Parker Automation

p/n 88-025360-01A

Ethernet Specification: ACR Series Products

Effective: December 2004

IMPORTANT

User Information

Warning!

ACR Series products are used to control electrical and mechanical components of motion control systems. You should test your motion system for safety under all potential conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel.

ACR series products and the information in this guide are the proprietary property of Parker Hannifin Corporation or its licensers, and may not be copied, disclosed, or used for any purpose not expressly authorized by the owner thereof.

Since Parker Hannifin constantly strives to improve all of its products, we reserve the right to change this guide, and software and hardware mentioned therein, at any time without notice.

In no event will the provider of the equipment be liable for any incidental, consequential, or special damages of any kind or nature whatsoever, including but not limited to lost profits arising from or in any way connected with the use of the equipment or this guide.

Technical Assistance

Contact your local automation technology center (ATC) or distributor.

North America and Asia
Parker Hannifin
5500 Business Park Drive
Rohnert Park, CA 94928
Telephone: (800) 358-9070 or (707) 584-7558
Fax: (707) 584-3793
Email: emn_support@parker.com
Internet: http://www.parkermotion.com

Germany, Austria, Switzerland
Parker Hannifin
Postfach: 77607-1720
Robert-Bosch-Str. 22
D-77656 Offenburg
Telephone: +49 (0) 781 509-0
Fax: +49 (0) 781 509-176
Email: eme_application@parker.com

Europe (non-German speaking)
Parker Hannifin
21 Balena Close
Poole, Dorset
England BH17 7DX
Telephone: +44 (0)1202 69 9000
Fax: +44 (0)1202 69 5750
Email: eme_application@parker.com

Italy
Parker Hannifin
20092 Cinisello Balsamo
Milan, Italy via Gounod, 1
Telephone: +49 (0) 781 509-0
Fax: +49 (0) 781 509-176
Email: sales.sbc@parker.com

Parker Automation

Technical Support E-mail
emn_support@parker.com

Introduction

For developers writing their own Ethernet drivers, this document describes the implementation of Ethernet for the ACR series controller.

If you are using the communications server (comserverACR.dll) supplied with ACR-View, you do not need this document.

This specification applies to the following:

ACR Series Controllers..... Operating System revision 1.18.14 or greater

Overview

Some ACR series motion controllers are equipped with an Ethernet interface for communications. The following ports are available:

- Port 5002: Unmanaged ASCII Command Communication
- Port 5003: Fast Status
- Port 5004: Connection Management (watchdog)
- Port 5006: Managed ASCII and Binary Command Communications

There are five ACR command streams, which provide the means to enter ACR commands and get responses: STREAM1, STREAM2, STREAM3, STREAM4, and STREAM5.

- Command connection STREAM1 is dedicated to the USB interface.
- Command connections STREAM2 through STREAM5 are served to clients and attached to ACR command streams by two servers for ports 5002 and 5006.

ACR Command Communication— Ports 5002 & 5006

There are two command connection servers, each of which listens for client connections on separate ports.

- Port 5002 serves connections that expect only ASCII commands. This is an unmanaged connection. It cannot accept binary commands, and does not require byte padding or packing from the client.
- Port 5006 serves connections that can accept ASCII and binary commands. This is a managed connection. The client must provide byte padding and packing—see below.

Byte Packing and Padding

A connection on port 5006 allows ASCII and binary commands to use a single communication stream. Packing and padding allow the stream to identify which data are ASCII or binary commands.

- Stream—A portion of the controller that parses the incoming bytes and composes the responses.

- Connection—A link through which bytes pass between the external clients and the internal streams.

The connection on an ACR controller receives data (binary and ASCII) and stores it in a buffer. The stream then processes the data in four-byte packets. If there are less than four bytes of data, the stream waits until there are four bytes to process.

Binary commands require a four-byte packet structure. The first byte of binary communication must be a zero—this identifies the data as binary. In addition, that same first byte must be at the beginning of a four-byte packet. The controller processes all data through the stream as ASCII unless it recognizes the Binary format as described above. For more information on binary data packets, see the ACR Users Guide (Online Help) in the ACR-View software.

ASCII commands require management (port 5006 only) so the controller can correctly identify and execute both Binary and ASCII commands. This is done by checking the byte count for ASCII characters, and padding them out to four-byte packets if they are not a multiple of four.

- If the ASCII characters are a multiple of four, then no further action is necessary. The ASCII characters are processed. If a Binary command follows, the stream recognizes the signifier—a zero in the first byte of a four-byte packet—and processes the Binary command.
- If the ASCII characters are not a multiple of four, then pad the end of the ASCII characters with zeros to create a four-byte packet. This ensures the command is processed immediately, and is not caught in the buffer waiting for additional bytes—remember, the stream only processes data in four-byte packets.

Echoes and responses from the controller can be binary or ASCII, and are not padded. Using the same method as the controller, a client can recognize a binary response from the zero in the first byte of a four-byte packet. However, the client must also be able to interpret the remaining four-byte packet in order to know how long the binary response is. For information about binary communication, see the ACR Users Guide (Online Help) in the ACR-View software.

Example
Suppose the client sends an ASCII command followed by a Binary command.

The user sends VEL22 in ASCII, setting the velocity to twenty-two, and is followed by a carriage return. The client determines the ASCII characters are only six bytes. The client then pads the ASCII to make a multiple of four. Following is the ASCII command and its translation in hexadecimal:

```
VEL22<cr>00 = 56 45 4C 32 32 0D 00 00
```

The padding of the ASCII command places the first byte of the subsequent binary command at the start of its own four-byte packet.

The subsequent Binary command requests the actual position of axes two, three, and five. The binary packet begins with zero in the first byte, indicating a binary command. It then specifies code thirty, index two, and masks four, eight, and twenty—this information is derived from the Axis Parameters tables. Following is the Binary command translated to hexadecimal:

```
00 30 20 2C
```

The connection receives the following transmission:

```
56 45 4C 32 32 0D 00 00 00 30 20 2C
```

The stream parses the data in four-byte packets, thereby separating the ASCII command from the Binary command:

```
56 45 4C 32
32 0D 00 00
00 30 20 2C
```

Fast Status—Port 5003 UDP

You can access Fast Status data using UDP connections on port 5003. Up to five clients can subscribe to the Fast Status data.

Note: You can configure the Fast Status data using the AcroBasic FSTAT commands.

Subscription Request Format

The Fast Status Subscription request consists of eight bytes per packet to configure the status update interval. The first four bytes set the update mode:

- If nonzero, then the next four bytes set the rate (in milliseconds) that status data is automatically transmitted.
- If zero, and this client had previously subscribed, then the connection is stopped—i.e. becomes unused.

Receive Packet (port 5003) Data Description		
Function	**Size**	**Description**
Update mode	4 bytes	Sets the mode to update fast status information if non-zero.
Update interval	4 bytes	Sets the fast status update interval in milliseconds.

Fast Status Transmit Format

The format of the fast status transmit packet is 80 pieces of 32-bit data configured by the FSTAT commands. There is no header or count information.

As with all multi-byte data sent in these Ethernet packets, the format is big-endian—the first byte in the packet is the most significant byte of the first piece of 32-bit data. The client must decide whether to interpret a piece of 32-bit data as an integer or floating point. This is determined by the data selected with the corresponding FSTAT command.

For more information, see the FSTAT command in the ACR Users Guide (Online Help) in the ACR-View software.

Transmit Packet (port 5003) Data Description		
Function	**Size**	**Description**
Group 0 Data	32 bytes	A block of eight 32-bit parameters. You can select this using the FSTAT0 command.

…	…	…
Group 9 Data	32 bytes	A block of eight 32-bit parameters. You can select this using the FSTAT9 command.

Connection Management—Port 5004

There are five connection management sockets available for services required to effectively manage the interface between the controller and its clients. The services are as follows: watchdog protection, controller to client alarms (interrupts), and operating system download over Ethernet.

The organization of management sockets is similar to ACR command sockets. There is a single TCP server listening on port 5004 for up to five connection requests. The management sockets will have to examine the incoming packets to determine the nature of the client's request.

Watchdog Function

The Watchdog feature is designed both to "re-assure" the client that the controller is alive, and to detect the loss of client activity and respond by closing an ACR command socket or fast status socket.

To identify which client is requesting watchdog protection, the client IP address and port number are included in the watchdog request. In addition, the requesting client specifies the time interval in seconds for watchdog, and the number of retries per time interval before watchdog times out. The client expects to receive at least one echo within the specified interval.

Receive/Transmit Echo

Watchdog Packet Data Description		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x01 identifies this packet as a watchdog request
Watchdog Timer	4 bytes	Time interval in milliseconds for watchdog. Zero disables the watchdog.
Watchdog Ticker	4 bytes	Number of retries per time interval before watchdog times out.
IP address	4 bytes	The IP address of the client connections to be guarded
ACR port	4 bytes	The port number of the client Command connection to be guarded
Fast status port	4 bytes	The port number of the client Fast Status connection to be guarded

Client Watchdog Support

In normal operation, the controller responds to a client watchdog packet by echoing the same packet back to the client. This occurs when the socket(s) described by the request are connected and alive. If the Command or Fast

Status connections described in the packet are not connected and alive, then the packet is not echoed.

Server Watchdog Function

The controller's server watchdog will detect the absence of client watchdog packets, given the duration as determined by the following: (Timer * Ticker). It then closes the connections specified in the most recent watchdog packet. Automatically closing the connection allows a command socket to be available for a subsequent client request.

A client can request that the controller stop guarding a command connection—if the value for "Timer" in a watchdog request packet is zero, the watchdog stops functioning.

Alarms to Clients

If the packet is an alarm subscription request, Connection Management socket parses the filter data from the packet and stores it in the socket's filters. These filters allow the Connection Management sockets to qualify the message content individually for each client connection by ANDing the message with the filter. If the result is nonzero, the message is sent to the client.

Alarms sent to clients are identified by the same ID word as the subscription request.

The user alarms are generated by changes in a user's program, such as changes to parameters. The controller alarms are generated by the controller's firmware.

Alarm Subscription Formats

Alarm Subscription Packet Data—User		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x02 identifies this packet as an user alarm subscription request
User Alarm Filter	4 bytes	Filter value for user invoked alarms

Alarm Subscription Packet Data—Controller		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x03 identifies this packet as an controller alarm subscription request
Controller Alarm Filter	4 bytes	Filter value for controller invoked alarms

Alarm Response Formats

Alarm to Client Packet Data—Client		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x02 identifies this packet as an user alarm
User Alarm long	4 bytes	User's value for user invoked alarms

Alarm to Client Packet Data—Controller		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x03 identifies this packet as an controller alarm
Controller Alarm long	4 bytes	Alarm code for controller invoked alarms
Object identifier	4 bytes	Identifies which controller object caused the alarm
Event Count	4 bytes	Number of times this event has occurred

Operating System Download

Important: This is for informational purposes only. We recommend using the ACR-View software for operating system downloads.

You can use the Connection Management sockets to download a new operating system to the controller. This is accomplished with four different types of messages packets from the client: request for the configuration file, download request, download termination request, and reboot request. Each plays a different role in the overall process of downloading an operating system. The process is as follows:

1. Get configuration file and verify that proposed operating system is compatible with ACR hardware.
2. Send download blocks using the download request packets.
3. Send download termination request and evaluate the controller's response packet. If the result is anything other than success, steps 2 and 3 may be repeated.
4. Send reboot command, even if steps 2 and 3 never succeeded, because the controller is now halted.

Configuration File Request

The request for the configuration file allows the client to examine the controller configuration to ensure that the hardware is compatible with the operating system that is about to be downloaded.

Receive

Packet Data from Client Description		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x04 identifies this packet as a configuration file request from the client

Transmit

Packet Data to Client Description		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x04 identifies this packet as a configuration file request result
Configuration file	1024 bytes	1020 file bytes plus 4 CRC bytes

Download Request

For each download request packet, the block index is examined first. The 1024 byte payload is then copied to the location corresponding to that index within an internally allocated operating system Loader file structure. This file structure is an array of 1024 byte blocks, and the provided block index will be used as the index into that array. The maximum size of an operating system is 1024 blocks of 1024 bytes. The actual size will be determined by noting the largest block index sent with a download request before a download termination request is sent.

Receive

Operating System Download Packets Data from Client Description		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x05 identifies this packet as an operating system download request
Block index	4 bytes	Index into array of 1024 byte blocks representing the new operating system
Data	1024 bytes	Partial operating system

Download Preparation

The following steps are performed automatically by the controller as it prepares to receive the new operating system:

- Obtain a semaphore that prevents any other Connection Management socket from also trying to download a new operating

system to the controller. This ensures only one socket is attempting to carry out the remaining steps.

- Stop the server tasks for command connections, management connections, and fast status connections. The connections themselves are closed next, but the servers must be closed first to prevent new connections from being opened.
- Close all fast status senders, all command connections, and all management connections other than the one requesting the operating system download. By explicitly closing the connections instead of just letting the task die, the client applications can perform orderly shutdowns also. With all connections closed, the watchdogs serve no purpose and are disabled.
- Prepare the controller for download. The controller will shut down all programs, all streams, fast status, and CANopen update.

Download Termination Request

If the incoming packet is an operating-system download-termination request, a Cyclic Redundancy Check (CRC) is performed over the entire received operating system Loader file to detect file errors, minus the CRC field itself.

The CRC algorithm will conform to the Ethernet standard.

Polynominal = 0x04C11DB7

Initial remainder = 0xFFFFFFFF

The calculated CRC is then compared to the CRC field. The result of this check is sent to the client in a packet identified by the same ID word as the download termination request.

Receive

Termination Request Packet Data		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x06 identifies this packet as an operating system download termination request

Transmit

Termination Result Packet Data		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x06 identifies this packet as an operating system download termination request result
Result	4 bytes	Zero = download successful, 1 = CRC failure, 2 = flash memory failure

If the CRC check is successful, the new operating system is written to flash memory in 1024 byte blocks. The Connection Management socket finally sends a download termination result packet to the host, with the format described above.

Reboot Request

If the CRC check was not successful, the new operating system is not written to flash memory. Whether or not the new operating system was copied, the controller must reboot because all processes have been shut down. If the incoming packet is an operating system reboot request, the controller reboots. When the controller comes up from reboot, the operating system (new or old) is loaded from flash memory.

A separate packet for an operating system reboot request allows the client the choice of several actions in the event that the CRC check does not pass.

The reboot request can be used outside the context of operating system download. When the request is received, the Connection Management socket determines if the controller has already closed its processes and connections. If not, the controller does so, allowing the client applications to end gracefully. It then closes its own connection and reboots the controller.

Receive

Reboot Request Packet Data Description		
Function	**Size**	**Description**
ID word	4 bytes	A value of 0x07 identifies this packet as an Reboot request

Opening Connections

There are many different ways you can open a connection. Much depends on the protocol you want to use, what you are trying to do, and how you want to accomplish it.

We recommend the following reference books:

- Comer, Douglas E. *Internetworking with TCP/IP Volume 1, Principles, Protocols, and Architecture.* Prentice Hall, 2000. ISBN 0130183806
- Stevens, W. Richard. *The Protocols (TCP/IP Illustrated, Volume 1).* Addison-Wesley Pub Co, 1993. IBSN 020163346
- Stevens, W. Richard. *The Implementation (TCP/IP Illustrated*, Volume 2). Addison-Wesley Pub Co, 1995. IBSN 020163354X

Closing Connections

Because the ACR controller supports a limited number of concurrent connections, you should negotiate a short shutdown with the host prior to closing a socket. Otherwise, the peer might not properly free up the allocated socket resources.

An ACR controller cannot make more socket connections if the supported, concurrent socket connections are already used up. If the ACR Controller believes it has valid connections on all its sockets but actually does not, the unit will require manual intervention to free up the sockets.

Note: Using the Watchdog function can help reduce the risk of closing a connection ungracefully.

Example

The following pseudo code uses WinSock v2.2 to implement a graceful shut down:

```
SOCKET mySocket = Create a new Socket Connection to the ACR;
... host uses mySocket to talk to the peer (ACR)
// Time to Close
shutdown(mySocket, SD_SEND);  // Actual Winsock command
// Now able to perform final read because peer returns FD_CLOSE event
// once ready to close.  Then it is safe to destroy socket.
closesocket(mySocket);  // Actual Winsock commands
WSACleanup();
```