

ICL-4300 *Modular DC Compatible Controller*

TCP/IP Support Kit Reference Manual



Revision C
P/N 60082003
revised 08/21/2000

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

© 1999-2000 Industrial Control Links, Inc. All rights reserved

License Agreement

The TCP/IP Support Kit (model #43-TCPIP-KIT) contains licensed software, including but not limited to Datalight Sockets (socketp.exe and socketm.exe) and the FTP support program (ftpapi.exe), referred to as the Software.

Acceptance.

By installing, copying, or otherwise using this Software you agree to be bound by the terms of this agreement. If you do not accept the terms of this agreement, do not install this Software and return it and other accompanying items to the place of purchase.

Software License.

Each ICL-4300 on which the Software is installed must have a separate license (Model #43-TCPIP-RTL). This license is purchased from Industrial Control Links and is indicated by a license sticker. Normally the license sticker is applied at the factory and is associated with the specific serial numbered unit. Under certain circumstances (TCP/IP used on non-ethernet hardware, for example) the license may be purchased separately and applied to a pre-existing unit. In any case, the license is associated with one specific serial-numbered unit and the license sticker is applied to that unit.

For each license purchased, Industrial Control Links, Inc. and Datalight Inc. grant you a non-exclusive license to use a copy of the Software on a single ICL-4300. You are authorized to load and execute the Software only on licensed ICL-4300 units which have the license sticker.

Industrial Control Links, Inc. and Datalight Inc. reserve the rights not expressly granted to the licensee. You own the physical media on which the Software is recorded or fixed, but Industrial Control Links, Inc. and Datalight Inc. retain the title and ownership of the Software. This Software and any accompanying written materials are protected by copyright laws and international treaty provisions. Unauthorized copying of the Software, including Software that has been modified or merged with other Software, or of any of the written material is forbidden. Subject to these restrictions, you may make one copy of the Software for working purposes and retain the original Software for archival purposes. This Software may not be rented, leased or distributed without the prior written consent of Industrial Control Links, Inc. and Datalight Inc. You may not disassemble, reverse engineer, modify, adapt, translate or create derivative works without prior written consent of Industrial Control Links, Inc. and Datalight Inc.

Limited warranty.

Industrial Control Links, Inc. warrants that (a) the Software will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of receipt, and (b) any support services provided by Industrial Control Links, Inc. shall be substantially as described in applicable written materials provided to you by Industrial Control Links, Inc., and Industrial Control Links, Inc.'s support engineers will make commercially reasonable efforts to solve any problems. To the extent allowed by applicable law, implied warranties on the Software, if any, are limited to ninety (90) days.

Industrial Control Links, Inc.'s entire liability and your exclusive remedy shall be, at Industrial Control Links, Inc.'s option, either (a) return of the price paid, if any, or (b) repair or replacement of the Software that does not meet Industrial Control Links, Inc.'s Limited Warranty. This Limited Warranty is void if failure of the Software has resulted from accident, abuse, or misapplication. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

Limitation of liability.

In no event shall Industrial Control Links, Inc. or its suppliers be liable for any special, incidental, indirect, or consequential damages whatsoever (including, without limitation, damages for loss of business profits,

business interruption, loss of business information or any other pecuniary loss) arising out of the use of or inability to use the software product or the provision of or failure to provide support services even if Industrial Control Links, Inc. has been advised of the possibility of such damages. In any case, Industrial Control Link, Inc.'s entire liability under any provision of this agreement shall be limited to the actual amount paid by you for the Software.

By installing the software you accept this agreement and are bound by its terms. If you do not agree, do not install the software.

Table of Contents

CHAPTER 1 – OVERVIEW	1
CHAPTER 2 – INSTALLING TCP/IP	3
Installing TCP/IP	3
SOCKETS Environment Variable	4
File Descriptions	5
NT4300.EXE.....	5
NETSET.EXE.....	5
PCKTDRV.COM.....	5
SOCKETP.EXE.....	6
SOCKETM.EXE.....	6
SOCKET.CFG.....	7
GOPD.BAT.....	8
XPING.EXE.....	8
CHAPTER 3 – SOCKETS CONFIGURATION	11
SOCKET.CFG Sample File	11
CHAPTER 4 – SOCKETS COMMAND REFERENCE	13
Syntax Rules	13
Commands	14
arp.....	14
bootp.....	15
domain.....	16
iface \ interface.....	17
ip.....	20
param.....	22
rip.....	26
route.....	28
tcp.....	31

CHAPTER 5 – PPP	35
PPP1.CFG Sample File	35
PPP2.CFG Sample File	37
CHAPTER 6 – FTP	39
FTPAPI.EXE	39
SOCKET.UPW	39
Sample user password file.....	40
CHAPTER 7 – HOSTS FILE	41
HOSTS	41
Sample HOSTS file	42

Chapter 1 – Overview

Support Kit

The ICL-4300 TCP/IP Support Kit is a software package which provides support for network communications using a TCP/IP stack on the ICL-4300. This support kit contains a group of utilities, drivers, samples and other files to facilitate the use of the TCP/IP stack on the ICL-4300. The specific files used from this kit will depend upon your application. This support kit has been designed to support TCP/IP in a DOS environment only. We recommend using ROM-DOS, which is the standard operating system loaded by the factory.

TCP/IP Stack

The TCP/IP Stack the ICL-4300 uses is an application, which runs as a TSR, called Sockets. The Sockets networking communication applications are licensed from Datalight, see the License Agreement at the beginning of this manual.

The TCP/IP Stack is group of protocols that provide communication over a network. The protocols include TCP (Transmission Control Protocol), UDP (User Datagram Protocol), IP (Internet Protocol) and ICMP (Internet Control Message Protocol).

Additional routing support protocols include RIP (Routing Information Protocol), BOOTP (Bootstrap Protocol), ARP (Address Resolution Protocol) and SNMP (Simple Network Management Protocol).

Media Support

The only TCP/IP media currently supported by ICL is Ethernet. The ICL-4300E is the model number for the ICL-4300 Controller with the Ethernet option installed. This Ethernet option uses a Crystal CS8900A Ethernet controller chip that connects via 10BaseT twisted pair cable to a network. The interface of the TCP/IP stack with the Ethernet controller chip requires a packet driver.

A packet driver transfers units of information to the physical layer. A packet driver is provided with this support kit and is loaded by DOS as a TSR.

Applications

Applications that use TCP/IP are typically developed using one of two methods. An application may utilize the TCP/IP stack by using the built-in features of the ICL-4300 API or with direct calls using the Sockets API.

The built-in features of the ICL-4300 API allow you to define and access network connections by creating network ports, sessions and events as described in the software development kits. For C applications, see the ICL-4300 C/C++ SDK. For configuration files that use the ESP (Easy Setup without Programming) application, see the ESP Users Manual. Note that ESP application is built upon the ICL-4300 API.

IsaGRAF users who want to use the TCP/IP stack would do so using a configuration file (same as the ESP format). All IsaGRAF communications must be setup with a configuration file, see the IsaGRAF SDK.

The Sockets API allows you to make direct calls to the TCP/IP stack through software interrupts. You can define and access network connections specific to your application through this interface. The ICL-4300 API also provides a C function wrapper around these software interrupts to facilitate application development. See the ICL-4300 C/C++ SDK for specific details and function prototypes.

Purpose and Organization of this Manual

The purpose of this manual is to help configure the ICL-4300 to allow your applications to use the Sockets TCP/IP stack. This applies whether you are using an application developed using one of the C/C++ SDKs, the standalone ESP program or the IsaGRAF SDK.

This manual is organized into the following chapters:

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

Chapter 2 – Installing TCP/IP: Installation and description of the TCP/IP files.

Chapter 3 – Sockets Configuration: Tutorial on sample Sockets configuration files.

Chapter 4 – Sockets Command Reference: Reference for Sockets configuration file syntax.

Chapter 5 – PPP: Description of PPP applications included.

Chapter 6 – FTP: Description of FTP applications included.

Chapter 7 – HOSTS File: Description of HOSTS file format.

Chapter 2 – Installing TCP/IP

Installing TCP/IP

To install the TCP/IP Support Kit, run the INSTALL.EXE program from the installation disk. The standard support files are normally installed in the following directory on your development PC:

```
C:\ICL4300\TCPIP
```

PPP support files are installed in the directory:

```
C:\ICL4300\TCPIP\PPP
```

FTP support files are installed in the directory:

```
C:\ICL4300\TCPIP\FTP
```

Sample files provided with this support kit are illustrated later in this manual.

In order for your applications to use the TCP/IP stack, you need to transfer the necessary files 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 (normally COM1), type the following command at the DOS prompt, followed by the Enter key:¹

```
R
```

Then start a Z-modem protocol transfer (sometimes called an "upload") using your terminal emulator, sending the files. The standard files are:²

```
NT4300.EXE  
NETSET.EXE  
PCKTDRV.COM  
SOCKETP.EXE  
SOCKETM.EXE  
SOCKET.CFG  
GOPD.BAT  
XPING.EXE
```

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

¹ This actually runs a DOS batch file (R.BAT) which contains the command "RSZ /R".

² If your unit was shipped with the Ethernet option the standard files for using TCP/IP are pre-loaded in the factory.

SOCKETS Environment Variable

There is an optional environment variable, `SOCKETS`, that is used by Sockets to find various files. For example, this environment variable can be set with the following command:

```
SET SOCKETS = C:
```

This environment variable is used by `SOCKETP.EXE`, `SOCKETM.EXE` and `FTPAPI.EXE`. These programs will be described more in the following sections.

The programs that use the `SOCKETS` environment variable essentially add a "\" themselves to the value. As a result, you should not put a "\" in the definition of the variable. (`SET SOCKETS = C:\` would *not* be correct).

File Descriptions

This section describes the purpose of each of the files required for using the TCP/IP stack.

NT4300.EXE

[NT4300.EXE](#) is a utility program that allows you to verify the configuration of the ICL-4300 Ethernet controller. This is a menu driven program that allows access to certain parts of the Ethernet hardware for purposes of test and verification.

This program will let you read back the manufacturing information and IEEE assigned Ethernet controller address configured by the factory. The ability to read back this information is useful, in that this verifies the Ethernet controller is properly installed in the unit and operating.

Press the <?> key to access the help screen of the current menu you are in for more information.

NETSET.EXE

[NETSET.EXE](#) makes configuration modifications to the ICL-4300 that will allow it to load and run the TCP/IP drivers. This program must be run first on the ICL-4300, before loading the packet driver and TCP/IP stack and before running a TCP/IP application.

The modifications that [NETSET](#) makes are:

- Zeros interrupt vectors 0x60 and 0x61. These are the standard vectors used by the packet driver and Sockets TSR's, respectively. These TSR's will not load if they think another ISR is already loaded into these vectors.
- Configures the 386Ex's INT2 signal to be connected to the internal ICU. This is a multiplexed signal that must be connected to the ICU to receive interrupts from Ethernet chip.

PCKTDRV.COM

[PCKTDRV.COM](#) is a TSR providing the interface layer that allows the TCP/IP stack to transfer data to/from the Ethernet device (the physical layer). The packet driver is typically loaded at vector 0x60. To load the packet driver enter the command as shown below:

```
PCKTDRV 0x60
```

The packet driver must be loaded after [NETSET](#) and before [SOCKETP](#).

SOCKETP.EXE

`SOCKETP.EXE` is a TSR that provides the TCP/IP stack. This TSR is accessed by software interrupt calls from your application. The Sockets stack allows your application to open connections and transfer data over a network. The Sockets stack is typically loaded at vector 0x61. To load the Sockets TCP/IP stack enter the command as shown below:

```
SOCKETP
```

The syntax is shown below:

```
SOCKETP  [/g=global_descriptors][/L=local_descriptor][/i=interrupt_number]
         [/v=interrupt_vector][/m=memory_size][/s=stack_size]
         [/p=print_delay] [cfg_file]
```

```
SOCKETP  [/u]
```

```

/g      number of global descriptors allowed (default is 8)
/L      number of local descriptors allowed (default is 8)
/i      interrupt number for the compatibility API (default is 0x61)
/v      interrupt number for the Sockets API (default is 0x7F)
/m      working memory size for the Sockets TSR (default is 16384)
/s      working stack size for the Sockets TSR (default is 2048)
/p      number of times a printer will be tested for busy status
/u      unloads the Sockets TSR
```

The *cfg_file* specifies the configuration file to use at startup (default is `SOCKET.CFG`)

SOCKETP searches for the configuration file in the following manner:

When you specify a configuration file on the command line, it will just use that file with the path as given. If it does not have a full path, it will look for it in the current directory. If you don't specify a configuration file on the command line, it will search for:

```

%SOCKETS%\DOS\SOCKET.CFG
%SOCKETS%\SOCKET.CFG
SOCKET.CFG (in the current directory)
```

Where `%SOCKETS%` means the value of the `SOCKETS` environment variable.

If you don't have a `SOCKETS` environment variable, it is the equivalent of:

```
SOCKETS=\DL\SOCKETS
```

The Sockets TSR must be loaded after `PCKTDRV` and before your application.

SOCKETM.EXE

`SOCKETM.EXE` is equivalent to `SOCKETP.EXE`, except that `SOCKETM.EXE` is designed to run over a serial port interface instead of Ethernet. It uses the same command line switches as `SOCKETP.EXE` (see above). Typically, protocols such as PPP and SLIP are used with a serial port interface. Direct and dialup connections are both supported for serial interfaces.

SOCKET.CFG

[SOCKET.CFG](#) is the configuration file that the Sockets TSR's (both [SOCKETP.EXE](#) and [SOCKETM.EXE](#)) use at startup. This configuration file defines the interface parameters that the TCP/IP stack will use when performing network communications. This includes such things as IP addresses definitions, hardware and software interfaces, routing and operational parameters.

For details on the commands used in this file, see the Sockets Configuration chapter later in this manual.

GOPD.BAT

`GOPD.BAT` is a batch file that executes the necessary programs to get the Sockets TCP/IP stack up and running over Ethernet. `GOPD.BAT` contains the following:

```
@echo off
:-----
:: Batch file to load Sockets with packet driver.
:: Supports TCP/IP over Ethernet.
:-----

:-----
:: Set SOCKETS environment variable
:-----
SET SOCKETS=C:

:-----
:: Set up interrupts for Sockets and packet driver.
:-----
netset

:-----
:: Load packet driver at vector 60 hex.
:-----
pcktdrv 0x60

:-----
:: Load Sockets TSR (TCP/IP stack).
:: Use the second line if you are going to use ftpapi, which takes more memory.
:-----
socketp
::socketp /m=30000

:-----
:: Load FTP support TSR (uncomment this line if FTP support is desired).
:-----
::ftpapi
```

Make sure `GOPD.BAT`, `NETSET.EXE`, `PCKTDRV.COM`, `SOCKETP.EXE`, `SOCKETS.CFG` (and optionally `FTPAPI.EXE`) are loaded into the root directory of your ICL-4300. Then enter `GOPD` from the command prompt and verify the programs load with no errors. You are now ready to run a TCP/IP application. If you would like to use FTP with your application, `FTPAPI.EXE` must be loaded. To accomplish this, uncomment the last line of the `GOPD.BAT` file. You must also load `SOCKETP` with additional memory reserved by enabling the second `SOCKETP` line (the one that loads `SOCKETP` with the `/m` command line switch) instead of the first `SOCKETP` line.

If the unit is reset or power cycles then `GOPD` must be run again. If your application terminates `GOPD` does NOT have to be run again, as the programs it loads remain resident.

XPING.EXE

`XPING.EXE` is an application that pings another host on your network. Use `XPING` to verify that your unit is properly connected and Sockets is up and running. To ping another host computer enter the following command at ICL-4300 DOS prompt:

```
XPING [ip_address]
```

`ip_address` host IP address in dotted decimal format, such as 192.168.1.100

You should see transmit and receive counts both increment if the ICL-4300 is able to ping the host you specified.

Also, try pinging your ICL-4300 from the other host to make sure your IP address is configured as you expect. If your host is running Win95/98/NT shell to a DOS prompt and enter the following command at the DOS prompt:

```
PING [ip_address]
```

`ip_address` ICL-4300 IP address specified in your `SOCKET.CFG` file, such as 192.168.1.200

The ping statistics should show 4 packets sent and 4 packets received.

Chapter 3 – Sockets Configuration

This chapter walks you through the details of the Sockets configuration sample file. For full details on the command syntax used in Sockets configuration files, see the Sockets Command Reference chapter later in this manual.

SOCKET.CFG Sample File

Let's look at the sample `SOCKET.CFG` file used to configure the Ethernet interface on the ICL-4300.

```
#-----  
# File:          SOCKET.CFG  
# File type:     ICL-4300 Sockets Configuration File  
# Project:       tutorial  
# Description:   This file configures the Ethernet interface on the  
#               ICL-4300.  
#-----  
  
#-----  
# Interface  
#-----  
  
ip address 192.168.1.199  
  
iface pdr if0 dix 1500 10 0x60  
  
#-----  
# TCP Options  
#-----  
  
tcp mss 1460  
tcp window 2920  
  
#-----  
# Routing  
#-----  
  
route add default if0  
  
#-----  
# Display info  
#-----  
  
ip address  
  
#-----  
# End of File  
#-----
```

Sockets configuration files are simple text files that may be created or edited with any ASCII text editor.

In this sample file, you'll notice that there are a number of comments. A comment starts with the pound '#' character, and continues to the end of the line.

This file contains a series of commands, parameters and associated value. It is these commands that define the interface of the Sockets TCP/IP stack. Let's look at each command.

- `ip address 192.168.1.199`

`ip` commands set values and options of the next interface to be defined.

This command sets the IP host address of the next interface to be defined. You should set this to a valid IP address allowed on your network. Remember all IP addresses have to be unique. Your network administrator should be able to assign you an address.

- `iface pdr if0 dix 1500 10 0x60`

`iface` commands define communication interfaces that are used at the network interface level. The command includes values for the interface type, name and class along with other related parameter.

This command defines a packet driver interface that has the following values:

<code>pdr</code>	the interface type uses a packet driver
<code>if0</code>	the interface name (this is an arbitrary name selected by the user)
<code>dix</code>	the interface class is an Ethernet Interface known as Ethernet I (Blue Book Ethernet)
<code>1500</code>	specifies the maximum transmission unit size (1500 for Ethernet)
<code>10</code>	specifies the number of incoming datagrams which may be queued at one time
<code>0x60</code>	specifies the interrupt vector number of the packet driver

- `tcp mss 1460`

`tcp` commands set various TCP operating settings.

This command sets the maximum segment size to 1460 bytes.

- `tcp window 2920`

This command sets the maximum receive window size to 2920 bytes.

- `route add default if0`

`route` commands create entries in the IP routing table.

This command sends all transmissions to IP addresses not defined by other route commands to the `if0` interface. Since no other route commands are specified, all transmissions will be sent to the `if0` interface.

- `ip address`

This command displays the current value of the IP host address.

Chapter 4 – Sockets Command Reference

This chapter provides a reference for the Sockets configuration files.

Sockets configuration files are simple text files that may be created or edited with any ASCII text editor, then downloaded to the ICL-4300.

Syntax Rules

- In the command summary, use is made of a parameter labeled *hostid*. This denotes a host, router, gateway, or network. *hostid* may be specified either by a symbolic name listed in the HOSTS file, or a numeric IP address in dotted decimal notation (such as 192.10.240.1). For information on the HOSTS file, see the HOSTS File chapter later in this manual.
- **bold** type indicates a reserved key word as part of the command syntax and is to be typed exactly as indicated.
- *italics* indicate that the term is a parameter to be specified by the user.
- [] Square brackets indicate that the enclosed item is optional.
- / A forward slash is used as a leading character for an optional switch in some commands. Both the forward slash and the switch that follows it form part of the command syntax. The equal sign before extra parameters is optional in most cases.
- | The vertical bar indicates that there is a choice between two or more selections, but that only one of the options indicated may be specified.
- # The pound character is to insert comments, which continue to the end of the line.
- Command and parameters are not case sensitive.

Commands

For each command, the parameter values that are relevant to the ICL-4300 are given. Default values are listed where applicable. Commands are listed in alphabetical order.

arp

arp adds a new entry to or deletes an entry from the Address Resolution Cache. **arp** with no parameters displays the contents of the Address Resolution Cache.

Syntax

arp add *hostid* ether | ipx | ieee *hw_addr*

arp drop *hostid* ether | ipx | ieee

Sub-commands

arp add includes a new entry in the Address Resolution Cache. Do not add entries with duplicate IP or hardware addresses, as this will cause malfunctioning of the network.

arp drop deletes an entry from the Address Resolution Cache.

Parameters

hostid

The IP address of a remote host that is to be added to the ARP cache. This value may be a symbolic name from the HOSTS file or a decimal (dotted) address.

hw_addr

Used to add the hardware (node) address of the remote host whose IP address is given in *hostid*. This must be a six-digit hexadecimal address separated by colons for Ethernet and a ten-digit hexadecimal address for IPX.

Example Commands

```
arp add unix_host ether 00:00:65:0D:E6:04
arp add 127.0.1.3 ether 00:00:65:0D:E6:04
arp drop brian ipx
arp drop 192.6.1.12 ether
```

Example Output

```
ARP:  Received 1  BadType 0  BadAddress 0
      RequestsIn 1  Replies 0  RequestsOut 2
IP addr      Type           Time      Q  Addr
192.6.1.4    10 MbEthernet  14        1  [unknown]
192.6.1.2    10 MbEthernet  891       00:00:d6:00:46:52
```

bootp

bootp changes the default retransmit time (5000 milliseconds) and timeout (30000 milliseconds) for **bootp** requests. A **bootp** request is made automatically as soon as the first **iface** statement is processed with a zero, or no, IP address. Consequently, the **bootp** specifications should occur before the **iface** statement. The **bootp** request is made after every retrans time until timeout has elapsed. To cancel the **bootp** request sooner, press any key.

Syntax

```
bootp retrans milliseconds  
bootp timeout milliseconds
```

Sub-commands

```
bootp retrans changes the default retransmit time value  
bootp timeout changes the default timeout value.
```

Parameters

milliseconds

An integer value equal to the requested time in milliseconds.

Example Commands

```
bootp retrans 5000  
bootp timeout 30000
```

domain

If a host name is not a decimal (dotted) address and it is not found in the HOSTS file and at least one Domain Name Server has been defined, an attempt is made to obtain the address from the defined DNS server(s). The number of times any server is polled (retries), in addition to the time to wait for a response, can also be specified. A suffix may be specified and is attached to all names not containing any dots.

All of the following sub-commands can be issued without the optional parameters to obtain information on the current status.

Syntax

```
domain server [host_name]  
domain retry [retry_count]  
domain time [wait_time]  
domain suffix [domain]
```

Sub-commands

domain server adds a DNS server address or lists the current servers if *host_name* not specified.

domain retry specifies the retry count for polling each server. **domain retry** lists the retry count if *retry_count* not specified.

domain time specifies or lists the time (milliseconds) to wait for a response before attempting retry.

domain time lists the time (milliseconds) to wait if *wait_time* not specified.

domain suffix specifies the domain suffix to add to all simple names; names that contains no dots.

domain suffix lists the domain suffix if *domain* is not specified.

Example Commands

```
domain retry 3  
domain server 196.2.1.1  
domain suffix myorg.co.za  
domain time 2000
```

iface \ interface

iface is a synonym for the **interface** command.

interface informs Sockets of the hardware or software communications interface(s) to be used at the network interface level. At least one network interface is required, and two or more are used in gateway (router) applications.

The class or mode of each interface defines the encapsulation used for packaging the data frame into the transport frame. Some types of interface support only one class.

When using more than one interface, Sockets assigns the previous given IP address in the .CFG file to this interface and uses its net mask to add a route to that net through this interface. Using the same IP address would result in multiple routes to the same network. The default route is set on the first interface with an IP address with a zero net mask (For example, ip address 19.63.10.11/0).

Syntax (general)

interface *type name class other parameters*

Syntax (specific)

interface pdr *name dix mtu numbuf intvec*

interface asy *name [slip | cslip | ppp] mtu buflim ioaddr iovec speed [modemfile]*

Parameters

Type

Type defines the type of hardware or software interface. **interface** supports the following interfaces.

Interface	Description
pdr	Packet driver
asy	Standard PC asynchronous interface (RS232 port)

name

name defines the name by which the interface is known on the local host. *name* is a symbolic name known only to the local host on which it is used.

name may be arbitrarily assigned. Each interface command on the same host must have a unique name assigned. This name is used by commands such as route, trace, param, and so on.

class

class specifies how IP datagrams are to be encapsulated in the link level protocol of the interface. Some interfaces offer a choice between classes while others use a fixed class. The following classes are available and are listed with their associated types.

Type	Class (defined in the following list)
<code>pdr</code>	dix, ieee, token, driver, slip
<code>asy</code>	slip, cslip, ppp

Class	Description
<code>dix</code>	The DEC/Intel/Xerox Ethernet interface also known as Blue Book Ethernet or Ethernet I.
<code>token</code>	IBM Token Ring. Source routing is supported for multiple rings.
<code>ieee</code>	IEEE: 802.3 Ethernet with SNAP headers.
<code>driver</code>	Use the default class for the packet driver.
<code>slip</code>	Serial Link Interface Protocol (SLIP) for point-to-point asynchronous links. This mode is compatible with UNIX SLIP.
<code>cslip</code>	Compressed Serial Link Internet Protocol (SLIP) for faster reaction over point-to-point synchronous links.
<code>ppp</code>	Point-to-point protocol.

mtu

mtu specifies the Maximum Transmission Unit size, in bytes. Datagrams larger than this limit are fragmented into smaller pieces at the IP layer. The maximum value of *mtu* for the various interfaces is:

- Ethernet - 1500
- IPX - 546
- Netbios - 512
- For serial links a standard value for *mtu* is 576. (576 is the maximum according to specifications, but may be increased on reliable connections as long as both sides use the same value.)

numbuf

numbuf specifies how many incoming datagrams may be queued on the receive queue at one time. If this limit is exceeded, further received datagrams are discarded. This mechanism is used to prevent fast interfaces from filling up memory when data cannot be handled fast enough.

buflim

buflim specifies the maximum number of outgoing datagrams (SLIP) or packets (SLIP_LAPB or HDLC) to queue before starting to discard datagrams. This mechanism is used to prevent the memory from filling up when a serial link goes down.

intvec

intvec specifies the software interrupt number (vector) in hexadecimal to use for resident packet driver.

ioaddr

ioaddr is the I/O base address in hexadecimal of a serial port. The standard values for the ICL-4300 serial ports are:

COM1	0x03F8
COM2	0x02F8
COM3	0xFC80
COM4	0xFC90
COM5	0xFCA0
COM6	0FCB0
COM7	0xE800

iovec

iovec is the hardware interrupt vector used by the serial port. The standard values for the ICL-4300 serial ports are

COM1	4
COM2	3
COM3	8
COM4	9
COM5	12
COM6	14
COM7	6

speed

speed specifies the transmission speed for serial interface devices (baud rate). If external clocking is to be used on scc devices, this parameter must be specified as 0. This value may be changed while Sockets is running by using the param command. Before using a serial connection you have to set flow control with the param command.

modemfile

A file containing the modem commands and scripts.

Examples

```
interface pdr if0 dix 1500 5 0x60
interface asy ser0 cslip 576 15 0x3f8 4 9600
```

ip

ip displays or sets the values of the options selected when defining the IP (internet protocol) host address of the next interface to be defined.

Syntax

ip address [*hostid* [*/net_bits*]

ip status

ip ttl [*number*]

Sub-commands

ip address sets the IP host address of the next interface to be defined. A route is automatically added to each interface for the default or specified net mask for its address. To make an automatic route the default, specify the net bits as zero. When specified without the optional parameters, **ip address** displays the current value(s) of the local host IP address(es). To assign different IP addresses to different interfaces on the same host, an **ip address** statement must precede each interface definition. The last IP address given is used in case of missing **ip address** statements.

ip status displays Internet Protocol (IP) statistics, such as total packet counts and error counters of various types. It also displays statistics on the Internet Control Message Protocol (ICMP). This includes the number of ICMP messages of each type sent or received.

ip ttl sets the default time-to-live value which is placed in each outgoing IP datagram. The ttl value limits the number of gateway hops the datagram is allowed to take in order to kill datagrams that get stuck in loops.

Parameters

hostid

hostid specifies the IP host address to assign to the next interface to be defined. This may be a symbolic name from the HOSTS file, or a dotted decimal address.

/net_bits

A net mask can be specified for the host. In the **ip address** command an optional */net_bits* can be used to indicate the number of bits in the network ID. The net mask is used to determine whether an incoming datagram is a broadcast and also for sending UDP broadcasts.

Net masks are more easily represented in binary or hexadecimal format. For example, the IP address 128.1.1.5/24 corresponds to a net mask of 255.255.255.0 (FFFFFF00h), 25 bits to 255.255.255.128 (FFFFFF80h) and 26 bits to 255.255.255.192 (FFF FFC0h).

The default net mask used corresponds to the class of address used if not explicitly specified.

Net Bits	Net Mask	Class	IP address range
8	255.0.0.0	A	0.x.x.x to 127.x.x.x
16	255.255.0.0	B	128.x.x.x to 191.x.x.x
24	255.255.255.0	C and higher	192.x.x.x or higher

If you want to subdivide your network, you can divide it by two for every net bit added. The following table provides information on converting between net bits and net mask. The number of net bits to add when changing a 0 in the net mask to:

Net Bits	Net Mask	Net Bits	Net Mask
1	128	5	248
2	192	6	252
3	224	7	254
4	240	8	255

number

When the *number* parameter is omitted, **ip ttl** displays the current value of the time to live parameter.

param

param invokes a device-specific control routine.

- `param ifname [up | down]`
- `param ifname clear`
- `param ifname disconnect`
- `param ifname address ip_address`

When executed without parameters, **param** displays defined interface names and device-specific flags. **param** operates differently for each interface type and interface mode. In many cases, it is used to interrogate the status of an interface. The **ifstat** and **param** commands perform similar and, in some cases, exactly the same function.

Syntax

```
param ifname [arg1...argn]
```

Parameters

ifname

ifname defines the name used in the interface command for the device to be controlled.

arg1...argn

These parameters depend on the type of interface in use.

Example

To display current serial link settings and restart the statistics on it, use:

```
param s10 clear
```

Address Change Sub-command

The **address** sub-command changes the current IP address on an interface. It is useful on serial connections to hosts that supply a different IP address after a connection is made.

Syntax

```
param ifname address ip_address
```

Example

```
param s10 address 192.17.138.23
```

Alternative Routing Control Sub-commands

The Alternative Routing Control Sub-commands set up and check the Sockets alternative route mechanism. More than one route can be specified to a target host or network. The first route that has an associated interface in the up state is used.

An interface is in the up state when it is defined by the interface command. It enters the query state when it does not receive valid input within a specified up-time period after sending data expecting a response. At this stage three ICMP echo requests (ping) are sent to query an IP address (catering for links with a high data loss). It enters the down state by a Sockets command or when it does not receive valid input within the specified up-time period after entering the query state. If an up-time has never been specified or a value of 0 is specified, the interface will stay in the up state whether valid input is received or not.

An interface enters the up state by a Sockets command or when valid input is received on that interface when in the down or query states. An ICMP echo request is sent on an interface in the down state every down-time period. If a down-time has never been specified or a value of 0 is specified, the ICMP echo request will not be sent. Up-time and down-time is specified in seconds.

Syntax

```
param ifname [ up | down ]
param ifname [ uptime | downtime ] time
param ifname query hostname
```

Example Alternative Routing

Two X.25 interfaces are used to get to the target network 192.6.1.0. The first interface, named **if0** should preferably be used, but if it stops receiving for a period of 20 seconds, it should try to ping 192.6.1.2 and if no response is received within another 20 seconds, **if1** should take over, but **if0** should be tried every five seconds. Interface **if1** should disconnect after 80 seconds of no traffic.

The Sockets configuration file should contain the following:

```
interface x25 if0 ... ..
param if0 uptime 20
param if0 downtime 5
param if0 query 192.6.1.2
interface x25 if1 ... ..
param if1 uptime 80
param if1 downtime 5
param if1 query 192.6.1.2
route add 192.6.1.0 if0
route add 192.6.1.0 if1
```

In the case of both **if0** and **if1** failing, both are tried every five seconds until one comes up. The return paths should also be maintained in a similar way with Sockets or by using RIP.

Break Sub-command for Serial Interfaces

The **break** sub-command activates or deactivates the "disconnect on break character" facility. It is useful for giving serial terminals connected to a Sockets terminal server, the opportunity to close a session by using the Break key. The default condition is off.

Syntax

param *ifname* break [on | off]

Clear Sub-command for Serial Interfaces

The clear sub-command displays the current status and statistics of a serial connection and then clears the statistics to start counting from zero again.

Syntax

param *ifname* clear

Disconnect sub-command for Serial Interfaces

The disconnect sub-command disconnects the modem connection by dropping the DTR for approximately 10 seconds.

Syntax

param *ifname* disconnect

COM Port Speed and Flow Control Sub-command

This **param** sub-command is only to be used on asy, aslink, scc or lcs type interfaces. On a class raw interface, this allows the baud rate, flow control and data parameters to be set. On a CSLIP or SLIP interface only the baud rate should be set.

Syntax

param *ifname* *speed* *out_flow* *in_flow* bits

Parameters

ifname

ifname is the interface name of an asy, aslink, scc or lcs interface.

speed

speed sets the baud rate for a serial link. The standard speeds are: 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 and 115200.

out_flow

out_flow sets the output flow control for a serial link. When not supplied it defaults to none.

in_flow

in_flow sets the input flow control for a serial link. When not supplied it defaults to none. The following list shows *out_flow* and *in_flow* selections.

<code>none</code>	no flow control
<code>Xon</code>	Xmit on/Xmit off (Ctrl-Q, Ctrl-S)
<code>DCD</code>	Data Carrier Detect modem signal
<code>CTS</code>	Clear To Send modem signal
<code>DSR</code>	Data Set Ready modem signal
<code>DTR</code>	Data Terminal Ready modem signal
<code>RTS</code>	Request To Send modem signal
<code>ixxx</code>	inverse of modem signal xxx

Invalid selections are ignored or replaced by more logical selections. The usage of Xon/Xoff is not recommended for most applications. The `ixxx` support is for non-standard equipment that uses a reversed signal on the required pin.

bits

bits sets the number of data bits per character, number of stop bits, and parity for a serial link. It is a three-character string consisting of `dsp` from the following table. When not supplied, *bits* defaults to value of `81n`.

Values for d	5, 6, 7, or 8 data bits
Values for s	1 or 2 stop bits
Values for p	o = odd parity e = even parity n = no parity m = mark s = space

Example

```
param asy1 57600 cts rts 81n
param slp0 4800 xon xon 72e
```

RIP Advertising Sub-command for Interfaces

When the `rip advertise` command has been used, this **param** sub-command makes allowance to disable and re-enable RIP advertising on a specific interface.

Syntax

```
param ifname [ ripadv | noripadv ]
```

Examples

```
param if0 noripadv
param if1 ripadv
```

rip

The Routing Information Protocol (RIP) allows a Sockets gateway to advertise routes and allows Sockets to recognize advertised routes from Sockets and other gateways. These two facilities can be individually selected. See also the **param** command for selecting **rip** on specific interfaces.

Syntax

```
rip advertise [time [1|2] [ self ]]  
rip use [time ]
```

Parameters

time

time is the number of elapsed seconds before the advertisement is repeated when used in the advertise sub-command.

time is the period during which routes are added or amended as a result of RIP, and are valid for *time* seconds. Such added or amended routes are dropped if another RIP advertisement is not received within *time*.

Advertise Sub-command

rip advertise causes an immediate advertisement of all relevant routes and repeats it every *time* seconds. The default value for time is 30 seconds. **rip** version 2 advertisements are sent by default. To send only version 1 advertisements, add a 1 on the command line.

When rip advertise is selected, all interfaces advertise all routes except those routes making use of that specific interface (split horizon) and routes marked Private. (To prevent certain interfaces from using RIP, see the parameter command.) A route which is dropped as a result of a RIP update or which becomes unavailable as a result of its associated interface going into the Down state, is immediately advertised as being infinite (metric = 16) and is not advertised until it becomes available again. The advertisement is sent as a sub-net broadcast using the net mask and IP address of the interface.

Example

```
rip advertise 30 self
```

Update RIP Sub-command

rip use causes a RIP request for routes and a continuous update of routes according to RIP advertisements. Routes added or amended as a result of RIP are valid for time seconds and are dropped if another RIP advertisement is not received within that time. The default value of time is 240 seconds.

When rip use is selected, routes are updated according to received RIP advertisements. Routes added or amended as a result of RIP have a timeout associated with them. If another RIP advertisement is not received during that time, the route is dropped. A route is also dropped if an advertisement of infinity (metric = 16) is received. To prevent dropping a route, it must be marked as Static.

Note on RIP and static routes:

The metric of a route marked **static** is never updated by a RIP advertisement. Instead, a duplicate route is added before the static route. If the duplicate route is dropped as a result of a timeout or RIP, the static route is used again.

Notes on RIP and mdd or x25 interfaces:

On **mdd** or **x25** interfaces, a RIP route is only sent when the last datagram was not a RIP route. This is done to allow an interface to time out and be disconnected when not used.

Also, the route on the receiving end will still time out and become unavailable. This means that inactivity timers should be shorter than route timers; as specified in rip use timer.

When an **aslink** to **mdd** association is broken, all non-static routes using the specific **mdd** are marked as having a hop-count of **infinity**; the metrics are set to 16.

route

route creates an entry in the IP routing table for Sockets to determine where to send data. When no arguments are specified, the current IP routing table is displayed. The route command also displays the remaining time for routes having been updated by RIP (Routing Information Protocol). **route** displays the following status indicators:

- X Proxy ARP used by route
- P Private route
- S Static route
- T Triggered update pending.

The Alternative Routing mechanism allows more than one route to be specified to a particular host or network. Failure of one route causes an automatic switch to the next route.

Refer also to the **ip** address command for specifying the net mask, because a route is automatically added to each interface for the default or specified net mask for that address. When multiple routes are defined to the same address, Sockets uses the route with the network size (largest number of bits in the net mask).

Syntax (general)

```
route [ add | drop destination ifname [gateid |none [ metric [proxy] [private] [static] ] ] ]
```

Syntax (specific)

```
route
route add [ hostid | netid ] ifname [gateid]
route add [ hostid | netid[/mask] ] ifname [gateid]
route add default ifname
route drop [ hostid | netid ]
route drop [ hostid | netid[/mask] ]
route drop default
```

Parameters

add or drop

Sub-command to add or drop (remove) a route from the routing table.

default

All transmissions to IP addresses not otherwise defined in routing commands are sent via the network interface specified by *ifname*.

hostid

hostid is the IP address of a destination remote host to which data must be sent, or a remote host which must be removed from the routing table (dropped).

netid

netid is the IP address of a destination network to which data must be sent. Any host with this IP network address is able to receive the data. Whether a particular host will use the data depends on the host portion of the specific IP address in the IP header.

mask

mask specifies the number of bits in the network portion of the address if sub-netting is used. If not used, the network portion of the address is determined according to the class (A, B or C) of the address.

ifname

ifname defines the name used in the interface command for the immediate network on which the data for the designated host must be sent. This is the network level interface to be used by the local host to reach the remote host.

gateid

gateid parameter specifies the IP address of a host, on the same physical network as the local host, which is used as a gateway or router to a different network. The gateway or router host specified in *gateid* must be directly reachable on the same physical network as the local host defining this gateway. In other words, this must be the nearest gateway to this local host.

metric

When using RIP or Proxy ARP a value from 0 to 16 for *metric* must be specified indicating the distance or cost of that route. A *metric* of 16 indicates that the route is down.

proxy, private and static

To support the Routing Information Protocol (RIP) the route command utilizes the proxy, private and static key words. These words can be used in any order following *metric*.

Proxy ARP should be used with care and not in conjunction with RIP. When more than one host responds to an ARP request, it can cause confusion and even lead to system crashes. This is possible in situations where more than one gateway implements Proxy ARP to a common destination.

When "RIP advertising" is selected, all interfaces advertise all routes except those routes making use of that specific interface (split horizon) and routes marked private. A route which is dropped as a result of a RIP update or which becomes unavailable as a result of its associated interface going into the down state, is immediately advertised as being infinite (metric = 16) and is not advertised until it becomes available again. In order for an interface to be used for advertising, a route without a gateway using that interface must be available. The advertisement is sent as a sub-net broadcast using the net mask of the host and the IP address of the interface.

When "RIP using" is selected, routes are updated according to received RIP advertisements. Routes added or amended as a result of RIP, have a timeout associated with them. If another RIP advertisement is not received during that time, the route is dropped. A route is also dropped if an advertisement of infinity (metric = 16) is received. To prevent dropping a route, it must be marked as static. The metric of a route marked static is never updated by a RIP advertisement. Instead a duplicate route is added before the static route. If the duplicate route is dropped as a result of a timeout or RIP, the static route is used again.

Examples

```
route (display current routing table entries)
route add default ipx0
route add unix_net eth0
route add unix_host ipx1 unx_gate
route add unix_net2 eth0 /eth 1
route add unix_net ipx0 unx_gate
route add subnet/26 eth0 sub_gw
route drop unix_net
```

route can specify a Proxy ARP on a route, as follows:

```
route add net interface gateway metric [proxy]
```

When using Proxy ARP, gateway and metric must be specified. If no gateway is used, none can be specified. For example:

```
route add 192.6.1.0 ifx25 none 5 proxy
```

tcp

tcp commands display or set various TCP operating parameters

Syntax

```
tcp irtt [time]  
tcp lport [port_number]  
tcp mss [size]  
tcp prod [con_number]  
tcp reset [con_number]  
tcp retry [number]  
tcp rtt con_number time  
tcp smss [size]  
tcp status [con_number]  
tcp window [size]
```

Parameters

time

time is the initial round-trip-time (IRTT) estimate in milliseconds and is used for new TCP connections until the actual value can be measured and adapted to. By increasing this value when operating over slow communication links, unnecessary retransmissions that otherwise occur before the smoothed estimate value approaches the correct value are minimized. The system default is 5000 milliseconds.

For **tcp rtt**, *time* is the new round-trip time in milliseconds.

port_number

port_number is the local port starting number.

size

For **tcp mss**, *size* is the maximum segment size in bytes sent on all outgoing TCP connect requests (SYN segments). *size* tells the remote host the size of the largest segment that may be received by this host. When changing the MSS value, any existing connections remain unchanged.

For **tcp smss**, *size* is the send maximum segment size in bytes sent on all outgoing TCP connect requests. This limits the size of the largest segment that may be sent by this host. When changing the SMSS value, any existing connections remain unchanged.

For **tcp window**, *size* is the size of the receive window in bytes for any new TCP connections. Existing connections are unaffected.

con_number

con_number is the connection number and can be obtained by executing a **tcp status** command.

number

number is the number of retries attempted without receiving an acknowledge from the remote host before the connection is broken. If the value exceeds 255, it implies an infinite number of retries; such a connection does not time-out. The default value for *number* is 6.

irrt Sub-command

tcp irtt displays or sets the initial round-trip-time estimate. When specified without an argument, the command displays the current values of TCP parameters including the initial round-trip-time in milliseconds.

To affect incoming connections, **tcp irtt** should be executed before the servers are started.

Example

```
tcp irtt 120
```

Sample Output

```
TCP: IRTT 5 ms Retry 6 MSS 1460 SMSS 1460 Window 2920
```

lport Sub-command

tcp lport specifies the local port starting number. When specified without a number the current value of the next free local port number is displayed.

Example

```
tcp lport 2004
```

Sample output

```
Lport = 2004
```

mss Sub-command

tcp mss displays or sets the TCP maximum segment size in bytes. When size is not specified, the current values of the TCP parameters, including the maximum segment size, are displayed. It is recommended to reduce the MSS and SMSS on bad network connections.

Example

```
tcp mss 1460
```

prod Sub-command

tcp prod forces a retransmission on the TCP connection indicated by *con_number*. Obtain *con_number* by executing a **tcp status** command. Retransmissions are done automatically, but the time interval between them lengthens after a retransmission with no response. Use the *tcp prod* to skip the waiting period after a problem has been rectified, such as when a communications cable has been plugged in.

reset Sub-command

tcp reset deletes the TCP connection specified by *con_number*. Any subsequent incoming traffic on this connection receives a RESET response.

retry Sub-command

tcp retry displays or sets the retry count before a connection is broken. When specified without the number parameter, **tcp retry** displays the current values of TCP parameters, including the retry count.

rtt sub-command

tcp rtt replaces the automatically computed round-trip time (RTT) for the specified connection with the time in milliseconds. Sockets calculates the RTT as a smooth average of past measured RTTs, starting with the IRTT on a new connection. To get the current RTT in use for a connection n, use the tcp status n command that will give the smoothed average RTT indicated by SRTT. Because this command provides a manual override of the normal back-off retransmission timing mechanisms, this command may be used to speed up recovery from a series of lost packets.

Example

```
tcp rtt 4 100
```

smss sub-command

tcp mss displays or sets the TCP send maximum segment size in bytes. When size is not specified, the current values of the TCP parameters, including the SMSS, are displayed. A small SMSS causes the remote to reduce its segment size. **tcp mss** can reduce the MSS and SMSS on bad network connections with high loss rates or where large packets get lost.

Example

```
tcp smss 512
```

status sub-command

tcp status displays **tcp** status information. When specified without the *con_number* parameter, the current **tcp status** values and information on all current **tcp** connections are displayed.

Example

```
tcp status
tcp status 4
```

Sample Output

```
TCP Status
ConOut 2  ConIn 3  ResetOut 0  Runt 0  ChksumErr 0  Bdcsts 0
Con#   Rcv-Q   Snd-Q   Local Socket   Remote Socket   State
0      0        0       192.6.1.5:2002 192.6.1.2:20    Time Wait
1      0        0       192.6.1.5:23   0.0.0.0:0       Listen (S)
2      0        288     192.6.1.5:23   192.6.1.5:2004  Established
3      0        10      192.6.1.5:2004 192.6.1.5:23    Established
4      0        0       192.6.1.5:2001 192.6.1.2:21    Established
5      0        0       192.6.1.5:21   0.0.0.0:0       Listen (S)
6      0        0       192.6.1.5:7    0.0.0.0:0       Listen (S)
7      0        0       192.6.1.5:25   0.0.0.0:0       Listen (S)
8      0        0       192.6.1.5:9    0.0.0.0:0       Listen (S)

TCP con# 4:
Local: 192.6.1.5:2001 Remote:192.6.1.2:21 State:Established
Init seq  Unack   Next      Resent   Cwind   Thrsh   Wind   MSS   Queue  Total
750000    750092  750092    0        1551    65535  2920  1460  0      91
996032704          996032991 0          2920    0      286
Timer stopped SRIT 1715 ms SRTT 550 ms Mean dev 2432 ms
```

window sub-command

tcp window displays or sets the default and maximum receive window size. When specified without the size parameter the current TCP parameters, including the current window size, are displayed.

Example

```
tcp window 2920
```

Chapter 5 – PPP

PPP (Point-to-Point Protocol) is used to establish a TCP/IP connection over a serial port interface. When used in conjunction with a network session/port on the ICL-4300, PPP may also be used to establish a dial-up modem connection. See the appropriate development kit (C++, ESP or ISaGRAF) for details on configuring of a network session/port for PPP dial-up support.

PPP1.CFG Sample File

Let's look at the sample `PPP1.CFG` file used to configure a PPP interface port on the ICL-4300.

```
#-----  
# File:          PPP1.CFG  
# File type:     ICL-4300 Sockets Configuration File  
# Project:       tutorial  
# Description:   This file configures COM7 on the ICL-4300 for a  
#               PPP server with password protection.  
#-----  
  
#-----  
# Interface  
#-----  
  
iface asy p7 ppp 1500 30 0xe800 6 19200  
  
#-----  
# Serial options  
#-----  
  
par p7 ipcpin local compress tcp 16 1  
par p7 ipcpin local address 192.168.1.199  
  
par p7 lcpin local accm 0  
par p7 lcpin local acfc on  
par p7 lcpin local pfc on  
par p7 lcpin local magic on  
  
par p7 ipcpin listen  
par p7 lcpin listen  
  
#-----  
# Routing  
#-----  
  
route add default p7  
  
#-----  
# Display info  
#-----  
  
ip address  
  
#-----  
# End of File  
#-----
```

This file contains a series of commands that configure the COM7 modem port on the ICL-4300. The port is configured to listen for a PPP connection that is initiated from a remote host. This makes the ICL-4300 act like a PPP "server".

- `iface asy p7 ppp 1500 30 0xe800 6 19200`

The `iface` command defines a communication interface that is used at the network interface level. The command includes values for the interface type, name and class along with other related parameter.

This command defines an asynchronous port interface that has the following values:

<code>asy</code>	the interface type uses standard PC asynchronous port (RS-232 port)
<code>p7</code>	the interface name (this is an arbitrary name selected by the user)
<code>ppp</code>	the interface class is point-to-point protocol
<code>1500</code>	specifies the maximum transmission unit size
<code>30</code>	specifies the number of outgoing datagrams which may be queued at one time
<code>0xe800</code>	specifies the base port address for the COM port
<code>6</code>	specifies the IRQ number for the COM port
<code>19200</code>	specifies the baud rate for the COM port

- `par p7 ipcpin local compress tcp 16 1`
`par p7 ipcpin local address 192.168.1.199`

```
par p7 lcpin local accm 0
par p7 lcpin local acfc on
par p7 lcpin local pfc on
par p7 lcpin local magic on
```

```
par p7 ipcpin listen
par p7 lcpin listen
```

The `par` command is used to set local and remote LCP/IPCP options. The `ipcpin` and `lcpin` parameters specify options for the incoming PPP connection. The `ipcp` and `lcp` parameters would be used to specify options for an outgoing PPP connection

In our example, the local address that will be assigned to the PPP interface on the ICL-4300 when a connection is established is `192.168.1.199`.

- `route add default p7`

The `route` command creates an entry in the IP routing table.

This command sends all transmissions to IP addresses not defined by other route commands to the `p7` interface. Since no other route commands are specified, all transmissions will be sent to the `p7` interface.

- `ip address`

This command displays the current value of the IP host address.

PPP2.CFG Sample File

Let's look at the sample `PPP2.CFG` file used to configure a PPP interface port on the ICL-4300 using password protection.

```
#-----  
# File:          PPP2.CFG  
# File type:     ICL-4300 Sockets Configuration File  
# Project:       tutorial  
# Description:   This file configures COM7 on the ICL-4300 for a  
#               PPP server with password protection.  
#-----  
  
#-----  
# Interface  
#-----  
  
iface asy p7 ppp 1500 30 0xe800 6 19200  
  
#-----  
# User passwords  
#-----  
  
user user1 pass1 192.168.1.201  
user user2 pass2 192.168.1.202  
  
#-----  
# Serial options  
#-----  
  
par p7 ipcpin local compress tcp 16 1  
par p7 ipcpin local address 192.168.1.199  
  
par p7 lcpin local accm 0  
par p7 lcpin local acfc on  
par p7 lcpin local pfc on  
par p7 lcpin local magic on  
  
par p7 lcpin local authen pap  
  
par p7 ipcpin listen  
par p7 lcpin listen  
  
#-----  
# Routing  
#-----  
  
route add default p7  
  
#-----  
# Display info  
#-----  
  
ip address  
  
#-----  
# End of File  
#-----
```

This configuration file is similar to the previous example except for a few commands that have been added to support PAP (Password Authentication Protocol). PAP is the standard protocol used with PPP to handle password verification.

- `user user1 pass1 192.168.1.201`
`user user2 pass2 192.168.1.202`

The `user` command is used create list containing each user name and password that is allowed access to the interface. The IP address assigned to the remote host may also be specified for each user.

In our example, two users are specified (`user1` and `user2`) with individual passwords (`pass1` and `pass2`). `User1` will be assigned IP address `192.168.1.201` and `user2` will be assigned IP address `192.168.1.202`.

- `par p7 lcpin local authen pap`

This parameter enables PAP as the authentication protocol to use for the specified interface port.

Chapter 6 – FTP

FTP (File Transfer Protocol) transfers binary or text (ASCII) files between host systems and provides for directory and file maintenance (rename, delete, etc.).

FTPAPI.EXE

The TCP/IP Support Kit can be used in conjunction with an application program (ESP, ISaGRAF or C) to support FTP client and server operations on the ICL-4300. To support FTP, the `FTPAPI.EXE` program must be loaded. This TSR provides FTP facilities that can be used by an application program (without the application program, it does nothing). The `FTPAPI.EXE` program should be loaded after Sockets. For an example of how to load `FTPAPI.EXE`, see the file `GOPD.BAT`.

Important note: if you load `FTPAPI.EXE`, Sockets must be started with additional memory reserved, using the `/m` command line switch as shown below:

```
socketp /m=30000
ftpapi
```

The default amount of memory (16,384 bytes) is insufficient and will result in program crashes.

SOCKET.UPW

`SOCKET.UPW` is the user password file that controls FTP file access. This file is an ASCII text file that contains user names, user passwords and file access privileges. The format for the file is:

```
username password directory access
```

<code>username</code>	symbolic name of the remote user
<code>password</code>	password that must be entered by the remote user (* = no password required)
<code>directory</code>	default directory which remote user is logged in to
<code>access</code>	access allowed by user as shown in table below

Access	Description
<code>r</code>	allows user read-only access
<code>s</code>	allows user read and write access
<code>c</code>	allows user to create new directories
<code>d</code>	allows user to access any directory or drive

The `SOCKET.UPW` file is searched for in the path specified by the `SOCKETS` environmental variable. If not set, the path `\DL\SOCKETS` is used as the search path.

When an FTP client attempts to connect, this file `SOCKET.UPW` is read by `FTPAPI.EXE`, verifying the user name and password.

Sample user password file

Let's look at the sample `SOCKET.UPW` used by the Sockets FTP server.

```
#-----  
# File:          SOCKET.UPW  
# File type:     ICL-4300 User Password File  
# Project:       tutorial  
# Description:   This file sets up user access and file privileges for the  
#               SFTPSERV server application program.  
#-----  
  
user1 pass1 \ wd  
anonymous * \public r  
admin all \ rwcd
```

In this sample file, you'll notice that there are a number of comments. A comment starts with the pound '#' character, and continues to the end of the line.

Note that user names and passwords are not case sensitive.

This file contains a list of user password entries, lets look at each.

- `user1 pass1 \ wd`

This entry defines a user named `user1` with a password of `pass1`. When this `user1` logs in, he will be logged into the drive's root directory. `User1` will be able to read/write files and change directories.

- `anonymous * \public r`

This entry defines an `anonymous` user login that does not require any specific password. Typically, your email address is used as the password for an anonymous login. The user will be logged into the directory `\public` and only allowed to read files.

- `admin all \ rwcd`

This entry defines a user named `admin` with a password of `all`. The `admin` user will be logged into the root directory and allowed all privileges. Typically, a network administrator would be allowed unrestricted access.

Chapter 7 – HOSTS File

The `HOSTS` file is one of the fundamental files used for communicating with other computers on your network. The structure and use of the `HOSTS` file is well defined in the TCP/IP world, even among different operating systems such as Win95, WinNT, UNIX or OS/2.

This file is created by you and stored on your local host computer, the ICL-4300. This file lists the symbolic names and IP addresses of other host computers known by your ICL-4300. This allows your application to reference other hosts on your network by their symbolic name as opposed to their IP address in dotted decimal format.

Both the Sockets configuration file commands and an application that makes calls to the Sockets compatibility API can take advantage of the `HOSTS` file. Whenever a host ID is required as a parameter, the symbolic name may be substituted for the IP address.

HOSTS

`HOSTS` is an ASCII text file that contains symbolic host names and host IP addresses. The `HOSTS` file does not have a file extension. The format for the file is:

```
host_name ip_address
```

```
host_name      symbolic name of host computer
ip_address     IP address of host computer in dotted decimal format
                (such as 192.168.1.200)
```

The `host_name` for each entry must start on the first column of each line and the `ip_address` must follow separated by one or more spaces.

The `HOSTS` file is searched for in the path specified by the `SOCKETS` environmental variable. If not set, the path `\SOCKETS` is used as the search path.

Sample HOSTS file

Let's look at a sample `HOSTS` file.

```
#-----  
# File:           HOSTS  
# File type:      ICL-4300 Hosts File  
# Project:        tutorial  
# Description:    This file lists names and IP addresses of host  
#                 computers which are connected to your network.  
#-----  
  
# my local computer  
192.168.1.200 localhost  
  
# other computers  
192.168.1.10  rtu1  
192.168.1.20  rtu2
```

In this sample file, you'll notice that there are a number of comments. A comment starts with the pound '#' character, and continues to the end of the line.

This file contains a list of host names and IP addresses, lets look at each.

- `192.168.1.200 localhost`

This entry is used to define a host name for your local host computer, such as the target ICL-4300 your application is going to be running on.

- `192.168.1.10 rtu1`
`192.168.1.20 rtu2`

These entries define the host names for other computers that are connected to your network. `Rtu1` and `rtu2` may be ICL-4300 controllers or other equipment, configured to be remote telemetry units, with which the local host may wish to communicate.