `c:\lonworks\bin\slta10` folder.

The SLTALink Manager application can associate a link with a LON device name and then interface with the LTS-20 NSI mode driver. The SLTALink Manager handles automatically dialing into the network from the PC host, providing the ability for applications with no knowledge of modems or phone numbers to run remotely through a pair of modems. The SLTALink Manager application can be used to connect or disconnect to a remote LTS-20-based node. In addition, the SLTALink Manager has a simple, programmatic way to interact with the LTS-20 NSI mode. This programmatic interface allows an application cause theLTS-20-based node to perform a number of functions, such as dial a phone number or hang up.

The SLTALink Manager includes many diagnostic functions.

*NOTE: Remote LTS-20-based nodess cannot be used with Windows NT 3.51 because Windows NT 3.51 does not include the 32-bit TAPI services used by the SLTALink Manager software.*

Upon invocation of the SLTALink Manager software (`SLTALINK.EXE`), the main screen appears, shown in figure 11.1.



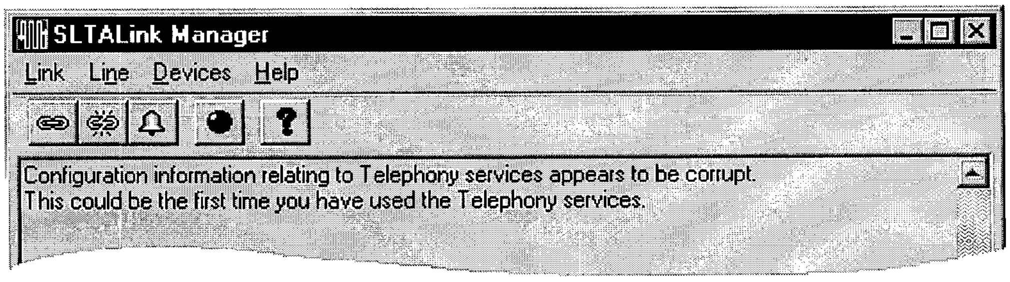
Selected link: Local SLTA-10

Idle

**Figure 11.1** SLTALink Manager Main Screen

## *Establishing a Communications Line for Dialing in to a Network*

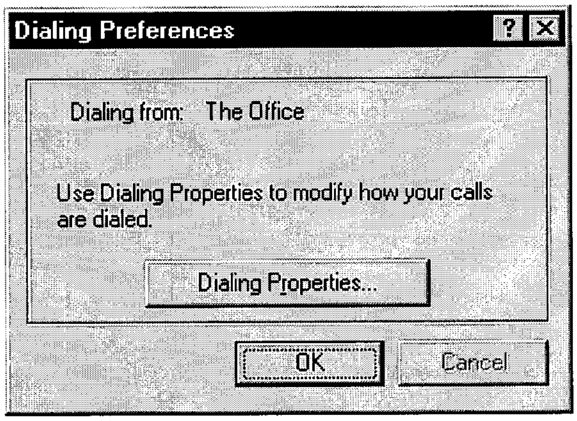
Establishing a communications line is the first task to be completed. Figure 11.2 displays the message that appears when Dialing Preferences is chosen from the Line menu. This message will only appear when telephony information has not been provided. This case usually occurs if the computer has never been configured to use a modem.



Configuration information relating to Telephony services appears to be corrupt.
This could be the first time you have used the Telephony services.

**Figure 11.2** First Time Use Message

This message in figure 11.2 may not be visible due to being covered by the SLTALink Manager Dialing Preferences window. Moving the Dialing Preferences window should reveal the message—if it exists. This leftmost window, shown in figure 11.3, will display "???" for the "Dialing from:" indicator if there has been no dialing location created/chosen.



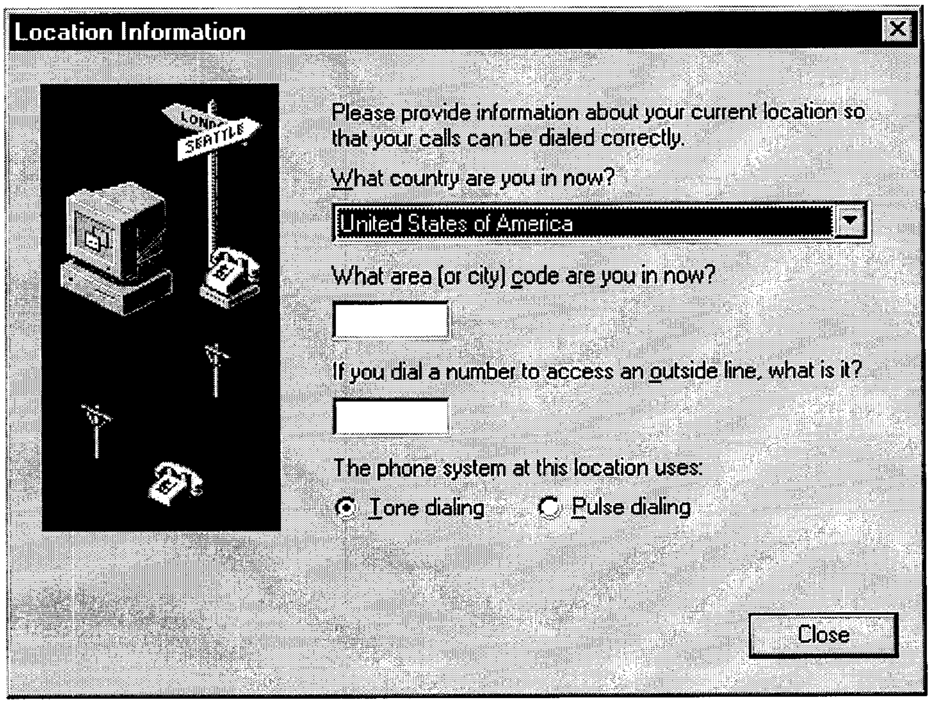
Dialing Preferences

Dialing from: The Office

Use Dialing Properties to modify how your calls are dialed.

**Figure 11.3** SLTALink Manager Dialing Preferences Window

Clicking on Dialing Properties will bring-up the Windows Location Information window (figure 11.4) if the "Dialing from:" indicator reads "???", or if TAPI information has been previously entered — as shown in figure 11.3 as "Dialing from: The Office" — the Windows Dialing Properties window (figure 11.5) will be displayed instead. The Dialing Properties window is a tabbed subset of the Windows Telephony Control Panel.



Location Information

Please provide information about your current location so that your calls can be dialed correctly.

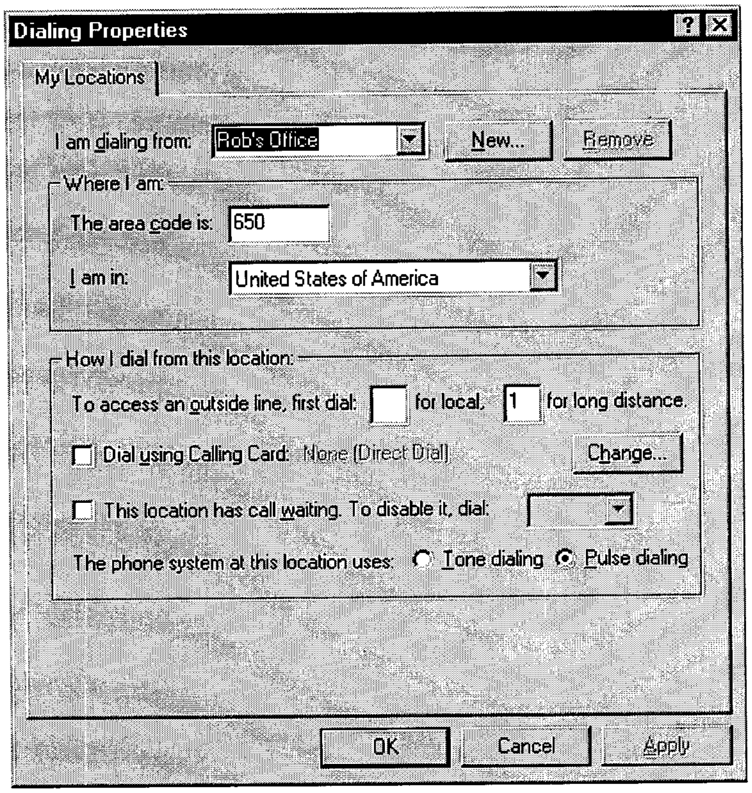
What country are you in now? United States of America

What area (or city) code are you in now?

If you dial a number to access an outside line, what is it?

The phone system at this location uses: [✓ Tone dialing] Pulse dialing

**Figure 11.4** Windows Location Information Window



I am dialing from: Rob's Office

Where I am:

The area code is: 650

I am in: United States of America

How I dial from this location:

To access an outside line, first dial: for local, 1 for long distance.

☐ Dial using Calling Card: None (Direct Dial)

☐ This location has call waiting. To disable it, dial:

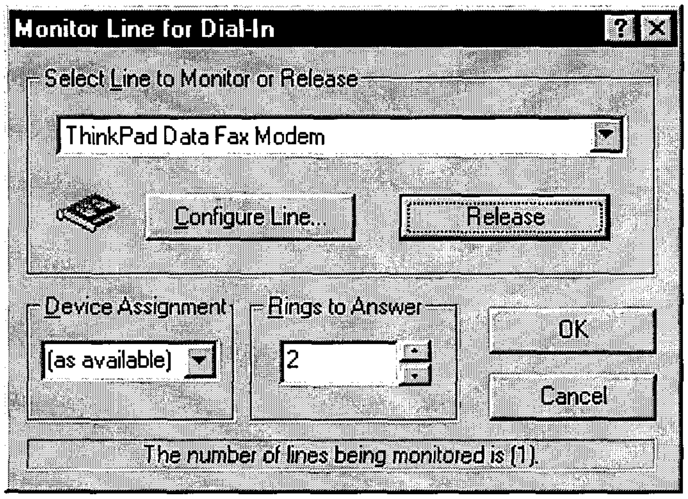
The phone system at this location uses: ○ Tone dialing [✓ ● Pulse dialing]

**Figure 11.5** Windows Dialing Properties Window

## *Establishing a Communications Line for Calls Dialed out to the PC*

The next step is to select a line/modem to monitor for incoming calls. Figure 11.6 shows the Monitor Line window that is displayed when "Monitor for SLTA dial-in" is chosen from the Line menu.

**Multiple phone lines or modem can be monitored (for receiving incoming calls) at the same time by the SLTALink Manager software.**



ThinkPad Data Fax Modem

Device Assignment: (as available)

Rings to Answer: 2

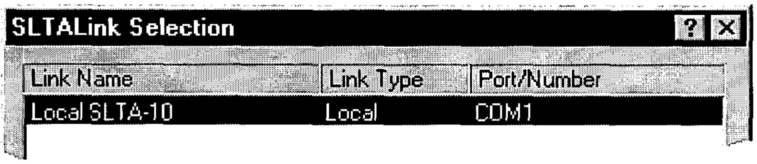
The number of lines being monitored is (1).

**Figure 11.6** SLTALink Manager Monitor Line Window

The option list box will display the list of modems which have been set-up for use on this computer. The list can be created/modified by using the Windows Modem Control Panel. Select the line/modem to be used for incoming calls, then click OK.

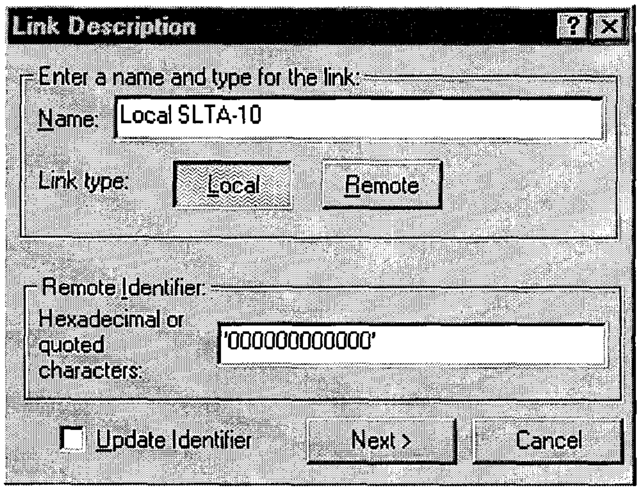
## Establishing Remote and Local Network Sites

Choosing Select/Action from the Link menu will display a screen similar to the screen shown in figure 11.7. Figure 11.8 shows the default local setup.



| Link Name | Link Type | Port/Number |
|---|---|---|
| Local SLTA-10 | Local | COM1 |

**Figure 11.7** Completed SLTALink Selection Window



Name: Local SLTA-10

Link type: Local

Hexadecimal or quoted characters: '000000000000'

☐ Update Identifier

**Figure 11.8** Default SLTALink Selection

Select “Local SLTA-10” and click Edit. This action will present a window allowing the ability to customize the connection—including changing it from Local to Remote, or modifying the name.

## SLTALink Configuration Script Formats

The SLTALink Configuration dialog can accept a script file for importing values to the dialog's user interface. Following this step, the configuration values can then be applied to the SLTA-10 by clicking Apply.

Individual configuration items are processed on a line-by-line basis. All values are not required to be in the script file, and any duplicated assignments are simply overwritten.

Any line may start with a semi-colon, which is treated as a comment line. Line length is limited to 80 characters. Assignments are structured as follows:

keyword=value (note that there are no blank spaces between the components).

Argument strings do not need to be quoted. Keywords are case-insensitive. Edit the script file as a simple text file with carriage returns and line feeds terminating each line.

The acceptable assignments are:

| **Keyword Format** | **Argument Description** |
|---|---|
| Password=password string | Password: Up to 8 characters. |
| Callback=switch value | Callback enable: Either a ‘1’ or a ‘0’ for enabled or disabled |
| HangupTimer=minutes value | Hangup timer: A number between 0 and 255. |
| ModemInit=string | Modem initialization string: A string whose length will be limited by the available EEPROM pool space in the SLTA. |
| ModemDialPrefix=string | Modem dial prefix: A string whose length will be limited by the available EEPROM pool space in the SLTA. |
| DialDir1=string to<br>DialDir5=string | Dial Directories 1 through 5: A string whose length will be limited by the available EEPROM pool space in the SLTA. |
| NVConnect=two digits | NV Auto-connect: Either two digits, or not digits if disabled. The first digit represents the starting dial directory number and the second digit represents the last dial directory. |

| | |
|---|---|
| NSIConnect=two digits | NSI Auto-connect: either two digits, or no digits if disabled. The first digit represents the starting dial directory number, the second digit represents the last dial directory number. |
| ClearEEPool=switch value | Either a '1' or a '0' to enable or disable clearing of the EE pool before applying. |

## Example

```
; An SLTA-10 Configuration Script.
Password=BIG DOG
Callback=1
HangupTimer=5
GuardTimer=20
ModemInit=ATE0V0&C1&D2S0=1M0
ModemDialPrefix=ATDT
DialDir1=14155557001
DialDir2=14155557002
DialDir3=12155557003
Dialdir4=
Dialdir5=
NVConnect=
NVConnect=1 2
```

## Name of Link

The name of the link should be descriptive enough to clearly define to users the connection and remote location.

## Remote Identifier

The Remote Identifier is used to identify a specific link when a dial-in to the computer occurs. It represents the remote SLTA-10/LTS-20-based node in a 12-byte string of characters or hexadecimal numbers. This value here should match the value stored in the remote SLTA-10/LTS-20-based node. It can be entered here as a string in single quotes, or as a series of hexadecimal numbers separated by dashes.

**If this field is blank or all zeroes (00-00-00-00-00-00-00-00-00-00-00-00) then the Remote Identifier will be captured and stored here the next time this connection is made.**

**If this field is all FFs (FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF-FF) then any Remote Identifier will be accepted. The identifier will not be stored on the PC. This is known as the wildcard Remote Identifier. The question mark (?) is also accepted as the wildcard Remote Identifier. The SLTALink Manager software translates "?" to all FFs.**

The Update Identifier checkbox indicates that the remote identifier will be read from the SLTA-10/LTS-20-based node by the SLTALink Manager and the new value will be stored in the .s10 file the next time this link is used.

The SLTALink Manager software allows a user to create two links with different names but the same Remote Identifier. However, when a network dials-out to a PC with multiple links each with the Remote Identifier, the user has no control over which link is selected, which could result in undesired behavior.

## Link Type

The type of link specifies whether the SLTA-10/LTS-20-based node is directly connected to the PC (Local), or if the SLTA-10/LTS-20-based node is at a different location and must be accessed via a set of modems (Remote).

## Configuring the Modem Line

Clicking on "Configure Line" will cause the selected modem's property window to appear. The property window will reflect the options available to the driver of the modem such as volume control and dial-tone detection.

TAPI services will handle the structuring of the call based on the Location Information (see figure 11.4).

## SLTA Password

The Password box allows the user to enter the password for a remote SLTA-10/LTS-20-based node. Up to eight characters may be entered. If entered, the password will be sent to the remote SLTA-10/LTS-20-based node when a connection is made.

**The password is not encrypted when stored on the host computer.**

## Invoking an Application

The SLTALink Manager provides a space to enter the startup application for this link. This may be a full executable path name, or the name of an application that can be found in the system's search path. Command line arguments may also added — including the special macros for link connection variables:

| | |
|---|---|
| %LINKNAME% | Expands to the name of the link, enclosed in quotes. |
| %DEVNAME% | Expands to the device name used by LONWORKS 32-bit applications to access the logical device. This serves the same purpose as %DOSNAME% does for DOS. |
| %NSSNAME% | Expands to the device name used by LonWorks LNS application, for SLTA-10s this will be "SLTALON*n*". |
| %DOSNAME% | Expands to the DOS device name for the logical device, provided the virtual device driver (VDD) is installed, which would be "LON*n*". |
| %ID% | Expands to the remote identifier. This is expressed as either a quoted ASCII string, or as a series of hexadecimal numbers if the identifier contains non-ASCII data. |

| | |
|---|---|
| %RESULT% | Expands to an unquoted word that represents the success or fail reason of the connection. |

The startup application will be launched when a dial-in occurs for this link, or optionally, when a manual connection is made to the link. It will **not** start up if the link is connected to due to an "auto-connect" case.

## Enabling a Callback

If a remote LTS-20-based node has callback enabled then it will expect a callback command whenever someone dials in to it. Check the Enable box if you need to enable the callback feature.

The callback command includes a directory index that points to a phone number stored in the remote SLTA-10/LTS-20-based node. If callback is enabled then one of the remote directory numbers ("Address 1" though "Address 5") can be selected to be used to call the host back.

See *Characteristics of a Well-Designed System* below for a description of how to successfully implement callback.

## Configuration

Use the following screen to configure your SLTA-10/LTS-20-based node.

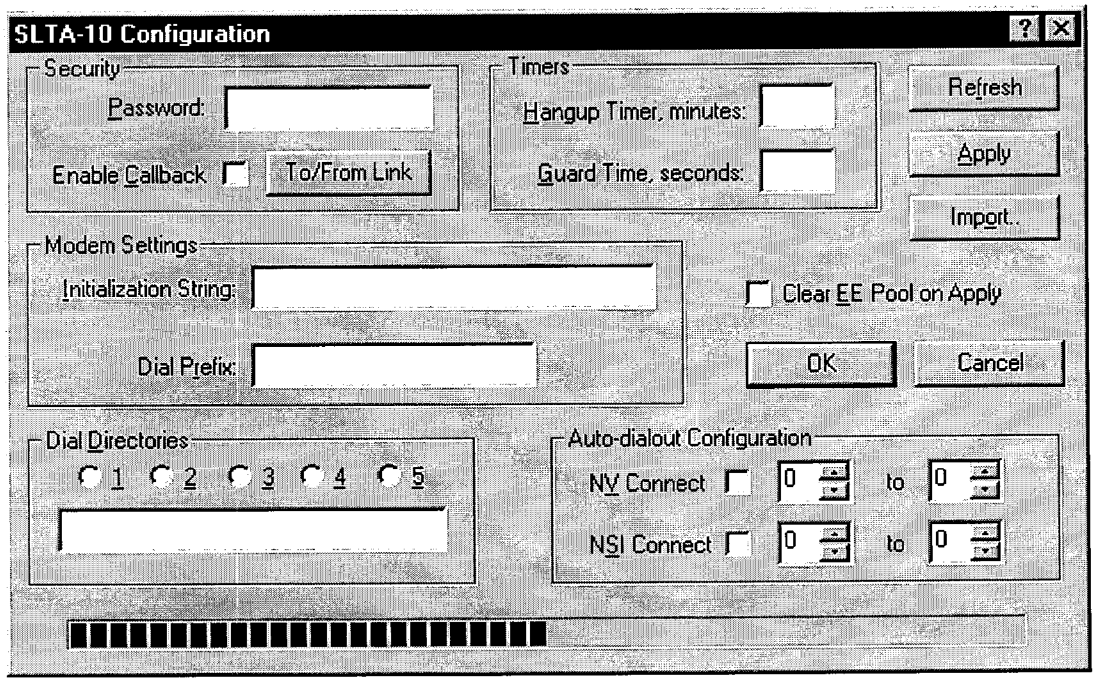


**Figure 11.9** Configuration Screen

## Security

### Password

The SLTA-10/LTS-20-based node may be configured to accept incoming calls and connect the network to the host. Incoming callers may be required to provide a password before the SLTA-10/LTS-20-based node will connect them to the network.

### Enable Callback

Check this box if you need to enable callback.

If a remote SLTA-10/LTS-20-based node has callback enabled then it will expect a callback command whenever someone dials in to it. The callback command includes a directory index that points to a phone number stored in the remote SLTA-10/LTS-20-based node.

## Timers

### Hangup Timer, minutes

Provides a space to enter a hangup timer value for the SLTA-10/LTS-20-based node. This must be within the range of 0-255, for 0 (disabled) to 255 minutes.

The hangup timer controls how many minutes of inactivity must pass before the SLTA-10/LTS-20-based nodehangs up (disconnects the modem).

### Guard Time, seconds

Provides a space to enter a guard time value for the SLTA-10/LTS-20-based node, in seconds. This must be within the range of 0-255.

The guard timer controls how long to wait before attempting to dial the next number for the auto-connect case.

## Modem Settings

### Initialization String

Provides a space to enter a modem initialization /configuration string for the SLTA-10/LTS-20-based node. This string is sent to the modem whenever the SLTA-10/LTS-20-based nodeis reset and it is not currently connected. Special characters may be embedded in the string:

| | |
|---|---|
| ! | Causes a carriage return to be sent. |
| ~ | Causes a 500ms pause. |

The carriage return is not required at the end of the string.

### Dial Prefix

Provides a space to enter a modem dial prefix for the SLTA-10/LTS-20-based node. This controls the characters sent to the modem before the actual phone number is sent. The default is "ATDT" for tone dialing.

| | |
|---|---|
| ! | Causes a carriage return to be sent. |
| ~ | Causes a 500ms pause. |

The carriage return is not required at the end of the string.

## Clear EE Pool on Apply

Check this box to clear the SLTA-10/LTS-20-based node's EEPROM pool before applying the configuration. This will force the clearing and re-programming of the strings that use the EEPROM pool: The Modem Initialization and Dial Prefix strings, and the Dial Directories. If not checked then only the SLTA-10/LTS-20-based node's strings that have been changed will be updated.

Use this feature if you are increasing the size of one string and decreasing the size of another in one step.

## Dial Directories

Provides a space to enter the dialout directory numbers, as strings. To chose which directory entry to edit, simply select one of the buttons above.

| | |
|---|---|
| ! | Causes a carriage return to be sent. |
| ~ | Causes a 500ms pause. |

The carriage return is not required at the end of the string.

## Auto-dialout Configuration

### NV Connect

Check this box to enable the Network Variable update auto-connect feature. When enabled, the SLTA-10/LTS-20-based nodewill attempt to connect to a remote host whenever the SLTA-10/LTS-20-based nodeis not connected and a non-broadcast host-bound NV update message is pending.

The SLTA-10/LTS-20-based nodestarts with the directory number first specified on the right. If this attempt fails then the next directory number is used, until the last directory number has been used. To use a single number simply specify the same directory number in both fields.

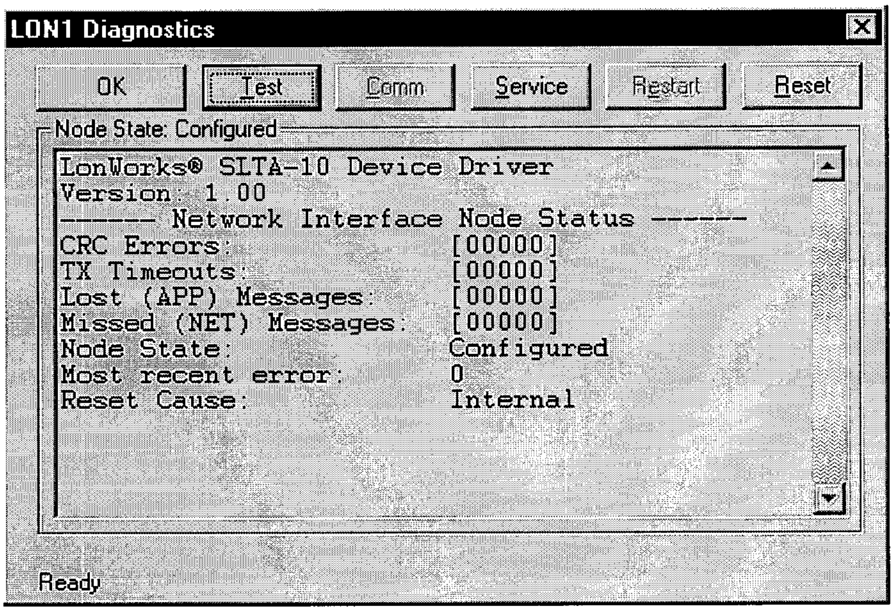
### NSI Connect

Check this box to enable the Network Services (NSI) auto-connect feature. When enabled, the SLTA-10/LTS-20-based nodewill attempt to connect to a remote host whenever the SLTA-10/LTS-20-based nodeis not connected and a host-bound NSI message is pending.

The SLTA-10/LTS-20-based nodestarts with the directory number first specified on the right. If this attempt fails then the next directory number is used, until the last directory number has been used. To use a single number simply specify the same directory number in both fields.

## Diagnostics

A number of Diagnostic and testing services are provided via the Diagnostic Screen, accessed through the Devices menu (see figure 11.10). The Test button retrieves status and error counts from the SLTA-10/LTS-20-based node. The Service button will cause the SLTA-10/LTS-20-based node to broadcast a service pin message on the network. The reset button causes a reset of the Neuron Chip in the SLTA-10/LTS-20-based node, but does not clear the Neuron Chip's system image.



```
LonWorks® SLTA-10 Device Driver
Version: 1.00
------- Network Interface Node Status ------
CRC Errors:                  [00000]
TX Timeouts:                 [00000]
Lost (APP) Messages:         [00000]
Missed (NET) Messages:       [00000]
Node State:                  Configured
Most recent error:           0
Reset Cause:                 Internal
```

Some buttons are left for future releases of additional features.

**Figure 11.10** Diagnostic Screen

## The SLTALink Manager Programmatic Interface

`SLTALINK.EXE` executes as a single process. If you try and run another copy of it, it will defer itself to the original process. However, command line arguments may be passed in this manner which will direct the existing process to perform certain types of tasks. These command line options are:

| | |
|---|---|
| **"Linkname"** | A link name is required for all actions. If the link name alone is passed then that link will be connected to. It must be enclosed in double quotes, since the link name can have embedded spaces.<br><br>`C:\lonworks\bin\sltalink.exe "Remote"` |
| **/D** | This causes the specified link to be disconnected.<br><br>`C:\lonworks\bin\sltalink.exe "Remote" /D` |
| **/# "number"** | This overrides the phone number for the link. If you have checked the "Use Country Code and Area Code" option for this link then the number must be in the 'canonical' format without a '+' sign and enclosed in double quotes. A canonical number is defined as a country code followed by a space, followed by an area code or city code enclosed in parenthesis, followed by a space and the rest of the phone number. TAPI will take this and decide how to translate it before performs the dial-out, based on your dialing preferences. Even if the call is local you should include the area code / city code. This is the same format as the number that appears in the link selection dialog under the Port/Number column.<br>Example: "1 (800) 555-1213"<br><br>If you have **not** checked the "Use Country Code and Area Code" option then the number will be used un-translated for dial-out. In this case you probably won't specify a canonical number.<br><br>`C:\lonworks\bin\sltalink.exe "Remote" /# "1 (650) 555 1213"` |
| **/P "passwd"** | This overrides the password for the link. It must be enclosed in double quotes.<br><br>`C:\lonworks\bin\sltalink.exe "Remote" /P "passwd"` |

Always include a space between each element of the command line arguments.

## Using the DOS "Stub" Driver

The DOS stub driver, which is added as part of the install, allows DOS and Windows 3.1x applications to run on top of the drivers for Windows 95 and NT. The following line is required in the `CONFIG.SYS` or `CONFIG.NT` file that is loaded on startup:

```
DEVICE=%SystemRoot%\system32\PCLTDOS.SYS /Dn
```

This makes the device 'LON*n*' available for DOS applications.

When LON*n* is opened by an application and the SLTALink Manager has been configured to associate LON*n* with a particular link, the SLTALink Manager will auto-connect to the SLTA-10/LTS-20-based node locally or remotely. That is, the SLTALink Manager automatically dials-in to the network defined by the link if required.

## Characteristics of a Well-Designed System

Well understood strategies used with the SLTA-10/LTS-20-based node and the SLTALink Manager for the following system functions are essential for reliable system design: Call Initiation, Call Termination, and Monitoring.

### *Call Initiation*

The four scenarios for call initiation are: dial-in to the network only, dial-out to the remote PC only, dial-in / dial-out, and callback.

#### Dial-In to the Network Only

In the most straight-forward case, a user launches an application. The application opens the driver, which is associated with a particular link. The SLTALink Manager application dials the phone number in the link (or the phone number the application passes down to the SLTALink Manager perhaps to a generic link) and establishes the connection. Similarly, the user could select the link from within the SLTALink Manager and cause a manual connection. At this point, either the SLTALink Manager would launch the pre-determined application from the information stored in the link file or the user could manually launch the application. In all of these cases, the user is assumed to initiate the call. The user could be a human operator or another application that initiates a dial-in based on a clock, for example.

For the dial-in only scenario, the system strategy issues primarily have to do with associating the link or phone number with the application. Where there are only one or two links, this is very easy. When one PC host can be connected to many different networks, we offer two standard solutions. The first is to have the user navigate the SLTALink Manager's GUI. Under the Link menu, the Select item lists approximately 40 links (of the 1000 possible). The second solution is a monitoring application that programmatically interacts with the SLTALink Manager to send down the appropriate phone number, perhaps to a generic out-going link.

#### Dial-Out to the Remote PC Only

The three common approaches for initiating a dial-out are: sending a network variable update to the SLTA-10/LTS-20-based node, sending an AddMyNSI message to the SLTA-10/LTS-20-based node, or sending an explicit message from a "Helper/Dialer" node on the network.

In the first case, the remote host application needs to have an explicitly-bound input network variable and the SLTA-10/LTS-20-based node's NSI mode EEPROM must be configured correctly. See the configuration section in this chapter or go to Chapter 17 for more information on configuring the SLTA-10/LTS-20-based node's EEPROM. See also *Call Termination* below.

In the second case, the remote host PC is assumed to have the NSS engine and a second NSI on the network (perhaps a service technician with a laptop running a PCC-10 card) is required to send the AddMyNSI message. Also, SLTA-10/LTS-20-based noder's NSI mode EEPROM must be configured correctly.

In the third case, a custom Neuron Chip application must be written.

All three cases could be used with the same SLTA-10/LTS-20-based node.

In the dial-out only case, besides the call initiation, the SLTALink Manager must be able to launch the appropriate application – with the correct database and device driver name. One system solution is to create a separate link for each SLTA-10/LTS-20-based node. Each link then stores the Remote Identifier of its SLTA-10/LTS-20-based node after the first connection. Upon connection, the appropriate application and command line arguments stored in the link get launched. A second viable approach is to create a generic link that uses the wild card as a Remote Identifier to launch a generic application using command line arguments to specify the appropriate network or database and device driver name. These arguments are available and described above under *Invoking an Application*.

**Note: if the device driver information used in the application does not match the device driver name being used by the link, the newly launched application can open a second device driver – which may result in an attempt to dial-in to the network. Since the modem is still presumably in use with the original dial-out call to the host PC, the second call will fail. The result is a system-level failure.**

## Dial-In / Dial-Out

These scenarios are the combinations and permutations of the above. However, it needs to be pointed out that not all dial-in strategies can co-exist with all dial-out strategies. For example, if the dial-out strategy involves having the SLTALink Manager match the incoming call to the wild card Remote Identifier and if the dial-in strategy requires a separate link for each Remote Identifier, then it is possible that a call initiated from the network will be received by the SLTALink Manager and will be matched with the link created for the dial-in case. The correct application may not be launched and a system-level failure may occur.

## Callback

The SLTA-10/LTS-20-based node callback functionality works as follows: A call is initiated from some remote PC to an SLTA-10/LTS-20-based node on a network, which must have its NSI mode EEPROM configured to require callback. The SLTA-10/LTS-20-based node answers; the remote PC identifies to the SLTA-10/LTS-20-based node one of the SLTA-10/LTS-20-based node directory entries to use for the callback. An SLTA-10/LTS-20-based node configured to require a callback will not accept any other direction from the host at this time. The original call is terminated, and the SLTA-10/LTS-20-based node calls the phone number indicated in its directory. Note: this does NOT need to be the phone number of the original remote PC that initiated the call. Typically, the SLTALink Manager on the remote host dialed answers the call and launches the appropriate application.

Several possible system-level failures include:

- The original remote host expects to receive the callback, but the directory index reflects the phone number of another remote host.
- The original remote host application opens an LNS database and initiates the first call. The callback is directed back to the original remote host PC. The SLTALink Manager on this PC receives the callback just like any other normal dial-out call and launches the application contained in the link. At this point there may be two copies of the application open. Depending on sharing configuration, the second application may fail because appropriate LNS database in already opened.

To prevent these failures, Echelon recommends that the initial call should either be a manual connection from within the SLTALink Manager or the initial call should originate from a "dummy" application that terminates itself without opening the LNS databases.

## Call Termination

The four scenarios for call termination include: termination of the host application, application controlled hang-up, a manual disconnect in the SLTALink Manager, and time-out. In all of these cases, the important system-level issues involve making sure that the termination strategy is compatible with the call initiation strategy.

The behavior of an application at termination is not always known. By default, applications based on the Object Server of LNS 1.5 and higher should exhibit two types of behavior. First, if the interface adapter is an SLTA-10/LTS-20-based node in NSI modem using modems and there are explicitly bound network variables to the host application, then at the LNS application's termination the host network variables and their connections should not be removed from the LNS database. This behavior facilitates the use of the first case described under *Dial-Out to the Remote PC Only* where a network variable update addressed to the SLTA-10/LTS-20-based node's NSI mode results in a call being initiated. Second, if the interface adapter is not an SLTA-10/LTS-20-based node in NSI modem using modems or there are no explicitly-bound network variables to the host application, then at the LNS application's termination, any host network variables and their connections are removed from the LNS database and the other nodes in the network. In addition, if the interface does not host the LNS database, upon termination of the LNS application the interface is deconfigured.

The second behavior described would be desirable in the event that multiple remote host PCs needed to be able to dial-in to the same network. As long as no explicitly-bound network variables were left when the host applications terminated, then several remote PCs could share one SLTA-10/LTS-20-based node. Note: this assumes that the LNS Server exists somewhere on the network and is not located on one of the remote PCs sharing the single SLTA-10/LTS-20-based node.

Both an application controlled hang-up (using the `SLTALINK.EXE` programmatic interface) and a manual disconnect in the SLTALink Manager will terminate the phone call; however, neither results in the termination of the host application. In these scenarios the host application remains running as the call can be re-established by the host application itself, by a manual connect in the SLTALink Manager, or a dial-out initiated on the network. The disadvantage of these system solutions is that they do not scale well to monitoring multiple networks on one PC. The result is many applications continually and concurrently running on the same PC. Also, one possible system failure to avoid is that the SLTALink Manager settings may result in multiple copies of the host applications.

By default, the SLTA-10/LTS-20-based node NSI mode will terminate a phone call after three minutes with no traffic going across the modems. As with the application-controlled hang-up and the manual disconnect in the SLTALink Manager, this scenario does not result in the application terminating. This scenario therefore carries the same advantages and disadvantages as those described in the previous paragraph.

## Monitoring: Application Termination Strategy

There are three strategies for terminating the remote LNS monitoring application.

The first strategy is to require user intervention to shut down or terminate the application. No special software must be written for this case. For the dial-in to a network scenario when the call is user initiated, the user is presumably available to shut down or terminate the application. In the dial-out case, the requirement of user intervention to terminate the application means that every call from a network is seen by an user. If the network is sending alarms, user intervention is highly desirable.

The second approach requires a node on the network to send a network variable update to the host when the network no longer requires the host application to be up and running. When the host application receives this network variable, it should enter a shut-down routine. This approach places the responsibility on the network to determine when the host application is needed and requires a hook in the monitoring application, but provides a level of automation when dial-out is used.

The third approach is the most blunt. In this scenario, the remote host application terminates itself either after completing a series of actions or based on a timer. We do not recommend this scenario for the general scenario, but it has certain appeal for some applications. For example, a connection may be established through dial-in or dial-out based on a timer. The host application could then take a series of measurements from the network, log them to a file, and then terminate itself.

## Monitoring: Missing Messages after a Dial-Out

In general, the message that triggers a dial-out from the SLTA-10/LTS-20-based node on the network to a remote PC host is lost.

When a network variable update is sent to the SLTA-10/LTS-20-based node and no phone connection is currently up and running, the SLTA-10/LTS-20-based node is typically configured to dial-out to the host. On the remote PC host, the incoming call is answered by the SLTALink Manager application. The SLTALink Manager then reads the Remote Identifier in the SLTA-10/LTS-20-based node and searches through all the .s10 files for a match. The SLTALink Manager application then typically launches the application listed in the link with optional command line arguments available, for example, to open the correct LNS database with the correct device name. The process of opening the LNS application results in the buffers of the SLTA-10/LTS-20-based node being over-written; thus, the original message is lost. Since this message network variable update is by definition important enough to warrant establishing a connection to the host, a system strategy is required so that this data reaches the host.

The two basic system strategies are: (1) to have the node on the network continue to send out the network variable update until the host application sends a message to the node telling it to stop, or (2) to have the host application seek out the information upon being launched. The first approach places the burden on nodes in the network; the second approach places the burden on the host application. The first is more direct and is likely to result in the information getting to the host more quickly. The second approach has the primary advantage that no special Neuron Chip application code is required; also, since the call initiation and host application launch may require minutes, the time it takes the host application to poll several network variables is not significant. In the best scenario, upon being launched the host application would first check with a datalogging node on the network that records alarms and also the system state, mode, or health information.

## Monitoring: LNS Application Design Issues

A well-designed LNS monitoring application using the SLTA-10/LTS-20-based node through modems should use the correct version and layer of LNS, should handle initialization of the application with the correct LNS database and device driver name, and should have a phone call session termination and an application termination strategy.

The remote monitoring application is based on LNS Object Server of LNS 1.5 or higher. Applications based on the LNS 1.0 and LNS 1.01 do not behave well by default and do not allow for certain dial-out scenarios. Applications written at the NSS layer are not supported for use with the SLTA-10/LTS-20-based node across a pair of modems.

The remote monitoring application should open the appropriate LNS and device driver. Specifically, the mapping of the Remote ID to the LNS database should be handled by the application. We suggest using the command line arguments with the information available from the SLTALink Manager. Note LNS capacity keys may need to be hard-coded in the monitoring application.

Finally, the LNS application needs to implement a termination strategy that meets the needs of the application and the system.

## Good Practices / Schemes that Work

Use these guidelines to avoid the system-level failures detailed above:

- When expecting the SLTA-10/LTS-20-based node to dial-out from the network to a remote PC, dedicate one SLTA-10/LTS-20-based node to dial-out and always have that SLTA-10/LTS-20-based node connect to the same remote PC. Make sure that the NSI EEPROM is configured correctly. See figure 11.11.
- If an LNS-based application is connected to the network through modems, dedicate one SLTA-10/LTS-20-based node in the network to handle dial-out and dial-in with this PC that has the LNS server. Make sure that the SLTA-10/LTS-20-based node's EEPROM is configured correctly. See figure 11.12.
- Several PCs can share one SLTA-10/LTS-20-based node as long as the calls are all initiated on the remote PC hosts (i.e., dial-in only) and each remote LNS application removes the bound connections to its host before terminating. See figure 11.13.

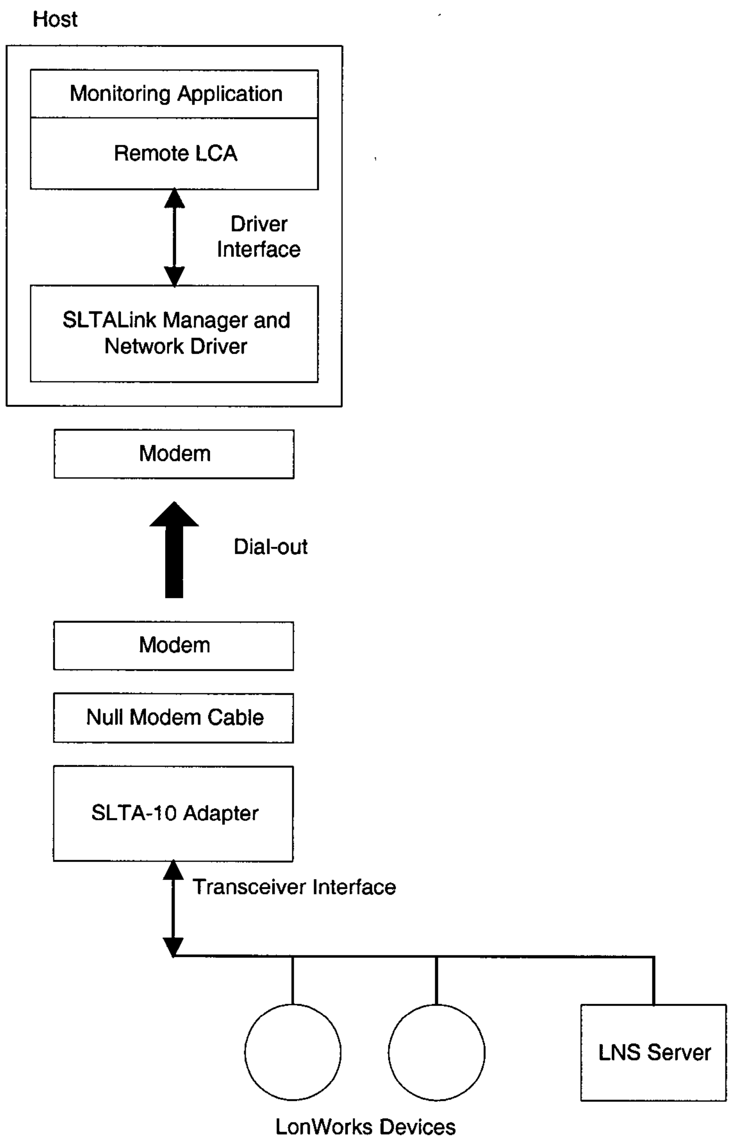


**Figure 11.11** Dedicated Adapter using Dial-out

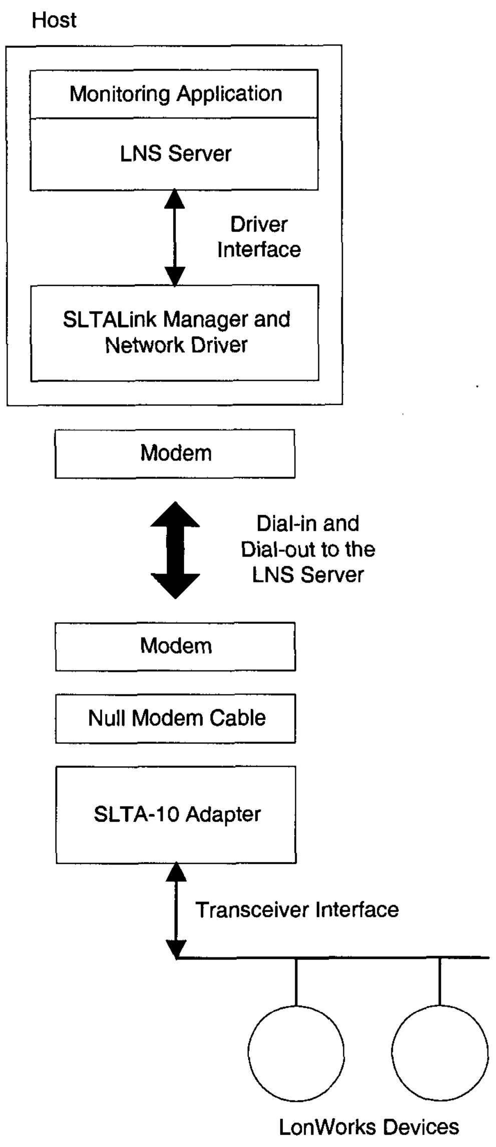


**Figure 11.12** Dedicated Adapter hosting the NSS

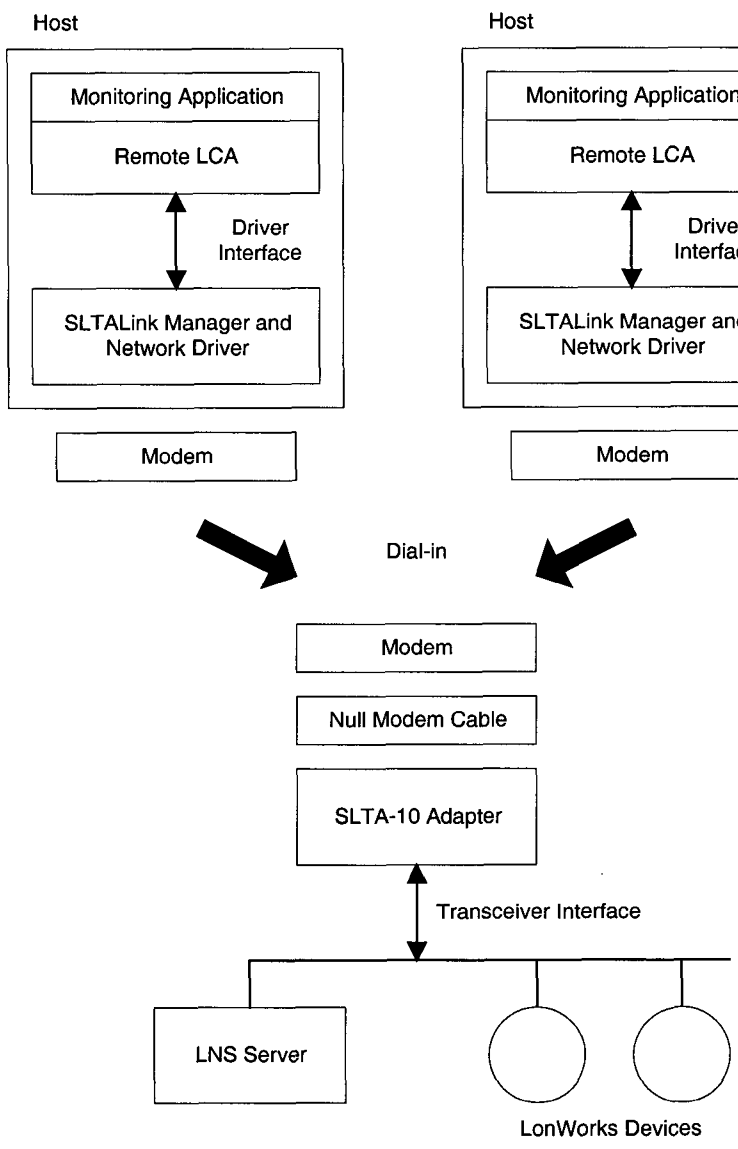


**Figure 11.13** Shared Adapter using Dial-in

# 12

# Using the DOS Driver with LTS-20 MIP Mode

This chapter describes the DOS network driver supplied with the Connectivity Starter. The driver also is available from the Developer's Toolbox on Echelon's web site at www.echelon.com. The DOS network driver provides a device-independent interface between a DOS or Windows 3.1x host application and the LTS-20 in MIP mode. The driver is configurable to use one of four PC/AT serial ports, COM1 through COM4, at one of eight serial bit rates.

**Skip this chapter if you are using the LTS-20 NSI mode.**

## Installing the LTS-20 MIP Mode Driver for DOS

**The LTS-20 software is shared in common with the SLTA-10 Serial LonTalk Adapter. Any references to the SLTA-10 NSI, including file names, apply to the LTS-20.**

*The DOS driver is supplied on the floppy diskette included with the Connectivity Starter Kit. The latest version of this driver may be obtained from the Echelon web site.*

The LTS-20 MIP mode network driver for DOS is installed by adding a DEVICE command to the DOS `CONFIG.SYS` file. Edit the `CONFIG.SYS` file to include the line:

```
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS [options]
```

Substitute your drive and directory name if other than `C:\LONWORKS\BIN`. Reboot the PC after adding this line to load the driver. For example, the following command would be used with a locally attached SLTA-10/LTS-20-based node in MIP mode installed on COM2 as device LON1 running at 115,200 bps with autobaud enabled:

```
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS /P2 /D1 /B115200
```

*Warning! The `/A` option must be present in the `CONFIG.SYS` entry if the AutoBaud function is Enabled, UP position, or the SLTA-10/LTS-20-based node will not function correctly. The /A option also may be left in the `CONFIG.SYS` entry if the Autobaud function is disabled (default.*

The available options for theMIP mode network driver for DOS are described in the following sections.

### Buffer Options

`/Onn` Sets the number of output (downlink) buffers within the driver to <*nn*>. The buffer sizes within the driver are pre-set to accommodate 255 byte packets. The LTS-20 in MIP mode has application output buffers that may be increased to as large as 255 bytes. There must be at least 2 buffers and the maximum allowed number for <*nn*> is limited by the size of the buffer (258) times the total number of input and output buffers within the driver. The entire buffer space plus the driver code itself cannot exceed 64Kbytes. The size of the driver code itself is 9Kbytes. The number of output buffers required is determined by the characteristics of the host application. If the host application always waits for an outgoing message completion before sending another message, then only two buffers are required. If the host application is set up to overlap transactions, more buffers may be required. In this case greater parallelism may be achieved at the expense of host application code complexity.

`/Inn` Sets the number of input (uplink) buffers within the driver to <*nn*>. The buffer sizes within the driver are pre-set to accommodate 255 byte packets. The LTS-20 has application output buffers may also be increased to as large as 255 bytes. There must be at least 2 buffers and the maximum allowed number for <*nn*> is limited by the size of the buffer (258) times the total number of input and output buffers within the driver. The entire buffer space plus the driver code itself cannot exceed 64Kbytes. The number of input buffers required is determined by the expected incoming traffic and the capability of the host application to process it. If the incoming traffic is bursty, more input buffers are required. If the application cannot process incoming traffic fast enough, the input buffer pool will fill up with unprocessed packets. In that case, the LTS-20 will not be able to pass any new data to the host, and the input application buffers in the LTS-20 will start to fill up. Once that occurs, messages will be lost, possibly causing incoming LonTalk transactions to be retried, and eventually causing the sender of the message to receive a failure indication.

## Serial Bit Rate Options

`/B`*`nnnnnn`*

Sets the serial bit rate to <*nnnnnn*>. The available serial bit rates are listed below.

Available serial bit rates are:

**`1200, 2400, 9600, 14400, 19200, 38400, 57600, 115200.`**

This rate represents the serial bit rate between the PC and the SLTA-10/LTS-20-based node when using a direct serial connection, and between the PC and the modem when using a remote connection. For remote connections, the PC-to-modem serial bit rate, telephone line speed, i.e., modem to modem serial bit rate, and the modem-to-SLTA-10/LTS-20-based node serial bit rate may be different. The PC-to-modem serial bit rate is controlled by the network driver on the PC using the `/B` option; the telephone line speed is selected by the modems based on modem configuration; and the modem-to-SLTA-10/LTS-20-based node serial bit rate is controlled by the hardware configuration of the SLTA-10/LTS-20-based node as described in Chapter 2 (Autobaud cannot be used in this configuration).

For local connections with the Autobaud option disabled, the serial bit rate specified by this driver option must match the rate specified by the configuration jumpers.

**`/A`**

Enables the Autobaud feature. This provides the autobaud sequence whenever the driver is opened. The default setting for the driver is *Autobaud Disabled.* If the Autobaud input on the SSLTA-10/LTS-20-based node is enabled, then this option must be specified. As described in Chapter 4, the default setting is disabled, so the `/A` option does not need to be specified with the default hardware configuration. This option may not be used with the modem support (`/M`) option.

*If you are using the default hardware configuration (autobaud disabled), the autobaud option (/A)* ***does not need to*** *be enabled .*

## DOS Device Options

**/P***n* Sets the serial port to <*n*> where <*n*> is 1-4 for COM1 - COM4. The default is COM1.

**/D***n* Defines the device unit number as <*n*>, where <*n*> is between 1 and 9, so that the DOS device name is "LON1" through "LON9". The default is 1 for "LON1". This option can be used to support multiple network interfaces on a single PC. For example, this device name is passed as a parameter to `lxt_open()` when using the LonManager API. When invoking the sample host application `HA`, the device may be specified with the `-D` option, for example:

```
HA -DLON2
```

**/U***n* Sets the serial port interrupt request number (IRQ) to a non-standard value <*n*>, where <*n*> is between 1 and 7. If the serial port in use is COM3 or COM4, you may want to use a unique, unused IRQ for that port. Many serial ports and internal modems allow the selection of a non-standard IRQ such as IRQ2 or IRQ5.

**/C** Enables communications interrupt chaining. Some PCs may incorporate up to four serial ports. If supported by the serial hardware, COM1 and COM3 may share the same interrupt (as do COM2 and COM4). This option may enable the driver to support the shared interrupt by "chaining" to the interrupt vector that was in place when the driver was loaded. This option is not necessary if your system does not use COM3 or COM4, or if COM3 or COM4 use a different interrupt request number. When installing two drivers on a system on COM1 and COM3 (or COM2 and COM4 with the same interrupt request number), the last installation of the driver should use this option. Here is an example of a `CONFIG.SYS` file entry for such a system.

```
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS /B38400 /A /P1
DEVICE=C:\LONWORKS\BIN\LDVSLTA.SYS /B38400 /A /P3 /C
```

**Table 12.1** Hardware Configurations COM Ports

| Device | I/O Address | Interrupt (*) |
|---|---|---|
| COM1 | 0x3F8 | 4 |
| COM2 | 0x2F8 | 3 |
| COM3 | 0x3E8 | 4 |
| COM4 | 0x2E8 | 3 |

(*) May be changed with the `/U` option

## Timing Options

| | |
|---|---|
| **/R***nn* | Defines the flush/retry count in 55ms intervals. This value is used in error states for re-transmitting requests and for terminating receive flushes when input errors occur. Normally, this option should not be specified. |
| **/W***nnn* | Includes a delay of *<nnn>* microseconds when transmitting downlink. This parameter can be used to pace the rate at which bytes are transmitted downlink to the SLTA-10/LTS-20-based node, and may be required for high-performance network management tools. The delay is executed following the transfer of each data byte to the host's UART, and only after the first 15 bytes have been sent. Since the SLTA-10/LTS-20-based node employs a 16-byte deep FIFO buffer in its UART, the first group of bytes sent do not need to be paced. The pacing delay will have no effect unless it is greater than the actual period it takes to transmit a single byte at the given serial bit rate. The time taken to transmit a byte is 173 $\mu$s at 57,600 bps, and 86 $\mu$s at 115,200 bps. This option should be used at 115,200 bps if messages greater than 16 bytes are to be transmitted. A value of `/W120` is suggested. This option is not required at the serial bit rate of 38,400bps or slower. |
| **/Z** | By default, the MIP mode firmware disables network communications after a reset by entering a `FLUSH` state. This state causes theSLTA-10/LTS-20-based node to ignore all incoming messages and prevents all outgoing messages, even service pin messages. The MIP mode network driver for DOS automatically enables network communications when the SLTA-10/LTS-20-based node is opened and when it receives an uplink message from the SLTA-10/LTS-20-based node indicating that it has been reset. However, the host application itself must explicitly enable network communications if the `/Z` option is specified and the CFG1 input is set to *Network Disabled.* <br><br> Host applications which need to configure the SLTA-10/LTS-20-based node prior to enabling network communications should use this option. This option should not be used with the LonManager API, LonManager LonMaker Installation Tool, or the LonManager DDE Server. More information about the `niFLUSH_CANCEL` message is provided in the *LONWORKS Host Application Programmer's Guide.* |

## Network Interface Protocol Options

| | |
|---|---|
| **/F** | Enables the full interrupt mode of the driver. If this option is <u>not</u> specified, the driver will disable interrupts for the duration of each link-layer transfer. This ensures that no data will be lost due to other system interrupts, and is acceptable at high serial bit rates. The driver will use interrupts for the first byte of each uplink interface buffer. When the uplink interrupt is received, the driver reads the rest of the message without interrupts via polled I/O. Interrupts are disabled during the uplink transfer. This assures that no other system interrupts will occur resulting in lost uplink data frames. Downlink transmissions are sent directly via polled I/O of the serial |

port from the write function call. The host write functions will not return until the message has been sent downlink. When using the ALERT/ACK link protocol, interrupt latency is not a problem, since the SLTA-to-host protocol includes an acknowledgment of the start of the message. The driver employs timeouts in order to prevent lockout of the write function, and timeouts for clearing various states of the transmitter/receiver when line errors occur.

When operating at lower serial bit rates, it becomes less desirable to disable interrupts for long periods. The trade-off with using the full interrupt mode is that other system interrupts may cause loss of data in the serial port's UART. If the `/F` option is specified, the driver uses interrupts for every uplink and downlink byte transferred. Downlink messages are buffered from the device write function and are sent downlink under interrupt control. Uplink messages are received under interrupt control and are buffered also. This option should be used for serial bit rates of 9,600 bps or slower.

**/M** Enables modem support and the reliable transport protocol. This option must be specified if the host is to communicate with the SLTA-10/LTS-20-based node via a modem connection. The SLTA-10/LTS-20-based node must be configured with CFG2 input in the *Remote Host* setting. When connected, the selected SLTA-10/LTS-20-based node and Host network interface link protocol is in effect. When disconnected the only allowable link layer traffic is of the 'modem direct' type, where ASCII strings are being exchanged between the host and the modem, for example, AT commands to dial out. Any other network interface traffic is not allowed when disconnected from the SLTA-10/LTS-20-based node. Calls to the read function will result in no network interface data messages (`LDV_NO_MSG_AVAIL`), and any call to a write function that needs to communicate with the SLTA-10/LTS-20-based node via the modems will result in a No Output Buffers Available error (`LDV_NO_BUFF_AVAIL`). Once the connection is made, normal network interface traffic may resume.

This option also enables the reliable transport protocol. This protocol includes the addition of a message sequence number and the end of message ACK/NACK code. See Chapter 4 for a description of this protocol.

**/N** Disables the ALERT/ACK network interface link protocol, and enables the buffered network interface link protocol. Network interface messages are sent without waiting for the ALERT ACK response. Both sides of the interface (the SLTA-10/LTS-20-based node and the driver) must have the same setting. This option should not be used with the */M* option.

**/Q** Allows modem responses to be sent uplink to the host. When the telephone link is disconnected, these messages are ASCII strings with the network interface command type `niDRIVER` (0xF0). If `/Q` is specified, the host application must be able to handle messages, such as `NO CARRIER`, that might come from the modem itself if problems occur in the connection.

| | |
|---|---|
| `/X` | Disables the buffer request protocol. When this option is enabled, the driver requests the buffer count from the SLTA-10/LTS-20-based node using the `niSBUFC (0xE7)` command whenever the interface is opened, or when the interface is reset, and reports an niRESET to the host. The driver keeps track of the number of available output buffers in the SLTA-10/LTS-20-based node by examining both uplink and downlink messages. This option prevents the use of one message type: A local network management command not using a request/response service. Normally this type of message is not used. One exception could be the `Set Node Mode: Reset` command, which would result in the node resetting and the buffer management recovering on its own anyway. Otherwise, if this type of message is used, no uplink response would occur and the driver could not track the fact that a new output buffer has been made available. |

**Table 12.2** Configuration Settings and DOS Driver Options

| ***Input*** | ***Input State*** | ***Driver Option*** |
|---|---|---|
| CFG 2 | Local Host; No Transport Protocol | `/M` not specified |
| CFG 2 | Remote Host; Reliable Transport Protocol | `/M` specified |
| CFG 3 | ALERT/ACK Link Protocol | `/N` not specified |
| CFG 3 | Buffered Link Protocol | `/N` specified |

## Calling the Network Driver from a Host Application

The MIP mode network driver for DOS supports the `open`, `close`, `read`, `write`, and `ioctl` DOS calls. See Chapter 4 of the *Host Application Programmer's Guide* for more details.

When the MIP mode network driver for DOS is loaded during execution of the `CONFIG.SYS` file, it does not attempt to communicate with the SLTA-10/LTS-20-based node.

When the network driver is opened with the DOS `open` call, it establishes communications with the SLTA-10/LTS-20-based node. The network driver returns an error if this fails, for example, if the SLTA-10/LTS-20-based node is disconnected, powered down, or configured incorrectly. If the open call succeeds, the driver enables network communications by clearing the SLTA-10/LTS-20-based node `FLUSH` state, if configured to do so.

The DOS `read` call is defined to return the number of bytes read from the device. Some LONWORKS standard network drivers return 0 if there are no uplink messages available. DOS reports this as an end-of-file condition and prevents further reads from succeeding. However, the SLTA-10/LTS-20-based node driver returns a length of 2, and sets the first byte of the caller's buffer (the cmd/queue byte) to 0 to indicate that there is no uplink message available.

Normally, the DOS `read` and `write` calls are not used with LONWORKS standard network drivers. This is because any error from the network interface will display the familiar `Abort, Retry, Fail?` error message from DOS, unless the caller has installed a critical error device handler. Therefore, DOS applications using a network device typically call direct entry points into the driver. This also allows more detailed error status to be returned to the application. The addresses of these entry points are obtained by calling the `ioctl()` function of the driver.

This function call is used as follows:

```
int ioctl(int handle, int func, void far *argdx, int argcx);
```

- `handle` is an integer returned by an earlier successful call to `open()`, specifying the LONWORKS network driver `LON`n to be opened.
- `func` is the value 2, meaning that the application is reading information from the driver. For LONWORKS standard DOS network drivers, the information returned is the network interface direct call structure.
- `argdx` is a pointer to a caller-declared structure that will contain the direct entry points into the driver. See the structure `direct_calls` in the file `NI_MSG.C` in the supplied example host application for usage.
- `argcx` is the size of the structure.

Function code 2 is supported by network drivers for DOS to return three direct entry points into the driver code. The network driver for the SLTA-10/LTS-20-based node supports an additional option to function code 2, as well as function code 3, which is used to manage the modem control state of the driver. These options are not used when the SLTA-10/LTS-20-based node is connected directly to a host. They are provided primarily for use while establishing communications with a remote host. For example, the host connect utility (HCU) described in Chapter 18 of this manual uses these functions. Host applications that only communicate to the SLTA-10/LTS-20-based node via an already-established telephone connection do not need to concern themselves with these functions. If you wish to establish or take down telephone connections during the execution of your host application, use the source code of HCU as a guide.

When function code 2 is used, `argdx` points to the `direct_calls` structure defined for all LONWORKS standard network drivers for DOS. If `argcx` is 13, the size of the standard direct calls structure, then three direct entry point addresses are returned as usual. If `argcx` is 4 (the size of the structure `ioctl_get_dcd_s`), then the state of the modem's DCD line is returned as a `TRUE` or `FALSE` value. Note that the status field is 16 bits in this structure, but 8 bits in the direct calls structure.

```
struct ioctl_get_dcd_s {
      unsigned ioctl_stat;   // 16 bit status
      unsigned dcd_state;    // Data Carrier Detect (TRUE or FALSE)
}
```

Function code 3 is used when the application wishes to write information to the driver. For the SLTA-10/LTS-20-based node driver, `argdx` points to the following structure, and `argcx` is its size:

```
struct ioctl_o_info_s {
    unsigned  ioctl_stat;       // 16 bit status
    unsigned  sub_command;      // use enum sub_command
    unsigned  mode;
    unsigned  mode_aux;
}
enum sub_command {
    SUBC_set_opt     = 1,       // set driver options
    SUBC_set_DTR     = 2,       // set DTR line
    SUBC_set_baud    = 3,       // set serial bit rate
};
```

There are three sub-commands, used to set the various modes of the driver, the state of the DTR (Data Terminal Ready) line to the modem, and serial bit rate of the serial interface.

When sub-command 1 is used, the `mode` field in the structure is a bit mask defining which of the driver modes is to be changed, and the `mode_aux` field specifies bits defining the new states of those modes. It is possible to set more than one of the modes by OR'ing the following bit-masks together:

| | |
|---|---|
| 0x0001 | Enables modem support. |
| 0x0002 | Allows modem responses to host - same as the `/Q` option. |
| 0x0004 | Forces direct modem mode. In this mode, the network driver is communicating directly with the modem. |
| 0x0010 | Enables the buffered link protocol and disables the ALERT/ACK link protocol - same as the `/N` option. |
| 0x0020 | Enables the reliable transport protocol. |

The `/M` option corresponds to 0x0021.

Sub-command 2 is used to set the state of the DTR line. In this case, the DTR signal is enabled (on) if the `mode` field is true.

Sub-command 3 is used to set the serial bit rate of the serial interface. The `mode` field determines the new bit rate as follows: 0:14,400; 1:1,200; 2:2,400; 3:9,600; 4:19,200; 5:38,400; 6:57,600; 7:115,200.

## Using the LTS-20 MIP Mode under Microsoft Windows 3.1x

In order to use the MIP mode network driver for DOS under Microsoft Windows 3.1x, an interface based on the DOS Protected Mode Interface (DPMI) must be provided. This type of interface, in the form of Windows DLL software, is supplied with the Connectivity Starter Kit, as well as with the LonManager API for Windows and LonManager DDE Server. The interface software is called WLDV.DLL. See Appendix A for information on using the Windows 3.1x DLL directly.

# 13

# Creating an LTS-20 MIP Mode Driver

This chapter describes the process of building a network driver for a host that is to be connected to an LTS-20 in MIP mode. This chapter also includes a description of the network interface protocol for the MIP mode. The network interface protocol defines the format of the data passed across the EIA-232 interface, and varies depending on the configuration of the LTS-20 and the network driver. If a LONWORKS standard network driver is used, the format of the data passed between the driver and the application is defined by the network driver protocol and is independent of the network interface protocol; the driver is responsible for providing the necessary translations. This chapter will therefore be of interest only to those needing to develop a network driver for a host other than DOS, Windows, or UNIX.

**Skip this Chapter if you are using the LTS-20 NSI mode.**

## Purpose of the Network Driver

The LTS-20 software is shared in common with the SLTA-10 Serial LonTalk Adapter. Any references to the SLTA-10 NSI, including file names, apply to the LTS-20.

The network driver provides a hardware-independent interface between the host application and the network interface. By using network drivers with consistent calling conventions, host applications can be transparently moved between different network interfaces. For example, the standard MIP mode DOS network driver, together with the Windows 3.1x DLL software, allows DOS and Windows 3.1x applications, such as those based on the LonManager API, to be debugged using the network driver for the LonBuilder Development Station. These applications can later be used with the network driver for the SLTA-10/LTS-20-based node, without modifying the host application.

A LONWORKS standard network driver must supply the functions defined under *Network Driver Services* in Chapter 4 of the *LONWORKS Host Application Programmer's Guide*. The Windows 3.1x DLL software is described in Appendix A.

## Example Network Drivers

The Connectivity Starter Kit includes source code for an example DOS network driver; the Echelon web site contains source code for an example UNIX network driver. The DOS driver is used for both DOS and Windows 3.1x applications. See the comments in the source code of the network drivers for an explanation of how the network drivers work. These drivers can be used as templates for a LONWORKS standard network driver. The DOS network driver is compatible with the LonManager APIs for DOS and Windows, the LonManager LonMaker installation tool, and the LonManager DDE Server. A sample host application for DOS is also supplied. The functions `ldv_open()`, `ldv_read()`, `ldv_write()`, and `ldv_close()` form a suitable operating-system independent definition for the network driver. These functions support multiple network interfaces, and hide the DOS-specific aspects of the DOS network driver.

The UNIX network driver is a source library that uses the UNIX serial device driver. It also supports the `ldv_open()`, `ldv_read()`, `ldv_write()`, and `ldv_close()` functions.

## Implementing an LTS-20 MIP Mode Network Driver

The network driver manages the physical interface with the SLTA-10/LTS-20-based node, implements the network interface protocol, performs flow control, manages input and output buffers, and provides a read/write interface to the host application.

Figure 13.1 illustrates how the network driver fits into the host application architecture.

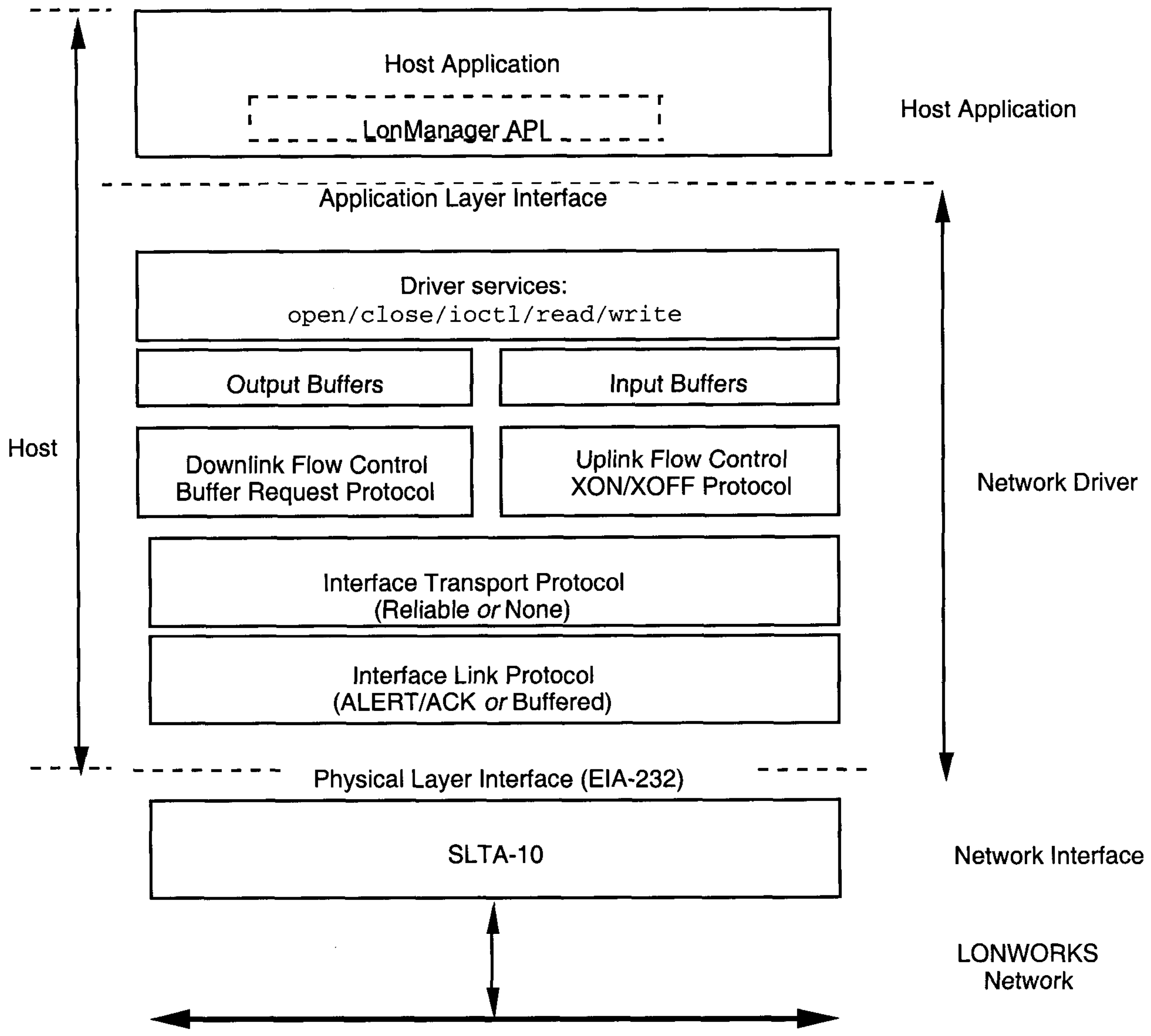


**Figure 13.1** Host Application Architecture

To implement a MIP mode network driver for a host other than DOS, Windows, or UNIX, follow these steps:

1 Implement and test low-level serial I/O. Serial I/O may be performed directly to the host's UART as is done in the DOS network driver, or may be performed by a serial I/O driver on the host as is done by the UNIX network driver. Serial I/O should be interrupt driven for better performance.

The UNIX network driver uses the UNIX serial port driver for all low-level serial I/O and interrupt support. This simplifies the driver and also simplifies porting between different versions of UNIX. The serial device is opened by the `ldv_open()` function and closed by the `ldv_close()` function. Data are read

from and written to the serial device using the UNIX `read()` and `write()` system calls.

The UNIX network driver includes a `ldv_post_events()` function that should be called periodically from the client application in order to assure that the SLTA-10/LTS-20-based node traffic is being processed.

The DOS network driver serial I/O functions are implemented by `MSD_SIO.C`, `MSD_UART.H`, and `MSD_IRQC.ASM`. These files may all be replaced as long as the required serial I/O functions in `MSD_SIO.C` are provided. The definitions of the UART registers are in `MSD_UART.H`. The DOS serial I/O interrupt service routines are in `MSD_IRQC.ASM`.

The DOS network driver uses the DOS system timer tick interrupt (vector 0x1C) and the serial I/O device interrupt for the relevant COM port to perform background processing of the serial network interface. The driver hooks into these interrupt vectors and executes driver code whenever the LON(n) device is opened. Flags internal to the driver prevent the interrupt code thread from interfering with the normal application foreground execution of functions within the driver.

The `smip_int_main()` function in the DOS network driver services the serial port connected to the network interface. The function `tick_int_main()` services the timer tick interrupt every 55 msec.

Both network drivers are fully buffered for both outgoing and incoming messaging. Read and write functions work with circular buffers within the driver. The host interrupt service routine handles the other ends of these buffer queues.

Both network drivers only support a single set of output buffers. An elaboration on this design could implement a set of priority output buffers. The write function could determine into which of the two buffer sets to place messages, and the driver service function could service the priority buffers first.

**2** Implement and test timer support functions. Timer support may be provided by a hardware timer as is done in the DOS network driver, by a system service as is done in the UNIX network driver, or by implementing a background software task. The UNIX network driver uses a once per second signal that is handled by the `second_service()` function. The DOS timer functions are implemented by `MDV_TIME.C` and `MDV_TIME.H`.

**3** Implement and test the host side of the network interface protocol. The network interface protocol is implemented by the `rx_process()` and `tx_process()` functions in the UNIX driver, and by the functions in `MSD_TXRX.C` for the DOS network driver.

**4** Implement and test raw modem I/O if you need to support a modem interface. Raw I/O manages the serial interface to the modem when the modem is not connected to a host and is used for modem initialization and control. The raw I/O interface is implemented in `MSD_RAW.C` for the DOS network driver, and is not implemented in the UNIX network driver.

**5** Implement and test the buffer request states, buffer management, and read/write interfaces. These functions are implemented by `MSD_EXEC.C` for the DOS

network driver. The read/write interface is implemented in the `ldv_read()` and `ldv_write()` functions for the UNIX network driver

The following files are unique to a DOS driver and would probably not be used in a port to another host: `MSD_DRVR.H`, `MSD_DIFC.C`, `MSD_FRST.C`, `MSD_LAST.C`, `MSD_SEGD.ASM`.

## Network Interface Protocol

The network driver implements the host side of the network interface protocol, providing an easy-to-use and interface-independent read/write interface to the host application. The network interface protocol is a layered protocol that includes the following layers:

- **Presentation Layer**. Defines packet formats for network variables and explicit messages. This is the only layer visible to the host application. The remaining layers are managed by the network driver.
- **Session Layer**. Manages flow control, buffer requests, and grants.
- **Transport Layer**. Ensures end-to-end reliability between the host and the SLTA-10/LTS-20-based node.
- **Link Layer**. Controls access to the serial link.
- **Physical Layer**. EIA-232 interface.

The physical layer is defined by the EIA RS-232 standard. The link, transport, session, and presentation layers are described in the following sections.

## Link Layer Protocol

The default interface link layer protocol is the *ALERT/ACK protocol*. This protocol may be used when the host is a microcontroller or microprocessor such as a PC running DOS or Windows. The alternative interface link protocol is the *buffered protocol*. This protocol is used with computer hosts that can asynchronously buffer an entire packet. All data are transmitted using 1 start bit, 8 data bits, no parity bits, and 1 stop bit.

### *ALERT/ACK Link Protocol*

The DOS network driver uses the ALERT/ACK link protocol by default (i.e. the `/N` option is not specified). See Chapter 8 for a description of the network driver options. The UNIX network driver uses the ALERT/ACK link protocol if the `alert_ack_prtcl` variable is set to `TRUE` in the source code (this is not the default). The CFG3 input of the SLTA-10/LTS-20-based node must be in the *ALERT/ACK* state.

When using this protocol, all transfers between the SLTA-10/LTS-20-based node and the host consist of serial data streams that start off with the link-layer header sequence described in figure 15.2. Whenever one device, either the SLTA-10/LTS-20-based node or the host, needs to send a command or message, the sender starts the sequence by transmitting the ALERT byte (value 01 hex).

When this byte is received by the receiver, that device responds by transmitting the ALERT ACK byte (value FE hex). This low level handshaking process prevents the sender from transmitting the rest of the sequence before the receiving device is ready. Once the ALERT ACK byte is received by the sender it sends the rest of the message without any other interactions.

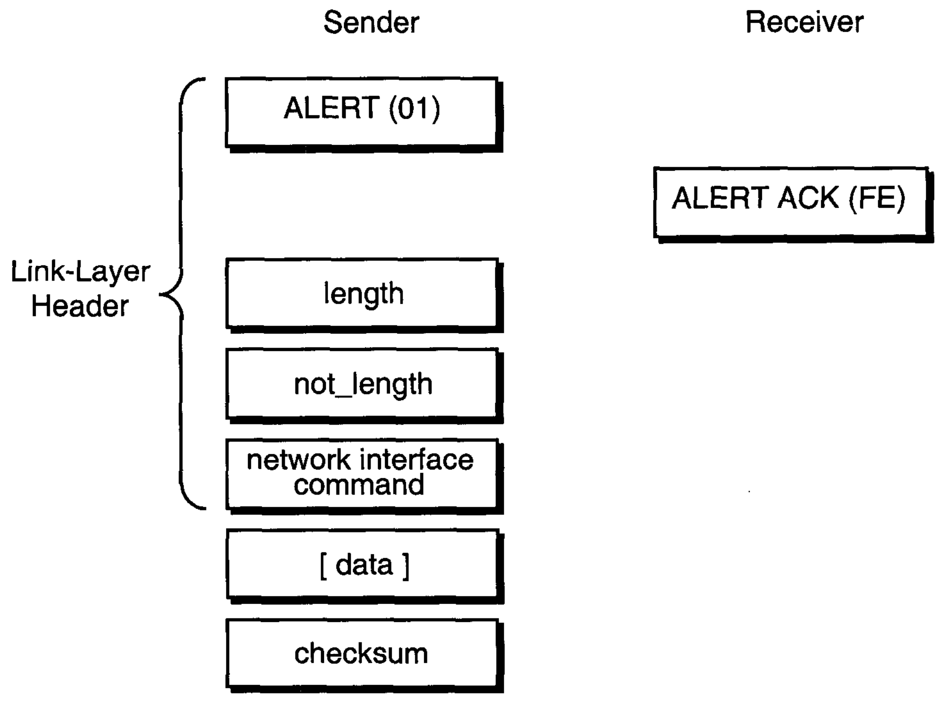


**Figure 13.2** LTS-20 Adapter ALERT/ACK Link Protocol

The link-layer header contains a length byte followed by a one's complement of the length byte. These values are always validated by the receiver before accepting the rest of the message. Following the length bytes is the network interface command. See Appendix D of the *Host Application Programmer's Guide* for a description of the command byte structure. If the message contains a data field it follows the command byte. Finally, a checksum terminates the sequence.

The length byte value describes the length of the network interface command byte plus the length of the data field. This value will always be at least 1. The checksum is a two's complement of the sum of the command byte and all of the bytes in the data field, if it exists. Checksum errors detected by the host will cause an error to be reported to the application, and checksum errors detected by the SLTA-10/LTS-20-based node will cause the message to be ignored.

The SLTA-10/LTS-20-based node places the following requirements on the rate of the received serial data stream. When receiving, the maximum wait period for the length byte following the transmission of the ALERT ACK byte is 100ms (or 1 second when attached to a modem). All subsequent bytes received must occur within 100ms after the previous byte, otherwise the SLTA-10/LTS-20-based node receive process will abort. Likewise, the SLTA-10/LTS-20-based node uses a wait period of 100ms (or 1 second when attached to a modem) before aborting for the reception of the ALERT ACK when transmitting a message. If the ALERT ACK is not received in time, the SLTA-10/LTS-20-based node repeats the process by transmitting another ALERT byte.

The SLTA-10/LTS-20-based node cannot support a full duplex communications process between it and the host. The network driver included with the SLTA-10/LTS-20-based node takes this into account. Data frames transmitted to the SLTA-10/LTS-20-based node while it is in the process of sending uplink messages will be lost if more than 16 bytes are sent to the SLTA-10/LTS-20-based node.

## Buffered Link Protocol

The DOS network driver uses the buffered link protocol when the `/N` option is specified. See Chapter 7 for a description of this option. The UNIX network driver uses the buffered link protocol if the `alert_ack_prtcl` variable is set to `FALSE` in the source code (this is the default). The Switch1/CFG3 input of the SLTA-10/LTS-20-based node, as described in Chapter 4, must be in the buffered protocol state (UP position).

When using this protocol, the link-layer header contains a length byte followed by a one's complement of the length byte. These values are always validated by the receiver before accepting the rest of the message. Following the length bytes is the network interface command. See Appendix D of the *Host Application Programmer's Guide* for a description of the command byte structure. If the message contains a data field it follows the command byte. Finally, a checksum terminates the sequence.

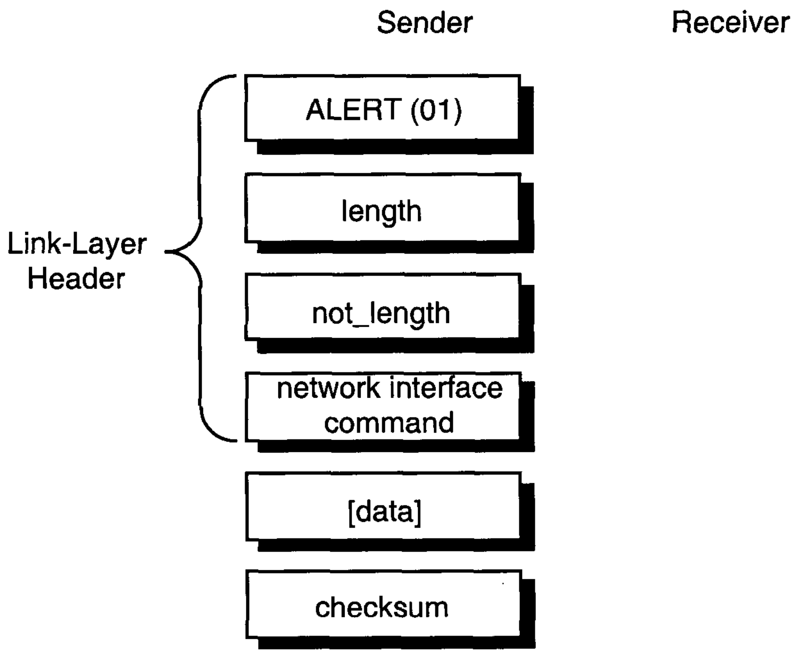


**Figure 13.3** LTS-20 Adapter Buffered Link Protocol

The length byte value describes the length of the network interface command byte plus the length of the data field. This value will always be at least 1. The checksum is a two's complement of the sum of the command byte and all of the bytes in the data field, if it exists. Checksum errors detected by the host will cause an error to be reported to the application, and checksum errors detected by the SLTA-10/LTS-20-based node will cause the message to be ignored.

This protocol is used when the host is capable of accepting asynchronously occurring input data without losing characters. The host is also relieved of the obligation of responding to an ALERT character within 50 ms. This protocol may therefore be used by an application-level handler calling an interrupt-driven buffered serial device driver. Drivers with these characteristics are typically provided with real time operating systems such as VRTX or time-sharing operating systems such as UNIX or VMS. In this case, these drivers should be set up for binary data communications without software flow control.

The buffered link protocol should not be used when the SLTA-10/LTS-20-based node is attached to a modem.

The buffered link protocol can only be used on multitasking operating systems such as UNIX if the host application executes often enough to empty any incoming buffers. For example, if the SLTA-10/LTS-20-based node is receiving 70 packets per second, and each packet is 25 bytes, the host will receive 1750 bytes per second. If the host has a serial input buffer of 256 bytes, the buffer will fill within 150 milliseconds if the host application is preempted. If the host application is preempted for longer than 150 milliseconds, incoming data will be lost due to lack of serial buffer space. In this case, the ALERT/ACK protocol should be used, or the buffer space increased to handle the worst case traffic during the maximum preemption period.

## Transport Layer Protocol

When used with a local host, the SLTA-10/LTS-20-based node assumes a reliable connection and does not use a transport layer protocol. When used with a remote host, the SLTA-10/LTS-20-based node assumes that the link may not be reliable and enables the reliable transport protocol. The reliable transport protocol adds an ACK/NACK transport protocol to the network interface protocol. A sequence number is also added to the link-layer header. This protocol can therefore recover from checksum errors on the host to SLTA-10/LTS-20-based node link.

The reliable transport protocol is enabled on the SLTA-10/LTS-20-based node with the *Remote Host* option selected by the CFG2 input as described in Chapter 4. The reliable transport protocol is enabled on the DOS network driver with the `/M` option as described in Chapter 8. The reliable transport protocol is not supported by the UNIX network driver.

The link-layer header contains an ALERT (0x01) byte, a sequence number, and a length byte followed by a one's complement of the length byte. These values are always validated by the receiver before accepting the rest of the message. Following the length bytes is the network interface command. See Appendix D of the *Host Application Programmer's Guide* for a description of the command byte structure. If the message contains a data field it follows the command byte. Finally, a checksum terminates the sequence.

The ALERT/ACK link protocol should be used with remote hosts. With this protocol, the sender will start the sequence by transmitting the ALERT byte. When this byte is received by the receiver, that device responds by transmitting the ALERT ACK byte (value FE hex). This low level handshaking process prevents the sender from transmitting the rest of the sequence before the receiving device is ready. Once the ALERT ACK byte is received by the sender it sends the rest of the message without any other interactions.

The length byte value describes the length of the network interface command byte plus the length of the data field. This value will always be at least 1. The checksum is a two's complement of the sum of the command byte and all of the bytes in the data field, if it exists. If the receiver receives a message in sequence, with a valid checksum, it responds with an ACK (0x06). Otherwise it responds with a NACK (0x15), requesting a re-transmission.

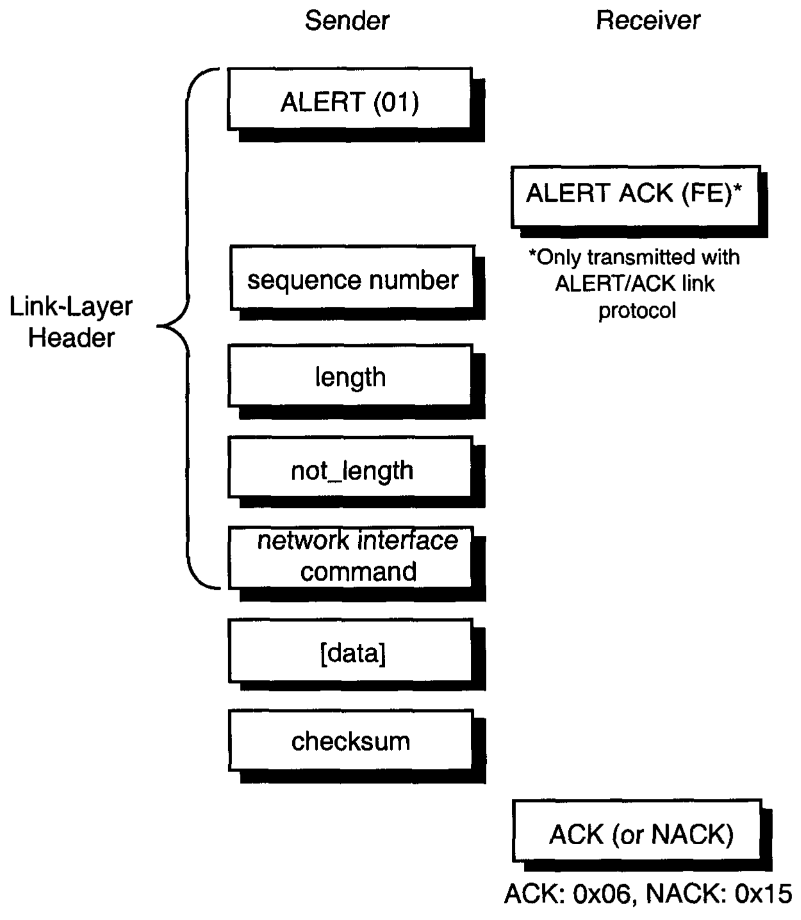


**Figure 13.4** LTS-20 Adapter Reliable Transport Protocol

## LTS-20 Timing Data

Certain aspects of the SLTA-10/LTS-20-based node link and transport layer protocols implement fail-safe timeouts in order to control the time spent waiting for protocol states to change when errors occur.

### *Downlink Byte-to-Byte Receive Timeout*

The downlink byte-to-byte receive timeout is the maximum allowable period between the end of a single byte data frame sent downlink to the SLTA-10/LTS-20-based node, to the end of the next single byte data frame sent downlink to the SLTA-10/LTS-20-based node. This period is 100ms in local host mode and 1 second in remote host mode. When this timeout occurs, the SLTA-10/LTS-20-based node discards the downlink buffer and returns to the NORMAL state. If the reliable transport protocol is enabled, the SLTA-10/LTS-20-based node also sends a NACK byte after this timeout.

### *Uplink Message Life*

The uplink message life is the maximum allowable period between the SLTA-10/LTS-20-based node sending an ALERT byte to the host and the host responding with an ALERT ACK byte. This period is 100ms in local host mode and 1 second in remote host mode. When this timeout occurs, the SLTA-10/LTS-20-based nodewill resend the ALERT byte. This process is repeated until 3 seconds have elapsed, after which the uplink message is discarded. This timeout only applies to the ALERT/ACK link protocol and is not used for the buffered link protocol.

### *ACK/NACK Receive Timeout*

When using the reliable transport protocol, the SLTA-10/LTS-20-based node will wait for the ACK or NACK byte to be sent downlink following the end of the uplink transmission of a message. This period is 1 second, after which the SLTA-10/LTS-20-based node will re-send the uplink message.

### *Uplink Timeout Message Retry Count*

When using the reliable transport protocol the SLTA-10/LTS-20-based node will re-send uplink messages whenever the ACK/NACK timeout period has elapsed. This retry process is limited to 5 retries, after which the uplink message is discarded. There is no retry limit applied to re-sends due to the reception of the NACK byte.

## Session Layer Protocol

The network interface link and transport protocols described above are used for all host-to-SLTA/LTS communications. Layered on top of these protocols is a downlink buffer request protocol and an uplink flow control protocol.

### *Downlink Buffer Request Protocol*

The network driver receives application buffers from the host application, translates them to interface buffers, and passes the interface buffers to the SLTA-10/LTS-20-based node. There are two types of downlink commands from the host to the SLTA-10/LTS-20-based node—commands that can be executed directly by the SLTA-10/LTS-20-based node, and commands that need to be buffered in the SLTA-10/LTS-20-based node.

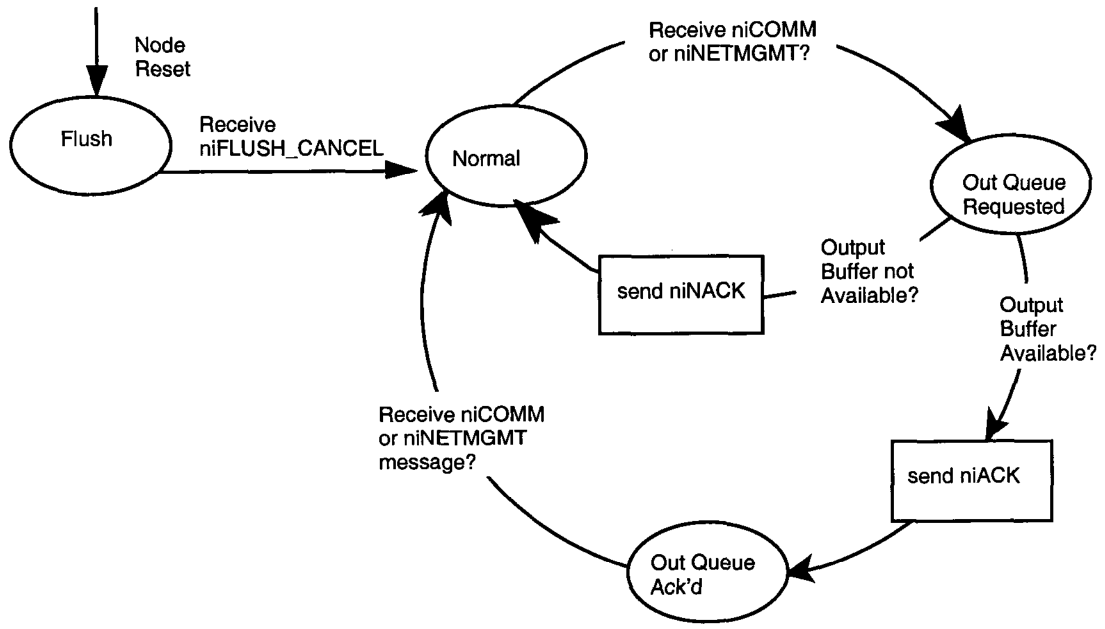
Downlink commands that are executed directly by the SLTA-10/LTS-20-based node are:

niRESET, niFLUSH_CANCEL, niONLINE, niOFFLINE, niFLUSH, niFLUSH_IGN, niPUPXOFF, niPUPXON, niSLEEP, and niSSTATUS.

See the *Host Application Programmer's Guide*, Appendix D, for a description of these commands.

The niSStatus command, when sent downlink, will cause the SLTA-10/LTS-20-based node to respond with a niSStatus command plus one byte of data. In MIP mode, this byte of data contains the settings of configuration switches, with BAUD0 being the least significant bit. In NSI mode, this byte of data contains 011 in the least significant bits followed by the XID information, making the SLTA-10/LTS-20-based node NSI mode consistent with the PCNSI Adapter.

Downlink commands that are buffered in theSLTA-10/LTS-20-based node are `niNETMGMT` (for network management commands to be executed by the SLTA-10/LTS-20-based node itself) and `niCOMM` (for messages to be sent out on the network, including network variables, explicit messages, and network management messages addressed to other nodes). For these two commands, a buffer request protocol is used to ensure that the SLTA-10/LTS-20-based node has a free application buffer for the data. The network driver must first request an output buffer before sending the interface buffer. The network driver must hold the buffers in an output queue until the SLTA-10/LTS-20-based node is ready to receive them. The network driver takes theSLTA-10/LTS-20-based node through 3 states to request a buffer and send the interface buffer. Figure 13.5 summarizes the downlink state transitions.



*Note: niNETMGMT commands are allowed in the Flush state.*

**Figure 13.5** LTS-20 Adapter Downlink Flow Control States

Following is the sequence of events for transferring an `niCOMM` or `niNETMGMT` command downlink to the SLTA-10/LTS-20-based node:

1 The SLTA-10/LTS-20-based node is initially in the `NORMAL` state.

2 The network driver requests an output buffer by sending a link-layer header (see figures 15.2 and 15.3) with a `niCOMM` or `niNETMGMT` command and the appropriate queue value (`niTQ`, `niTQ_P`, `niNTQ`, `niNTQ_P`). The data portion of the interface buffer is not sent with the buffer request. This puts theSLTA-10/LTS-20-based node in the `OUTPUT QUEUE REQUESTED` state.

3 If an output buffer is not available, the SLTA-10/LTS-20-based node responds with a `niNACK` (0xC1) command. The SLTA-10/LTS-20-based node returns to the `NORMAL` state, and the driver starts again at step 2.

4 When an output buffer is available, the SLTA-10/LTS-20-based node responds with a `niACK` (0xC0) command. The SLTA-10/LTS-20-based node is now in the `OUTPUT QUEUE ACKNOWLEDGED` state. While in this state, the network driver can only transfer downlink LonTalk messages, uplink source quench commands (`niPUPXOFF`), uplink source resume commands (`niPUPXON`), or reset commands (`niRESET`) since the SLTA-10/LTS-20-based node is waiting for a message in this state. All other network interface commands sent downlink will be ignored, and will return the SLTA-10/LTS-20-based node to the `NORMAL` state.

5 Upon receiving the `niACK` acknowledgment, the network driver transfers the entire interface buffer to the SLTA-10/LTS-20-based node. This buffer has the

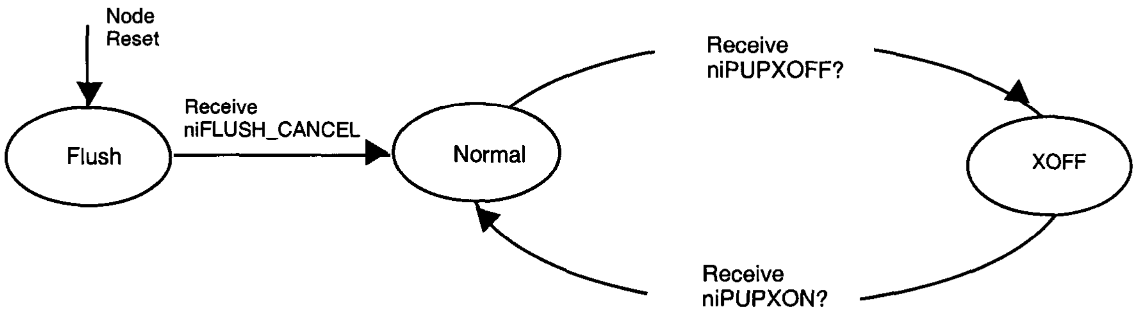
same command and queue value sent in step 2, and also contains the data and checksum. Upon completion of this transfer, the SLTA-10/LTS-20-based node returns to the `NORMAL` state.

The network driver must preserve the continuity of the type of buffer request and the type of message sent downlink. For example, if the network driver sends the `niCOMM+niTQ_P` command requesting a priority output buffer, and follows this with a message transfer with the non-priority `niCOMM+niTQ` command, theSLTA-10/LTS-20-based node will incorrectly store the message in a priority output buffer, the type originally requested.

## *Uplink Flow Control Protocol*

Uplink traffic may be incoming LonTalk messages, output buffer request acknowledgments, completion events, or local commands. The network driver translates the interface buffers to application buffer format and stores the buffers in a queue until the host application is ready to read them.

There is no buffer request protocol for uplink traffic. The network driver is normally assumed to have sufficient buffers. The network driver can suspend or resume uplink traffic when no network driver input buffers are available by sending the *Uplink Source Quench* (`niPUPXOFF`) command to the SLTA-10/LTS-20-based node. This prevents the STLA from sending any LonTalk messages uplink. When the network driver senses that network driver input buffers are available, it sends the *Uplink Source Resume* (`niPUPXON`) command to resume uplink transfers. Figure 13.6 summarizes the uplink state transitions.



*Note: Responses to niNETMGMT and niSSTATUS commands are allowed in the Flush state.*

**Figure 13.6** LTS-20 Uplink Flow Control States

The host may chose to sidestep the downlink buffer request protocol. In this case, the complete message is sent downlink without any buffer request step. If the SLTA-10/LTS-20-based node has a free output buffer, then the message will be transferred into the SLTA-10/LTS-20-based node successfully. If not, there will be no indication and the message will be lost. The exception to this case is when using the transport layer protocol, in which case the SLTA-10/LTS-20-based node will send the NACK to the host, which should force the host to re-send the message. Otherwise, in order to use this feature successfully, the host driver must manage the number of available output buffers within the SLTA-

10/LTS-20-based node. This feature has been added to the DOS driver for the SLTA-10/LTS-20-based node.

## Presentation Layer Protocol

The network driver exchanges LonTalk packets with the host application at the presentation layer. The LonTalk packet enclosed in a command of type `niCOMM` or `niNETMGMT` is described in detail in the *Host Application Programmer's Guide*. It is summarized here for convenience.

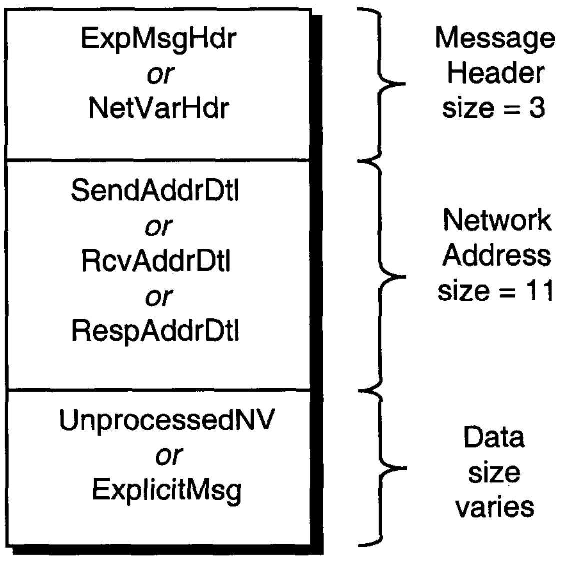


**Figure 13.7** Application Buffer Format

The SLTA-10/LTS-20-based node firmware is configured with explicit addressing enabled, and therefore the 11-byte network address field is always present in an uplink or downlink buffer. The firmware is also configured with host selection enabled, so the data field of the buffer is either an unprocessed network variable or an explicit message.

# 14

# Initialization and Installation

This chapter describes initializing, communicating with, and installing the LTS-20 as a network node.

## Initializing an LTS-20-based Node

**The LTS-20 software is shared in common with the SLTA-10 Serial LonTalk Adapter. Any references to the SLTA-10 NSI, including file names, apply to the LTS-20.**

After an SLTA-10/LTS-20-based node and its host processor are powered, the host application must initialize the SLTA-10/LTS-20-based node. When an SLTA-10/LTS-20-based node is initially powered-up or reset, it disables network communications by entering the FLUSH, unconfigured state, unless the CFG1 input is set to *Network Enable*. The FLUSH state prevents the SLTA-10/LTS-20-based node from responding to network management messages before the host application has initialized the SLTA-10/LTS-20-based node. The unconfigured state prevents the SLTA-10/LTS-20-based node from responding to application messages before the node has been installed in a network. The host application will not be able to send or receive application messages until after it cancels the `FLUSH` state and the SLTA-10/LTS-20-based node leaves the unconfigured state.

An SLTA-10/LTS-20-based node leaves the unconfigured state when it is installed in a network and assigned an address in one or two domains on that network. There are two ways this can happen. If the SLTA-10/LTS-20-based node is used as a network interface for a network management tool, then the network management application itself normally configures the network interface. For example, a network management application based on LNS will configure the network interface (in this case an SLTA-10/LTS-20-based node in NSI mode) when the network interface device is opened. Also, a network management application based on the LonManager API or the LonMaker installation tool will configure the network interface when the network interface device is opened. In this case the PC running the network management application sends a local message to the SLTA-10/LTS-20-based node to change its state to configured.

Alternatively, the SLTA-10/LTS-20-based node may be installed in a network by some other network management tool. In this case, the network management tool sends a message to the SLTA-10/LTS-20-based node across the network to change its state and to assign it an address.

Once theSLTA-10/LTS-20-based node is in the configured state, it will retain that state and its address assignment across power-cycles, because that information is stored in the internal EEPROM. However, if CFG1 is set to *Network Disable*, the `FLUSH` state must be canceled each time the SLTA-10/LTS-20-based node is reset.

If the host application attempts to send a message while the SLTA-10/LTS-20-based node is in the `FLUSH` state, the SLTA-10/LTS-20-based node will return a failed response for acknowledged messages and a success response for unacknowledged messages. The message will not be sent in either case.

If the host application attempts to send a message while the SLTA-10/LTS-20-based node is in the `NORMAL`, unconfigured state, the SLTA-10/LTS-20-based node will always return a success response even though the message will not be sent.

To initialize an SLTA-10/LTS-20-based node, follow these steps (for more detail, see the *LONWORKS Host Application Programmer's Guide*):

1 Reset the SLTA-10/LTS-20-based node from the host application by sending the `niRESET` command. If installed correctly, the SLTA-10/LTS-20-based node will respond with an uplink `niRESET` message upon completion of the reset. The first message from an SLTA-10/LTS-20-based node after power-up will also be an uplink `niRESET` message informing the host that the SLTA-10/LTS-20-based node has reset.

2 Cancel the `FLUSH` state in the SLTA-10/LTS-20-based node. LNS applications automatically handle this, and it is done automatically by the MIP mode DOS network driver after an open command or uplink `niRESET` if the `/Z` flag is not specified. The `FLUSH` state can be manually canceled by sending the `niFLUSH_CANCEL` message. The `FLUSH` state can also be disabled by a configuration switch on the SLTA-10/LTS-20-based node as described earlier in this chapter.

3 Install the network interface in one or two domains using the *Update Domain* network management message. This may be done by a network management tool across the network, or may be done directly by the host application by sending the *Update Domain* message as a local network management command.

4 Change the state of the network interface to configured. This may be done by a network management tool across the network, or may be done directly by the host application by sending the *Set Node Mode* network management message.

## Installing an LTS-20-based Node on a Network

An SLTA-10/LTS-20-based node attached to a network appears as a standard LONWORKS node to other nodes on the network. The SLTA-10/LTS-20-based node node is logically installed on a network with a network management tool. Installation scenarios are described in the *LONWORKS Installation Overview* engineering bulletin (number 005-0006-01). Unique installation requirements of host applications are described in Chapter 3 of the *LONWORKS Host Application Programmer's Guide.*

## *Installing with LNS, the LonMaker for Windows Integration Tool, or the LNS DDE Server*

With an LNS-based application, the SLTA-10/LTS-20-based node is initialized automatically as part of the system open. Prior to the system open, the application selects the desired network interface. See the Chapter 3 *Initializing and Terminating LCA Applications* in the *LCA Object and Data Server Programmer's Guide* for code fragments.

## *Installing with the LonBuilder Tool*

An SLTA-10/LTS-20-based node node can be installed on a development network using the LonBuilder Network Manager. Chapter 6 of the *LonBuilder User's Guide* describes how to define and install nodes in a development network using the LonBuilder Network Manager. A prerequisite to creating application node target hardware and node specifications is to define the channels that will be included in the network as defined under *Defining Channels* in Chapter 10 of the *LonBuilder User's Guide*. Select the standard transceiver type compatible with the transceiver on your SLTA-10/LTS-20-based node node.

When creating a hardware properties definition for a custom node to represent the SLTA-10/LTS-20-based node, set the input clock rate to 10MHz. Then create a hardware definition for the custom node specifying these hardware properties. The SLTA-10/LTS-20-based node will not accept an incorrect clock rate or hardware properties. This may occur, for example, if you specify `default_hw_props` instead of the correct one. To install the SLTA-10/LTS-20-based node in a LonBuilder network using the service pin, you must either connect the SLTA-10/LTS-20-based node to a host, and open the network driver (for example, by running `HA`), or else set CFG1 to the *Network Enable* state.

When installing the SLTA-10/LTS-20-based node, the channel definition must match the transceiver on the SLTA-10/LTS-20-based node. If it does not, the SLTA-10/LTS-20-based node will not accept the new values. A 'No' response is required to the prompt, `Do you want to install communications parameters?` *DO NOT use the Yes response to the prompt:* *`Do you want to install communications parameters?`* *unless the channel and hardware definitions are compatible with the transceiver and input clock on the SLTA-10 Adapter.*

When defining the application image, the `App Image Origin` field should be set to `Interface File`, and the `App Image Name` should be set to the name of an external interface file (XIF)created as described under *Binding to a Host Node* in Chapter 3 of the *LONWORKS Host Application Programmer's Guide.*

*WARNING: This is NOT the default setting for the* *`App Image Origin`*. *If you specify a Neuron C source file as the* *`App Image Origin`*, *the SLTA-10 Adapter may be rendered unusable.*

If the SLTA-10/LTS-20-based node is accidentally configured with the wrong communication parameters, it may be rebooted with the `NODEUTIL` application available on Echelon's web site. See the `README.TXT` file included with `NODEUTIL` for a description.

## *Installing an LTS-20-based Node with LonManager API, the DOS-based LonManager LonMaker for DOS Installation Tool, or the LonManager DDE Server*

When an SLTA-10/LTS-20-based node is used as a network interface for a host application based on the LonManager API, these installation steps are automatically handled by the LonManager API `lxt_open()` function call.

When an SLTA-10/LTS-20-based node is used as a network interface for a network management PC running LonMaker for DOS installation tool or a LonManager API-based application, these installation steps are automatically handled by the LonMaker for DOS `Attach` command or the LonManager API `lxt_open()` function call.

When an SLTA-10/LTS-20-based node is used as a network interface for a control and monitoring PC running the LonManager DDE Server, these installation steps are automatically handled by the LonManager DDE Server.

# 15

# Using the LTS-20 with a Modem

This chapter describes the operation of the LTS-20-based node when a remote host computer is connected via a pair of modems to the LTS-20-based node. In this set-up, any node on the network may request the LTS-20-based node to initiate a dial-out operation to connect to the host. In addition, if the LTS-20 is functioning in NSI mode, the LTS-20-based node itself may be configured to initiate a dial-out operation to connect to the host. The LTS-20 can store a telephone directory of commonly called numbers, as well as accept commands to dial any other number.

The LTS-20 may also be configured to accept incoming calls and connect the network to the host. Incoming callers may optionally be required to provide a password before the LTS-20-based node will connect them to the network.

See Chapter 11, *Using the Windows 95 and NT Drivers and SLTALink Manager with LTS-20 NSI Mode*, for configuring the LTS-20 NSI mode EEPROM through the SLTALink Manager application.

## Overview

**The LTS-20 software is shared in common with the SLTA-10 Serial LonTalk Adapter. Any references to the SLTA-10 NSI, including file names, apply to the LTS-20.**

The SLTA-10/LTS-20-based node may be attached to the host processor using modems and the switched telephone network. Figure 15.1 illustrates this option.

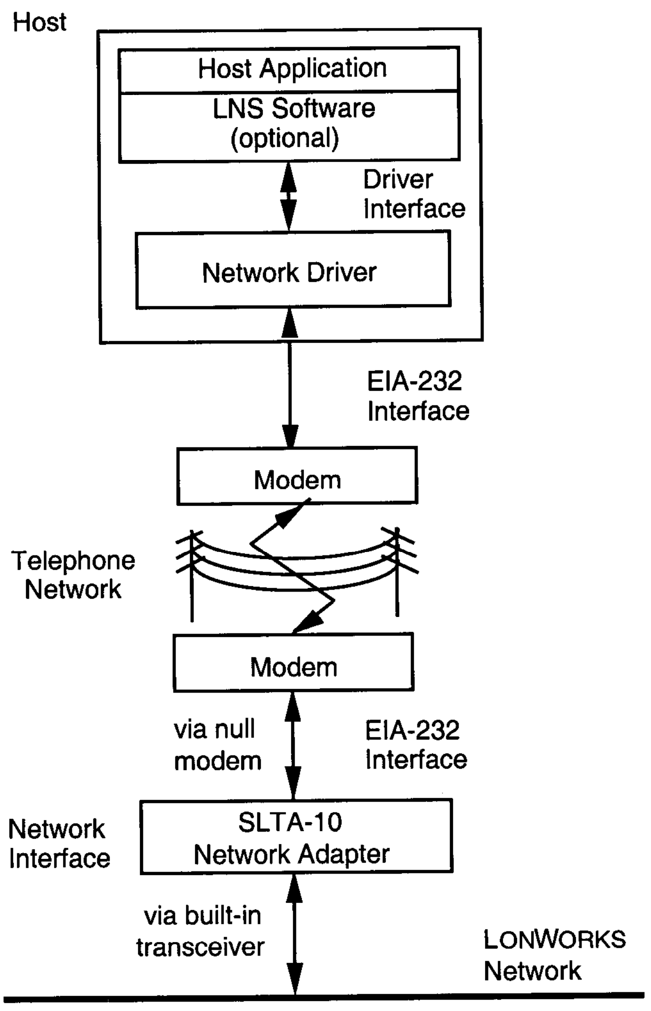


**Figure 15.1** Using the SLTA-10/LTS-20 with Modems

When the SLTA-10/LTS-20-based node is configured for modem support, it will respond to special network management messages that cause it to communicate with the attached modem. The modem used must support the Hayes standard AT command set allowing it to dial-out, auto-answer incoming calls, and set various parameters.

Once a connection is established, the host computer communicates transparently with the network as though the LTS-20 were attached locally.

In NSI mode, the SLTA-10/LTS-20-based node can be configured to initiate the connection with the host PC when LNS network management messages and/or network variable updates are addressed to the SLTA-10/LTS-20-based node. Alternatively, another node on the local network can command the SLTA-10/LTS-20-based node to initiate the connection.

In MIP mode, the SLTA-10/LTS-20-based node cannot itself initiate any connection; it must be commanded to do so by another node on the local network, or else by the modem's detection of an incoming call. This means, for example, another node on the local network must initiate the dialing procedure when an alarm is detected that needs to be reported to the host. Once a connection has been established, however, any node on the local network can communicate with the host by addressing messages to the SLTA-10/LTS-20-based node.

In order to support the modem functions, the CFG2 input must be set to the *Remote Host* state. CFG3 should be set to the default ALERT/ACK link protocol. This automatically enables the reliable network interface transport protocol. See Chapter 2 for details of the configuration inputs. See Chapter 11 for using the SLTALink Manager with the LTS-20 NSI mode or Chapter 7 for LTS-20 MIP mode DOS network driver options.
Note that the packet throughput of the SLTA-10/LTS-20-based node is substantially reduced when using a modem because of the overhead associated with modem support.

## LTS-20 Connection States

When the SLTA-10/LTS-20-based node is operating in the remote host mode, several internal states (or *connection states*) will control its behavior:

- The `IDLE` state is entered after power-up reset. In this state any uplink bound messages are ignored since the SLTA-10/LTS-20-based node is not connected to a host. The `IDLE` state is also entered whenever the telephone connection is broken and the modem drops the DCD (Data Carrier Detect) line.
- The `CALL_IN_PROCESS` state is entered once a connection is initiated by a node on the network connected to the SLTA-10/LTS-20-based node. In this state uplink traffic is still discarded while the SLTA-10/LTS-20-based node monitors the modem for connection completion or connection failed events to occur.
- The `CONNECTED` state is entered once the connection is complete. The normal network interface protocol resumes between the SLTA-10/LTS-20-based node and the remote host. This state may be entered as a result of a node on the local network initiating a call, or as a result of a remote host calling up this SLTA-10/LTS-20-based node.
- The `FAILED` state is entered if the connection process failed. This state is operationally the same as `IDLE`.

The connection state of the SLTA-10/LTS-20-based node is preserved across software resets, allowing normal network management resets to occur without breaking the connection. The SLTA-10/LTS-20-based node will not preserve the connection state after it has been through a power reset.

## Command Set Assumptions

The SLTA-10/LTS-20-based node uses the following strings received from the modem to interpret the connection state. These strings are consistent with all Hayes AT compatible modems operating in the word response mode (alphabetic responses). [CR] is the hex 0D character.

```
CONNECT [any text] [CR]
BUSY [CR]
VOICE [CR]
NO [any text][CR]
ERROR [CR]
```

The "CONNECT" string may be, and typically is, followed by other informative text, such as connection serial bit rate or error correction methods in use.
These other four states indicate a failure to make the connection.

## Translated Characters

All strings that are sent specifically to the modem (as commands) by the SLTA-10/LTS-20-based node are scanned for certain characters by the SLTA-10/LTS-20-based node firmware. These characters are then translated into specific functions or characters unless they are preceded by a backslash ("\"). The characters are:

| | |
|---|---|
| ~ | The tilde will cause the SLTA-10/LTS-20-based node to pause 500ms before sending the next character to the modem. The tilde itself is not sent. |
| ! | The exclamation point will cause the SLTA-10/LTS-20-based node to send a carriage return (0x0H) to the modem. The exclamation point itself is not sent. |

The tilde is provided for users familiar with existing modem packages. It can be used, for example, in the command string which causes the modem to hang-up. When dialing out, Hayes AT-compatible modems use the comma character to insert delays between portions of a dial command; the comma character should continue to be used for dialing out.

## DTE Connections

In addition to the basic three wire connections—Transmit Data (TXD), Receive Data (RXD), and Signal Ground— there are two additional signals that must be connected: Data Carrier Detect (DCD) and Data Terminal Ready (DTR). The modem must also be configured to use these signals. DCD is used by the SLTA-10/LTS-20-based node to determine that a connection has been made and DTR is used to terminate a connection by hanging up. Note that many modems default to ignore these two signals and must be configured to enable them. The following AT command enables these two signals on many modems.

```
AT&C1&D2[CR]
```

## Network Management Messages

Network management messages are used to configure the operation of the network, as opposed to delivering application data during operation of the network. All LONWORKS nodes respond to the standard network management messages as described in Appendix B of the *Neuron Chip Data Book*. For nodes based on the SLTA-10/LTS-20, additional network management messages are defined to configure and control the SLTA-10/LTS-20-based node. The message codes for network management messages are reserved, and therefore applications need not be concerned about possible conflicts in the choice of message codes. The additional SLTA-10/LTS-20-based node network management messages use the reserved message code 7D hex. The first data byte is used as a sub-code, and is 01 hex to indicate an SLTA-10/LTS-20-based node function. The second data byte is a specific application command code. Table 15.1 summarizes the network management messages specific to the SLTA-10/LTS-20-based node.

**Table 15.1** LTS-20 Network Management Messages

| ***Message Code*** | ***Message Sub-code*** | ***Application Command*** | ***Function*** |
|---|---|---|---|
| 0x7D | 1 | 1 | Product Query |
| 0x7D | 1 | 2 | Send Modem String |
| 0x7D | 1 | 3 | Modem Status Readback |
| 0x7D | 1 | 4 | Modem Response Query |
| 0x7D | 1 | 5 | Connection Status Query |
| 0x7D | 1 | 6 | Install Directory Entry |
| 0x7D | 1 | 7 | Dial From Directory |
| 0x7D | 1 | 8 | Hang-up |
| 0x7D | 1 | 9 | Install Password |
| 0x7D | 1 | 10 | Install Modem Configuration String |
| 0x7D | 1 | 11 | Install Hangup String |
| 0x7D | 1 | 12 | Install Dial Prefix |
| 0x7D | 1 | 13 | Install Hangup Timer |
| 0x7D | 1 | 14 | Configure Modem |
| 0x7D | 1 | 15 | Request/Release Modem |
| 0x7D | 1 | 16 | Clear EEPROM Pool |
| 0x7D | 1 | 17 | Install NV Connect |
| 0x7D | 1 | 18 | Install NSI Connect |
| 0x7D | 1 | 19 | Install Callback Enable |
| 0x7D | 1 | 20 | Report SLTA EEPROM contents |

These network management messages may be sent from any node on the network to the SLTA-10/LTS-20-based node. If an application node wishes to send modem-control network management messages to the SLTA-10/LTS-20-based node, it does so using explicit messaging. See Chapter 4 of the *Neuron C Programmer's Guide* for details on explicit messaging. The message should be delivered using request/response service, and the message code for modem control messages is always 7D hex. The data portion of the message always begins with a sub-code value of 1, indicating that the message is addressed to an SLTA-10/LTS-20-based node, followed by an application command byte indicating which modem-control command this is. This is followed by parameters specific to the application command. These messages may be sent by a Neuron C-based node, a node based on the Microprocessor Interface Program (MIP), or a node based on another SLTA-10/LTS-20-based node.

See the supplied Neuron C program `DIALOUT.NC` for an example of an application that sends a message to an SLTA-10/LTS-20-based node to cause it to dial-out using an attached modem. The example `GIZSETUP.NC` sends messages to an SLTA-10/LTS-20-based node to configure its modem strings using a Gizmo I/O module as the user interface. A node can send these messages using either implicit or explicit addressing. When implicit addressing is used, a network management tool binds the output message tag in the node wishing to send the message to the SLTA-10/LTS-20-based node as the destination. When explicit addressing is used, the node wishing to send the message must explicitly insert the destination address in the outgoing message.

Note that the SLTA-10/LTS-20-based node will be unable to receive any messages, including these network management messages, if network communications is disabled, i.e., it is in the `FLUSH` state. This will normally be the case if it is not connected to a host. Therefore, in order to be able to dial-out and connect to a host, the SLTA-10/LTS-20-based node must be configured to initialize with network communications enabled, i.e., the `NORMAL` state, with the CFG1 input. See Chapter 4 for more details.

The messages should be sent with request/response service, and the response code should be checked to see if the message was executed correctly - it will be 0x3D if there was no error, and 0x1D if there was. Possible failure conditions are noted under each message. Some of these network management messages return data to the sender in the response structure as noted below.

Note that some of these messages cause one or more bytes of EEPROM in the SLTA-10/LTS-20-based node to be written. Each byte of EEPROM takes 20ms to write, and the response is not sent by the SLTA-10/LTS-20-based node until after the command is executed. This time should be taken into account when setting the transaction timer (`tx_timer`) in the message tag connection for implicit addressing, or in the destination address for explicit addressing. If the LonBuilder or LonManager API binder is used to create the connection, the transaction timer may be explicitly specified.

Example:

```
msg_tag SLTA_tag;
when ( ... ) {
   msg_out.code = 0x7D;          // network mgmt msg code
   msg_out.tag = SLTA_tag;
   msg_out.service = REQUEST;
   msg_out.data[0] = 1;          // sub-code for SLTA-10
   msg_out.data[1] = app_command; // specific command
```

```
        msg_out.data[2] = .....         // additional parameters
        msg_send();
    }
    when (msg_fails(SLTA_tag)) {
            // SLTA-10 Adapter did not respond to the message
            ...
    }

    when (msg_succeeds) { ... }

    when (resp_arrives(SLTA_tag) {
            // SLTA-10 Adapter did respond to the message
            if (resp_in.code == 0x3D)
                    // command executed successfully
                    ...
```

Certain SLTA-10/LTS-20-based node network management messages which are designed to send a command string directly to the modem will fail if the connection status is CONNECTED. This applies to the following messages: *Send Modem String*, *Dial From Directory*, and *Configure Modem*. The telephone connection must be broken and the SLTA-10/LTS-20-based node returned to the IDLE state for these messages to be issued.

Many of these network management messages, for example the "Install..." messages, may be sent to the SLTA-10/LTS-20-based node from the host computer via the telephone link if the SLTA-10/LTS-20-based node is in the CONNECTED state. In this case, they should be sent using the niNETMGMT network interface command so that they are addressed to the SLTA-10/LTS-20-based node itself.

Structures are defined in the file SLTA_ANM.H for each of the application commands, and include the sub_code and app_command fields, as well as any additional parameters for the specific application command. If the response contains data other than the response code, a structure for the returned data is also defined.

## EEPROM String Pool Management

The SLTA-10/LTS-20-based node's EEPROM is used to store a number of varying-length configurable strings which are used to control the modem. These strings can be set by sending network management messages to the SLTA-10/LTS-20-based node, either from the modem side or the network side. Since the strings are varying in length and the EEPROM space is limited, a pool management system is used to optimize the usage of EEPROM. **The MIP mode EEPROM string pool and the NSI mode EEPROM pool are different! Changing the mode and cycling power will destroy the EEPROM pool.**

For the MIP mode, the EEPROM pool consists of the following strings:

- Up to eight dial-out directory entries. These may be used by index to initiate a dial-out connection, and may contain any combination of AT commands and numbers.
- One modem initialization string. This string is used to initialize the modem as required.
- One modem hang-up string. This string is used to hang-up the modem and break the connection when DTR control does not function.
- One dial-out prefix string. This string is sent as a prefix to any dial-out operation to specify the modem dial command and to indicate whether tone or pulse dialing should be used.
- An 8-byte dial-in password string.

EEPROM storage and allocation for these strings is managed by the MIP mode EEPROM pool. This allows flexible utilization of the MIP mode EEPROM space. The MIP mode pool consists of 21 blocks, each with 9 bytes of data storage space. A string occupies one or more blocks.

For the NSI mode, the EEPROM pool consists of the following strings:

- Up to five dial-out directory entries. These may be used by index to initiate a dial-out connection, and may contain any combination of AT commands and numbers.
- One modem initialization string. This string is used to initialize the modem as required.
- One dial-out prefix string. This string is sent as a prefix to any dial-out operation to specify the modem dial command and to indicate whether tone or pulse dialing should be used.
- An 8-byte dial-in password string.
- A 1-byte code that enables an auto dial-out on a network variable message.
- A 1-byte code that enables an auto dial-out on a NSI message.
- A 1-byte code that enables callback.

EEPROM storage and allocation for these strings is managed by the NSI mode EEPROM pool. This allows flexible utilization of the NSI mode EEPROM space. The NSI mode pool consists of 8 blocks, each with 12 bytes of data storage space. A string occupies one or more blocks.

The NSI mode EEPROM pool does not require the exclamation point (translated to a carriage return) in the dial directories; whereas, it is required in the MIP mode EEPROM pool. In addition, the NSI mode EEPROM pool does not require a null terminator on exact sector size strings. In the MIP mode EEPROM pool, if a string winds up ending on the last byte of a sector, a subsequent sector is required to hold the null terminator.

The network management functions that install these strings allow incremental EEPROM writes, and include a `total_size` field. Incremental writes allow long strings to be installed without the requirement of large buffer sizes on the SLTA-10/LTS-20-based node. If the `total_size` field is greater than the amount of EEPROM storage space available then the network management message response will indicate a failed status. In all cases the strings should include a null terminator.

The `offset` field in the install messages indicates the starting point in the complete string of the current string piece to be installed. The first byte of the complete string is at offset zero. If the offset is such that the string piece would not fit in the allocated total size for the string, the total size for that string is automatically extended, if space is available.

All pieces of a string should specify the same value for `total_size`. Otherwise it means that you are starting over with a new string.
Once a particular string has been installed in the pool, it may be deleted by re-installing it with a total size of 0. Alternatively, the *Clear EEPROM Pool* message may be issued to clear the whole EEPROM string pool.

## Product Query

This message may be used to check the type of interface product that the node is running. The value returned for the SLTA-10/LTS-20-based node is a '1'. Application nodes will respond with a failed code of 0x1D returned for this or any message where the request message code is 0x7D.

```
typedef struct  {
      byte sub_code;                  // always #1
      byte app_command;         // value = 1
} ANM_product_query_request;

typedef struct {
      byte product;                   // for SLTA-10 Adapter, = 1
} ANM_product_query_response;
```

## Send Modem String

This message is typically used to send AT command character strings to the modem from another node on the local network. This provides full control of the modem and its internal control registers and settings. Normal use of this feature is to dial-out to a number that is not in the telephone directory of the SLTA-10/LTS-20-based node, or to change or set modem control registers and options. This puts complete control of the modem in the hands of any application node on the network. The message definition is:

```
typedef enum {
      NO_CHANGE          = 0,
      DIAL_OUT                 = 1,
      HANGUP_DIAL_OUT    = 2
} STR_mode;

typedef struct {
      byte  sub_code;         // always #1
      byte  app_command;      // value = 2
      STR_mode mode;
      char  modem_string[];
} ANM_send_str_request;
```

The length of `modem_string` is limited to the application and network buffer sizes within the SLTA-10/LTS-20-based node and the node with which it is communicating. The SLTA-10/LTS-20-based node as shipped has buffers sizes allowing for a maximum of 46 characters in the modem string sent with this message. The string must be null terminated.

If a large string needs to be sent to the SLTA-10/LTS-20-based node, use a series of these requests with a single carriage return in the last string. Note that many modems have a limited input buffer size, typically 32 to 80 bytes.

The `mode` parameter is used to control the connection state of the SLTA-10/LTS-20-based node. The values for this parameter are:

0 Make no change to the SLTA-10/LTS-20-based node's modem connection state. Send only if not CONNECTED. Otherwise, respond with a failed status.

1 Initiate a dial-out connection. If the SLTA-10/LTS-20-based node is already currently connected, preserve that connection, and ignore the message. The SLTA-10/LTS-20-based node's connection status changes from `IDLE` to `CALL_IN_PROCESS`, unless the connection is already made, in which case the state stays at CONNECTED. The dial-out prefix is sent first.

2 Same as '1', but disconnect (hang-up) if currently connected before initiating the new connection.

## Modem Response Query

ASCII strings received from the modem when the SLTA-10/LTS-20-based node is not connected to a host will be buffered and may be read back by a node via this message. The storage for this string is limited, and so it will contain only the first 16 characters received from the modem in the disconnected state. Executing the *Send Modem String* message will always clear this buffered string. It will not be cleared when the modem disconnects, aiding in diagnosis of connection problems.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command; // value = 4
      byte max_length;  // limits the size of the response.
} ANM_modem_response_query_request;


typedef struct {
      char response[];  // null terminated string
} ANM_modem_response_query_response;
```

## Connection Status Query

The SLTA-10/LTS-20-based node's connection status may be polled with this message. Most modems may be configured with various time-outs for the different stages of establishing a connection. Consult your modem's documentation for details.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command; // value = 5
} ANM_connect_state_request;

typedef enum {
      IDLE              = 0,
      FAILED                  = 1,
      CALL_IN_PROCESS   = 2,
      CONNECTED         = 3
} CONN_state;

typedef struct {
      CONN_state connection_state;
} ANM_connect_state_response;
```

## Install Directory Entry

This message stores a dial-out directory entry in the SLTA-10/LTS-20-based node's EEPROM string pool. Up to 5 dial-out entries can be stored in the NSI mode EEPROM pool; up to 8 can be stored in the MIP mode EEPROM pool. The entries are numbered 0 to 4 (or 0 to 7) as specified by the `dir_num` field.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command;       // value = 6
      byte dir_num;           // value = 0-7 for MIP; 0-4 for
NSI
      byte total_size;        // of the data string
      byte offset;            // for piecemeal writes
      char dir_string[];
} ANM_install_dir_request;
```

See the section on *EEPROM String Pool Management* above for details on the EEPROM string pool.

## Dial From Directory

Using one of the directory entries specified by `dir_num`, dial-out to a remote host and establish a connection. Based on the `mode` argument, the directory string may be sent to the modem and the SLTA-10/LTS-20-based node enters the `CALL_IN_PROCESS` state. Connection progress may then be checked periodically by interested nodes using the *Connection Status Query* message. The `mode` parameter is used to control the connection state of the SLTA-10/LTS-20-based node. The values for this parameter are:

1 Initiate a dial-out connection. If the SLTA-10/LTS-20-based node is already currently connected, preserve that connection, and ignore the message. The SLTA-10/LTS-20-based node's connection status changes from `IDLE` to `CALL_IN_PROCESS`, unless the connection is already made, in which case the state stays at `CONNECTED`. The dial-out prefix is sent first.

2 Same as '1', but disconnect (hang-up) if currently connected before initiating the new connection.

```
typedef enum {
      DIAL_OUT          = 1,
      HANGUP_DIAL_OUT   = 2
} STR_mode;

typedef struct {
      byte  sub_code;      // always #1
      byte  app_command;   // value = 7
      STR_mode mode;
      byte  dir_num;       // value = 0-7 for MIP; 0-4 for NSI
} ANM_dial_dir_request;
```

If the directory entry does not exist, the response to this message will indicate a failure. A successful response to this message indicates that the SLTA-10/LTS-20-based nodehas sent the dial-out command to the modem, not that the modem has successfully dialed. The *Connection Status Query* message should be sent to determine whether a successful connection has been established.

## Hang-up

This message causes the SLTA-10/LTS-20-based node to pulse the EIA-232 DTR signal (Data Terminal Ready) low for 500ms. If the DCD signal is still ON following this step, then the SLTA-10/LTS-20-based node will send the hangup string (see below) if it is not a null string. This will terminate the connection, and the SLTA-10/LTS-20-based nodeenters the `IDLE` state. The response will not be sent until this process is complete. Therefore the transaction timer for this message should be set to at least 768ms if it is sent from a node on the network. If this message is sent from a remote host, no response should be expected, since the connection will have been broken before the response can be sent.

```
typedef struct {
      byte  sub_code;          // always #1
      byte  app_command;       // value = 8
      boolean     if_config;
} ANM_hangup_request;
```

If the SLTA-10/LTS-20-based node is forced to send the hangup string, and this string does not exist in the EEPROM configuration, the response to this message will indicate a failure. If the `if_config` byte of this message is non-zero, the SLTA-10/LTS-20-based node sends the configuration string to the modem following the hangup process. This provides a remote host with the ability to dial up a remote SLTA-10/LTS-20-based node, change the configuration string, hang-up, and reconfigure the modem.

## *Install Password*

This message stores a dial-in password in EEPROM. The default setting for this string is a null string, which results in no password requirement by the SLTA-10/LTS-20-based noder. Any node on the network may change the password, but an external host must have already connected and used the existing password in order to change it, unless it was blank. Any installed SLTA-10/LTS-20-based node should use a password, otherwise an intruder might change the password to another setting.

The password string is limited to 8 characters, and may be any sequence of non-zero eight bit values. The string must be null-terminated.

```
typedef struct {
      byte sub_code;                // always #1
      byte app_command;       // value = 9
      char passwd_string[8];
} ANM_install_passwd_request;
```

If a password is installed in the SLTA-10/LTS-20-based node, the host must issue the `niPASSWD` network interface command to the SLTA-10/LTS-20-based node after the connection is established. The code for this command is 0xE4, and it should be followed by up to eight password characters, null terminated. For details on sending network interface commands, see the *LONWORKS Host Application Programmer's Guide.*
If modem support is selected, a password is present in the SLTA-10/LTS-20-based node's configuration, and the connection state changes from `IDLE` to `CONNECTED`, the SLTA-10/LTS-20-based node will not process any other network interface commands until the correct password is sent downlink. If the password is correct theSLTA-10/LTS-20-based node will respond with the `niACK` code. Otherwise the SLTA-10/LTS-20-based node will respond with the `niNACK` code.

## *Install Modem Configuration String*

Whenever the SLTA-10/LTS-20-based node is either powered or reset, the connection state is not `CONNECTED`, or is commanded to by the *Configure Modem* message, the SLTA-10/LTS-20-based node will send its modem configuration string to the modem. This string is stored in EEPROM, and may be changed by this message. The default setting for this string is "`ATE0&C1&D2S0=1!`". The commands specified in this string are:

- `E0` is the AT command to disable local echo of characters received by the modem.
- `&C1` is the AT command to enable the Data Carrier Detect (DCD) output of the modem, which is active when the modem detects the carrier signal of another modem on the line.
- `&D2` is the AT command to enable the Data Terminal Ready (DTR) input of the modem. The modem will hang-up, enter command state, and disable auto-answer when it detects an on-to-off transition of the DTR input.
- `S0=1` is the AT command to set register S0 to 1, meaning that the modem should auto-answer incoming calls on the first ring. This option should be used if remote hosts will be dialing in to the SLTA-10/LTS-20-based node. Use "`S0=0`" if the SLTA-10/LTS-20-based node will only be used for dialing out to remote hosts.

See *Modem Compatibility* later in this chapter for additional sample modem configuration strings.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command; // value = 10
      byte total_size;  // of the data string
      byte offset;            // for incremental writes
      char cfg_string[];
} ANM_install_cfgs_request;
```

This message uses the EEPROM pool to store the modem configuration string. See the section on *EEPROM String Pool Management* above for details.

## Install Hangup String (MIP mode only)

This message installs a hang-up string, which is used to terminate a connection if the DTR control fails to do so. The default setting for this string is "`~~~+++~~~ATH0!`". This particular sequence is useful for Hayes modems, or other modems that have licensed the use of the Guard Time feature from Hayes Corporation. Some so-called Hayes-compatible modems use other sequences. A 1.5 second pause, followed by the string '+++', followed by another 1.5 second pause will cause a Hayes-compatible modem to enter its command state if it is in the connected state. The command `ATH0` causes the modem to hang-up.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command; // value = 11
      byte total_size;  // of the data string
      byte offset;            // for incremental writes
      char hups_string[];
} ANM_install_hups_request;
```

This message uses the EEPROM pool to store the hang-up string. See the section on *EEPROM String Pool Management* above for details.

## Install Dial Prefix

The default setting for this string is "ATDT". This string is sent as a prefix for any dial-out operations. This particular sequence instructs the modem to dial using Touch-Tone (DTMF) signaling. If pulse dialing is required, the prefix should be set to "ATDP."

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command;       // value = 12
      byte total_size;        // of the data string
      byte offset;            // for incremental writes
      char dpre_string[];
} ANM_install_dpre_request;
```

This message uses the EEPROM pool to store the dial prefix. See the section on *EEPROM String Pool Management* above for details.

## Install Hangup Timer

The hangup timer is controlled by an eight bit value in EEPROM. The default setting for this value is zero, which results in no hangup timer action. If the hangup timer is a non-zero value, the SLTA-10/LTS-20-based node will hang-up and break a connection (if in the CONNECTED state) when the number of minutes specified by `timer_value` have elapsed and no uplink or downlink activity has occurred. The default value for this timeout is zero, meaning that the phone connection will remain active indefinitely, even if there is no activity on the link.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command;       // value = 13
      byte timer_value;
} ANM_install_hangt_request;
```

## Configure Modem

This message will cause the SLTA-10/LTS-20-based node to send its Modem Configuration String to the modem provided it is not in the CONNECTED state. See the *Install Modem Configuration String* message for further details. If the modem is in the CONNECTED state, the failed status will be returned.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command; // value = 14
} ANM_modem_config_request;
```

## Request/Release SLTA

This message may be used to grant ownership access of the SLTA-10/LTS-20-based node to any node on the local network. In a design where there may be more than one network node that wishes to control the SLTA-10/LTS-20-based node's connection states, Request/Release provides a method of managing this control. In this case a node will request the SLTA-10/LTS-20-based node, and the response to this request will be successful (code 0x3D) if access was granted, or failure (code 0x1D) if it was denied. At the end of such a session the node which was granted ownership must release the SLTA-10/LTS-20-based node, allowing other nodes access to it. This process only works if all nodes employ this mechanism. The request state is not preserved across resets of the SLTA-10/LTS-20-based node.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command; // value = 15
      boolean req_rel;  // TRUE if request, FALSE if release
} ANM_request_slta_request;
```

## Clear EEPROM Pool

This message will clear the entire EEPROM pool usage, and will remove the following configurable strings. The strings will be set to null strings, not their default values.

- Modem configuration string
- Hangup string
- Dial prefix string
- All dial-out directory entries

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command;       // value = 16
} ANM_clear_eepool_request;
```

## Install NVConnect (NSI mode only)

This message writes the NVConnect byte. A value of 0xFF (the default) will disable this feature. When this feature is enabled, the SLTA-10/LTS-20-based node initiates a dial-out when it receives a network variable update.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command;       // value = 17
      byte NVConnect;         // two 4-bit fields
} ANM_install_nvconnect;
```

## Install NSIConnect (NSI mode only)

This message writes the NSIConnect byte. A value of 0xFF (the default) will disable this feature. When this feature is enabled, the SLTA-10/LTS-20-based node initiates a dial-out when it receives an AddMyNSI message.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command;       // value = 18
      byte NSIConnect;        // two 4-bit fields
} ANM_install_nsiconnect;
```

## Install CallbackEnable (NSI mode only)

The CallbackEnable configuration byte is written as a Boolean value using this command. A valus of zero (the default) will disable the callback configuration. When this feature is enabled and a remote host dials-in to the SLTA-10/LTS-20-based node, the SLTA-10/LTS-20-based node terminates the call and initiates a dial-out using the address entry requested by the initial dial-in.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command;       // value = 19
      byte NSIConnect;        // Non-zero enables callback
} ANM_install_callbackenable;
```

## Report SLTAEE (NSI mode only)

This command will result in a response that includes the address of the SltaEE data, the revision of the data structure, and its length. This information can be used to read back any portion of the structure for analysis and deconstruction.

```
typedef struct {
      byte sub_code;          // always #1
      byte app_command;       // value = 20
} ANM_report_sltaee;

The response is:
typedef struct {
      word abs_address;       // 16-bit absolute address of
                              //  structure
      byte version;           // currently 1
      byte length;
} ANM_report_sltaee_resp;
```

## Modem Compatibility

Nodes based on the LTS-20 have been tested with the following modems:

| | |
|---|---|
| **Best Data** | Smart One external modem 33,600 bps Data/Fax modem |
| **Diamond** | SupraExpress 336e external faxmodem |
| **Hayes** | Accura 336 external fax modem |
| **US Robotics** | Sportster Voice external 28.8 faxmodem |

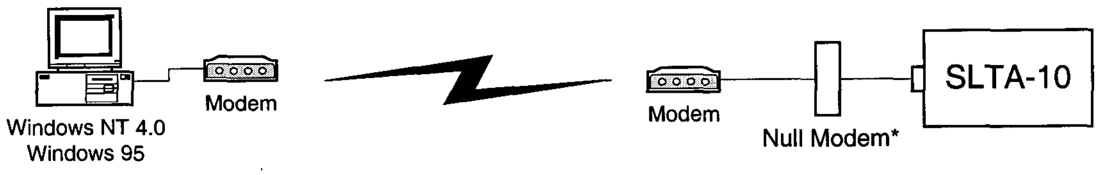
Synchronous communication should be disabled when synchronous modems are connected at lower serial bit rates (less than 4800 bps). Alternatively, data compression can be disabled at lower bit rates. For example, two V.42 or V.32bis modems connected at 2400 bps will have very low throughput due to the slow serial bit rate.

**You should disable XON/XOFF flow control in the modem when using it with the LTS-20.**

Note: When using the Hayes Accura external Fax Modem with Simultaneous Voice and Data 33.6 modem with the US Robotics 28.8 Faxmodem with Personal Voice Mail, an incompatibility in communication occurs. Testing the two modems via *Hyperterminal* reveals that the CD signal (Carrier Detect) is incorrect. Due to this communication incompatibility, do not use these two modems together with the SLTA-10/LTS-20-based node.

## Remote LTS-20 Deployment

The LTS-20 software is shared in common with the SLTA-10 Serial LonTalk Adapter. Any references to the SLTA-10 NSI, including file names, apply to the LTS-20.
The LTS-20 is an NSI capable network interface that supports a wide variety of application configurations to support several different network interface requirements for LONWORKS networks. This application brief describes the steps to setup an LTS-20-based node at a remote site to support dial-in access by a Windows 95, 98, or Windows NT 4.0 hosted LNS application. The goal configuration is shown in figure 15.2.



*See *SLTA-10 Adapter User's Guide* tables 3.3, 3.4

**Figure 15.2** LTS-20-based Node Remote Dial-in Setup

### Configuration

The LTS-20-based node should be configured as follows:

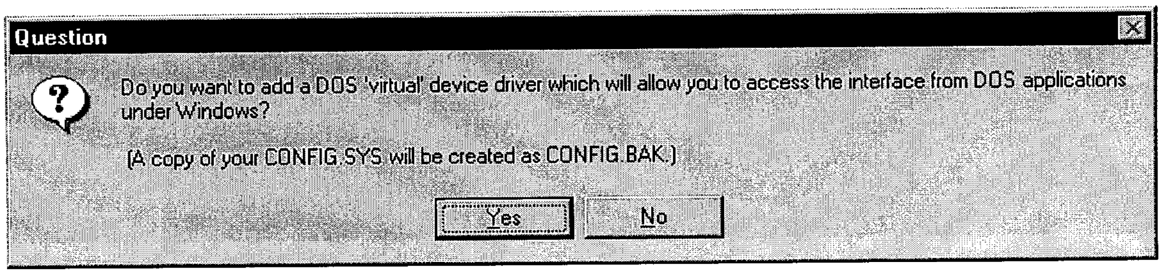
ALERT/ACK Link Protocol
Remote Host (modem)
Network Enable
NSI Mode
Autobaud Disabled

115,200 bps (configured assuming the use of a Diamond **SupraExpress 336e modem)**

## Software Setup

Your PC must have Windows 95, 98, NT installed. Chapter 5, *The LTS-20 NSI Mode Software*, describes how to install the software for Windows NT. This section describes the procedure for Windows 95.

Run the `setup.exe` supplied for the SLTA-10/LTS-20 Driver and SLTALink Manager software. After common installation queries are addressed, the setup program prompts you with the dialog box shown in figure 15.3. If you intend to access the SLTA-10/LTS-20-based node using 16-bit, LonManager API for Windows based tools you must answer 'Yes' to this prompt. If you intend to use current generation 32-bit LNS based applications (LonMaker for Windows), you can click 'No' to this prompt.



Do you want to add a DOS 'virtual' device driver which will allow you to access the interface from DOS applications under Windows?

(A copy of your CONFIG.SYS will be created as CONFIG.BAK.)

**Figure 15.3.** Answer 'Yes' if You Are Using 16-bit Windows Applications to Access the SLTA-10/LTS-20

Reboot the PC after installing the driver software.

## SLTALink Manager

The SLTALink Manager software provides the interface to setup the SLTA-10/LTS-20-based node for different modes of operation. This section describes how to configure it for remote dial-in access. The remote PC is assumed to have the an LNS 1.5 or higher application. The LonMaker for Windows integration tool is the application used for this description.

The SLTALink Manager (`sltalink.exe`) is installed to launch automatically from the `Start\Programs\StartUp` folder when Windows starts up. This application must run prior to accessing SLTA-10/LTS-20-based node from the LonMaker for Windows tool. The SLTALink Manager 's icon appears in the System Tray of the taskbar as shown in figure 15.4.

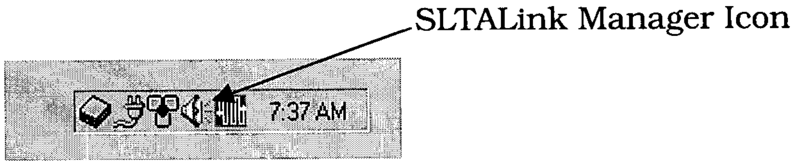


**Figure 15.4** SLTALink Manager Icon in the System Tray

Setting up an SLTA-10/LTS-20-based node and modem to leave onsite for remote dial-in access is accomplished as follows:

1. Temporarily connect the serial port of the network management PC directly to the SLTA-10/LTS-20-based node. You typically use the straight-through cable described in Hardware Setup.

2. Click on the SLTALink Manager in the System Tray You should see the screen shown in figure 15.5.

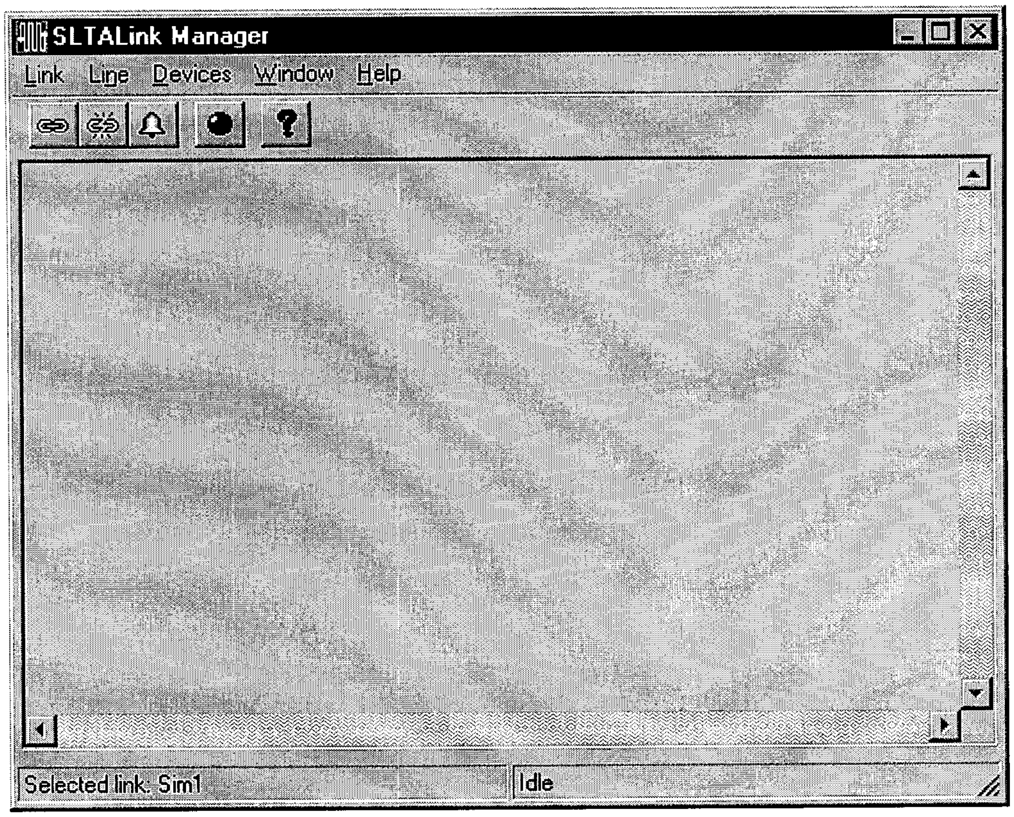
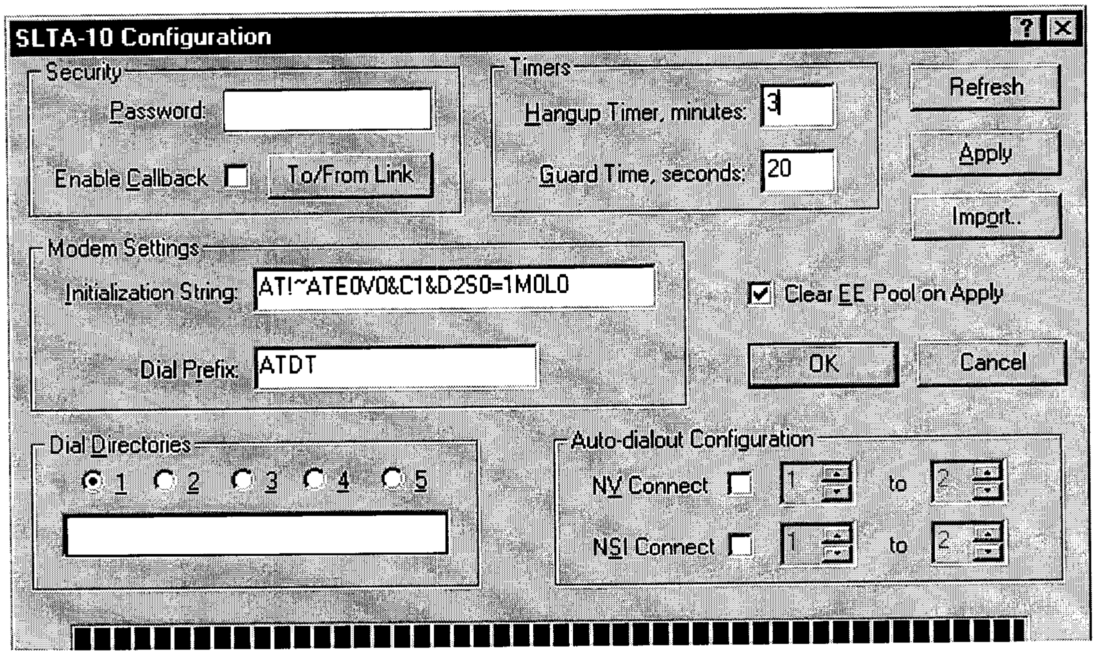


**Figure 15.5** SLTALink Manager Application

3. Confirm that the left panel of the SLTALink status bar shows the 'Local SLTA-10'. If not, click on the `Link.Select>` menu to select the Local SLTA-10.

4. Click on the `Manual Connect Link` speed button (the left most button in the toolbar) to establish a connection to the local SLTA-10. A successful connection will immediately be reported in the trace frame of the SLTALink Manager.

5. In a dial-in only scenario, the main configuration concern is the password used to gain access to the SLTA-10/LTS-20-based node remotely. Using the `Device.Configure SLTA...` dialog, you can set the password used to control access to the SLTA-10/LTS-20-based node. Figure 15.6 shows this dialog box. In this example, we checked the `Clear EE Pool on Apply`, and unchecked the `Auto-dialout` options. When `Apply` button is clicked, the appropriate portions of the currently connected SLTA-10/LTS-20-based node are modified.



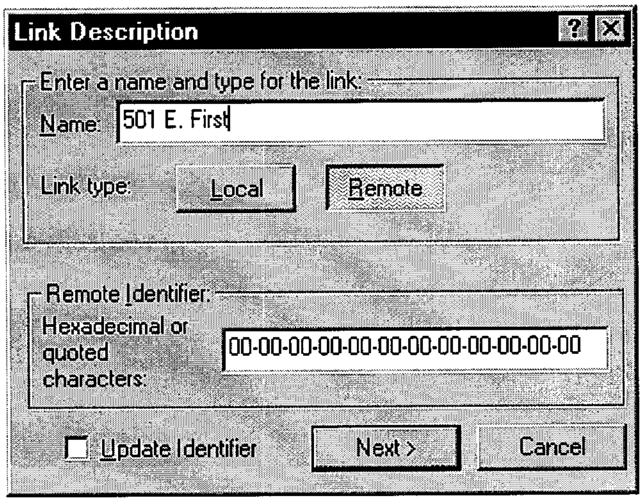
Password:
Enable Callback ☐
Hangup Timer, minutes: 3
Guard Time, seconds: 20
Initialization String: AT!~ATE0V0&C1&D2S0=1M0L0
Dial Prefix: ATDT
☑ Clear EE Pool on Apply
Dial Directories: ● 1 ○ 2 ○ 3 ○ 4 ○ 5
NV Connect ☐ 1 to 2
NSI Connect ☐ 1 to 2

**Figure 15.6** SLTA-10/LTS-20 Configuration Dialog Setup for Remote Dial-in

6. At this point you can test the SLTA-10/LTS-20-based node by running the LonMaker for Windows tool and selecting the SLTALON1 network interface to use the current active local SLTA-10/LTS-20-based node as the network interface. With the LonMaker for Windows tool, doing a `LonMaker.Status Summary...` command will provide a quick confirmation that the SLTA-10/LTS-20-based node functions properly, and can reach all devices in the target network. With the integrity of the adapter established, it is time to shutdown the LonMaker tool, and manually disconnect the local SLTA-10/LTS-20-based node using the SLTALink Manager.

7. Connect the proper null modem cable between the SLTA-10/LTS-20-based node and the modem. Verify power to both devices.

8. From SLTALink Manager, use the `Link.New...` command to define the remote connection description. See figures 15.6 – 15.8.


Link Description

Enter a name and type for the link:

Name: 501 E. First

Link type: Local | Remote

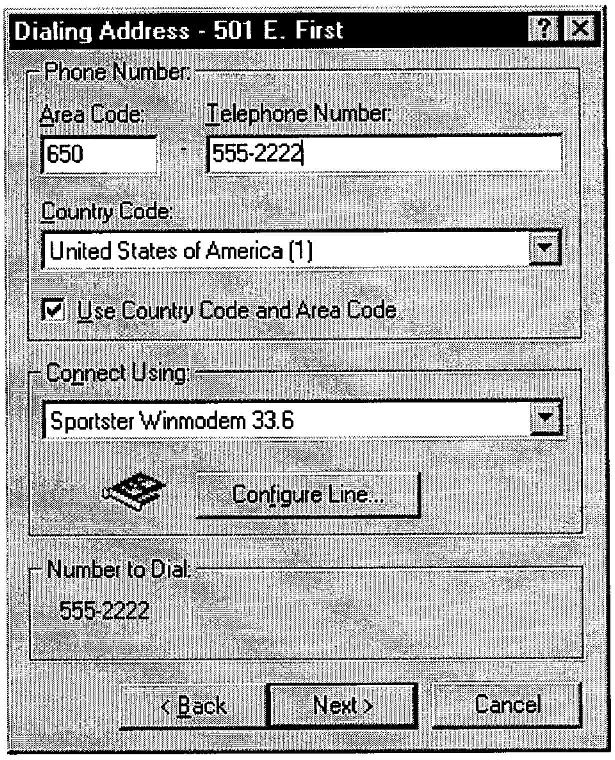
Remote Identifier:

Hexadecimal or quoted characters: 00-00-00-00-00-00-00-00-00-00-00-00

☐ Update Identifier

**Figure 15.7** Link Description Wizard Step 1

9. Provide the dialing information as shown in figure 15.8.


Dialing Address - 501 E. First

Phone Number:

Area Code: 650

Telephone Number: 555-2222

Country Code: United States of America (1)

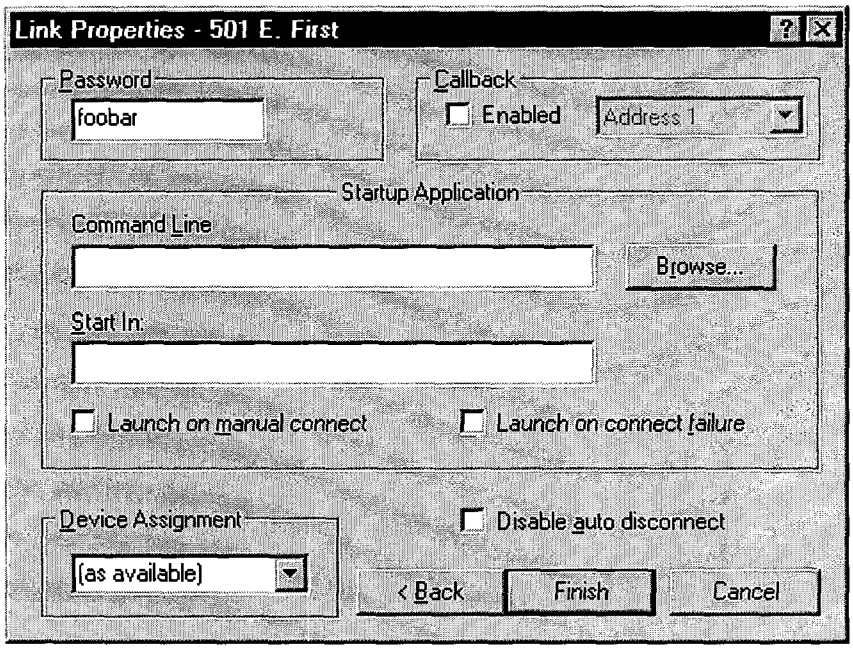
☑ Use Country Code and Area Code

Connect Using: Sportster Winmodem 33.6

Number to Dial: 555-2222

**Figure 15.8** Dialing Information for the Remote Link

10. In the next screen of the Link description wizard you need to provide the password that was configured during the local connection back in step 5.



Password: foobar

Callback: [ ] Enabled — Address 1

[ ] Launch on manual connect

[ ] Launch on connect failure

Device Assignment: (as available)

[ ] Disable auto disconnect

**Figure 15.9** Setting the Password for the Remote Link Description

11. Assuming the remote SLTA-10/LTS-20-based node modem is attached to the telephone network, you now can dial-in and perform all network management functions required to diagnose and maintain the network as if you were present at the site. To establish the remote connection, connect the network management PC's modem to the telephone network, select the remote link just defined using the `Link.Select>` menu function, and click on the `Connect` speed button. A successful connection to the remote SLTA-10/LTS-20-based node will be reported in the trace log section of the link manager software. At this point, a network management tool such as the LonMaker for Windows tool can attach to the `SLTALON1` device.

# 16

# Using the Host Connect Utility with the LTS-20 MIP Mode

The Host Connect Utility, or HCU, is a standalone DOS utility designed to dial out and make a connection to a remote LTS-20-based node in MIP mode. This utility may be used prior to executing an application based on a LonManager product, or may be called directly from such a product. The source code for HCU is supplied with the Connectivity Starter Kit so that it may be used as a model for host applications that need to establish connections directly with a remote LTS-20-based node, as well as host applications on platforms other than PCs running DOS.

The features provided by this program are:

- *Dial Out or Hangup Operations*
- *User Defined Modem Strings, Used for Initialization or Dialing*
- *Modem Control via the LDVSLTA Network Driver*

**Skip this Chapter if you are using the LTS-20 NSI mode.**

## HCU Usage

**The LTS-20 software is shared in common with the SLTA-10 Serial LonTalk Adapter. Any references to the SLTA-10 NSI, including file names, apply to the LTS-20.**

The command line arguments for HCU are:

HCU [options] [string1 .. string(n)] | [@filename]

The optional [options] arguments may include:

| | |
|---|---|
| `-C` *or* `-H` | To indicate to HCU that this is a Connection operation (the default), or a **H**angup operation. For example, a host application that needs to communicate with a remote SLTA-10/LTS-20-based node can be invoked from a DOS batch file, with preceding and subsequent calls to HCU. Prior to invoking the application, HCU is called to connect to the remote SLTA-10/LTS-20-based node. Following execution of the host application, HCU is called to disconnect from the remote SLTA-10/LTS-20-based node. |

**-D***devname* To select a non-default device name, where *devname* is the device name, default LON1.

**-P***password* To indicate a password, where *password* is a string of up to 8 characters, which will be sent downlink to the remote SLTA-10/LTS-20-based node once the connection is made. Each character in the password string may be any eight bit value. Non printable characters may be represented by hex or octal values in the same format as for C strings, such as "\x10", or "\020".

| | |
|---|---|
| **-T***nnn* | To select a non-default connection wait time of *nnn* seconds; the default is 60 seconds. This value limits the period for which HCU will wait for a connection to be completed. |
| **-B***bps* | To change the LDVSLTA network driver serial bit rate. The default is the current setting of the network driver. This controls the link rate between the host computer and the modem. The acceptable values for the *bps* value are the same as those available on the SLTA-10/LTS-20-based node: 1200, 2400, 9600, 14400, 19200, 38400, 57600, and 115200. |
| **-N** | To enable the buffered link protocol and disable the ALERT/ACK link protocol. This option is not recommended for connection operations. |
| **-X** | To exit HCU with the network driver's modem support mode disabled. Use with the -H option to disconnect from a remote SLTA-10/LTS-20-based node for subsequent connection to a local SLTA-10/LTS-20-based node. |

All options must precede string arguments on the command line when evaluated from left to right.

The string arguments are modem command strings, typically Hayes-compatible AT commands. These strings are sent directly to the modem. For details of the allowable commands, see the documentation supplied by your modem manufacturer. If more than one string is included, HCU will wait for the OK response from the modem before sending the next string. This requires that word (alphabetic) modem responses are enabled in the modem. If the OK response does not appear within 4 seconds, HCU will stop waiting and proceed to the next string.

In most cases the command string arguments need to be terminated with a carriage return. The carriage return is represented by the "!" character. HCU will interpret the "!" character by sending a carriage return (0x0D) to the modem. HCU will also interpret the "~" character by pausing 500ms. If either of these characters themselves need to be sent to the modem they can be escaped with a leading backslash, "\!" or "\~". HCU does not send implied carriage returns at the end of each string.

Spaces may be included in the string arguments. If you are including space in an argument you must enclose the string in double quotes or else the DOS command line interpreter will view the spaces as argument delimiters.

A filename may be used as a source of the string arguments. The filename should be preceded by the "@" character, and may be any full pathname. The structure of the file should be a series of text lines, where each line is a separate string argument. Command line strings and filename arguments may be intermixed.

## Theory of Operation

Upon execution, HCU will open the network driver, set the driver's operational mode to modem support on, force direct mode on (disables any SLTA-10/LTS-20-based node network interface protocols, enabling communications with the modem itself), modem responses on (enables modem responses to be passed to the host), and reliable transport protocol on. If a serial bit rate has been indicated, this also is set within the driver.

Next, if the HCU hangup operation is selected, HCU will drop DTR for 500ms. See *Using the LTS-20 with a Modem*, Chapter 15.

Next, all string arguments are sent to the modem. Acceptable modem responses to these strings are OK or ERROR.

Next, if the HCU connection operation is selected, HCU will wait for the connection to be established. This wait will be terminated by one of the following conditions:

- A response from the modem indicating either success or failure. These responses may be one of the following. [CR] is the carriage return (0x0D) character.

| | |
|---|---|
| CONNECT [any text] [CR] | This indicates a successful connection. |

These indicate a failed connection:

`NO` [any text] [CR], `BUSY` [CR], `VOICE` [CR]

- Any keyboard action. This will abort the HCU process.

Once the connection is considered made, HCU will process any additional modem responses for a few seconds. It will then change the operational mode of the network driver to force direct mode off and modem responses off. If this is a connection operation, the selected network interface protocol is enabled. If the `-X` option is specified, modem support is disabled.

Finally, if the user has indicated that a password is to be used, HCU will send the password command plus the password to the remote SLTA-10/LTS-20-based node, and wait for a response. If the response does not appear within five seconds, or if the response is not an acknowledgment, the process is repeated up to two additional times.

HCU will exit with a status of zero if the connection or hangup was successful. Otherwise the exit status will be '1'. Upon successful exit the network driver state will be set to modem responses off. This implies that if the connection is broken during the course of a host application execution the network driver will not start sending modem responses back to the host application, since it may not know how to handle them.

If operating under Microsoft Windows 3.1x, dialing out to a remote SLTA-10/LTS-20-based node requires running the DOS program `HCU.EXE` in a session prior to running the Windows API application. The HCU source code is available from Echelon for integration into a DOS application.

When using HCU with the Windows 95 operating system, use the following procedure:

- Start Windows 95.
- Start a DOS box. Make the DOS box a small window.
- Run HCU in the DOS box.
- After the message "successfully connected" and the DOS prompt appears, EXIT (type c:\>exit) DOS box.
- Then run the application.

## Usage Examples

To connect at 9600 bps with a modem initialization string included in the process:

HCU -b9600 -pSLTA_2.0 "ATM1E1S8=1!" "ATDT9,555-1234!"

It is important to remember to include the carriage return ('!') at the end of each command string argument.

To hangup:

```
HCU -h "~~~+++~~~ATH0!"
```

which includes the Hayes AT command mode escape sequence.

The following is a DOS batch file which will make the connection using a script 'dial.cmd', execute a host program 'hostapp.exe', and then hangup:

```
@echo off

hcu -b9600 -pSLTA_2.0 @dial.cmd

if not errorlevel 1 hostapp.exe
hcu -h "~~~+++~~~ath0!"
```

If using a Windows-based network management application, a connection can be created by using HCU and then closing the DOS box before initializing the connection with `lxt_open()`.

## Suggested Modem Configurations

Following is a list of configuration settings that are suggested for optimal operation of both HCU and for SLTA-10/LTS-20-based node to host phone links. These should be included in the string arguments to HCU if they are not the modem defaults. When possible, the corresponding AT command is included.

- Command echo: enabled ("E1"). This is a personal preference, and will result in modem commands being included in the HCU display.
- Send modem responses: enabled ("Q0"). This causes the modem to respond to commands sent to it with a result code.
- Word responses ("V1"). The modem result codes will be expressed as full word codes (alphabetical) rather than as numerical codes.
- Full response set ("X4" or "X2"). This will include the BUSY and NO DIALTONE responses, which will expedite the process when these cases occur.
- Data Carrier Detect signal (DCD): reflects carrier state ("&C1").
- DTE flow control: disable ("S58=0" for Telebit modems, "&K0" for Hayes modems). **If your modem uses software (XON/XOFF) flow control, you must disable it since the** SLTA-10/LTS-20-based node **link layer is a binary one.**
- Auto-answer: enabled ("S0=1"). If you need your Host modem to answer to an incoming call from a remote SLTA-10/LTS-20-based node and modem, you will need to enable this feature.

## Status and Error Reporting

HCU will print its progress to the standard output device, which defaults to the CRT screen. The format of this display is:

** HCU mm/dd/yy hh/mm/ss *progress string*

where `mm/dd/yy` is the current month/day/year, and `hh/mm/ss` is the current time. All modem responses are also displayed.

**Table 16.1** HCU Progress Strings

| | |
|---|---|
| *devname*: No such file or directory **or** Not ready reading device *devname* | Indicates that HCU could not open the device *devname*. |
| Dialing **or** Hanging up | Indicates the HCU operation. |
| Device initialization error, *devname* | Indicates a problem with initializing the network driver. |
| Could not open file *filename* | Indicates a problem with opening the string argument file. |
| Password Format error, abort | Indicates there was a problem interpreting the password argument. |
| Modem Response Timeout | Indicates that the modem did not respond to a command string. |
| DCD state test error, *devname* | Indicates a problem communicating with the network driver. |
| Connect timeout, *devname* | Indicates that the connection was not established within the connection timeout period. |
| User Abort | Indicates that the connect wait process was aborted by the user. |
| Connected **or** Connection Failed | Indicates the status of the connection. If the message Connection Failed appears following what appears to be a successful modem connection there may be a problem with the DCD signal from the modem. |
| Downlink Password Failure | Indicates a problem sending the password command downlink to the SLTA-10/LTS-20-based node. The connection may have been broken unexpectedly. |
| Password Validation Fail | Indicates that the remote SLTA-10/LTS-20-based node has responded to the password command with a negative acknowledgment. You are probably using the wrong password. |
| Password Validation Complete | Indicates a successful password command transfer with the remote SLTA-10/LTS-20-based node, and that HCU has successfully communicated with the remote SLTA-10/LTS-20-based node. |

| | |
|---|---|
| Downlink Password Timeout | Indicates no password acknowledgment or negative acknowledgment was received. Either the remote SLTA-10/LTS-20-based node is not operational, or the SLTA-10/LTS-20-based node link layer protocol does not match, or there is a problem with transferring binary data across the modem connection. |
| Hangup Failed, Still Connected | Indicates HCU still thinks that the connection is made following a hangup. This could be the result of a persistant DCD ON level. |

# 17

# Using a Programmable Serial Gateway

This chapter describes how to develop a gateway for a user-defined EIA-232 serial protocol. The LTS-20 product is shipped with SLTA firmware that allows it to be used as a LonTalk network interface for network management, monitoring, and control. When the SLTA firmware is installed, the data passed between the firmware and the host application via the EIA-232 port is in the format of LonTalk protocol application buffers. Serial gateways are developed using the model 73390 PSG-20 Serial Gateway Core Module. This gateway allows you to replace the SLTA firmware with your own firmware, enabling custom protocols to be translated into LonTalk.

## Creating a Serial Gateway

For applications of the LTS hardware platform, which cannot use the LonTalk network interface protocol on the EIA-232 link, the LTS hardware may be programmed for other data formats using Neuron C. For example, the LTS hardware may be used to connect ASCII devices such as printers, modems, and terminals to a LONWORKS network. The LTS hardware may also be programmed to build a gateway between a foreign EIA-232-based protocol, such as the serial port on a Programmable Logic Controller (PLC), and the LonTalk protocol. The LTS hardware without the firmware (e.g., without the PROM) is called a *programmable serial gateway*.

The programmable serial gateway is used to create a LONWORKS application node, so all the standard Neuron Chip firmware features described in the *Neuron C Programmer's Guide* are available to the application programmer. The firmware images for NSI and MIP mode, as well as capabilities described in the *LONWORKS Host Application Programmer's Guide* are not available. For example, there is no support for modem control or 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, LonManager DDE Server, or LNS-based applications, although it can still be installed and managed by such an application.

## LTS / PSG History

Echelon manufactures (or has manufactured) the following LONWORKS serial devices. Each device ships with firmware:

| **Product** | **Description** | **Firmware** | **Status** |
|---|---|---|---|
| LTS-10 | First generation SIP | SLTA/2 functionality in SIP package | Replaced by LTS-20 |
| **LTS-20** | Second generation SIP | SLTA-10 functionality in SIP package | Current Product |

Each of the serial adapters listed above may also be ordered *without* firmware. The hardware is identical, but no firmware is shipped with the device. The user must provide firmware using the functions described below.

| Product | Description | Firmware | Status |
|---|---|---|---|
| PSG-10 | First generation SIP; Hardware identical to SLTA-10 | None | Replaced by PSG-20 |
| **PSG-20** | Second generation SIP; hardware identical to LTS-20 | None | Current Product |

Depending on the hardware you intend to use, one of the following `#defines` must be defined in your source (`.NC`) just before the `PSG.H` file is included.

| | |
|---|---|
| `#define SLTASIP` | When targeting PSG-10 |
| `#define SLTA2` | When targeting PSG/2 |
| `#define PSG20` | When targeting PSG/3 or PSG-20 |

## Programmable Serial Gateway Hardware Resources

The PSG-20 hardware includes a UART attached to the I/O pins of a Neuron Chip, memory, oscillator, reset circuitry, and a twisted pair transceiver, as shown in figure 17.1.

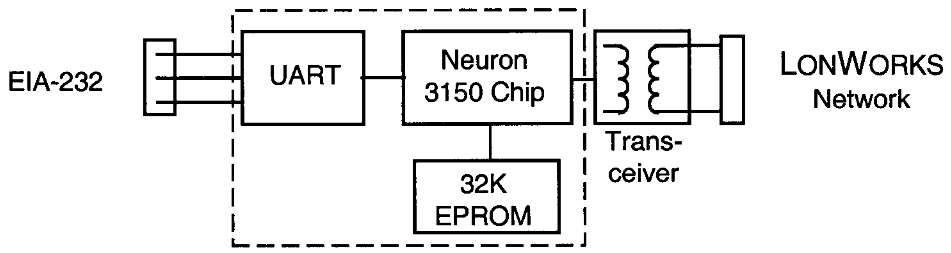


**Figure 17.1** PSG-20 Block Diagram (identical to an LTS-20)

## Developing a PSG Application with the NodeBuilder Development Tool

Developing an application for the PSG-20 serial gateways is similar to developing an application for any other custom device. Follow the device creation procedure outlined in Chapter 5 of the *NodeBuilder User's Guide*.

**Warning:** The PSG software relies on the software update provided in NodeBuilder Patch 5. Use the following steps to make certain this patch has been applied to your NodeBuilder development tool before building your PSG images.

1. Select About NodeBuilder... in the NodeBuilder HELP menu.
2. Check to be sure the version listed in the "About" dialog box is Version 1.5 **Build 05**.

### *PSG Software Installation*

The PSG/3 software diskette contains the following files:

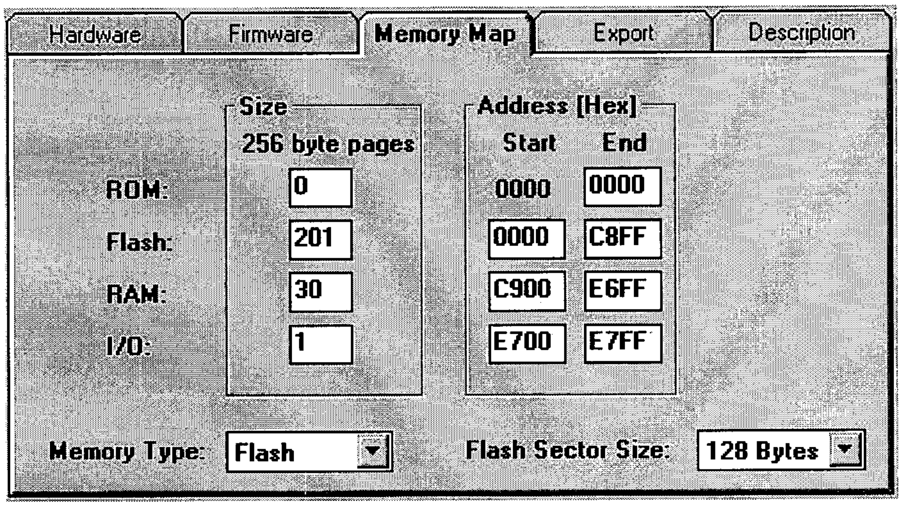
| | |
|---|---|
| `PSG20R.DTM` | //Device Template for release |
| `Readme.txt` | //Last minute notes |
| `PSG.LIB` | //Neuron C library containing PSG functions |
| `PSG.H` | //Function prototypes |
| `PSGREGS.H` | UART register and address definitions |

To install the PSG software on your development tool:

1. Copy the device template (`PSG20R.DTM`) to your device template directory. By default, this directory is `c:\lonworks\template`.
2. The two header files (`PSG.H` and `PSGREG.H`) should be copied to your development tool include directory. By default, this directory is `c:\lonworks\neuronc\include`.
3. The final step is to copy the `PSG.LIB` file to your development tool's library directory. By default, this directory is `c:\lonworks\images`.

## PSG20R.DTM

This template requires that the resistor described in Chapter 2 be in place (PSG-20 in NSI mode).



| | Size (256 byte pages) | Address [Hex] Start | Address [Hex] End |
|---|---|---|---|
| ROM: | 0 | 0000 | 0000 |
| Flash: | 201 | 0000 | C8FF |
| RAM: | 30 | C900 | E6FF |
| I/O: | 1 | E700 | E7FF |

Memory Type: Flash

Flash Sector Size: 128 Bytes

## Firmware Library Support

To aid in programming the custom serial gateway application, a firmware library (`PSG.LIB`) provides hardware access functions callable from Neuron C. (`PSG.LIB` replaces and supercedes the `SLTA.LIB` library that shipped with the SLTA/2 and PSG-10 products.) The include files also have been renamed to `PSG.H` and `PSGREG.H`. The new include files contain functions that are backward compatible with the functions declared in `SLTA.H` and `SLTAREG.H`.

In most cases, this library provides all the functions necessary to control the UART hardware and to read and write data from the serial port. Your application program will normally make use of some or all of the firmware functions included in the library `PSG.LIB`. Prototypes for the following functions, and the enumeration literals used in the following descriptions, are in the include file `PSG.H`.

This library is available for free by downloading it from the licensed software section of the Developer's Toolbox at www.echelon.com. The library also is provided with the NodeBuilder development tool.

## Usage

A single programmable serial gateway library (`PSG.LIB`) is included with the PSG software. Depending on the hardware you intend to use, one of the following `#defines` must be defined in your source (`.NC`).

| | |
|---|---|
| `#define SLTASIP` | When targeting PSG-10 |
| `#define SLTA2` | When targeting PSG/2 |
| `#define PSG20` | When targeting PSG/3 or PSG-20 |

This will control which low-level I/O access functions are used. Include the `#define` before the `#include <PSG.H>` statement. For example:

```
#include <netdbg.h>

#define PSG20

#include <psg.h>

...

when (reset)

{

    ...

}
```

## Code Development and Debugging

The PSG applications can be debugged only with the NodeBuilder Development Tool. The LonBuilder Development Tool may be used to develop/load/export applications, however no debugging facilities are available.

1. Use the NodeBuilder tool with PSG20R.DTM device template described above to create the following default [dummy] program:

```
#include <netdbg.h>

#define PSG20

#include <psg.h>

when (reset)

{

}
```

2. Export the program to Motorola S Records or Intel HEX format. See Chapter 5 of the *NodeBuilder User's Guide* for details.
3. Load an AT29C010 or AT29C512 FLASH part with your PROM programmer.
4. Insert the FLASH into the PSG and power the PSG. Be sure that your NodeBuilder tool is connected to the PSG using the LonWorks network. (The same way you would connect to any other custom node.)
5. It is now possible to create a real application and download it over the LonWorks network to the PSG. The NodeBuilder debugging features are available if you included `<netdbg.h>` in your source.

***Note:*** You should remove the reference to `<netdbg.h>` when creating your final distribution build.

## PSG.LIB Functions

```
void slta_init(slta_format, slta_baud, slta_intfc);
```

This function initializes the UART. It sets up the frame format, the serial interface bit rate, and the modem handshake lines. The frame format parameters are listed in PSG.H and may be set to:

| ***Format Parameter*** | ***Data Bits*** | ***Parity*** | ***Stop Bits*** |
|---|---|---|---|
| `format_8N1` | 8 | no | one |
| `format_7E1` | 7 | even | one |
| `format_7O1` | 7 | no | one |
| `format_7N1` | 7 | no | one |
| `format_8E1` | 8 | even | one |
| `format_8O1` | 8 | odd | one |
| `format_6N1` | 6 | no | one |
| `format_6O1` | 6 | even | one |
| `format_5N1` | 5 | no | one |
| `format_5E1` | 5 | even | one |
| `format_5O1` | 5 | odd | one |

The `slta_baud` parameter may be set to `baud_300`, `baud_600`, `baud_1200`, `baud_2400`, `baud_4800`, `baud_9600`, `baud_14400`, `baud_19200`, `baud_38400`, `baud_57600`, or `baud_115200`. The EIA-232 interface parameter may be set to `intfc_3wire`, or `intfc_8wire`. Configuration inputs are ignored–all configuration information is taken from the parameters supplied to `slta_init()`. This function should be called once in the `reset` clause of the application program.

Example:
```
     when (reset) {
          slta_init(format_8N1, baud_38400, intfc_3wire);
     }
```

### `unsigned slta_config(void);`

This function reads the configuration inputs. Each input corresponds to a bit in the value returned by `slta_config()`. All of the configuration inputs are available for application use. The PSG-20 input pin allocation described in Chapter 3 is only valid for the standard LTS firmware.

An application that uses the software in `PSG.LIB` may allocate these inputs for any purpose. To set the serial bit rate and other parameters, see the function `slta_init()`. The PSG-20 inputs return a logic 0 when the input is low and a logic 1 when the input is high. The `BAUD0` input corresponds to the least significant bit, followed by the `BAUD1`, `BAUD2`, `AUTOBAUD`, and the `CFG0` through `CFG3` inputs. The `CFG3` input corresponds to the most significant bit.

| MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| CFG3 | CFG2 | CFG1 | CFG0 | AUTOBAUD | BAUD2 | BAUD1 | BAUD0 |

`boolean slta_txrdy(void);`

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

`void slta_putchar(unsigned data);`

This function waits until the UART is ready, and then transmits the `data` character. If the UART is busy, this can take up to one character time. Since the UART is buffered, this function can return before the character is actually transmitted.

`void slta_puts(const char *s)`

This function waits until the UART is ready and then outputs a null-terminated string to the UART. Since the UART is buffered, this function can return before the string is completely transmitted. The terminating NUL is not transmitted. For other useful string-manipulation functions, see the *Neuron C Reference Guide*.

Example:
```
network input SNVT_str_asc text_message;
when(nv_update_occurs(text_message)) {
      slta_puts(text_message.ascii);
      slta_puts("\r\n");
}
```

`boolean slta_rxrdy(void);`

This function returns `TRUE` if the UART has one or more characters in its input FIFO buffer and `FALSE` otherwise. The UART used in the programmable serial gateways has a 16-character input FIFO buffer.

`long slta_getchar(void);`

This function tests to see if a character is waiting in the UART's input FIFO buffer. If there is no character waiting, the function returns `-1`. If there is a character waiting, it is returned in the least significant byte, and zero is returned in the most significant byte.

Example:
```
when (slta_rxrdy()) {                  // keep polling UART
      char c;
      c = (char) slta_getchar();      // get the character
}
```

## Advanced Applications

For most applications, the functions in `PSG.LIB` are all that is necessary to create custom programs for a programmable serial gateway using Neuron C. However, for specialized applications, the registers of the UART and the programmable serial gateway may be accessed directly by the application software. This section describes these registers and how they are accessed. For complete documentation, obtain a data sheet for the UART. The UART is an Exar Model 16C550I (http://www.exar.com/products/st16c550.html).

The registers are accessed using the `slta_write_uart()` and `slta_read_uart()` functions, which can be used to access any of the locations in the PSG/3 or PSG-20 I/O space. When using these functions, include the file `PSGREG.H`, which contains definitions for many of these locations.

```
void slta_write_uart (unsigned addr, unsigned data);
```

This function writes the data byte to the memory mapped I/O location defined by `addr`.

```
    extern unsigned slta_read_uart (unsigned addr);
```

This function reads the data byte from the memory mapped I/O location defined by `addr`, and returns that value.

The map for the extended address space is shown in the following sub-sections. Neuron C declarations for these addresses and bit assignments are provided in the include file `PSGREG.H` on the PSG software diskette.

## UART Registers

The UART registers are located at address 0xE780 – 0xE787. See the UART datasheet for register usage information.

## PROM / FLASH Specifications

The following specifications apply to PROM and FLASH.

PROM (90ns)

FLASH 29C010, 29C512 (90ns)

## Differences Between PSG-10 and PSG-20

The following differences exist between PSG-10 and PSG-20.

- PSG-20 uses a 29C010 or 29C512 flash.
- PSG-20 no longer supports `STL_BYTE` Board Control Register.
- PSG-20 no longer supports `STAT_BYTE` Board Status Byte.
- PSG-20 supports larger memory map with expanded RAM and ROM/FLASH options.

- `SLTA.H` has been replaced by `PSG.H`.
- `SLTAREG.H` has been replaced by `PSGREG.H`.
- `SLTA.LIB` has been replaced by `PSG.LIB`.

## Porting PSG-10 Code to the PSG-20

Use the following steps to port an existing application from a PSG-10 to a PSG-20.

1. Use the device template supplied in the PSG-20 software distribution.
2. Change all references to <`slta.h`> to reference <`psg.h`>.
3. Change all references to <`sltareg.h`> to reference <`psgreg.h`>.
4. Change #define `SLTASIP` or #define `SLTA2` to #define `PSG`.
5. Recompile your application and link with the `PSG.LIB` library.

# 18

# Modem Troubleshooting

This chapter provides solutions to problems that may arise with a modem attached to an LTS-20-based node.

## Troubleshooting

A Host/Modem - Modem/SLTA-LTS configuration has many user-selected options including the choice of modems, configuration of the modems, the operating system of the host, the network interface link protocol, and the serial bit rates of both the host/modem link layer and the SLTA-10/LTS-20-based node modem link layer.

**The most common problem is a failure to use the correct cable between the SLTA-10/LTS-20-based node and the modem.**

### *LTS-20-based Node and Modem Do Not Answer or Pick Up*

The modem attached to the LTS-20-based node must be configured to auto-answer if you want it to pick up and connect when dialed up. Set the modem's S0 register to a non-zero value.

### *Modems Will Not Connect*

This is an unusual case with modems which utilize modulation scheme fallbacks and error control negotiation fallbacks. One rule for modem modulation speed configuration is that if your modem can be configured to connect at the speed of the last 'AT' command, do so. Also, be sure that the modem's connect/carrier (Register S7) wait time is sufficient. Start with 60 seconds.

### *LTS-20-based Node- to-Host Link Fails Completely*

This can be observed as repeated retries at the link layer when the first actual downlink or uplink transfer is attempted. This can be due to any of the following conditions (in order of likelihood):

- Mismatched network interface link protocols. One end is using the ALERT/ACK link protocol and another is using the buffered link protocol. These settings are determined by CFG3 and in the host by network driver switches. The ALERT/ACK link protocol should be used.
- Using the Host Connect Utility in the wrong network interface link protocol. HCU can and may modify the current configuration of the DOS network driver. Ensure that the command line switches for the HCU maintain the desired network interface link protocol and serial bit rate settings. For example, you may have used the `/N` option with the DOS device driver, but did not use the `-N` switch with HCU.
- Using XON/XOFF flow control in your modem. Since the SLTA-10/LTS-20-based node network interface protocol is a binary one, this configuration will interfere with, or lock up, your modem. Be sure that this feature is disabled in your modems.

## LTS-20-based Node-to-Host Link Fails Partially

This can be observed as retries at the link layer when certain downlink or uplink transfers are attempted. This can be due to any of the following conditions (in order of likelihood):

- Extreme-case delays in either the modem or the connection. In this case the DOS network driver's calculated timeout values are too short. Increase the basic timeout value for the driver using the `/Rnn` option. Start with 10 for *nn* and go up.
- Use with Microsoft Windows 3.1x, particularly at higher serial bit rates (9600 or greater). This is always a problematic case. The priority of the serial I/O interrupts for PC/ATs is always lower than the DOS tick interrupt, which is used by Windows to perform many multi-tasking services. During these services, the serial I/O interrupts may not be serviced in time, resulting in lost uplink data. One solution is to lower the serial bit rate. Another solution may be to replace the PC/AT's UART with the 16550 UART, which has a 16 byte FIFO buffer built into it. This only works for external modem configurations, and will add about 16ms of interrupt headroom at 9600 bps, because of the hardware FIFO buffer. Another suspect in this area can be a disk caching program. These programs also perform services under the DOS tick interrupt, such as flushing data onto the disk drive, which can postpone serial I/O interrupts for lengthy periods.
- Modem serial bit rate overrun. For example, if the SLTA-10/LTS-20-based node serial bit rate is set to 38,400 bps and the modem telephone line speed is set to 2,400 bps, the modem will likely be overrun by sending it data faster than it can transmit it. This can occur since no flow control schemes can be used to restrict the rate that data is sent to the modem. In general, set the modem link rate equal to the telephone line speed. In certain cases it will be acceptable to exceed the telephone line speed - for instance, with a 14,400 bps V.32bis modem with data compression enabled, it may be possible to run the modem link at 19,200 bps.
- Full duplex FIFO overrun. This is caused by excessive full duplex traffic when using the buffered link protocol. The ALERT/ACK link protocol should be used instead.

## LTS-20-based Node Sends Modem Configuration String, But It Has No Effect

Most modems will determine the serial bit rate based on the assumption that the first two characters sent to them while in the command mode are the characters "AT". This means that a type of bit rate detection is being performed when these characters are sent to the modem. If the modem has been power cycled it will need to repeat this process. Some modems cannot handle being sent an entire configuration string following these first two characters during this link rate sensing phase. One way to accommodate these modems is to add an extra AT command, followed by a delay, to the front of the configuration string. For example:

```
"AT!~ATE0&C1&D2S0=1!"
```

This will allow the modem time to become fully synchronized with the new link bit rate before sending any actual command strings to the modem.

# Appendix A

## Communications Parameters

The LTS-20 will automatically set the transceiver communication parameters based on transceiver ID jumper settings. Parameters for LONMARK-approved transceivers correspond to the parameters defined by the *LONMARK™ Layers 1-6 Interoperability Guidelines*. The parameters specified as "Configurable" may be changed by a network management tool. The following table presents the transceiver IDs for different channel types.

| ***Parameter*** | ***TP/XF-78*** | ***TP/XF-1250*** | ***FT-10*** | ***TP-RS485-39*** |
|---|---|---|---|---|
| Transceiver ID | 1 (01 hex) | 3 (03 hex) | 4 (04 hex) | 5 (05 hex) |
| Media | Isolated Twisted Pair | Isolated Twisted Pair | Free Topology /Link Power | RS-485 Twisted Pair |
| Neuron Chip to Transceiver Interface (Comm Mode) | Differential | Differential | Single-Ended | Single-Ended |
| Interface Bit Rate (Comm Rate) | 78kbps | 1.25Mbps | 78kbps | 39kbps |
| Input Clock | 5/10MHz* | 10MHz | 5/10MHz* | 5/10MHz* |
| Minimum Clock | Default = 5MHz | Default = 10MHz | Default = 5MHz | Default = 5MHz |
| Number of Priority Slots | Default = 4 slots | Default = 16 slots | Default = 4 slots | Default = 4 slots |
| Average Packet Size | Default = 15 bytes | Default = 15 bytes | Default = 15 bytes | Default = 15 bytes |
| Oscillator Accuracy | 200ppm | 200ppm | 200ppm | 200ppm |
| Oscillator Wakeup | 0μsec | 0μsec | 0μsec | 0μsec |
| Collision Detect (CD) | No | No | No | No |
| CD Term after Preamble | N/A | N/A | N/A | N/A |
| CD Through Packet End | N/A | N/A | N/A | N/A |
| Bit Sync Threshold | 5 bits | 7 bits | 4 bits | 4 bits |
| Hysteresis | 2 | 0 | N/A | N/A |
| Filter | 1 | 0 | N/A | N/A |
| Channel Rate | N/A | N/A | N/A | N/A |
| Alternate Rate | N/A | N/A | N/A | N/A |
| Wakeup Pin Direction | N/A | N/A | N/A | N/A |
| XCVR Controls Preamble | N/A | N/A | N/A | N/A |
| General Purpose Data | N/A | N/A | N/A | N/A |
| Allow Node Override | N/A | N/A | N/A | N/A |
| Receive Start Delay | 2.9 bits | 14.0 bits | 9.0 bits | 2.0 bits |
| Receive End Delay | 0.0 bits | 0.0 bits | 0.0 bits | 0.0 bits |
| Indeterminate Time | 24.0 bits | 25.0 bits | 24.0 bits | 4.0 bits |
| Min. Interpacket Time | 0.0 bits | 0.0 bits | 0.0 bits | 0.0 bits |
| Turnaround Time | 0μsec | 0μsec | 0μsec | 0μsec |
| Missed Preamble | 1.0 bits | 4.0 bits | 4.0 bits | 1.0 bits |
| Preamble Length | N/A | N/A | N/A | N/A |
| Use Raw Data | No | No | No | No |

*The input clock rate is 10MHz.

| ***Parameter*** | ***TP-RS485-625*** | ***TP-RS485-1250*** | ***TP-RS485-78*** |
|---|---|---|---|
| Transceiver ID | 10 (0A hex) | 11 (0B hex) | 12 (0C hex) |
| Media | RS-485 Twisted Pair | RS-485 Twisted Pair | RS-485 Twisted Pair |
| Neuron Chip to Transceiver Interface (Comm Mode) | Single-Ended | Single-Ended | Single-Ended |
| Interface Bit Rate (Comm Rate) | 625kbps | 1.25Mbps | 78kbps |
| Input Clock | 5/10MHz* | 10MHz | 5/10MHz* |
| Minimum Clock | Default = 5MHz | Default = 5MHz | Default = 5MHz |
| Number of Priority Slots | Default = 4 slots | Default = 16 slots | Default = 4 slots |
| Average Packet Size | Default = 15 bytes | Default = 15 bytes | Default = 15 bytes |
| Oscillator Accuracy | 200ppm | 200ppm | 200ppm |
| Oscillator Wakeup | 0μsec | 0μsec | 0μsec |
| Collision Detect (CD) | No | No | No |
| CD Term after Preamble | N/A | N/A | N/A |
| CD Through Packet End | N/A | N/A | N/A |
| Bit Sync Threshold | 4 bits | 4 bits | 4 bits |
| Hysteresis | N/A | N/A | N/A |
| Filter | N/A | N/A | N/A |
| Channel Rate | N/A | N/A | N/A |
| Alternate Rate | N/A | N/A | N/A |
| Wakeup Pin Direction | N/A | N/A | N/A |
| XCVR Controls Preamble | N/A | N/A | N/A |
| General Purpose Data | N/A | N/A | N/A |
| Allow Node Override | N/A | N/A | N/A |
| Receive Start Delay | 2.0 bits | 2.0 bits | 2.0 bits |
| Receive End Delay | 0.0 bits | 0.0 bits | 0.0 bits |
| Indeterminate Time | 4.0 bits | 4.0 bits | 4.0 bits |
| Min. Interpacket Time | 0.0 bits | 0.0 bits | 0.0 bits |
| Turnaround Time | 0μsec | 0μsec | 0μsec |
| Missed Preamble | 1.0 bit | 1.0 bit | 1.0 bit |
| Preamble Length | N/A | N/A | N/A |
| Use Raw Data | No | No | No |

*The input clock rate is 10MHz.

| ***Parameter*** | ***PL-10*** | ***PL-20N*** | ***PL-20C*** |
|---|---|---|---|
| Transceiver ID | 9 (09hex) | 17 (11 hex) | 16 (10 hex) |
| Media | FCC Power Line | CENELEC C-band Power Line, Protocol Off | CENELEC C-band Power Line, Protocol On |
| Neuron Chip to Transceiver Interface (Comm Mode) | Special Purpose | Special Purpose | Special Purpose |
| Interface Bit Rate (Comm Rate) | 625kbps | 1.25Mps | 1.25Mps |
| Input Clock | 10MHz | 10MHz | 10MHz |
| Minimum Clock | Default = 5MHz | Default = 2.5MHz | Default = 2.5MHz |
| Number of Priority Slots | Default = 8 slots | Default = 8 slots | Default = 8 slots |
| Average Packet Size | Default = 15 bytes | Default = 15 bytes | Default = 15 bytes |
| Oscillator Accuracy | 200ppm | 200ppm | 200ppm |
| Oscillator Wakeup | 0$\mu$sec | 0$\mu$sec | 0$\mu$sec |
| Collision Detect (CD) | Yes | N/A | N/A |
| CD Term after Preamble | Yes | N/A | N/A |
| CD Through Packet End | N/A | N/A | N/A |
| Bit Sync Threshold | N/A | N/A | N/A |
| Hysteresis | N/A | N/A | N/A |
| Filter | N/A | N/A | N/A |
| Channel Rate | 9412bps | 3987bps | 3987bps |
| Alternate Rate | 9412bps | N/A | N/A |
| Wakeup Pin Direction | Output | Output | Output |
| XCVR Controls Preamble | Yes | Yes | Yes |
| General Purpose Data | 00 0A 00 00 00 00 00 | 0E 01 00 10 00 00 00 | 4A 00 00 10 00 00 00 |
| Allow Node Override | Yes | Yes | Yes |
| Receive Start Delay | 1.0 bit | 7.3 bits | 7.3 bits |
| Receive End Delay | 10.4 bits | 1.6 bits | 1.6 bits |
| Indeterminate Time | 0.0 bits | 10.1 bits | 10.1 bits |
| Min. Interpacket Time | 0.0 bits | 17.5 bits | 17.5 bits |
| Turnaround Time | N/A | N/A | N/A |
| Missed Preamble | N/A | N/A | N/A |
| Preamble Length | 3900 $\mu$sec | 33.5 bits | 33.5 bits |
| Use Raw Data | No | No | No |

| ***Parameter*** | ***PL-20A*** | ***PL-30*** | ***FO-10*** |
|---|---|---|---|
| Transceiver ID | 15 (0F hex) | 18(12hex) | 24 (18 hex) |
| Media | CENELEC A-band Power Line, Protocol On | CENELEC A-band Power Line | Fiber Optic |
| Neuron Chip to Transceiver Interface (Comm Mode) | Special Purpose | Special Purpose | Single-Ended |
| Interface Bit Rate (Comm Rate) | 1.25Mps | 625kbps | 1.25Mbps |
| Input Clock | 10MHz | 10MHz | 10MHz |
| Minimum Clock | Default = 2.5MHz | Default = 5MHz | Default = 10MHz |
| Number of Priority Slots | Default = 8 slots | Default = 8 slots | Default = 16 slots |
| Average Packet Size | Default = 15 bytes | Default = 15 bytes | Default = 15 bytes |
| Oscillator Accuracy | 200ppm | 300ppm | 200ppm |
| Oscillator Wakeup | 0$\mu$sec | 0$\mu$sec | 0$\mu$sec |
| Collision Detect (CD) | N/A | Yes | Yes |
| CD Term after Preamble | N/A | Yes | Yes |
| CD Through Packet End | N/A | N/A | Yes |
| Bit Sync Threshold | N/A | N/A | 4 bits |
| Hysteresis | N/A | N/A | N/A |
| Filter | N/A | N/A | N/A |
| Channel Rate | 3987bps | 1882bps | N/A |
| Alternate Rate | N/A | 1882bps | N/A |
| Wakeup Pin Direction | Output | Output | N/A |
| XCVR Controls Preamble | Yes | Yes | N/A |
| General Purpose Data | 0E 01 00 10 00 00 00 | 00 8A 00 00 00 | N/A |
| Allow Node Override | Yes | Yes | N/A |
| Receive Start Delay | 6.8 bits | 1.0 bit | 4.0 bits |
| Receive End Delay | 1.6 bits | 10.4 bits | 4.0 bits |
| Indeterminate Time | 0.0 bits | 0 bits | 4.0 bits |
| Min. Interpacket Time | 17.5 bits | 0 bits | 8.0 bits |
| Turnaround Time | N/A | N/A | 0$\mu$sec |
| Missed Preamble | N/A | N/A | 4.0 bits |
| Preamble Length | 33.5 bits | 19500 $\mu$sec | N/A |
| Use Raw Data | No | No | No |

| ***Parameter*** | ***DC-78*** | ***DC-625*** | ***DC-1250*** |
|---|---|---|---|
| Transceiver ID | 27 (1B hex) | 28 (1C hex) | 29 (1D hex) |
| Media | Direct Connect | Direct Connect | Direct Connect |
| Neuron Chip to Transceiver Interface (Comm Mode) | Differential | Differential | Differential |
| Interface Bit Rate (Comm Rate) | 78kbps | 625kbps | 1.25Mbps |
| Input Clock | 10MHz | 10MHz | 10MHz |
| Minimum Clock | Default = 10MHz | Default = 10MHz | Default = 10MHz |
| Number of Priority Slots | Default = 0 slots | Default = 0 slots | Default = 0 slots |
| Average Packet Size | Default = 15 bytes | Default = 15 bytes | Default = 15 bytes |
| Oscillator Accuracy | 200ppm | 200ppm | 200 pm |
| Oscillator Wakeup | 0$\mu$sec | 0$\mu$sec | 0$\mu$sec |
| Collision Detect (CD) | No | No | No |
| CD Term after Preamble | N/A | N/A | N/A |
| CD Through Packet End | N/A | N/A | N/A |
| Bit Sync Threshold | 4 bits | 4 bits | 4 bits |
| Hysteresis | 0 | 0 | 0 |
| Filter | 0 | 0 | 0 |
| Channel Rate | N/A | N/A | N/A |
| Alternate Rate | N/A | N/A | N/A |
| Wakeup Pin Direction | N/A | N/A | N/A |
| XCVR Controls Preamble | N/A | N/A | N/A |
| General Purpose Data | N/A | N/A | N/A |
| Allow Node Override | N/A | N/A | N/A |
| Receive Start Delay | 1.0 bit | 1.0 bit | 1.0 bit |
| Receive End Delay | 0.0 bits | 0.0 bits | 0.0 bits |
| Indeterminate Time | 0.0 bits | 0.0 bits | 0.0 bits |
| Min. Interpacket Time | 0.0 bits | 0.0 bits | 0.0 bits |
| Turnaround Time | 0$\mu$sec | 0$\mu$sec | 0$\mu$sec |
| Missed Preamble | 0.0 bits | 0.0 bits | 0.0 bits |
| Preamble Length | N/A | N/A | N/A |
| Use Raw Data | No | No | No |

# Appendix B

## Windows DLL Files for LTS-20 MIP Mode

This appendix describes the function and use of the LONWORKS DLL driver software provided with Echelon's Connectivity Starter Kit. Microsoft Windows supports access to DOS drivers through an interface layer called DOS Protected Mode Interface (DPMI). This interface standard defines the requirements to switch the processor between protected (Windows) and Real (DOS) mode operation, and also the mechanisms for proper data transfer between code running in these operating environments. Using DPMI, the same driver may be used in both DOS and Windows without modification.

The DPMI layer that allows access to the `LDVSLTA.SYS` and other DOS drivers provided by Echelon is contained in the Windows DLL, `WLDV.DLL`. This DLL is part of the LonManager® API for Windows, and the LonManager DDE Server. It is also supplied on the Windows DLL diskette. See Chapter 4 for information on installing the Windows DLL software. The LonManager API provides high level functions for network installation, maintanence, monitoring, and control. The LonManager DDE Server provides a simple Dynamic Data Exchange (DDE) interface for other client Windows applications to access to a LONWORKS based network for monitoring and control.

Programs specify a logical network driver name when first requesting access to the network. The `WLDV.DLL` supports simultaneous access to a maximum of eight (8) DOS drivers. The functions provided by `WLDV.DLL` are described in the following section.

## ldv_close

### Purpose

Terminates access to the network interface hardware.

### Syntax

```
#include <ldv.h>
short ldv_close(short handle);
```

### See Also

`ldv_open()`

### Returns

| | |
|---|---|
| `LDV_OK (0)` | Device closed successfully. |
| `LDV_NOT_OPEN (3)` | Invalid handle or device not open. |

### Parameters

| | |
|---|---|
| `handle` | `short`<br>Device identifier returned by `ldv_open()`. |

## ldv_get_version

### Purpose

Returns the current version of the driver DLL as a text string. Format of the version string is "M.mm[.sss]" where M is the major release number, mm is the minor release number, and [.sss] is an option sub-release number. All numbers are decimal. Using this function allows your application to verify that a compatible version of the driver `WLDV.DLL` is loaded.

### Syntax

```
#include <ldv.h>
const char far *ldv_get_version(void);
```

### See Also

None.

### Returns

| | |
|---|---|
| `char far *` | Character pointer to text string containing the `WLDV.DLL` version number. |

### Parameters

None.

## ldv_open

### Purpose

Initializes the network interface hardware for access by a Windows application. A Windows application can open multiple network interfaces. In the case of DOS drivers, this is done by loading multiple drivers in `CONFIG.SYS`. Initialization required to prepare the LTS-20 in MIP mode, or any Echelon provided network interface, for network messages is performed by this function. Different drivers and hardware interfaces could require different initialization and configuration requirements. Each driver must provide its own mechanism for providing these services. In the case of DOS device drivers, this is assumed to be command line options specified at the time the driver is loaded.

**Note:** A driver should only allow itself to be opened once. If the driver is already open, it should return the error value 2.

### Syntax

```
#include <ldv.h>
short ldv_open(char far *device_id_p, short far *handle)
```

### See Also

`ldv_close()`

### Returns

| | |
|---|---|
| `LDV_OK (0)` | Device successfully opened. |
| `LDV_NOT_FOUND (1)` | Hardware does not exist or is not accessible. |
| `LDV_ALREADY_OPEN (2)` | Device already open. |
| `LDV_DEVICE_ERR (4)` | Error occurred accessing device. |
| `LDV_INVALID_DEVICE_ID (5)` | Invalid device ID. |
| `LDV_NO_RESOURCES (8)` | No device handles available. |

### Parameters

`device_id_p` — `char far *`

Pointer to a character string identifying the network interface hardware device to be accessed. The following naming conventions are used to identify the type of device driver being used:

| | |
|---|---|
| `LON`*n* | DOS Device Driver named *LONn*, where *n* is a number from 1 to 9. |

`handle` — `short far *`

Pointer to an integer in which the open function will return a handle to be used to identify this device in other driver functions.

## ldv_read

### Purpose

Retrieves an available message from the network interface hardware. The function returns immediately when no messages are available. An error is returned when the next available message is longer than the specified buffer length.

### Syntax

```
#include <ldv.h>
short ldv_read(short handle, void far *msg_p, short len);
```

### See Also

`ldv_write()`

### Returns

| | |
|---|---|
| `LDV_OK (0)` | Message read and placed in the buffer pointed to by `msg_p`. |
| `LDV_NOT_OPEN (3)` | Invalid handle or device not open. |
| `LDV_DEVICE_ERR (4)` | Error occurred accessing device. |
| `LDV_NO_MSG_AVAIL (6)` | No message available. |
| `LDV_INVALID_BUF_LEN (9)` | Invalid buffer length. |

### Parameters

| | |
|---|---|
| `handle` | `short`<br>Device identifier returned by `ldv_open()`. |
| `msg_p` | `void far *`<br>Pointer to the buffer into which the message will be placed. |
| `len` | `short`<br>Length of buffer, in bytes. |

## ldv_write

### Purpose

Delivers a message to the network interface hardware.

### Syntax

```
#include <ldv.h>
short ldv_write(short handle, void far *msg_p,
            short len);
```

### See Also

`ldv_read()`

### Returns

| | |
|---|---|
| `LDV_OK (0)` | Message written successfully. |
| `LDV_NOT_OPEN (3)` | Invalid handle or device not open. |
| `LDV_DEVICE_ERR (4)` | Error occurred accessing device. |
| `LDV_NO_BUFF_AVAIL (7)` | No message buffers available. |

### Parameters

| | |
|---|---|
| `handle` | `short`<br>Device identifier returned by `ldv_open()`. |
| `msg_p` | `void far *`<br>Pointer to buffer containing the message to be delivered to the network. |
| `len` | `short`<br>Length of outgoing message, in bytes. |

# Appendix C

## Software License Agreements

Copies of the software license agreements for the LTS-20 products are included in this appendix.

## LONWORKS® Windows Driver Interface
## Software License Agreement

Echelon Corporation ("Echelon") grants you a non-exclusive, non-transferable license to use the copy of the software and documentation contained in this package and any updates or upgrades thereto provided by Echelon according to the terms set forth below. As used herein, "Licensed Software" means all software contained in this package, "Network Interface" means Echelon's products known as the PCLTA, PCLTA-10, PCLTA-20 LonTalk Adapters, SLTA/2, SLTA-10 Serial LonTalk Adapters, LTS-10, LTS-20 Module, NSS-10, NSI-10 Network Services Module, and any updates thereof and replacements thereto, and "Host Application Software" means software that executes on a microprocessor attached to a Network Interface and that, in conjunction with a Network Interface, acts as a device on a LONWORKS networks and which includes the Licensed Software or modifications thereof made by Licensee. If the software contained in this package is being provided to you as an update or upgrade to software which you have previously licensed, then you agree to destroy all copies of the prior release of this software within thirty (30) days after opening this package; provided, however, that you may retain one copy of the prior release for backup, archival and support purposes.

### LICENSE

**You may:**

(a) use and incorporate with additional software the Licensed Software to create Host Application Software, and use the accompanying documentation to support such efforts,

(b) make a limited number of copies of the Licensed Software solely to exercise the rights granted above and for backup purposes, provided that you reproduce, unaltered, all proprietary notices on or in the copies, and

(c) reproduce and distribute Host Application Software and the Licensed Software, in binary form only, solely for use with a Network Interface, provided that you reproduce, unaltered, all proprietary notices on or in the copies. You remain solely responsible for support, services, upgrades or other technical assistance with respect to your Host Application Software, and will indemnify and hold Echelon harmless from all claims, liability and damages arising from your use or distribution of your Host Application Software,

**You may not:**

(a) copy the Licensed Software (except as expressly permitted above), or copy the accompanying documentation,

(b) modify, translate, reverse engineer, decompile or disassemble of the Licensed Software (except to the extent that such activities may not be prohibited under applicable law), or

(c) distribute, rent, transfer or grant any rights in the Licensed Software or modifications thereof or accompanying documentation (except as expressly permitted above) in any form to any person without the prior written consent of Echelon.

This license is not a sale. Title and copyrights to the Licensed Software, accompanying documentation and any copy made by you remain with Echelon. Unauthorized copying of the Licensed Software or the accompanying documentation, or failure to comply with the above restrictions, will result in automatic termination of this license and will make available to Echelon other legal remedies. You may not use Echelon's name, logo or trademarks, except to state that your Host Applications Software incorporates the Licensed Software in accordance with Echelon's guidelines for use of its trademarks.

### LIMITED WARRANTY AND DISCLAIMER

Echelon warrants that, for a period of ninety (90) days from the date of delivery to you, the diskettes on which the Licensed Software is furnished under normal use will be free from defects in materials and workmanship and the Licensed Software under normal use will perform substantially in accordance with the Licensed Software specifications contained in the Network Interface documentation.

Echelon's entire liability and your exclusive remedy under this warranty (which is subject to your returning the Licensed Software to Echelon) will be, at Echelon's option, to use reasonable commercial efforts to attempt to correct or work around errors, to replace the Licensed Software or diskettes with functionally equivalent Licensed Software or diskettes, as applicable, or to refund the purchase price and terminate this Agreement. EXCEPT FOR THE ABOVE EXPRESS LIMITED WARRANTIES, ECHELON MAKES AND YOU RECEIVE NO WARRANTIES OR CONDITIONS, EXPRESS, IMPLIED, STATUTORY OR IN ANY COMMUNICATION WITH YOU, AND ECHELON SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT AND THEIR EQUIVALENTS.

Echelon does not warrant that the operation of the Licensed Software will be uninterrupted or error free or that the Licensed Software will meet your specific requirements.

SOME STATES OR OTHER JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSIONS MAY NOT APPLY TO YOU. YOU MAY ALSO HAVE OTHER RIGHTS THAT VARY FROM STATE TO STATE AND JURISDICTION TO JURISDICTION.

**LIMITATION OF LIABILITY**
IN NO EVENT WILL ECHELON BE LIABLE FOR LOSS OF DATA, LOST PROFITS, COST OF COVER OR OTHER SPECIAL, INCIDENTAL, PUNITIVE, CONSEQUENTIAL OR INDIRECT DAMAGES ARISING FROM THE USE OF THE LICENSED SOFTWARE OR ACCOMPANYING DOCUMENTATION, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY. THIS LIMITATION WILL APPLY EVEN IF ECHELON OR AN AUTHORIZED DISTRIBUTOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL ECHELON'S LIABILITY EXCEED THE AMOUNTS PAID FOR THE NETWORK INTERFACE. YOU ACKNOWLEDGE THAT THE AMOUNTS PAID BY YOU FOR THE NETWORK INTERFACE REFLECT THIS ALLOCATION OF RISK.

SOME STATES OR OTHER JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATIONS AND EXCLUSIONS MAY NOT APPLY TO YOU.

## LANGUAGE

The parties hereto confirm that it is their wish that this Agreement, as well as other documents relating hereto, have been and shall be written in the English language only.

Les parties aux présentes confirment leur volonté que cette convention de même que tous les documents y compris tout avis qui s'y rattache, soient rédigés en langue anglaise.

## GENERAL

This Agreement shall not be governed by the 1980 U.N. Convention on Contracts for the International Sale of Goods; rather, this Agreement shall be governed by the laws of the State of California, including its Uniform Commercial Code, without reference to conflicts of laws principles. This Agreement is the entire agreement between us and supersedes the license agreement on the box containing this package and any other communications or advertising with respect to the Licensed Software and accompanying documentation. If any provision of this Agreement is held invalid or unenforceable, such provision shall be revised to the extent necessary to cure the invalidity or unenforceability, and the remainder of the Agreement shall continue in full force and effect. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subdivision (c)(1)(ii) of the Rights in Technical Data and computer software clause at DFARS 252.227-7013, FAR 52.227-19 or equivalent rights under the regulations of any agency supplement.

Last Revised 03/29/95, 279-0202-01 Rev 0295