

ICL-4300

PC Compatible Controller

EasyLink Users Manual



Revision A
P/N 60069001
revised 07/12/98
compatible with EasyLink v2.00

INDUSTRIAL CONTROL LINKS, INC.
12840 Earhart Avenue Auburn, CA USA 95602
Tel: (530) 888-1800 FAX: (530) 888-1300
www.iclinks.com

© 1998 Industrial Control Links, Inc. All rights reserved

Table of Contents

CHAPTER 1 – OVERVIEW	1
CHAPTER 2 – INSTALLING AND RUNNING EASYLINK	5
Installing EasyLink.....	5
Running EasyLink	6
CHAPTER 3 – USING EASYLINK	7
Simple EasyLink Configuration File	7
Modbus Master -- Reading	11
Modbus Master -- Writing.....	15
Time Based Net Events	15
Triggers	16
Net Events With Multiple Conditions.....	16
BrickNet Communications.....	17
Sending "Non-Standard" Data Types Using Modbus.....	18
Textual User Interface (TUI).....	20
Configuring the TUI.....	20
Using the TUI	27
Defines	28
Defaults.....	29
Options	30
Message Routing	31
BrickNet Routing Example.....	31
Modbus Routing Example	34

CHAPTER 4 – EASYLINK CONFIGURATION FILE REFERENCE _____ 37

- Syntax37
- Outline of the EasyLink Configuration File39
- Records and Parameters45**
 - [DEFINES]45
 - [OPTIONS]46
 - [DEFAULTS]47
 - [REGALLOC]48
 - [IOMAP]49
 - [IOSCALE]50
 - [REGSCALE]51
 - [TRIGGER]52
 - [NETPORT]53
 - [NETSESSION]54
 - [NETEVENT]57
 - [NETROUTE]58
 - [TUI]59
 - [TUIPAGE]60
 - [TUIFIELD]61
 - [VOICE]62

Chapter 1 – Overview

The ICL-4300 is a modular PC compatible controller that supports various optional hardware configurations. The ICL-4300 hardware options are described in more detail in the ICL-4300 Hardware Reference Guide.

ICL-4300 Hardware Features

- Intel 386EX Processor
- 33MHz Internal Clock
- 16-bit External Bus
- 512K battery backed SRAM
- Additional 512K SRAM (Super Size option)
- 2M Flash Memory (standard) or 4M Flash Memory (Super Size option)
- Real Time Clock
- Up to 7 Serial Communication Ports (including RS-232, RS-485 and internal dialup modem)
- Up to 64 analog or digital I/O points onboard
- 4 I/O board slots
- Expandable I/O via remote I/O modules (up to 32 modules)

Operating System Features

- BIOS Support
- ROM-DOS 6.22
- Flash File System (non-volatile storage)
- Multitasking kernel (ICL XINU) available

The ICL-4300 comes with the BIOS/ROM-DOS/Flash File System software pre-installed.

ICL-4300 Utilities

A set of useful utilities is available in the ICL-4300 Utilities package. This package also contains ROM-DOS documentation on diskette. The standard utilities are documented in the ICL-4300 Utilities Reference Manual.

EasyLink

An application program called EasyLink comes with the ICL-4300 Utilities package. The EasyLink program has the following features:

- A configuration file (INI file) determines its operation
- Supports multiple communications protocols (Modbus & BrickNet)
- Supports a configurable Textual User Interface (TUI) for viewing and editing register values
- No run-time royalties or license fees
- EasyLink configuration files can also be used in programs developed with the C or ISaGRAF Software Development Kits

Without any programming, the EasyLink program can be used to implement:

- an expansion or remote I/O module
- a "dumb" RTU (remote telemetry unit) that doesn't require any control logic
- a networked user interface display, when connected to a terminal or terminal emulator

Software Development Kits

If you need to implement any control logic, or other features that are not built into the EasyLink program, you'll need to use a software development kit. The ICL-4300 is available with the following types of software development kits (SDKs):

ICL-4300 C/C++ Standard SDK

Allows development of single tasking ICL-4300 applications using Borland C or C++.

ICL-4300 C/C++ Multitasking SDK

Provides additional components for development of multitasking ICL-4300 applications using Borland C or C++.

ICL-4300 ISaGRAF SDK

The ICL-4300 also supports the ISaGRAF IEC-1131 SDK, which allows the development of ICL-4300 applications using IEC-1131 languages. IEC-1131 is an international standard consisting of five languages:

- Sequential Function Chart
- Function Block Diagram
- Ladder Diagram
- Structured Text
- Instruction List

Purpose and Organization of this Manual

The purpose of this manual is to help you develop EasyLink configuration files, whether you are using the EasyLink program, or an application that was developed using the C SDKs. Most of the information applies to either of these two approaches. Differences are noted where appropriate. The use of EasyLink configuration files with the ISaGRAF SDK is somewhat different, and is described separately in the ISaGRAF SDK documentation.

This manual is organized into the following chapters:

Chapter 1 – Overview: The chapter you are now reading.

Chapter 2 – Installing and Running EasyLink: Describes how to install and run the EasyLink program.

Chapter 3 – Using EasyLink: Tutorial on how to use EasyLink and configuration files.

Chapter 4 – EasyLink Configuration File Reference: Complete description of the EasyLink configuration file.

Chapter 2 – Installing and Running EasyLink

Installing EasyLink

The EasyLink program (EASYLINK.EXE) is part of the ICL-4300 Utilities package. To install EasyLink, run the INSTALL.EXE program from the installation disk. The EasyLink program (EASYLINK.EXE) is normally installed in the following directory on your development PC:

```
C:\ICL4300\EASYLINK
```

A set of EasyLink sample files are installed in the directory:

```
C:\ICL4300\EASYLINK\SAMPLE
```

These sample files are referred to throughout this manual.

If you will be using an EasyLink configuration file as part of a C application, it is helpful (but not necessary) to run the EasyLink program as an aid to getting your configuration file correct.

In order to run EasyLink on your ICL-4300, you need to transfer the program to the ICL-4300. This may be accomplished using the standard RSZ utility, as described below. The RSZ utility comes pre-installed on the ICL-4300.

Using a terminal emulation program (such as HyperTerminal) connected to the console port on the ICL-4300, type the following command at the DOS prompt:

```
RSZ /R
```

Then start a Z-modem protocol transfer (sometimes called an "upload") using your terminal emulator, sending the file EASYLINK.EXE.

For additional details, see the ICL-4300 Utilities Reference Manual.

Running EasyLink

Running the EasyLink program (EASYLINK.EXE) on the ICL-4300 is relatively simple. The syntax is shown below:

```
EASYLINK filename [-p]
```

The *filename* parameter tells the EasyLink program which configuration file to use. The `-p` command line switch tells the EasyLink program to pause after reading (parsing) the configuration file, before beginning to run. This allows you to see any error messages that may have been displayed.

An example is shown below:

```
EASYLINK SAMP1.INI
```

This will cause EasyLink to read and execute the configuration file `SAMP1.INI`. The configuration file must be in the current directory on the ICL-4300 for this to work. If it is not, you need to specify a complete path, as shown below:

```
EASYLINK C:\INIFILES\SAMP1.INI
```

If you run EasyLink without any command line parameters or switches, it will display the program version number and help information.

Chapter 3 – Using EasyLink

This chapter shows you how to use EasyLink, walking you through some of the basic features using a tutorial approach. Using EasyLink primarily consists of creating an EasyLink configuration file, and modifying it until you get the desired behavior. For all the details on this configuration file, see the EasyLink Configuration File Reference chapter later in this manual.

Simple EasyLink Configuration File

Let's start by looking at a relatively simple configuration file, and walking through the various parts of it (this file is the same as the SAMP1.INI file in the samples directory):

```

;*****
; File:          SAMP1.INI
; File type:    ICL-4300 EasyLink Configuration File
; Project:      tutorial
; Description:  This file allocates registers, maps I/O to the registers, and
;              configures COM2 as a Modbus RTU Mode slave.
;*****

;=====
; REGALLOC Records
; Allocates register banks, used to store values that can be accessed by the
; I/O system, communication system, and Textual User Interface (TUI).
;=====

;-----
; Allocate 16 Boolean registers, with a bank ID of 1.
;-----
[REGALLOC]
regType=bool          ; register bank type
blockSize=16         ; number of registers to allocate in bank
bankId=1              ; number that identifies bank

;=====
; IOMAP Records
; Maps registers to physical I/O.
;=====

;-----
; Map digital inputs in slot 1 to registers in bank 1, starting with register 1.
;-----
[IOMAP]
slot=1                ; slot number of the I/O board (1-4)
sectType=DI           ; identifies the I/O type of the section being mapped
startReg=1 1         ; starting bank ID/index to associate with I/O

```

continued on next page

continued

```

;=====
; NETPORT Records
; Defines ports to be used for network communications & sets their parameters.
;=====

;-----
; Setup COM2.
;-----
[NETPORT]
name=ModbusPort           ; name to identify this record
port=COM2                 ; which port (COM1-7)
baud=115200              ; baud rate to use on the port
parity=none               ; parity for this port
dataBits=8                ; data bits for this port
stopBits=1                ; stop bits for this port

;=====
; NETSESSION Records
; Defines the protocol to be used on each NETPORT & sets protocol parameters.
;=====

;-----
; Assign Modbus RTU slave protocol, address 1, to net port.
;-----
[NETSESSION]
name=ModbusSlave          ; string that identifies this netsession
netport=ModbusPort        ; which netport to use for this session
netAddr=1                 ; network address for this unit on this netsession
protocol=ModbusRtuSlave   ; protocol to use with this netsession
regLink=DI 1              ; link bank 1 to messages that reference DI's

;*****
; End of file
;*****

```

EasyLink configuration files are created and modified with any ASCII text editor, then downloaded to the ICL-4300.

In this sample file, you'll notice that there are a number of comments. A comment starts with the semicolon ';' character, and continues to the end of the line. The file is organized as a series of records. Each record starts with a record heading enclosed in bracket characters '[']'. The content of each record consists of a series of parameters and values. For more details on the syntax, see the EasyLink Configuration File Reference chapter in this manual.

The first record defined in this sample file allocates a register bank consisting of 16 Boolean (on/off) values, with a bank ID of 1:

```

[REGALLOC]
regType=bool             ; register bank type
blockSize=16            ; number of registers to allocate in bank
bankId=1                 ; number that identifies bank

```

Register banks are used as places to store information. Registers can be associated with physical inputs, causing the input states/values to be automatically scanned and placed in the registers. Registers may also be associated with physical outputs, causing the outputs to be automatically driven based on the values stored in the registers. In addition, registers are accessible using communication protocols. Each register bank has an ID, a number greater than 0 that is used to refer to the bank. Individual registers within the bank are referenced by index. Indexes must also be greater than 0.

The next record defined in the sample file makes an association (mapping) between the allocated registers and an I/O card installed in slot 1:

```
[IOMAP]
slot=1           ; slot number of the I/O board (1-4)
sectType=DI      ; identifies the I/O type of the section being mapped
startReg=1 1     ; starting bank ID/index to associate with I/O
```

There are four I/O slots in an ICL-4300, numbered 1 through 4. In this IOMAP record, register 1 of bank 1 is mapped to the first digital input of the digital input section of the I/O card in slot 1. Some I/O cards consist of all one type of input or output, and others have several different types mixed on the same card. Additional registers are mapped in sequence to the remaining digital inputs on the I/O card.

The remaining two records in the sample configuration file set up the communications system. The NETPORT record defines and configures a serial port to be used in communications:

```
[NETPORT]
name=ModbusPort ; name to identify this record
port=COM2       ; which port (COM1-7)
baud=115200    ; baud rate to use on the port
parity=none     ; parity for this port
dataBits=8     ; data bits for this port
stopBits=1     ; stop bits for this port
```

In this case, COM2 is configured for the desired settings and a name is associated with the record so that it can be used elsewhere in the configuration file.

The last record attaches a communications protocol to the serial port:

```
[NETSESSION]
name=ModbusSlave ; string that identifies this netsession
netport=ModbusPort ; which netport to use for this session
netAddr=1        ; network address for this unit on this netsession
protocol=ModbusRtuSlave ; protocol to use with this netsession
regLink=DI 1     ; link bank 1 to messages that reference DI's
```

This NETSESSION record causes the ICL-4300 to act as a Modbus RTU mode slave on COM2. It references the port that was defined in the previous record, by using its name (ModbusPort). The NETSESSION record is also given a name (ModbusSlave), but in this sample configuration file, the name is not used and therefore not really required. Slave address 1 is associated with the ICL-4300 on this COM port. The last line in the NETSESSION record makes an association between register bank 1 and DI (digital input) registers. Whenever a Modbus master reads digital input registers from this ICL-4300, it will access bank 1. Multiple regLink associations can be defined for one NETSESSION.

Four different register types are defined for Modbus, and may be used in regLink associations:

Register Type	Abbreviation
Analog Input	AI
Analog Output	AO
Digital Input	DI
Digital Output	DO

In summary, this sample configuration sets up the ICL-4300 as a Modbus slave device, capable of reading digital inputs from an I/O card in slot 1, and communicating the digital input values to a Modbus master. In this configuration, the ICL-4300 could be used as an expansion I/O or remote I/O module.

Now let's look at a slightly more complex example.

Modbus Master -- Reading

The following configuration file sets up the ICL-4300 as a Modbus master. It polls a slave device on COM2 for some digital and analog inputs. The input information received from the slave is sent to the digital outputs and analog outputs on the master ICL-4300. This application is a reasonably common one: multiplexing discrete analog and digital signals over a communication link from one unit to another. It can be used to reduce wiring costs, when compared with wiring the individual analog and digital signals. The communication link could be one of several different media, including RS-232, RS-485 and radio.

```

;*****
; File:          SAMP2.INI
; File type:    ICL-4300 EasyLink Configuration File
; Project:     tutorial
; Description:  This file allocates registers, maps I/O to the registers, and
;              configures COM2 as a Modbus RTU Mode Master. The master polls a
;              slave unit for some digital inputs and analog inputs. These
;              input values are then sent to the local digital and analog
;              outputs (because the registers are mapped to the outputs).
;*****
;=====
; REGALLOC Records
; Allocates register banks, used to store values that can be accessed by the
; I/O system, communication system, and Textual User Interface (TUI).
;=====
;-----
; Allocate 16 Boolean registers, with a bank ID of 1.
;-----
[REGALLOC]
regType=bool          ; register bank type
blockSize=16         ; number of registers to allocate in bank
bankId=1              ; number that identifies bank
;-----
; Allocate 16 analog registers, with a bank ID of 2.
;-----
[REGALLOC]
regType=int16         ; register bank type
blockSize=16         ; number of registers to allocate in bank
bankId=2              ; number that identifies bank
;=====
; IOMAP Records
; Maps registers to physical I/O.
;=====
;-----
; Map digital outputs in slot 1 to registers in bank 1, starting w/register 1.
;-----
[IOMAP]
slot=1                ; slot number of the I/O board (1-4)
sectType=DO           ; identifies the I/O type of the section being mapped
startReg=1 1         ; starting bank ID/index to associate with I/O

```

continued on next page

continued

```

;-----
; Map analog outputs in slot 2 to registers in bank 2, starting w/register 1.
;-----
[IOMAP]
slot=2           ; slot number of the I/O board (1-4)
sectType=AO      ; identifies the I/O type of the section being mapped
startReg=2 1     ; starting bank ID/index to associate with I/O

;=====
; IOSCALE Records
; Defines scaling to be used for analog inputs and outputs.
;=====

;-----
; Scale analog outputs to a range of 0 to 20000. (microamps)
;-----
[IOSCALE]
startReg=2 1     ; bankId and index
range=0 20000   ; min/max engineering units
blockSize=16    ; number of registers to scale
calMode=0to20mA ; calibration mode

;=====
; NETPORT Records
; Defines ports to be used for network communications & sets their parameters.
;=====

;-----
; Setup COM2.
;-----
[NETPORT]
name=ModbusPort ; name to identify this record
port=COM2       ; which port (COM1-7)
baud=115200     ; baud rate to use on the port
parity=none     ; parity for this port
dataBits=8      ; data bits for this port
stopBits=1     ; stop bits for this port

;=====
; NETSESSION Records
; Defines the protocol to be used on each NETPORT & sets protocol parameters.
;=====

;-----
; Assign Modbus RTU Master protocol to net port.
;-----
[NETSESSION]
name=ModbusMaster ; string that identifies this netsession
netport=ModbusPort ; which netport to use for this session
protocol=ModbusRtuMaster ; protocol to use with this netsession

```

continued on next page

continued

```

;=====
; NETEVENT Records
; Defines messages to read and write information over the network.
;=====

;-----
; Poll slave for digital inputs 1-16, store locally in bank 1, registers 1-16.
;-----
[NETEVENT]
event=cyclic 1          ; event type [every cycle]
action=read           ; event action to take
blockSize=16          ; number of registers to get
src=DI 1              ; starting point on slave
dst=1 1               ; where to put received data
netSession=ModbusMaster ; netSession to use to execute this netevent
netAddr=1             ; network address of remote unit

;-----
; Poll slave for analog inputs 1-16, store locally in bank 2, registers 1-16.
;-----
[NETEVENT]
event=cyclic 1          ; event type [every cycle]
action=read           ; event action to take
blockSize=16          ; number of registers to get
src=AI 1              ; starting point on slave
dst=2 1               ; where to put received data
netSession=ModbusMaster ; netSession to use to execute this netevent
netAddr=1             ; network address of remote unit

;*****
; End of file
;*****

```

As before, registers are allocated with REGALLOC records. You will notice that a new data type, `int16`, is introduced in the second register bank allocation. As you may have guessed, the `int16` type is a 16-bit signed integer. For a listing of all available data types, please see the Configuration File Reference chapter in this manual.

In this example, you can see that the digital outputs on the I/O card in slot 1 are mapped to bank 1, starting with register 1. The analog outputs on the I/O card in slot 2 are mapped to bank 2, starting with register 1.

The analog output registers are scaled, using the IOSCALE record. Scaling makes it so the registers associated with the analog outputs may be accessed in engineering units instead of arbitrary device units. In this case, the scaling is set up so that the full range of the analog outputs (0.00 to 20.00 mA) corresponds to a scaled range of 0 to 20,000 (microamps). If the registers were not scaled, the full range of values would be 0-4095, corresponding to the 12-bit digital to analog converter that is used by the hardware. In the IOSCALE record, the entry `calMode=0to20mA` selects the calibration mode to be used on the I/O board. In general, each analog I/O board contains two calibration tables (for example, one for current mode and one for voltage mode). When the I/O boards are manufactured they are calibrated for these two different modes, and the calibration data is stored on the I/O board itself in a non-volatile memory device (EEPROM). Setting the `calMode` selects between these two different tables, or may be used to ignore the tables altogether (`calMode=none`). For the different calibration modes associated with each I/O board, see the EasyLink Configuration File Reference chapter. Registers associated with analog inputs may be scaled in the same manner as registers associated with analog outputs.

Notice that the protocol is defined as Modbus RTU Master in the NETSESSION record for this example. Also, there is no network address required in the NETSESSION record, because a Modbus master has no address. In general, some parameters are optional in each record type. If a parameter is not specified, it defaults to a reasonable value. The Configuration File Reference chapter indicates for each parameter whether it is optional, and what the default values are. No regLink association is needed for a Modbus master NETSESSION. This is because a Modbus

master does not receive any unsolicited messages from the slaves (there can be only one master on a given communication link). Any messages generated by the master are configured on the master, and contain all the information necessary without using the `regLink` parameter.

A new type of record, the `NETEVENT` record, is shown at the end of this sample file. `NETEVENT` records define communication messages and the flow of information between the local and remote units. Notice that the `NETEVENT` is associated with a `NETSESSION`, by referencing the `NETSESSION` name.

In this example, the first `NETEVENT` defines a read message that polls slave address 1 for a block of 16 digital inputs, starting with DI 1. When the information is received, it is stored in bank 1, starting at register 1. The event parameter determines when this message will be triggered. In this case, it will be triggered on every event cycle, as will the second `NETEVENT`. An event cycle is complete when all pending `NETEVENTS` for a given port are serviced. Consequently, the ICL-4300 will alternate between the two `NETEVENTS`.

The second `NETEVENT` defines a read message that polls slave address 1 for a block of 16 analog input values, starting with AI 1. The information received is to be stored in bank 2, starting with register 1.

When *reading* information from a Modbus slave, the type of message generated depends on the `src` parameter in the `NETEVENT`. The source of information can be one of four different register types:

Register Type	Abbreviation
Analog Input	AI
Analog Output	AO
Digital Input	DI
Digital Output	DO

As you will see in the next section, when you *write* to a Modbus slave, the `dst` parameter determines the type of message generated. The general rule is that a remote Modbus slave device is referred to in terms of register types. The local ICL-4300 is referred to in terms of bank IDs.

Modbus Master -- Writing

Imagine that we also wanted to *write* information from the ICL-4300 master to the Modbus slave. How would we accomplish that? We could add the following two NETEVENTS to our previous example (see the resulting file SAMP3.INI):

```
[NETEVENT]
event=cyclic 1           ; event type [every cycle]
action=write            ; event action to take
blockSize=16           ; number of registers to write
src=1 17               ; where to get information on master
dst=DO 1               ; where to send information on slave
netSession=ModbusMaster ; netSession to use to execute this netevent
netAddr=1              ; network address of remote unit

[NETEVENT]
event=cyclic 1           ; event type [every cycle]
action=write            ; event action to take
blockSize=16           ; number of registers to write
src=2 17               ; where to get information on master
dst=AO 1               ; where to send information on slave
netSession=ModbusMaster ; netSession to use to execute this netevent
netAddr=1              ; network address of remote unit
```

The first NETEVENT causes 16 Boolean registers to be written from bank 1, starting at register 17, to slave 1. The information is to be stored on the slave starting at digital output 1. The second NETEVENT causes 16 integer registers to be written from bank 2, starting at register 17, to slave 1. The information is to be stored on the slave starting at analog output 1. Both NETEVENTS are to be serviced every cycle (*event=cyclic 1*), giving each equal priority. If you wish to lower the relative priority of an event, so that it happens less frequently, use a higher cycle count value. For example, if you set one event up as *cyclic 3*, it will happen one-third as often as those events that are set up as *cyclic 1*.

Time Based Net Events

In addition to cycle-based NETEVENTS (i.e. *event=cyclic 1*), NETEVENTS may be activated based on time. Suppose, for example, that you wanted to read some information once every ten seconds. The following NETEVENT would accomplish this:

```
[NETEVENT]
event=timerSec 10      ; event type (every 10 seconds)
action=read            ; event action to take
.
.
.
```

Time values are given in seconds. There is no guarantee that the message will be transmitted exactly every ten seconds – this depends on system load. When a time-based event is "triggered" (the time has expired and it is ready to be serviced), it will be serviced at the next available opportunity, as the communications system cycles through the list of NETEVENTS.

Triggers

NETEVENTS can also be based on triggers. Triggers are defined elsewhere in the configuration file and may be referenced by name in one or more NETEVENTS. Here is an example of a trigger that is based on change of a group of integer registers (see the file SAMP4.INI):

```
[TRIGGER]
name=DeltaAITrig           ; user definable trigger name
trig=delta 10             ; trigger type [change in value by 10]
startReg=3 1              ; starting bank ID/register to monitor for change
blockSize=16              ; number of registers to monitor for change
```

The trigger will be activated whenever any of registers 1-16 in bank 3 change by 10 or more. The change is detected using a comparison value that is stored for each register. The comparison values are stored each time the trigger is activated. The trigger may be used in a NETEVENT, as follows:

```
[NETEVENT]
event=trig DeltaAITrig    ; event type [trigger based]
action=write              ; event action to take
blockSize=16              ; number of registers to write
src=3 1                   ; where to get information on master
dst=AO 1                  ; where to send information on slave
netSession=ModbusMaster   ; netSession to use to execute this netevent
netAddr=1                 ; network address of remote unit
```

As you can see, the trigger is referenced by its name. The result of this trigger and NETEVENT combination is that any time one or more of the registers change by 10 or more, the register block will be transmitted. This approach to communications has powerful implications. Instead of constantly sending data, when no significant changes are occurring, the communications system can be set up to transmit data only on change, thus dramatically reducing communication traffic. This approach can be especially effective with a protocol that supports peer-to-peer communications, such as the BrickNet protocol. BrickNet is supported on the ICL-4300, and is described further in this manual.

Net Events With Multiple Conditions

In addition to the simple NETEVENTS that have been shown so far, more complex NETEVENTS can be created in which multiple conditions are specified which cause the event to be triggered. The example below (see the file SAMP5.INI) shows how you can set up such a NETEVENT:

```
[NETEVENT]
event=trig DeltaAITrig    ; trigger this event based on a defined trigger
event=startup             ; trigger this event on application startup
event=timerSec 60         ; trigger this event every 60 seconds
action=write              ; event action to take
blockSize=10              ; number of registers to write
src=1 1                   ; where to get information on master
dst=AO 1                  ; where to send information on slave
netSession=ModbusMaster   ; netSession to use to execute this netevent
netAddr=1                 ; network address of remote unit
```

BrickNet Communications

You may use the BrickNet communications protocol in much the same way as Modbus, but with substantially more flexibility. The BrickNet protocol is designed as a peer-to-peer protocol, which allows any network node to originate messages and send/receive information. It has features that make it well suited for use on many different types of media, including radio, dial-up telephone, leased-line, RS-485 and RS-232. It supports complex message routing schemes (sometimes referred to as store and forward). Multiple data types are supported: bit, integer, floating point, and buffer.

A simple BrickNet EasyLink configuration file can be found in the file `SAMP6.INI`. It is similar to a Modbus master configuration file, with some differences as noted in below.

The register allocation record looks like this:

```
[REGALLOC]
regType=float           ; register bank type
blockSize=32           ; number of registers to allocate in bank
bankId=1                ; number that identifies bank
```

Notice a new register type has been introduced (`float`). The BrickNet protocol directly supports floating point numbers (both single precision and double precision). For double precision, use the data type `double`.

As shown below, `NETPORT` records are defined in the same way as for Modbus protocol.

```
[NETPORT]
name=BrickNetPort      ; name to identify this record
port=COM4              ; which port (COM1-7)
baud=19200             ; baud rate to use on the port
parity=odd             ; parity for this port
dataBits=8            ; data bits for this port
stopBits=1            ; stop bits for this port
```

Notice also, with the BrickNet protocol, each node has a network address. As shown in the `NETSESSION` below, the address assigned to this ICL-4300 port is 1.

```
[NETSESSION]
name=BrickNetSession  ; string that identifies this netsession
netport=BrickNetPort  ; which netport to use for this session
protocol=BrickNet     ; protocol to use with this netsession
netAddr=1             ; network address for this unit on this netsession
```

As shown below, when referring to registers in BrickNet `NETEVENTS`, both source and destination use a bank ID and index.

```
[NETEVENT]
event=trig DeltaAITrig ; trigger this event based on a defined trigger
event=startup          ; trigger this event on application startup
event=timerSec 60     ; trigger this event every 60 seconds
action=write           ; event action to take
blockSize=16          ; number of registers to write
src=1 1               ; where to get information on this ICL-4300
dst=1 1               ; where to send information on remote ICL-4300
netSession=BrickNetSession ; netSession to use to execute this netevent
netAddr=100           ; network address of remote unit
```

When configuring BrickNet communications, make sure that the data type of both the source and destination banks is the same. If the data type does not match properly on both ends, the transfer of information will not take place. The receiving node will generate an error response message, and the transmitting node will display/log the error message.

Sending "Non-Standard" Data Types Using Modbus

The Modbus protocol was designed to transport only Boolean (bit) and 16-bit integer information. Sometimes it is desirable in an application, however, to send other data types, such as floating point and 32-bit integer values. The ICL-4300 provides a mechanism for doing this. Non-standard data types may be encoded by the sender within 16-bit integer registers in a Modbus message and be decoded in the same way by the receiver. Other Modbus implementations (SCADA systems, operator interfaces, etc.) implement similar schemes. There is not complete agreement on the byte ordering within the Modbus messages, however, so the ICL-4300 supports two different byte orderings (LSB first, and MSB first).

This mechanism does not apply to BrickNet protocol, because the BrickNet protocol is designed to transmit all of the different data types without requiring any special "tricks."

Two sample configuration files are provided which illustrate transmitting data of type double from a Modbus master to a Modbus slave. The master configuration file is `SAMP7M.INI` and the slave configuration file is `SAMP7S.INI`. Portions of each file are shown below.

In `SAMP7M.INI`:

```
[REGALLOC]
regType=double           ; register bank type
blockSize=8             ; number of registers to allocate in bank
bankId=1                ; number that identifies bank

[NETPORT]
name=ModbusPort         ; name to identify this record
port=COM2               ; which port (COM1-7)
baud=9600               ; baud rate to use on the port
parity=none             ; parity for this port
dataBits=8              ; data bits for this port
stopBits=1             ; stop bits for this port

[NETSESSION]
name=ModbusMaster       ; string that identifies this netsession
netport=ModbusPort      ; which netport to use for this session
protocol=ModbusRtuMaster ; protocol to use with this netsession
regMode=MsbFirst       ; encode register data into messages MSB first

[NETEVENT]
event=timerSec 10      ; trigger this event every 10 seconds
action=write           ; event action to take
blockSize=8           ; number of source registers (doubles) to write
src=1 1               ; where to get information on master
dst=A0 1              ; where to send information on slave
netSession=ModbusMaster ; netSession to use to execute this netevent
netAddr=1             ; network address of remote unit
```

The master is configured to have a register bank of type double. It sends 8 registers from this bank to the slave. The registers are encoded into the messages MSB (most significant byte) first, because `regMode` is set to `MsbFirst`. If you wish to use LSB first ordering, set `regMode` equal to `LsbFirst`.

In SAMP7S.INI:

```
[REGALLOC]
regType=double           ; register bank type
blockSize=8              ; number of registers to allocate in bank
bankId=1                 ; number that identifies bank

[NETPORT]
name=ModbusPort          ; name to identify this record
port=COM2                ; which port (COM1-7)
baud=9600                ; baud rate to use on the port
parity=none              ; parity for this port
dataBits=8               ; data bits for this port
stopBits=1              ; stop bits for this port

[NETSESSION]
name=ModbusSlave         ; string that identifies this netsession
netport=ModbusPort       ; which netport to use for this session
netAddr=1                ; network address for this unit on this netsession
protocol=ModbusRtuSlave  ; protocol to use with this netsession
regLink=AO 1             ; link bank 1 to messages that reference AO's
regMode=MsbFirst        ; encode register data into messages MSB first
```

The slave configuration file contains a `NETSESSION` entry which associates messages that reference AO's with bank 1 (a bank of doubles). It also sets `regMode` equal to `MsbFirst`, so that the same byte ordering is agreed-upon by master and slave. This agreement is necessary or the information will be scrambled.

Textual User Interface (TUI)

The textual user interface (TUI) is a powerful feature that allows you to define a user interface that operates through one or more of the serial ports on the ICL-4300. To use the TUI, you must connect a terminal or terminal emulator to the TUI port. This terminal must be capable of interpreting and displaying standard ANSI-BBS control codes. For example, you may use the HyperTerminal program that comes with Windows 95, running on a PC. Also, you may use a dedicated user interface terminal (such as a 4 by 20, character-mode LCD display that operates as a "dumb terminal").

The TUI is capable of displaying data from any register bank that has been defined. This includes all the different data types. Multiple display pages can be defined, which consist of a background (text that does not change) overlaid with fields that contain register information.

Configuring the TUI

To configure the TUI, you set up the port, pages and fields in the configuration file. Text files are independently created to represent the background of each page.

A simple example is given in the files `SAMP8.INI` (configuration file) and `SAMP8P1.TXT` (page 1 background file). For reference, they are listed in their entirety here.

`SAMP8.INI`:

```

;*****
; File:          SAMP8.INI
; File type:    ICL-4300 EasyLink Configuration File
; Project:      tutorial
; Description:  This file illustrates use of the TUI. Analog inputs 1-16 are
;              displayed
;*****

;=====
; REGALLOC Records
; Allocates register banks, used to store values that can be accessed by the
; I/O system, communication system, and Textual User Interface (TUI).
;=====

;-----
; Allocate 16 16-bit integer registers, with a bank ID of 1.
;-----
[REGALLOC]
regType=int16          ; register bank type
blockSize=16          ; number of registers to allocate in bank
bankId=1              ; number that identifies bank

```

continued on next page

continued

```

;=====
; IOMAP Records
; Maps registers to physical I/O.
;=====

;-----
; Map analog inputs in slot 1 to registers in bank 1, starting with register 1.
;-----
[IOMAP]
slot=1           ; slot number of the I/O board (1-4)
sectType=AI      ; identifies the I/O type of the section being mapped
startReg=1 1     ; starting bank ID/index to associate with I/O

;=====
; NETPORT Records
; Defines ports to be used for network communications & sets their parameters.
;=====

;-----
; Setup COM1.
;-----
[NETPORT]
name=TuiPort     ; name to identify this record
port=COM1        ; which port (COM1-7)
baud=115200      ; baud rate to use on the port
parity=none      ; parity for this port
dataBits=8       ; data bits for this port
stopBits=1       ; stop bits for this port

;=====
; TUI Records
; Defines serial ports to be used for the textual user interface (TUI).
;=====

;-----
; Configure COM1 for use with TUI.
;-----
[TUI]
name=MyTui       ; name to associate with this TUI
netPort=TuiPort  ; network port name
header="TUI SAMPLE" ; string displayed on top line of all pages

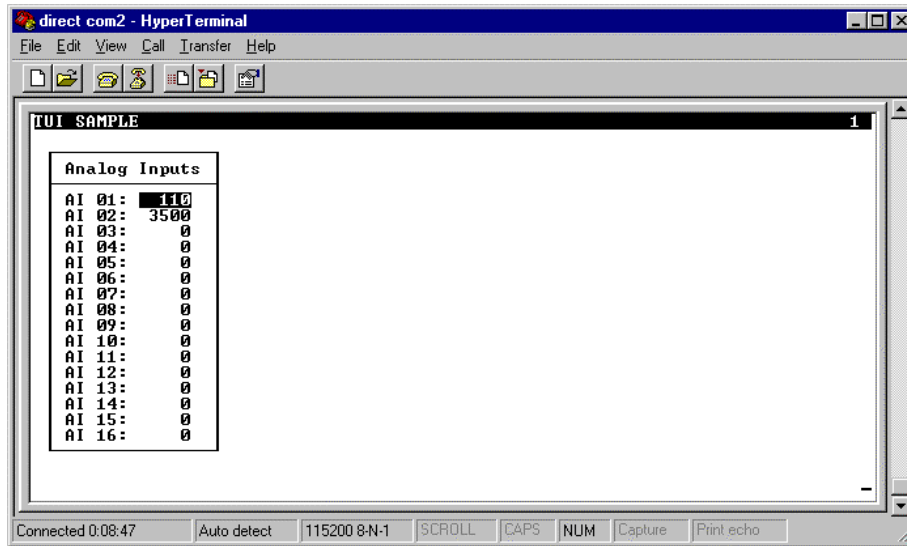
;=====
; TUIPAGE Records
; Defines TUI pages.
;=====

;-----
; Define page 1.
;-----
[TUIPAGE]
name=PAGE1       ; TUI page name
tui=MyTui        ; name of TUI to use for this page
pageNum=1        ; TUI page number
background=SAMP8P1.TXT ; page background filename

```

continued on next page

When viewed with HyperTerminal, executing on an ICL-4300, the sample looks like this:



Let's look at each of the TUI records from `SAMP8.INI` in more detail.

```
[TUI]
name=TuiPort
netPort=TuiNetPort
header="TUI SAMPLE"
```

Each textual user interface that is defined (you may have more than 1, on different serial ports) must have a name. This name is used in the other records to identify the TUI. The defined header string is displayed at the top of each TUI page, and is typically used for title information.

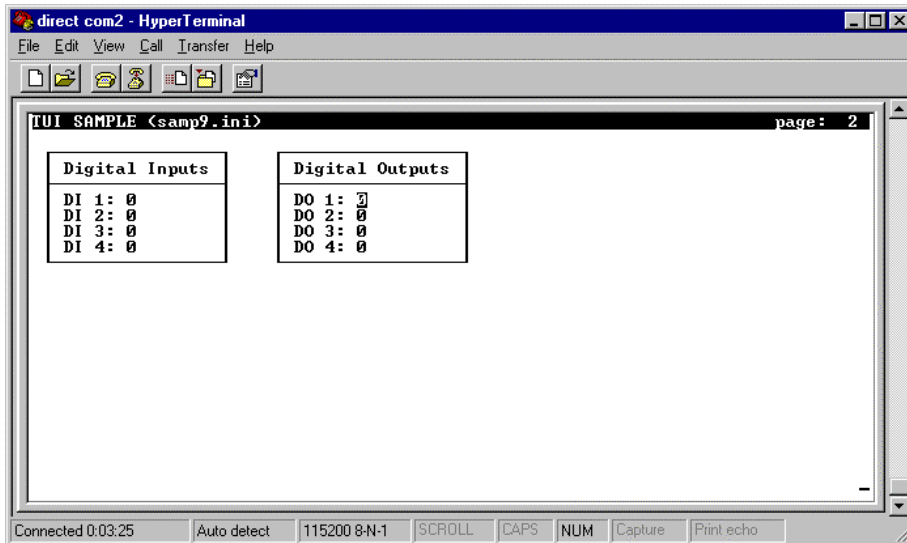
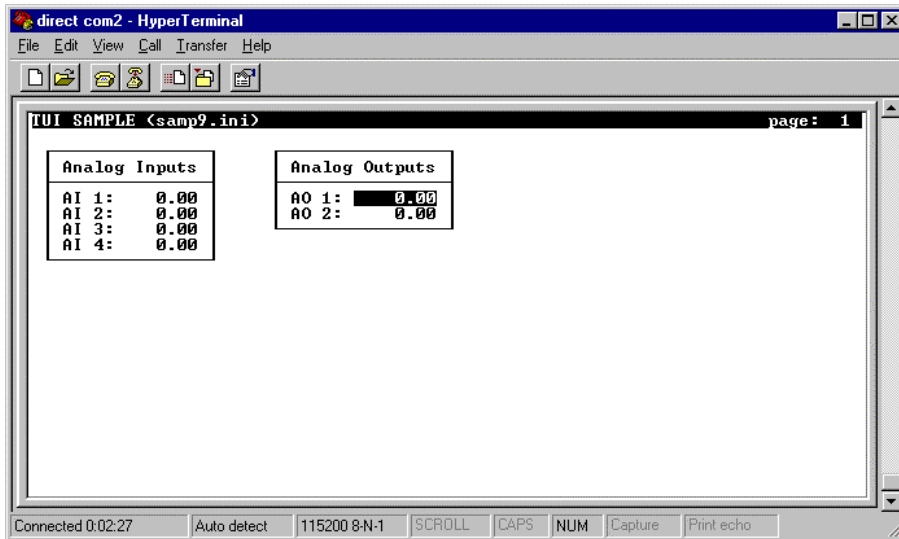
```
[TUIPAGE]
name=PAGE1           ; TUI page name
tui=MyTui            ; name of TUI to use for this page
pageNum=1            ; TUI page number
background=SAMP8P1.TXT ; page background filename
```

This `TUIPAGE` record is identified with the name `PAGE1`, which is referred to in the `TUIFIELD` record. Each `TUIPAGE` record must have a name. The second line of this record (`tui=MyTui`) identifies which TUI this record pertains to. Each page has a page number, which allows the user to navigate between pages, using the 'P' key, followed by the page number. The last line in this record (`background=SAMP8P1.TXT`) defines the file to be used as the background for this page. The `background` field is optional.

```
[TUIFIELD]
page=PAGE1           ; name of page on which to display this field
position=12 6        ; x/y coordinates of field data starting point
startReg=1 1         ; bankId and index
blockSize=16         ; number of registers to display vertically
format="AI ?: #####" ; specify tag and format for registers
```

This record causes a block of 16 values to be displayed, starting with register 1 from bank 1. The position is given in x/y coordinates, with the upper left-hand corner of the display corresponding to position 1/1. The `format` parameter controls the formatting of each data field. The '?' character is replaced with the register index, and the '#' characters are replaced with the register value. For floating point numbers you may include a decimal point and indicate the number of decimal digits which should be displayed (such as `#####.##`).

Another sample, represented by the files SAMP9 . INI (configuration file), SAMP9P1 . TXT and SAMP9P2 . TXT (page backgrounds), illustrates additional data types and features of the TUI. This sample has two pages defined, with two groups of information being displayed on each page. The two pages look like this:



Let's look at some of the records that make up this sample (SAMP9P1 . INI).

```
[TUI]
name=MyTui           ; name to associate with this TUI
netPort=TuiPort     ; network port name
header="TUI SAMPLE (samp9.ini)           page:"
refreshSec=10       ; how often to completely refresh display
scanRateMs=10       ; how often to scan fields for change
```

Notice that the header parameter contains the text "page" to label the current page number. By default, the current page number is shown in the upper right hand corner of the display. If you wish to move the page number to a different location, use the `pageNumLoc` parameter. You will notice two parameters: `refreshSec` and `scanRateMs`. Periodically, the TUI refreshes the entire display. By default, this happens every 30 seconds. If you wish to have this happen more or less frequently, you can change it with the `refreshSec` parameter. To disable this periodic refreshing, set this parameter to 0. To control how often the system checks registers to determine if the display needs updating, you may adjust the `scanRateMs` parameter. By default, the system checks all displayed registers every 100 milliseconds.

One of the `TUIFIELD` records from the file `SAMP9 . INI` looks like this:

```
[TUIFIELD]
page=PAGE1           ; name of page on which to display this field
position=4 6         ; x/y coordinates of field data starting point
startReg=1 1         ; bankId and index
blockSize=4          ; number of registers to display vertically
format="AI ? : ####.##" ; specify register display format
readOnly=yes         ; make field read-only (can't change inputs)
```

Notice the parameter `readOnly`, which controls whether or not the user is allowed to enter numbers into the field. In this, case the field is set as read-only, because the data being displaying is coming from analog inputs, and it would not make any sense to write to an analog input. The default value for this parameter is `no`, so if you want to allow the user to write to a field, you may omit this parameter.

Using the TUI

To switch between pages, you may use any of the following keys:

P <num> Enter	switch to a specific page number (1, 2, 3...)
N	switch to the next page
L	switch to the last (previous) page

To move between fields, use the Tab key or up/down arrow keys.

To edit a displayed data field, press the Enter key to access edit mode. Once you are in edit mode, you may use the following keys:

- numbers, the minus sign, and the decimal point for numeric fields
- alphanumeric characters are accepted for buffer fields
- the Delete key clears the field
- the Backspace removes character to the left of cursor
- Left/right arrow keys move the cursor between digits
- Up/down arrow keys change individual digits/characters
- the Escape key restores the last value and exits edit mode

When you are finished editing the field, press Enter to accept the new value, or Escape to cancel your changes and restore the previous value.

Pressing the Escape key when you are not in edit mode will cause the current page to refresh (redraw).

Defines

"Defines" are a feature that can be used to improve readability and maintainability of your configuration files. The use of defines is entirely optional. A define consists of a symbol and an associated value. Whenever the symbol appears in the configuration file, its defined value is substituted. An example taken from the sample file `SAMP10.INI` will help clarify the idea:

```
[DEFINES]
BOOL_BANK=1                ; bank ID for Boolean bank
BOOL_BLOCK_SIZE=16        ; Boolean bank block size
SLAVE_ADDRESS=1           ; Modbus address of slave

[REGALLOC]
regType=bool              ; register bank type
blockSize=BOOL_BLOCK_SIZE ; number of registers to allocate in bank
bankId=BOOL_BANK         ; number that identifies bank

[NETEVENT]
event=cyclic 1            ; event type [every cycle]
action=read              ; event action to take
blockSize=BOOL_BLOCK_SIZE ; number of registers to get
src=DI 1                 ; starting point on slave
dst=BOOL_BANK 1          ; where to put received data
netSession=ModbusMaster  ; netSession to use to execute this netevent
netAddr=SLAVE_ADDRESS    ; network address of remote unit
```

In this case, the symbol `BOOL_BANK` is defined with a value of 1. Wherever `BOOL_BANK` appears in the rest of the configuration file, its value is substituted. The same type of substitution occurs with the symbols `BOOL_BLOCK_SIZE` and `SLAVE_ADDRESS`. One advantage of using defines is that it allows you to specify in one location values that might change in the future, instead of scattering the values throughout the entire configuration file. When you need to change the value, you can simply change it in one place, instead of multiple places.

Defaults

Defaults are another feature (like defines) which are not required, but which can make creating and maintaining configuration files easier. In a configuration file record, whenever you omit an optional parameter, that parameter takes on a default value. The default feature allows you to specify what the default value should be for omitted parameters. The sample file `SAMP11.INI` illustrates the use of defaults. An excerpt is shown here:

```
[DEFAULTS]

; netPort parameters -----
baud=19200           ; set to the default baud rate
dataBits=8           ; set to the default number of data bits
stopBits=1          ; set the default number of stop bits
parity=even         ; set the default parity

[NETPORT]
name=ModbusPort1    ; name to identify this record
port=COM1           ; which port (COM1-7)
```

By setting up appropriate defaults for baud, data bits, stop bits and parity, these parameters do not have to be specified for each net port record. As a result, the entire configuration file is shorter and contains less redundant information.

For a complete list of all the parameters that can be defaulted, and what their normal default values are, see the EasyLink Configuration File Reference chapter.

Options

Miscellaneous settings can be configured using the `OPTIONS` record, as shown below:

```
[OPTIONS]
logDisplay=no      ; 'yes' if error log messages should be displayed to console
aiAverage=15      ; set analog input averaging to 15 for all slots & channels
aiAverage=3 2 5   ; set analog input averaging to 3 for slot 2, channel 5
```

The `logDisplay` parameter indicates whether system error log messages should be displayed to the console (typically COM1). The default setting is `yes`.

The `aiAverage` parameter controls the software filtering (or "averaging") of analog inputs. The averaging may be set individually for each channel. If you do not override the analog input averaging, it defaults to a count of 5, which means that 5 consecutive samples are averaged together for each reading that is presented. The averaging is specified as a sequence of three numbers. The first number is the averaging count. The second number is the slot. The third number is the channel. If you omit the channel, the averaging setting applies to all channels in the slot. If you omit the slot and channel, the averaging setting applies to all channels in all slots. If multiple `aiAverage` parameters are specified, the parameters that appear later override the previous parameters. This effect is used in the example shown above.

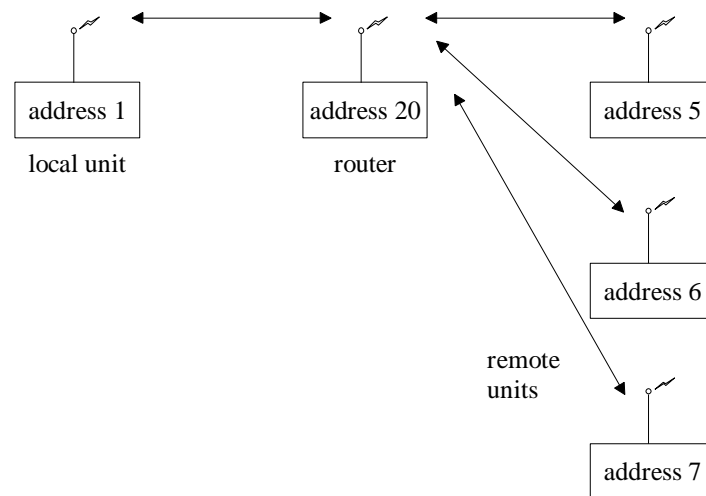
Message Routing

The ICL-4300 is capable of acting as a message router. A message received can be redirected to another node on the network, using either the same port or a different port.

A routing table determines message routing operation, and is set up using `NETROUTE` records. The routing table determines which messages should be routed and where they should be sent.

BrickNet Routing Example

An example of BrickNet protocol routing is shown below. In this example a radio communications system is used. The local station would like to communicate with each of the remote stations directly, but due to signal propagation limitations of the radios and antennas used, direct communication is not possible with the remote stations. Therefore, an intermediate station with which the local station *can* directly communicate is set up as a router between the local stations and the remote stations. In addition, direct communication may occur between remote stations, which should not be routed.



Three sample files correspond with this example:

SAMP12A.INI configuration file for the local unit, address 1
 SAMP12B.INI configuration file for the router, address 20
 SAMP12C.INI configuration file for one of the remote units, address 5

Excerpt from SAMP12A.INI (local unit):

```
[NETSESSION]
name=BrickNetSession      ; string that identifies this netsession
netport=BrickNetPort      ; which netport to use for this session
protocol=BrickNet         ; protocol to use with this netsession
netAddr=1                 ; network address for this unit on this netsession
pathID=1                  ; routing path identifier for transmitted messages

[NETEVENT]
event=timerSec 5          ; trigger this event every 5 seconds
action=write              ; event action to take
blockSize=16              ; number of registers to write
src=1 1                   ; where to get information on this ICL-4300
dst=1 1                   ; where to send information on remote ICL-4300
netSession=BrickNetSession ; netSession to use to execute this netevent
netAddr=5                 ; network address of remote unit
```

Notice the parameter `pathID` in the `NETSESSION` record. When messages are sent using this `NETSESSION`, the path ID byte in the BrickNet message will take on a value of 1. Unit 20 is configured to route messages with a path ID of 1. Any other messages will not be routed. In the `NETEVENT` record above, the destination network address is given as 5. Notice that this is the address of one of the remote units, which is the ultimate destination. The `NETEVENT` is not addressed to the intermediate destination (the router, unit 20).

Excerpt from SAMP12B.INI (router):

```
[NETROUTE]
routeType=peer            ; routing entry type
inSes=BrickNetSession     ; inbound network session name
inAddrDst=5 7             ; messages destined for addresses 5 through 7
inAddrSrc=1               ; messages received from address 1
pathId=1                  ; route only messages with a path ID of 1

[NETROUTE]
routeType=peer            ; routing entry type
inSes=BrickNetSession     ; inbound network session name
inAddrDst=1               ; messages destined for address 1
inAddrSrc=5 7             ; messages received from addresses 5 through 7
pathId=1                  ; route only messages with a path ID of 1
```

These two `NETROUTE` records set up an outbound and return path for messages between the local unit and remote units. The `routeType` parameter must be set to `peer` for BrickNet protocol. The first `NETROUTE` record applies to received messages with a final destination address of 5, 6 or 7, and an intermediate source address of 1. The intermediate source address is the unit from which the message last came (in this case it is the same as the originator, since there are no additional hops in the route). The second `NETROUTE` record applies to received messages with a final destination address of 1, and an intermediate source address of 5, 6 or 7. Notice that in both cases, the `pathId` parameter is set to 1. This means that only messages with a path ID of 1 will be routed. This mechanism helps the BrickNet protocol discriminate between messages that should be routed and those that should not be routed. Local message traffic between units 5, 6 and 7 would have a path ID of 0, and therefore would not be routed.

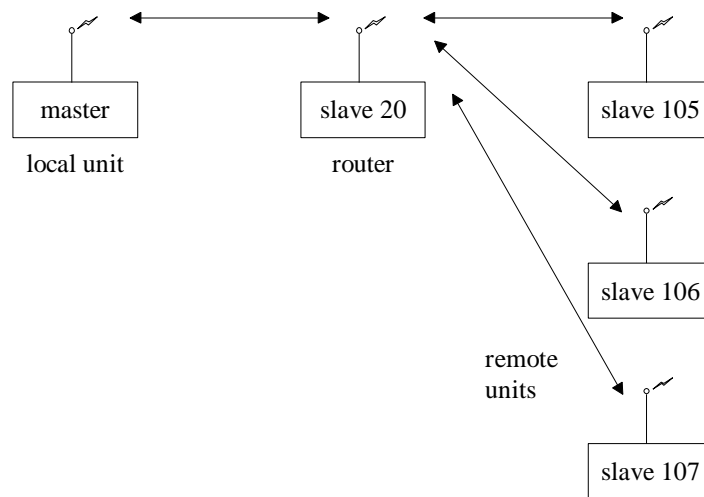
Exerpt from SAMP12C.INI (remote unit):

```
[NETSESSION]
name=BrickNetSession      ; string that identifies this netsession
netport=BrickNetPort      ; which netport to use for this session
protocol=BrickNet         ; protocol to use with this netsession
netAddr=5                 ; network address for this unit on this netsession
```

Nothing particularly unusual is seen in the `NETSESSION` record for the remote unit. It is sufficient to simply set the address to 5. Configuring a path ID on this `NETSESSION` is not necessary. When this unit generates a message in response to an incoming command from unit 1, it will use the path ID that was contained in the incoming message. This allows the response to be routed back to the originator using the same path.

Modbus Routing Example

An example of routing using the Modbus protocol is shown below. It is similar to the previous example in that a radio system is used, with an intermediate station acting as a router between a local unit and several remote units. Because of the way the Modbus protocol is designed, address translation must be used so that when the master hears the router re-transmitting a message, the master will not interpret this as a response to its own message. The master must, however, be able to reject messages that do not have the expected slave address. The router translates incoming messages from the master such that address 5 translates to address 105, 6 translates to 106 and 7 translates to 107. When the router receives responses from the slaves, it translates the addresses in the reverse. Address 105 translates to address 5, 106 translates to 6 and 107 translates to 7. Thus, as far as the master is concerned, it is communicating with slaves 5, 6 and 7.



Three sample files correspond with this example:

SAMP13A.INI configuration file for the Master
 SAMP13B.INI configuration file for the router, address 20
 SAMP13C.INI configuration file for one of the remote units, address 105

Excerpt from SAMP13A.INI (master):

```
[REGALLOC]
regType=int16           ; register bank type
blockSize=32           ; number of registers to allocate in bank
bankId=1                ; number that identifies bank

[NETSESSION]
name=ModbusSession     ; string that identifies this netsession
netport=ModbusPort     ; which netport to use for this session
protocol=ModbusRtuMaster ; protocol to use with this netsession

[NETEVENT]
event=timerSec 5       ; trigger this event every 5 seconds
action=write           ; event action to take
blockSize=32           ; number of registers to write
src=1 1                ; where to get information on this ICL-4300
dst=AO 1               ; where to send information on remote ICL-4300
netSession=ModbusSession ; netSession to use to execute this netevent
netAddr=5              ; network address of remote unit
```

The master transmits a block of 32 16-bit integer registers to address 5. Notice that with Modbus protocol (unlike BrickNet) there is no path ID.

Excerpt from SAMP13B.INI (router):

```
[NETSESSION]
name=ModbusSession     ; string that identifies this netsession
netport=ModbusPort     ; which netport to use for this session
protocol=ModbusRtuSlave ; protocol to use with this netsession
netAddr=20              ; network address for this unit on this netsession

[NETROUTE]
routeType=command       ; routing entry type
inSes=ModbusSession     ; inbound network session name
inAddrDst=5 7           ; messages destined for addresses 5 through 7
outAddr=105             ; translates addresses 5->105, 6->106, 7->107

[NETROUTE]
routeType=response      ; routing entry type
inSes=ModbusSession     ; inbound network session name
inAddrSrc=105 107       ; messages received from addresses 105 through 107
outAddr=5               ; translates addresses 105->5, 106->6, 107->7
```

The router is set up as a Modbus slave. Its address doesn't really matter for routing purposes, except that it can't be in the range 5-7 or 105-107. In this case, it is 20. The first NETROUTE entry routes command messages that are received from the master with a destination address of 5-7. In the Modbus protocol, master command messages contain the address of the slave (the master has no address). The router retransmits these messages, translating the address to 105-107. The outAddr parameter accomplishes this translation. The second NETROUTE entry routes response messages that are received from the slaves. In the Modbus protocol, slave response messages contain the address of the slave. The router retransmits these messages to the master, translating addresses 105-107 to 5-7. Once again, the outAddr parameter accomplishes this translation.

Excerpt from SAMP13C.INI (slave):

```
[REGALLOC]
regType=int16           ; register bank type
blockSize=32           ; number of registers to allocate in bank
bankId=1                ; number that identifies bank

[NETSESSION]
name=ModbusSession     ; string that identifies this netsession
netport=ModbusPort     ; which netport to use for this session
protocol=ModbusRtuSlave ; protocol to use with this netsession
netAddr=105            ; network address for this unit on this netsession
regLink=AO 1           ; associate AO's with bank 1
```

Notice in this configuration file that the address for the Modbus slave is 105. Also, the `regLink` parameter is used to associate AO registers with bank 1. This will cause the incoming data that was transmitted from the master to be stored in bank 1.

Chapter 4 – EasyLink Configuration File Reference

This chapter provides a reference for the EasyLink configuration file. Some additional parameters that were not explained in the tutorial section are listed here, along with a complete set of possible values for each parameter, and default values where applicable.

Syntax

The configuration file is an ASCII text file, with a syntax patterned after the familiar (to some, anyway) Windows INI file. The configuration file defines a configuration database, which consists of 1 or more records of various types. The basic format is:

```
; comment  
[recordname]      ; comment  
param=value      ; comment  
param=value      ; comment  
param=value      ; comment  
param=value      ; comment  
.  
.  
.  
[recordname]      ; comment  
param=value      ; comment  
; comment  
param=value      ; comment  
.  
.  
.
```

Syntax Rules:

- Comments start with a semicolon, and go to the end of the line.
- Record names are enclosed in square brackets – "[]".
- Each record consists of a list of parameters.
- The parameter name appears on the left side of the equals sign, and its value appears on the right side.
- The parameter names and allowed values are specific to each record type.
- Record names, parameter names and values are not case-sensitive.
- The order in which records appear in the file is not critical, except as noted.
- The order of parameters within a record is not critical, except as noted.
- User-defined symbols/identifiers can be up to 32 characters.
- User-defined symbols/identifiers must begin with a non-numeric character and may contain the following characters: a-z, A-Z, 0-9 and _.
- String values with embedded spaces must be placed in double quotes (as in "my string"). Quotes are optional otherwise.
- Lines with more than 120 characters will be truncated.

Outline of the EasyLink Configuration File

An outline of the configuration file is shown beginning on the following page. Each section is described in more detail in the remainder of this chapter.

Typographic conventions are shown below.

Items that are found on the left-hand side of the equal sign (parameter names):

<u>param</u>	normal underline	required parameter for record
<u>param</u>	wave underline	parameter may be optional
param	no underline	parameter is optional

Items that are found on the right hand side of the equal sign (parameter values):

<i>val</i>	italics	value you supply (number or string)
val	bold	default value for optional parameters
symbol	normal	keyword, appears literally in file
[<i>val</i>]	brackets	optional value for parameter
	vertical bar	separates different optional values (means "OR")

Notes/comments that are not part of the syntax are enclosed in braces - "{}".

```
{defines values that can be used elsewhere in the file}
```

```
{NOTE: multiple defines may be under the same record}
```

```
[DEFINES]
```

```
name = definition                                {equate name to definition}
```

```
{miscellaneous options}
```

```
[OPTIONS]
```

```
logDisplay = yes|no                                {yes to display log messages to console}
aiAverage = aveCount [slot [chan]]                {sets averaging for one or more AI channels}
                                                    {aveCount=0-255, slot=1-4, chan=1-16}
diInvert = slot [chan]                             {enables inversion of digital input channels}
                                                    {slot=1-4, chan=1-16}
```

```
{sets the default value for various parameters}
```

```
[DEFAULTS]
```

```
{REGALLOC parameters}
```

```
storage = ram|nvram
```

```
{NETPORT parameters}
```

```
baud = 300|1200|2400|4800|9600|19200|38400|57600|115200
parity = none|odd|even|mark|space|mark9bit|space9bit
dataBits = 8|7|6|5
stopBits = 1|2
rtsMode = alwaysOn|alwaysOff|xmtOn
leadDelayMs = time|0
trailDelayMs = time|0
rcvBufSize = size                                {normal default is protocol-dependent}
xmtBufSize = size                                {normal default is protocol-dependent}
```

```
{NETSESSION parameters}
```

```
protocol = modbusRtuMaster|
           modbusRtuSlave|
           modbusAsciiMaster|
           modbusAsciiSlave|
           brickNet
netAddr = address|0
regMode = MsbFirst|LsbFirst
readyMode = always|
            timeMs readyTime
accessDelay = [min|0] max|50
hopLimit = limit|5
eventGapMs = time|0
sesGapMs = time|0
respDelayMs = time|0
respTimeoutMs = time|1000
respRetry = count|3
rcvTimeoutMs = time|500
probeTimeSec = time|0
```

{allocates space for register banks}	
[REGALLOC]	
<u>regType</u> =	
bool	{1-bit Boolean - holds true/false or on/off values}
int8	{8-bit integer}
int16	{16-bit integer}
int32	{32-bit integer}
float	{32-bit single precision IEEE floating point}
double	{64-bit double precision IEEE floating point}
buffer	{array of bytes}
<u>blockSize</u> = <i>size</i>	{number of registers to allocate}
<u>bankId</u> = <i>id</i>	{identifies the bank, id=1-10000}
<u>storage</u> = ram nvram	{volatile or non-volatile RAM}

{maps registers to physical I/O}	
[IOMAP]	
<u>slot</u> = <i>slotNum</i>	{number of slot which contains I/O board, slotNum=1-4}
<u>sectType</u> =	{board section type}
DI	{digital input}
DO	{digital output}
AI	{analog input}
AO	{analog output}
<u>startReg</u> = <i>bankID regIndex</i>	{starting register}

{scales analog I/O from device units to engineering units}	
[IOSCALE]	
<u>startReg</u> = <i>bankID regIndex</i>	{starting register}
<u>range</u> = <i>min max</i>	{minimum/maximum engineering units}
<u>percent</u> = <i>low high</i>	{low/high converter percent for range}
<u>clamp</u> = <i>min max</i>	{min/max to clamp}
<u>blockSize</u> = <i>size</i> 1	{number of registers to scale}
<u>calMode</u> =	{calibration mode (default is I/O board's 1st cal table)}
none	{do not use calibration information}
0to5V	{use 0 to 5 volt mode}
0to10V	{use 0 to 10 volt mode}
neg5to5V	{use -5 to +5 volt mode}
neg10to10V	{use -10 to +10 volt mode}
0to20mA	{use 0 to 20 mA mode}

{scales and copies values from one group of registers to another (source to destination)}	
[REGSCALE]	
<u>srcReg</u> = <i>bankID regIndex</i>	{source starting register}
<u>dstReg</u> = <i>bankID regIndex</i>	{destination starting register}
<u>srcRange</u> = <i>min max</i>	{minimum/maximum values for source registers}
<u>dstRange</u> = <i>min max</i>	{corresponding min/max values for destination registers}
<u>clamp</u> = <i>min max</i>	{min/max to clamp destination registers to}
<u>blockSize</u> = <i>size</i> 1	{number of registers to scale}

```

{defines a trigger for NETEVENT to use}
[TRIGGER]
name = trigName                                {trigger name (referenced by NETEVENT)}
trig =                                           {trigger type}
    delta threshold |                            {triggered when any reg in block changes by threshold}
    deltaReg bankID regIndex |                 {same as delta, but threshold is stored in register}
    edge either|rise|fall |                     {detects change in any Boolean register in block}
    cyclic cycleCount |                         {triggered every cycleCount cycles}
    timerSec timeVal |                          {time based (in seconds)}
startReg = bankID regIndex                     {required if trig = delta, deltaReg or edge}
blockSize = size|1                              {avail if trig = delta, deltaReg or edge}

```

```

{network communications port}
[NETPORT]
name = portName                                {network port name}
port = com1|com2|...com7                        {serial port to use}
baud = 300|1200|2400|4800|9600|                {baud rate to use}
    19200|38400|57600|115200
parity = none|odd|even|mark|space|            {parity to use}
    mark9bit|space9bit
dataBits = 8|7|6|5                              {data bits to use}
stopBits = 1|2                                  {stop bits to use}
rtsMode = alwaysOn|alwaysOff|xmtOn           {how RTS signal is to be controlled}
leadDelayMs = time|0                           {time from RTS activation to data transmission}
trailDelayMs = time|0                           {time from transmission complete to RTS de-activation}
rcvBufSize = size                               {receive buffer size, default is protocol-dependent}
xmtBufSize = size                               {transmit buffer size, default is protocol-dependent}

```

```

{network protocol session}
[NETSESSION]
name = sesName                {network session name}
netPort = netPort             {name of network port to use}
protocol =                    {protocol to use for session}
    modbusRtuMaster|          {Modbus RTU Mode Master}
    modbusRtuSlave|           {Modbus RTU Mode Slave}
    modbusAsciiMaster|       {Modbus ASCII Mode Master}
    modbusAsciiSlave|        {Modbus ASCII Mode Slave}
    brickNet                  {BrickNet}
netAddr = address|0          {network address}
regLink =                     {register bank to associate with message type}
    DI bankID|                {associate bankID with messages referencing DI's}
    DO bankID|                {associate bankID with messages referencing DO's}
    AI bankID|                {associate bankID with messages referencing AI's}
    AO bankID|                {associate bankID with messages referencing AO's}
regMode = MsbFirst|LsbFirst  {register/message byte ordering}
readyMode =                   {how media is determined to be ready}
    always|                   {media is always ready}
    timeMs readyTime          {media ready when nothing received since readyTime}
accessDelayMs = [min|0] max|50 {min/max access backoff when not ready}
pathId = id|0                 {routing path identifier for transmitted messages}
hopLimit = limit|5            {routing hop limit count}
eventGapMs = time|0           {time gap to insert after each event serviced}
sesGapMs = time|0             {time gap to insert after all events on same session serviced}
respDelayMs = time|0          {delay time before sending a response to incoming message}
respTimeoutMs = time|1000     {timeout before retry after no response to outgoing message}
respRetry = count|3           {retry attempts on no response for the same message}
rcvTimeoutMs = time|500       {receiver byte timeout to restart message}
probeTimeSec = time|0         {probe interval to reconnect failed node (0=disabled)}
altNetSes = sesName           {alternate session to use on failure (same protocol & port)}

```

```

{network event, causes message transmission}
[NETEVENT]
name = eventName              {network event name}
netSession = session          {name of network session to service event}
event =                       {condition that triggers event}
    startup|                   {on application startup}
    cyclic cycleCount|         {every cycleCount cycles}
    timerSec timeVal|          {time based every timeVal seconds}
    trig trigName              {trigger based}
action =                       {action to take when event triggered}
    read|                       {read information from remote device}
    write|                      {write information to remote device}
    probe                       {send probe message to determine if contact can be made}
blockSize = size              {number of registers to read/write (not required for probe)}
src = bankID regIndex         {source of data (not required for probe)}
dst = bankID regIndex         {destination of data (not required for probe)}
netAddr = address             {network address of remote unit}

```

{NOTE: multiple event parameters may be specified for the same record.}

```

{sets up message routing}
[NETROUTE]

```

<code>routeType =</code>	<code>{ routing entry type }</code>
<code>peer </code>	<code>{ used for peer-to-peer protocols (BrickNet) }</code>
<code>command </code>	<code>{ command msg, used for master/slave protocols (Modbus) }</code>
<code>response</code>	<code>{ response msg, used for master/slave protocols (Modbus) }</code>
<code>inSes = session</code>	<code>{ inbound network session name }</code>
<code>inAddrDst = min [max]</code>	<code>{ min/max destination addresses allowed for route path }</code>
<code>inAddrSrc = min [max]</code>	<code>{ min/max src addresses allowed for route path, rq'd for peer }</code>
<code>peer}</code>	
<code>outSes = session</code>	<code>{ outbound network session name (defaults to inSes value) }</code>
<code>outAddr = addr</code>	<code>{ outbound starting address (defaults to inAddrDst min) }</code>
<code>pathId = id</code>	<code>{ path identifier required for peer }</code>

`{ defines a textual user interface (TUI) }`

`[TUI]`

<code>name = tuiName</code>	<code>{ TUI port name }</code>
<code>netPort = portName</code>	<code>{ name of net port to use }</code>
<code>header = headerString</code>	<code>{ string to be displayed on top line of all pages }</code>
<code>refreshSec = refreshTime 30</code>	<code>{ TUI page refresh rate in seconds }</code>
<code>scanRateMs = scaneTime 100</code>	<code>{ TUI field scan rate in milli-seconds }</code>
<code>pageNumLoc = x y</code>	<code>{ x/y coordinates of page number location }</code>
<code>quit = quitKey</code>	<code>{ character to terminate program }</code>

`{ defines a TUI page }`

`[TUIPAGE]`

<code>name = pageName</code>	<code>{ name used to refer to this page }</code>
<code>tui = tuiName</code>	<code>{ TUI to associate this page with }</code>
<code>pageNum = pageNumber</code>	<code>{ page number to associate with this page }</code>
<code>background = filename</code>	<code>{ file to use for page background }</code>

`{ defines a TUI data field }`

`[TUIFIELD]`

<code>page = pageName</code>	<code>{ name of page on which to display this field }</code>
<code>position = x y</code>	<code>{ x/y coordinates of field data starting point }</code>
<code>startReg = bankId index</code>	<code>{ starting register which contains data to display }</code>
<code>blockSize = count 1</code>	<code>{ number of registers to display vertically }</code>
<code>readOnly = yes no</code>	<code>{ yes if field is to be read-only, no for read/write }</code>
<code>format = formatSpec #</code>	<code>{ format of data field to display (such as "AI ? : ###.##") }</code>
<code>clamp = min max</code>	<code>{ min/max range to clamp user TUI input }</code>

`{ defines voice messages, useful only in conjunction with a C or ISaGRAF application program }`

`[VOICE]`

<code>msg = refNum msgLen [tagName]</code>	<code>{ defines msg w/reference number, max length & tag name }</code>
	<code>{ refNum=0-65535, msgLen=0.3-240.0 seconds }</code>

`{ NOTE: multiple 'msg' parameters may be listed under the same record. }`

Records and Parameters

The descriptions and tables which follow contain additional information for each record type.

[DEFINES]

Defines allow you to create symbolic names with equivalent values. These symbolic names can be used throughout the configuration file where the value normally would go. The configuration file parser looks up and substitutes the value for the name wherever you use the name. This allows you to define a value in one place and use it in many places. If the value needs to change, then you only need to change the define and not all the places it is used. Make sure that you do not use a keyword as a define name.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
name	text	any text string, up to 32 characters	no	none	A define can be used within a define, but they must be defined in order of inclusion.

[OPTIONS]

Miscellaneous options are controlled by the OPTIONS record.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
logDisplay	enumerated	yes no	no	yes	Determines whether system error log messages will be displayed to the console. A setting of yes causes messages to be displayed.
aiAverage	compound: aveCount (integer) slot (integer) chan (integer)	aveCount: 0-255 slot: 1-4 chan: 1-16	no	aveCount=5 slot=all chan=all	Software filtering (or "averaging") is applied to analog input signals. This parameter determines how much filtering is used. An aveCount value of 0 or 1 means that no filtering should be applied. A value greater than 1 means that some filtering should be used. The larger the aveCount value, the more filtering is applied. If you omit the slot and channel, the averaging setting is applied to all slots and channels. If you omit the channel, the averaging setting is applied to all channels in the slot.
diInvert	compound: slot (integer) chan (integer)	slot: 1-4 chan: 1-16	no	chan=all	Determines whether digital input channels are inverted, by slot and channel. If you omit the channel, the inversion setting is applied to all channels in the slot.

[DEFAULTS]

The `DEFAULTS` record allows you to set the default values for a number of parameters. This overrides the standard default values. As a result, the rest of the configuration file can be more terse. For the proper syntax, meaning and allowed range of values, see the description for the appropriate record/parameter elsewhere in this chapter. The table below lists all of the different parameters which can appear in the `DEFAULTS` record.

Record Type	Parameter
REGALLOC	storage
NETPORT	baud
	parity
	dataBits
	stopBits
	rtsMode
	leadDelayMs
	trailDelayMs
	rcvBufSize
	xmtBufSize
NETSESSION	protocol
	netAddr
	regMode
	readyMode
	accessDelayMs
	hopLimit
	eventGapMs
	sesGapMs
	respDelayMs
	respTimeoutMs
	respRetry
	rcvTimeoutMs
	probeTimeSec

[REGALLOC]

Register bank allocation records allow you to allocate memory for register banks, in either volatile or a non-volatile RAM.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
regType	enumerated	bool int8 int16 int32 float double buffer	yes	none	Determines the type of bank allocated.
blockSize	integer	1-65535	yes	none	Determines the number of registers allocated, not necessarily the number of bytes.
bankId	integer	1-10000	yes	none	The bank ID is used to reference the bank elsewhere within the configuration file, as well as within your C application program.
storage	enumerated	ram nvram	no	ram	Selects the storage location for the register bank -- either volatile RAM or non-volatile RAM.

[IOMAP]

I/O mapping records allow you to map registers to physical I/O.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
slot	integer	1-4	yes	none	Indicates which slot the I/O board that you are mapping registers to is installed in.
sectType	enumerated	DI DO AI AO	yes	none	Identifies the type of the section you are mapping. Some I/O boards have multiple I/O sections.
startReg	compound: bankID (integer) regIndex (integer)	Any valid register of a compatible type for section, as listed: DI, DO: Bool Int8 Int16 Int32 AI, AO: Int8 Int16 Int32 float double	yes	none	Starting point of the block of registers that is to be mapped to the I/O section. As many registers as there are I/O points in the section are mapped to the board.

[IOSCALE]

IOSCALE records allow you to scale analog inputs and outputs using engineering units instead of arbitrary device units.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
startReg	compound: bankID (integer) regIndex (integer)	any valid register which has been mapped to analog I/O	yes	none	Beginning point of the block of registers that is to be scaled. The registers need to be associated with analog I/O using IOMAP records.
range	compound: min (float) max (float)	any range, where min \neq max	yes	none	Specifies the range of engineering unit values that are to be scaled to the analog input or output.
percent	compound: low (float) high (float)	0-100, where low < high	no	min = 0 max = 100	This parameter is used if a sub-range of the analog to digital converter (for AI's) or digital to analog converter (for AO's) is to be mapped to the specified engineering unit range. Useful for 4 to 20 mA scaling: set min to 20 and max to 100. (20% is the 4 mA point).
clamp	compound: min (float) max (float)	any valid range, where min < max	no	none	This parameter is used to clamp, or limit the resulting scaled values to the specified min and max.
blockSize	integer	1-65535	no	1	Specifies the number of registers to scale.
calMode	enumerated	none 0to5V 0to10V neg5to5V neg10to10V 0to20mA	no	depends on I/O board (see below)	Specifies the calibration mode that is to be used. This causes the correct calibration values to be read from the I/O board. Specifying None causes the ICL-4300 to ignore the calibration values.

Calibration Modes:

I/O Board	Available Modes	Default Mode
#43-AI16-14	none, 0to20mA, Neg10to10V	0to20mA
#43-AO16-12I	none, 0to20mA	0to20mA
#43-AO16-12V	none, 0to10V, Neg5to5V	0to10V
#43-MINICOMBO-24 / 120 AI section	none, 0to20mA, 0to5V	0to20mA
#43-MINICOMBO-24 / 120 AO section	none, 0to20mA	0to20mA

[REGSCALE]

REGSCALE records allow you to apply scaling operations between two registers or two sets of registers. The source register bank is automatically scaled and copied to the destination register bank.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
srcReg	compound: bankID (integer) regIndex (integer)	any valid integer or floating point register	yes	none	Beginning point of the source block of registers that is to be scaled and copied to the destination block.
dstReg	compound: bankID (integer) regIndex (integer)	any valid integer or floating point register	yes	none	Beginning point of the destination block of registers that is to receive the scaled values.
srcRange	compound: min (float) max (float)	any range, where min \neq max	yes	none	Specifies the range of values that are to be scaled from.
dstRange	compound: min (float) max (float)	any range, where min \neq max	yes	none	Specifies the range of values that are to be scaled to.
clamp	compound: min (float) max (float)	any valid range, where min < max	no	none	This parameter is used to clamp, or limit the resulting scaled values to the specified min and max.
blockSize	integer	1-65535	no	1	Specifies the number of registers to scale.

[TRIGGER]

TRIGGER records allow you to specify conditions which can be used to activate NETEVENT records.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
name	string	any text string, up to 32 chars	yes	none	Used to identify the record, so other records can refer to it.
trig	compound: delta (keyword) threshold (float)	threshold can be any valid floating point number	yes	none	The delta trigger monitors a block of registers for a change equal to greater than the specified threshold. The comparison values are captured each time the trigger is activated.
	compound: deltaReg (keyword) bankID (integer) regIndex (integer)	bankID/regIndex can refer to any existing register of any type except "buffer"			The deltaReg trigger works the same as the delta trigger, except that the threshold value is held in a register.
	compound: edge (keyword) either rise fall (keywords)				The edge trigger is used for Boolean registers. The trigger can be activated on either edge, the rising edge only, or the falling edge only.
	compound: cyclic (keyword) cycleCount (integer)	cycleCount: 1-65535			A cyclic trigger becomes active every cycleCount cycles. A cycle is complete when every pending NETEVENT for the same device has been serviced.
	compound: timerSec (keyword) timeVal (integer)	timeVal: 1-65535 sec			The timerSec trigger is activated every time the specified number of seconds passes.
startReg	compound: bankID (integer) regIndex (integer)	any valid register of a compatible type as listed: delta, deltaReg: int8 int16 int32 float double edge: bool	for delta, deltaReg, and edge only	none	The starting register of the block of registers to monitor for change, when used with the delta, deltaReg or edge trigger types.
blockSize	integer	1-65535	for delta, deltaReg, and edge only	none	The number of registers in the register block to monitor for change, when used with the delta, deltaReg or edge trigger types.

[NETPORT]

NETPORT records define and configure network communication ports.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
name	string	any text string, up to 32 characters	yes	none	Used to identify the record, so other records can refer to it
port	enumerated	COM1, COM2, COM3, COM4, COM5, COM6, COM7	yes	none	Specifies which serial port this record refers to.
baud	integer	300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200	no	9600	Specifies the baud rate to use on this serial port.
parity	enumerated	none, odd, even	no	none	Specifies the parity to use on this serial port.
dataBits	integer	5, 6, 7, 8	no	8	Specifies the number of data bits to use on this serial port.
stopBits	integer	1, 2	no	1	Specifies the number of stop bits to use on this serial port.
rtsMode	enumerated	alwaysOn alwaysOff xmtOn (on for transmit)	no	alwaysOn	Specifies how to control the RTS signal.
leadDelayMs	integer	0-65535 ms	no	0	Time from RTS assert to begin transmit.
trailDelayMs	integer	0-65535 ms	no	0	Time from transmit end to RTS de-assert.
rcvBufSize	integer	0 (means use default) 1-65535	no	protocol-based: Modbus=256 BrickNet=512	Determines the size (in bytes) of the receive buffer used with this serial port. It is recommended that you use this parameter only to increase the buffer size, and not to decrease it.
xmtBufSize	integer	0 (means use default) 1-65535	no	protocol-based: Modbus=256 BrickNet=512	Determines the size (in bytes) of the transmit buffer used with this serial port. It is recommended that you use this parameter only to increase the buffer size, and not to decrease it.

[NETSESSION]

NETSESSION records allow you to attach a communication protocol to a network port.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
name	string	any text string, up to 32 characters	yes	none	Used to identify the record, so other records can refer to it.
netPort	string	any text string, up to 32 characters	yes	none	Identifies NETPORT record (must exist) to use for this session.
protocol	enumerated	ModbusRtuMaster ModbusRtuSlave ModbusAsciiMaster ModbusAsciiSlave BrickNet	yes	none	Defines the protocol used for this session.
netAddr	integer	0-65535	only for BrickNet, Modbus Slave, & Optomux	0	Used to identify this node on the network.
regLink	compound: msgType (enumerated) bankID (integer)	msgType: DI, DO, AI, AO bankID: must reference an allocated bank	for Modbus RTU/ASCII Slave only	none	Used for Modbus slave. Identifies which banks should be associated with which register types in messages. You may have four entries in each NETSESSION record, one for each register type.
regMode	enumerated	msbFirst lsbFirst	no	msbFirst	Applies only to Modbus protocols. Determines the byte ordering that is used to encode and decode registers in messages. (Both ends must agree on the same byte ordering).
readyMode	enumerated: always		no	always	BrickNet only. Indicates how the communications media should be determined to be ready (not in use by another network node).
	compound: timeMs readyTime (integer)				
accessDelayMs	compound: min (integer) max (integer)	0-65535 ms, min <= max	no	0	BrickNet only. Min is optional, defaults to 0 when not supplied. When the media is determined to be ready, the access delay is applied before the

					media is re-checked and accessed. A random time between min and max is used. This helps prevent collisions between multiple units trying to access the media at the same time.
pathID	integer	0-255	no	0	BrickNet only. When messages are transmitted using this NETSESSION, this path ID is encoded in the message. The path ID is used to control message routing. A path ID of 0 essentially means that no routing will be performed.
hopLimit	integer	0-255	no	5	BrickNet only. When messages are transmitted using this NETSESSION, the hop limit is encoded in the message. The hop limit is used to control message routing. If a message travels more than the specified number of "hops" when being routed to its destination, the message will be terminated. This prevents a message from hopping around the network indefinitely due to an error in the routing tables.
eventGapMs	integer	0-65535 ms	no	0	Delay inserted after each individual event message.
sesGapMs	integer	0-65535 ms	no	0	Delay inserted after all pending event messages for a network port have been serviced.
respDelayMs	integer	0-65535 ms	no	0	Time to wait before transmitting a

					response to a received message.
respTimeoutMs	integer	0-65535 ms	no	5000	Time to wait before re-sending a message to which there was no received response.
respRetry	integer	0-65535	no	3	Number of times to retry sending a message to which there was no received response.
rcvTimeoutMs	integer	0-65535 ms	no	500	BrickNet only. If a message is being received, and the incoming byte stream is interrupted for longer than this time, the message will be discarded.
probeTimeSec	integer	0-65535 sec	no	30	Interval at which the ICL-4300 will try to reestablish communications with a remote device that has failed to respond. A value of 0 means that probing is disabled.
altNetSes	string	any valid NETSESSION name, up to 32 characters	no	none	Alternate session to use if communication fails on this NETSESSION. The alternate session must be defined with the same protocol and port, and must come <i>before</i> this NETSESSION. Communication will switch back to this NETSESSION when probing determines that it is operational again.

[NETEVENT]

NETEVENT records define messages to read and write information over the network.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
name	string	any text string, up to 32 characters	no	none	Used to identify the record.
netSession	string	any text string, up to 32 characters	yes	none	name of the session (must exist) to use for this event
event	enumerated: startup compound: cyclic cycleCount (integer) timerSec timeVal (integer) trig trigName (string)	cycleCount: 1-65535 timeVal: 1-65535 sec trigName: valid trigger name	yes	none	Specifies the condition that triggers the event. A startup event is triggered on application startup. Cyclic events are triggered every cycleCount cycles. A cycle is complete when all pending NETEVENTs for a port have been serviced. A trig event is triggered by a TRIGGER record that is defined elsewhere.
action	enumerated	read write probe	yes	none	Specifies the action to take when the event is triggered. Read gets information from remote. Write sends information to remote. Probe sends a probe message to determine if contact can be made with the remote unit.
blockSize	integer	1-65535	for read and write only	0	Number of registers to read or write.
src	compound: bankID (integer) regIndex (integer) msgType (enumerated) regIndex (integer)	any valid register or message type (DI, DO, AI, AO) compatible with dst	for read and write only	none	Source of data.
dst	compound: bankID (integer) regIndex (integer)	any valid register or message type (DI, DO, AI, AO) compatible with src	for read and write only	none	Destination of data.
netAddr	integer	0-65535	yes	0	Used to identify the remote node on the network with which to communicate.

[NETROUTE]

NETROUTE records allow you to set up a network message routing table.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
routeType	enumerated	peer command response	no	peer	"Peer" is used only for BrickNet. "Command" is used to route Modbus command messages. "Response" is used to route Modbus response messages.
inSes	string	name of existing NETSESSION, up to 32 characters	yes	none	NETSESSION for incoming messages to which this routing record applies.
inAddrDst	compound: min (integer) max (integer)	0-65535	yes	none	Range of destination addresses which should be routed.
inAddrSrc	compound: min (integer) max (integer)	0-65535	for peer only	none	BrickNet only. Range of intermediate source addresses which should be routed.
outSes	string	name of existing NETSESSION, up to 32 characters	no	inSes	NETSESSION for outgoing routed messages.
outAddr	integer	0-65535	no	inAddrDst min	Outbound message starting address. Used for message address translation. The outgoing destination address is calculated as: incoming dst addr – inAddrSrc min + outAddr
pathID	integer	1-65535	for peer only	0	BrickNet only. Incoming messages must match this path ID in order to be routed.

[TUI]

A TUI record allows you to define a textual user interface and associate it with a serial port.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
name	string	any text string, up to 32 characters	yes	none	The name to be associated with this instance of the TUI. Used elsewhere within the configuration file to associate pages with the TUI.
netPort	string	any text string, up to 32 characters	yes	none	Indicates which port to run the TUI on. Must reference a port that is defined by a NETPORT record.
header	string	any text string	no	empty	The header will automatically be displayed at the top of each TUI page. It is typically used as a system title.
refreshSec	integer	0-65535	no	30	The current TUI page will be periodically refreshed at this rate. If the refresh rate is set to zero, it has the effect of disabling the refresh.
scanRateMs	integer	0-65535	no	100	All the currently displayed registers are periodically examined for changes that need to be reflected on the display. This setting determines how frequently the registers are examined. A setting of 0 will be interpreted to mean "as fast as possible."
pageNumLoc	compound: x (integer) y (integer)	For an 80 x 25 display, x: 1-80 y: 1-25	no	x=78 y=1	Determines where the current page number is displayed.
quit	string	Any character.	no	none	Determines which key, when pressed, will terminate the current application. If you leave this parameter out, then no key will be defined to have this functionality.

[TUIPAGE]

A TUIPAGE record defines one page of a textual user interface (TUI).

Parameter	Value Type	Allowed Values	Required	Default	Remarks
name	string	any text string, up to 32 characters	yes	none	The name used to refer to this page elsewhere in configuration file.
tui	string	any text string, up to 32 characters	yes	none	The TUI, defined by a [TUI] record, with which to associate this page.
pageNum	integer	1-65535	yes	none	The page number that is used for this page.
background	string	any valid DOS filename	no	none	Indicates the file to be used as a background for this page.

[TUIFIELD]

TUIFIELD records define areas on TUI pages for displaying register data.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
page	string	any text string, up to 32 characters	yes	none	The page on which to display this field.
position	compound: x (integer) y (integer)	For an 80 x 25 display, x: 1-80 y: 1-25	yes	none	Determines where the field is displayed. The coordinates refer to the upper left-hand corner of the field.
startReg	compound: bankID (integer) regIndex (integer)	any valid register	yes	none	Source of data to display.
blockSize	integer	1-255	no	1	The number of registers to display.
readOnly	enumerated	yes no	no	no	Determines if the field should be read-only, or read/write.
format	string	quoted format specifier consisting of 1) literal characters, 2) an optional '?' character to display the register index, 3) and a combination of '#' characters and a single optional '.' character	no	#	Determines the format for register data display. includes characters that are printed literally, and special characters ('?' and '#') for which numbers are substituted. If the '?' character appears, the register index is substituted. Register values are substituted for the '#' characters. The number of # characters before the decimal point indicates the minimum number of digits to allow for. If more digits are required, the field will be expanded as needed. The number of # characters after the decimal point indicate the number of decimal places to display, for floating point numbers. For integer numbers, leave out the decimal point and the # characters to the right of the decimal point. Example: "AO ? = ###.##"
clamp	compound: min (float) max (float)	full range	no	none	Minimum and maximum values to clamp user input.

[VOICE]

VOICE records define voice messages. This record type is useful only in conjunction with a C or ISaGRAF application program. Multiple `msg` parameters may be defined within one VOICE record.

Parameter	Value Type	Allowed Values	Required	Default	Remarks
<code>msg</code>	compound: refNum (integer) msgLen (float) tagName (string)	refNum: 0-65535 msgLen: 0.3-240.0 sec tagName: any name, up to 32-characters	no	none	The reference number is used to identify each voice message. Messages 0 through 99 are reserved and should not be used for application messages. The message length resolution is 0.3 seconds. The tag name is an optional identifier/description associated with the voice message.