User Manual

Gesytec
Gesellschaft für Systemtechnik
und Datenverarbeitung mbH
Pascalstr. 6
52076 Aachen, Germany

Tel. + (49) 24 08 / 9 44-0
Fax + (49) 24 08 / 94 4-100

email: info@gesytec.de
www.gesytec.de

Doc. ID: LPS/UserDoc/LPS_Manual-E.doc
V1.3, May 2009

Gesytec

This manual …

… provides you with all the information which you will need to use the Easylon® Serial Socket Interface. However, this manual will neither explain aspects of Echelon's® LONWORKS® technology, nor Echelon's Microprocessor Interface Program (MIP) used on this network interface card. The network interface has been designed in accordance with the driver specifications of the Echelon Corporation.

After a general presentation of the Easylon Interface card in Chapter 1, Chapter 2 describes the necessary steps to install it.

Chapter 3 contains the technical description.

Chapter 4, "Programming Instructions", contains some information which will be helpful, if you should wish to develop your own network driver for the card.

References

- "WLDV32 – Programming API for LonWorks Access", Gesytec
- 078-0160-01E SLTA manual, starting at page 9-8; Echelon Corp. Description of the serial reliable transport protocol used for this device
- 078-0016-01B Host Application Programmers Guide; Echelon Corp.

Contents

1 Product Information

This manual describes the Easylon Serial Socket Interface

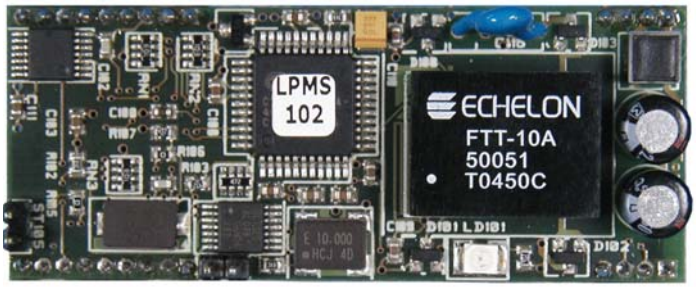

Figure 1-1 Easylon Serial Socket Interface

1.1 Scope of Delivery

- PC plug-in card with Echelon‘s MIP/P20 firmware
- Installation and Documentation CD including
  - WLDV32.DLL for Windows and Windows CE (a driver for Linux is available on request)
  - This documentation
  - Sample design for a carrier board (Protel DXP)

1.2 Overview

The Easylon Serial Socket Interface realizes a LON-serial connection as a socket module, to be integrated into OEM devices. The serial connection to the CPU board is made by a connection designed according the Conexant Socket Modem standard. Power supply uses this connector as well.

A FTT-10A transceiver (option: FTX smart transceiver) with a NEURON Chip running the MIP/P20 firmware connects to the LonWorks network. Service button and ~LED are available via the connector and have to be implemented on the carrier board. A sample design for a carrier board is available on the Drivers & Documentation CD, delivered with the interface. Firmware can be downloaded via the serial connection.

As an OEM module a certain flexibility with respect to customer specific requirements is observed, e.g. with respect to the connector types or positions. The

actual module may therefore be different from the description in this documentation.

Depending on the firmware of the module there are two basically different usages:

- The Easylon Serial Socket Interface can be used as a serial LonTalk adapter. In this application variant a serial protocol compatible to standard applications is implemented allowing for easy integration into existing applications.
- In a second application variant the interface module operates as a serial gateway. A host application running on the module allows implementation of network variables – even more than the usual 62. Thus more data points than in simply Neuron based solutions can be used. The module's processor with large integrated memory enables implementation even of complex protocol.

Additionally the module can be used for OEM applications. Either Gesytec or the customer can implement a dedicated application for the module.

NOTE: This documentation describes just the functionality as standard LonWorks interface.

2 Installation

The Easylon Interface cards are delivered in status "unconfigured". Prior to using it as a LONWORKS network interface it has to be set "configured". Standard applications available from the market, such as network management tools, automatically set this status or offer an appropriate command.

For customer specific applications which shall use this Easylon Interface the status setting has to be taken care of. Chapter 4 gives further hints on this subject.

The external interface files (.xif) can be found in the XIF directory of the installation CD.

2.1 Insertion of the Card

When inserting the Easylon Serial Socket Interface card in your computer, please be sure to observe all the computer manufacturer's instructions regarding the insertion of additional interface cards.

- Insert the Easylon Serial Socket Interface card into an available Conexant socket. The serial communication with the module has to be set to 57600 baud.
- Connect the interface card with an appropriate cable to the LONWORKS network.

2.2 Network Interface API

For usage with Windows and Windows CE there is no special driver for the Easylon Serial Socket Interface. Gesytec's WLDV32.DLL is prepared to interface to the Easylon Serial Socket Interface. Please refer to the manual of the WLDV32.DLL.

To install it, start setup.exe in the respective directory of the "Drivers & Documentation" CD.

Optionally a Linux driver is as well available for the device.

2.2.1 EasyCheck

Separately on the driver CD there is a setup for the utility “EasyCheck” which can be used to perform basic tests and settings with any Gesytec LonWorks interface.

For usage of the Easylon Serial Socket Interface, there have to be made some registry settings. There is a REG file, named “LPS.reg” on the Drivers & Documentation CD, which has to be imported into the registry. Afterwards the Serial Socket Interface will be available as COM1-115200 for EasyCheck. The REG file contains following settings:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\LonWorks\DeviceDrivers\COM1-115200]
"device name"=" LPS/1:115200"
```

The EasyCheck utility opens the interface. It will then check and display the version of the driver. For a hardware communication test it will then send a “query status” message to the local unit. „read memory“ command EasyCheck will show if the device is running MIP-firmware. A correctly installed Easylon Interface card will send an appropriate response. Finally, EasyCheck will close the interface.

3 Technical Description

3.1 Network Interface

The Easylon Serial Socket Interface cards is based on a NEURON 3120® Chip. Under MIP/P20 firmware the NEURON Chip is operated with up to 1096 byte RAM for network and application buffers. It is connected to an embedded host CPU in slave_b mode. The host CPU implements the serial protocol for communication with an external host.

On board there is a service LED for the NEURON Chip. A service push button can be realized by connecting a push button to the dedicated pins of the module. Additionally two status LEDs can be connected to the module

3.2 Service LED

The service LED signal is available on board and via the socket connector. It signals the card status. Additional to the service LED signals defined by Echelon the following status signals are used:

Service LED	Status	Remarks
Flash (1 Hz)	No firmware installed or firmware failure.	
Blink (1/2 Hz)	Driver installed, node is “unconfigured”[1].	Configure the node.
Permanently ON	Node is “applica-tionless” and “unconfig-ured”.	
Permanently OFF	Installation ok	Normal operation

Table 3-1 Service LED

[1] boards are delivered “unconfigured”

3.3 Serial Interface

The serial interface is just using TXD and RXD lines. The additionally mentioned hardware handshake lines RTS and DTR are used for on-board programming of the host CPU.

The baud rate is fixed to 115 200 baud. There is no autobaud feature implemented. To obtain the maximum possible baud rate of 57 600 baud, pleas ask Gesytec for devices with the corresponding firmware.

3.4 Power Supply

The serial communication between the Easylon Socket Interface and the PC is running on TTL level. Therefore the following requirements with respect to the power supply have to be met:
Socket interface and serial device must use the same voltage source.
Both devices must be supplied simultaneously. If not, a latch-up may occur on the interface or device using it, inhibiting correct operation.

3.5 Connector Pin Assignments

The Easylon Serial Socket Interface follows the Conexant Socket Modem pin layout using a part of the 64 connector pins.

PIN 1 position cf. to board dimensions (next page)
grey: pin does not exist.

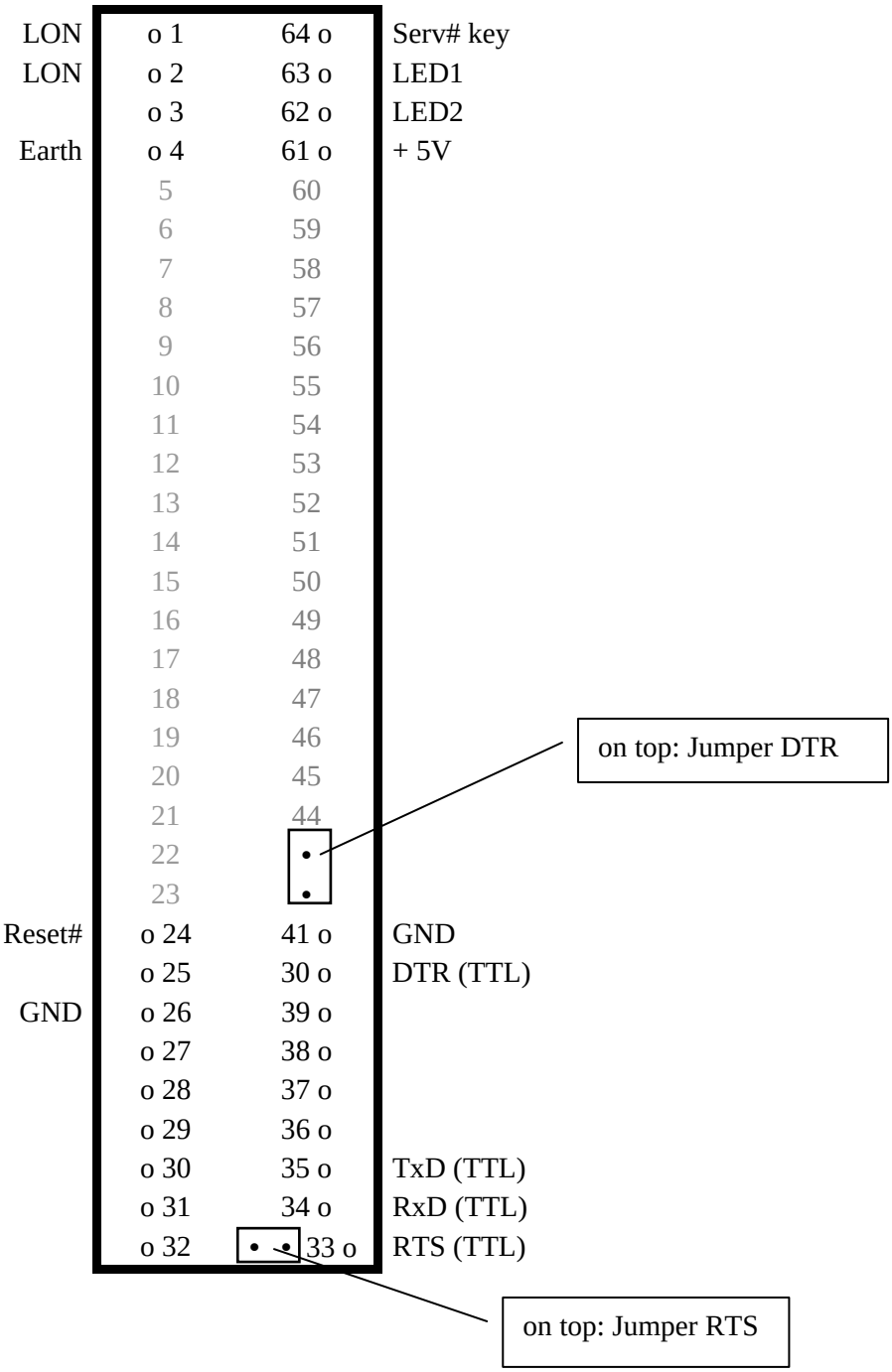

Figure 3-1 Connector pin assignment

3.6 Technical Specifications

Network Interface

CPU	Neuron 3120E4, 10–40 MHz
Transceiver variants	FTT-10A, FTX
Connector	pin connector
Service LED	on board or via Conexant connector
Service push button	via Conexant connector
Compatibility	LonTalk, EIA 709.1

CPU

Processor	8031 compatible, 11.059 MHz
Memory	Flash 64 Kbytes, RAM 8 Kbytes

Serial Interface

Type	TTL, Conexant standard, RxD, TxD, RTS, CTS
Connector	pin connector
Transmission	115,2 kbaud, (57,6 kbaud optional)

Power supply

Voltage	5 V DC +- 5%, externally from Conexant socket
Power consumption	< 100 mA

Dimensions & Environmental Characteristics

Temperature	operating 0 – +50 °C storage -20 – +70 °C
extended range	operating -40 – +85 °C storage -40 – +85 °C
Humidity	class F accord. DIN 40040, no condensation
Dimension	65,6 x 26,6 [mm]

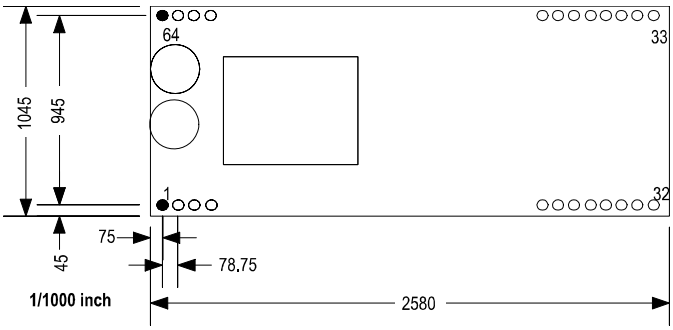

Figure 3-2 Dimensions and connector pins

4 Programming Instructions

This chapter gives programming instructions to the Easylon Serial Socket Interface.

4.1 LONWORKS Network Node

The Easylon Serial Socket Interface card is a network node in the LONWORKS network. It is operated under Echelon's Microprocessor Interface Program MIP/P20 firmware using the NEURON 3120 Chip as communication processor. The appropriate external interface file (.xif) is on the installation disk. Which .xif-file is describing which interface card variant is explained the table below.

Network Interface	Transmission rate	XIF-file
FTT	78 kbps	lolp075f.xif

Table 4-1 .xif files and interface card variants

4.2 Device Status

Applications have to take care of the status of the Easylon Serial Socket Interface card. As an example some parts of code are shown below. The structures used are taken from the so called HOST APPLICATION of the Echelon Corp. This application is available from the Echelon web site: www.echelon.com.

```
#pragma pack(1)
#define NM_update_domain    0x63
#define NM_set_node_mode    0x6C
#define SVC_request         0x60
#define niRESPONSE          0x16
#define niLOCAL             0x22
#define niRESET             0x50
#define LDV_OK              0

typedef struct {
    BYTE cmq;       // cmd[7..4]                    queue[3..0]
    BYTE len;
    BYTE svc_tag;   // 0[7] Service[6..5] auth[4]   tag[3..0]
    BYTE flags;     // prio path cplcode[5..4] expl altp pool resp
    BYTE data_len;
    BYTE format;    // rcv: domain[7] flex[6]
```

```
    union {
        struct {
            BYTE dom_node;      // domain[7] node/memb[6..0]
            BYTE rpt_retry;     // rpt_timer[7..4]     retry[3..0]
            BYTE tx_timer;      //                     tx_timer[3..0]
            BYTE dnet_grp;      // destination subnet or group
            BYTE nid[6];        // NEURON ID
        } send;
            struct {
            BYTE snet;          // source subnet
            BYTE snode;         // source node
            BYTE dnet_grp;      // destination subnet or group
            BYTE dnode_nid[7];  // destination node or NEURON ID
        } rcv;
        struct {
            BYTE snet;          // source subnet
            BYTE snode;         // source node
            BYTE dnet;          // destination subnet
            BYTE dnode;         // destination node
            BYTE group;
            BYTE member;
            BYTE reserved[4];
        } resp;
    } adr;
    BYTE code;                  // message code or selector MSB
    BYTE data[239];
} ExpAppBuf;

ExpAppBuf msg_out;    // Explicit message buffer for outgoing messages
ExpAppBuf msg_in;     // Explicit message buffer for incoming messages
ExpAppBuf msg_rsp;    // Explicit message buffer for response messages
int ni_handle;
BYTE my_domain[15] =
    {0,0,0,0,0,0, 0x01, 0xC0, 0, 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};

int send_local( int len ) {
    int ldv_err;
    msg_out.cmq = niLOCAL;
    msg_out.svc_tag = SVC_request;
    msg_out.flags = 8;
    msg_out.len = len + 15;
    msg_out.data_len = len + 1;
    if( ldv_write( &msg_out, len + 17 ) ) return(0);
    while( 1 ) {
        ldv_err = ldv_read( &msg_in, 256 );
        if( ldv_err == LDV_OK ) {
            if(msg_in.cmq == niRESET) return(0);      // Local reset
            if(msg_in.cmq == niRESPONSE) {
                memcpy(&msg_rsp, &msg_in, msg_in.len + 2);
                return(1);                             // Ok
            }
        }
```

```
    }
    return(0);
}

int set_config_online() {
    msg_out.code = NM_update_domain;
    msg_out.data[0] = 0;                          // Domain index 0
    memcpy( &msg_out.data[1], &my_domain, 15 ); // Subnet 1, Node 64
    if( !send_local(16)) return(0);

    msg_out.code = NM_set_node_mode;
    msg_out.data[0] = 3;                          // Change state
    msg_out.data[1] = 4;                          // Configured online
    if( !send_local(2)) return(0);
    return(1);                                    // Success
}
```

4.3 Linux Driver

There is a Linux driver available in source code. It implements a device named „slta“. The driver has an interface identical to all other Gesytec Linux drivers.

4.3.1 Implementation

4.3.1.1 Serial Line Discipline

Using the „serial line discipline“ concept this Linux driver provides a flexible method to realize different protocols for serial transmission. The following advantages result for the slta driver:

- Maximum compatibility with different kernel versions as only an “intermediate layer” of the serial port has to be implemented. This means a minimum of dependencies from kernel specific functions.
- Independence from the serial hardware used. The driver can be used with every tty interface.

In order to make the driver compatible to all systems the ID of a line discipline already known to the Linux kernel was used. Usage of a new ID would have required a kernel modification. Therefore the ID N_MASC of a „Mobitex Moduls“ is used, which is reserved but not used.

Note: The ID of the Line Discipline are defined in /usr/include/asm//termios.h.

4.3.1.2 Reliable Transport Protocol

The driver is based on Echelon's Solaris driver. This driver is using a receive- and transmit state machine supporting reliable transport protocol

4.3.2 Installation

4.3.2.1 Source Code Compilation for Driver and Utilities

The kernel source code has to be installed in order to compile the driver. Depending on the distribution this will require the kernel development package to be installed. Driver installation requires administrator rights.

The following commands will compile the driver and the slta-attach program:

```
make
make tools
```

The commands

```
make install
make setup
```

will copy the driver and the slta-attach program into the system.

4.3.2.2 Registering the Module Alias

To connect the line discipline used with the correct driver the following line has to be added to the configuration file /etc/modprobe.conf or /etc/modprobe.conf.local.

```
alias tty-ldisc-8 slta
```

This connects index 8 of the serial line discipline to the slta driver module. An additional line passes the stated option to the driver at loading time.

```
options slta slta_debug=1
```

4.3.2.3 Init Script

The driver has to be loaded and the program slta-attach started at system start to activate the slta driver. In most Linux distributions this is done by means of a SysV-Initscript. An example initscript.suse for SuSE-Linux is provided. Copy this script into the SysV-Init directory.

```
cp initscript.suse /etc/init.d/slta
```

To activate the slta driver then use the Yast Runlevels editor to make sure that the init script is automatically started at system start. There are other setup tools for other distributions. Please refer to the system documentation. Alternatively the script can be started manually to start the driver.

```
/etc/init.d/slta start
```

4.3.2.4 Using the slta Driver

The init script has to contain the device files of the serial ports to which sltas are connected. This means that the following line of the init script has to be adapted.

```
DAEMON_PORTS="/dev/ttyS0 /dev/ttyS1"
```

Furthermore, the init script starts device files with following names:

```
/dev/lon1, /dev/lon2, usw.
```

The increasing index of these device files follows the enumeration sequence of the serial ports in the DAEMON_PORTS variable.

4.3.2.5 Device Files and Access Authorization

Serial device are identified by entries in the /dev directory. Unfortunately there is no naming convention and different naming s exist in different distributions. Mostly /dev/ttyS0 refers to the first serial port (corresponding to COM1: in DOS). With the device file system devfs the name /dev/tts/0 was introduced. The current development allows a flexible naming. But not all distributions go for a standardized naming environment. However, in most cases /dev/ttyS0 is used for COM1: –also with SuSE 10.x and Fedora Core 3 and later. A serial USB port usually is named /dev/ttyUSBn or /dev/usb/tts/0 respectively, if devfs is used.

Using the slta driver requires the corresponding access authorization to the serial port for the user account. Depending on Linux or kernel version different approaches can be followed.

Depending on Distribution

a) SuSE 10.x, SuSE 9.x, Fedora Core

The serial port can be used by all members of the uucp group. Just add the user account under which the application program shall run to this group.

Depending on System- and Kernel Configuration

a) Using static device files

In a static /dev directory read/write authorization for a serial port can be assigned to all users with the file system rights using the command

```
chmod 0666 /dev/ttyS0
```

b) Using devfs

If the device file are dynamically started by devfs at system start, access authorization has to be defined in the configuration file /etc/devfsd.conf.

Note: You can see if devfs is used by the existence of the file /dev/.devfsd

The line

```
REGISTER ^tts/.* PERMISSIONS root.dip 0660
```

defines that all device files for serial ports belong to the user root and the group dip. The user root and all dip members have read/write authorization, all other users are denied the port access. To grant authorization to all users the rights have to be set to

```
0666
```

Alternatively authorization can be granted to all members of the group users. This requires the definition

```
root.users
```

c) Using udev

From kernel version 2.6.13 onward devfs can not be used any more. The userspace program udev is used instead. The corresponding configuration file can be found the in the /etc/udev/rules.d directory usually called 50-udev.rules.

Note: You can see if udev is used by the existence of a /etc/udev directory.

The line

```
KERNEL="ttyS*", GROUP="uucp", MODE="0660"
```

defines that all device files for serial ports belong to the user root and the group uucp. The user root and all uucp members have read/write authorization, all other users are denied the port access. To grant authorization to all users the rights have to be set to

```
MODE="0666"
```

4.4 Windows CE – Application Interface

For usage with Windows CE there is a Windows CE version of the WLDV32.DLL, which can be found on the Drivers & Documentation CD-ROM.

5 List of Figures

6 List of Tables

7 Index