



# Apple® MacTCP™ Programmer's Guide

 **APPLE COMPUTER, INC.**

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual or the software may not be copied, in whole or in part, without written consent of Apple, except in the normal use of the software or to make a backup copy of the software. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose.

© Apple Computer, Inc., 1989  
20525 Mariani Avenue  
Cupertino, CA 95014  
(408) 996-1010

Apple, the Apple logo, AppleTalk, and Macintosh are registered trademarks of Apple Computer, Inc.

LocalTalk, MacTCP, and MPW are trademarks of Apple Computer, Inc.

ITC Garamond is a registered trademark of International Typeface Corporation.

Microsoft is a registered trademark of Microsoft Corporation.

PostScript is a registered trademark of Adobe Systems Incorporated.

Simultaneously published in the United States and Canada.

# Contents

Figures and tables / vii

## **Preface / ix**

What's in this guide / ix

Who should read this guide / ix

Document conventions / ix

Related documents / x

## **1 Introduction / 1**

Architectural overview / 2

Application, presentation, and session layers / 2

Transport layer / 3

    User Datagram Protocol (UDP) / 3

    Transmission Control Protocol (TCP) / 3

Network layer / 3

Link access layer / 4

Physical layer / 4

## **2 The MacTCP Driver / 5**

The PBOpen call / 6

The PBControl call / 7

The PBClose call / 8

Implementation notes / 8

    Breakpoints / 8

    Fragmentation and reassembly / 8

    Routing / 9

## **3 User Datagram Protocol / 11**

Data structures / 12

    Write Data Structures / 12

    Receive buffer area / 13

- Using UDP / 13
- UDP routines / 14
  - UDPCreate / 15
  - UDPRead / 16
  - UDPBfrReturn / 17
  - UDPWrite / 18
  - UDPRelease / 19
  - UDPMTU / 20
  - UDP asynchronous notification routine / 21
- C parameter block definitions / 23

## **4 Transmission Control Protocol / 25**

- Data structures / 26
  - Read and Write Data Structures / 26
  - Receive buffer area / 27
- Using TCP / 27
  - Streams and connections / 27
  - Asynchronous notification routine / 27
  - Connection opening / 28
  - Receiving data / 28
  - Sending data / 28
  - Time-outs / 28
  - Pushed data / 29
  - Urgent mode / 29
  - Connection closing / 30
  - Network management information / 30
  - Formatting MacTCP commands / 30
- TCP routines / 31
  - TCPCreate / 32
  - TCPPassiveOpen / 33
  - TCPActiveOpen / 36
  - TCPSend / 37
  - TCPNoCopyRcv / 38
  - TCPBfrReturn / 40
  - TCPRcv / 41
  - TCPClose / 43
  - TCPAbort / 44
  - TCPStatus / 45

TCPRelease / 47  
TCPGlobalInfo / 48  
TCP asynchronous notification routine / 50  
C parameter block definitions / 54

## **5 Name-to-Address Resolution / 59**

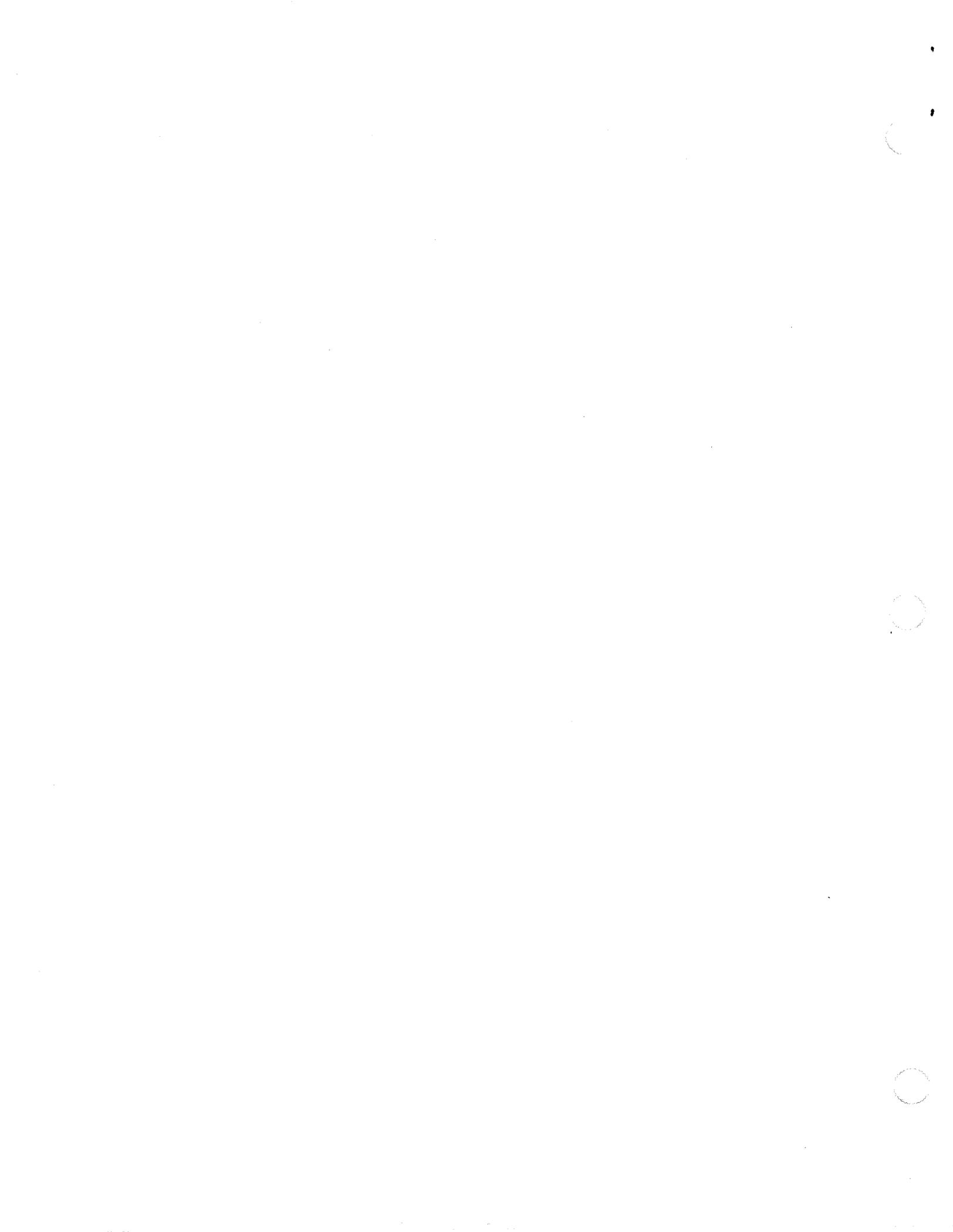
The AddressXlation.h header file / 60  
The OpenResolver call / 60  
The StrToAddr call / 60  
The AddrToStr call / 61  
The EnumCache call / 62  
The CloseResolver call / 63  
The AddrToName call / 63  
Binding the DNR to the application / 64  
DNR operation / 64

## **6 Miscellaneous Interfaces / 67**

MacTCPCommonTypes / 68  
Result codes / 68  
Miscellaneous types / 70  
Internet Control Message Protocol report structures / 71  
GetMyIPAddr / 72

## **Appendix Constants / 73**

Command codes / 74  
UDP asynchronous event codes / 74  
TCP asynchronous event codes / 74  
Reasons for TCP termination / 75



# Figures and Tables

**CHAPTER 1 Introduction / 1**

Figure 1-1 MacTCP protocols and OSI network layers / 2

**CHAPTER 3 User Datagram Protocol / 11**

Figure 3-1 UDP Write Data Structure / 12

Table 3-1 UDP routines / 14

**CHAPTER 4 Transmission Control Protocol / 25**

Figure 4-1 TCP Read and Write Data Structures / 26

Table 4-1 TCP routines / 31



# Preface

THIS GUIDE DESCRIBES how to create application programs for the MacTCP™ driver. MacTCP, Apple® Computer's implementation of the protocol suite known as Transmission Control Protocol/Internet Protocol (TCP/IP), increases the Macintosh® computer's ability to interoperate in a heterogeneous computer environment.

## **What's in this guide**

This guide is divided into six chapters and one appendix that contain the following information:

- Chapter 1, "Introduction," provides an overview of the MacTCP architecture.
- Chapter 2, "Making MacTCP Calls," describes the PBOpen, PBControl, and PBClose calls.
- Chapter 3, "User Data Protocol," describes UDP routines.
- Chapter 4, "Transmission Control Protocol," describes TCP routines.
- Chapter 5, "Name-to-Address Resolution," describes how textual names are resolved to IP addresses.
- Chapter 6, "Miscellaneous Interfaces," describes the types that are found throughout the programmatic interfaces supplied with MacTCP.
- The Appendix, "Constants," presents command codes, UDP asynchronous event codes, TCP asynchronous event codes, and reasons for TCP termination.

## **Who should read this guide**

This guide is designed both for Macintosh programmers who are not familiar with TCP/IP and for TCP/IP programmers who are not familiar with the Macintosh programming environment.

## **Document conventions**

This document reflects the Macintosh Programmer's Workshop (MPW™) conventions for types and sizes of variables and fields.

## Related documents

You might find the following reference materials useful.

- The *MacTCP Administrator's Guide*.
- *Inside Macintosh*, Volumes I-V.
- Douglas Comer, *Internetworking with TCP/IP* (Prentice-Hall, 1988).
- MacTCP implements protocols that conform to the following Request For Comments (RFC) and Military Standards (MIL-STD):
  - RFC 791, 894; MIL-STD 1777 (Internet Protocol)
  - RFC 768 (User Datagram Protocol)
  - RFC 793; MIL-STD 1778 (Transmission Control Protocol)
  - RFC 826 (Address Resolution Protocol)
  - RFC 903 (Reverse Address Resolution Protocol)
  - RFC 792 (Internet Control Message Protocol)
  - RFC 951, 1048 (Bootstrap Protocol)
  - IDEA004 (Routing Information Protocol)
  - RFC 1034, 1035 (Domain Name Resolver)
  - RFC 950 (Internet Subnetting)
  - RFC 1010 (Internet Assigned Numbers)

## Chapter 1 Introduction

MACTCP™ IS A SOFTWARE DRIVER for the Macintosh® Operating System that allows developers to create Macintosh applications for network environments that use the Transmission Control Protocol/Internet Protocol (TCP/IP). TCP/IP is a widely used industry standard for connecting multivendor computers. The TCP/IP protocol layers are fully compatible across all implementations on different hardware platforms, allowing different vendors' computers that run TCP/IP to interoperate and share data and services.

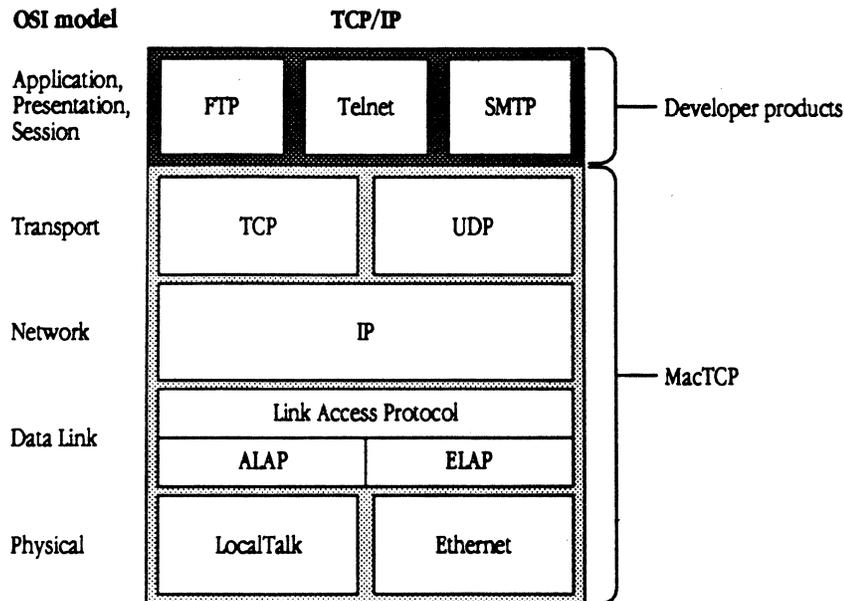
This chapter provides an overview of the MacTCP architecture. ■

---

## Architectural overview

MacTCP protocols partially conform to the International Standards Organization (ISO) Open Systems Interconnection (OSI) layers of networking functionality. Figure 1-1 shows a comparison of the OSI and MacTCP communications architecture.

■ **Figure 1-1** MacTCP protocols and OSI network layers



The TCP/IP protocols shown in Figure 1-1 are described in the following sections.

---

### Application, presentation, and session layers

The services of TCP, the User Datagram Protocol (UDP), and IP are augmented by application services in higher-level protocols such as the File Transfer Protocol (FTP), Telnet, and the Simple Mail Transfer Protocol (SMTP). These application services correspond to the application, presentation, and session layers of the OSI model.

---

## Transport layer

The UDP and TCP provide services at the transport layer of the OSI model.

### User Datagram Protocol (UDP)

UDP provides unreliable transmission of data between processes. UDP's transport of data is unreliable because, unlike TCP, it does not provide error checking, it does not acknowledge that data has been successfully received, and it does not order incoming messages. UDP messages can be lost, duplicated, or arrive out of order. Like TCP, UDP messages are encapsulated in IP datagrams for delivery.

The advantage of UDP is that the overhead associated with establishing and maintaining an error-free TCP session is avoided. Upper-layer protocols that don't require reliability use UDP to transmit data. For instance, the domain name system uses UDP because reliability is not critical; if there is no response to a domain name query, the resolver simply retransmits.

### Transmission Control Protocol (TCP)

TCP provides reliable transmission of data between processes. It ensures that data is delivered error-free, without loss or duplication, and in sequence.

Upper-layer protocols such as Telnet pass data to TCP for delivery to peer processes. TCP encapsulates the data into segments and passes the segments to IP, which puts the segments into datagrams and passes them across the internet. TCP at the receiving end checks for errors, acknowledges error-free segments, and reassembles the segments for delivery to upper-layer protocols. If a segment is lost or damaged, it will not be acknowledged, and the sending process will retransmit.

TCP has a flow control mechanism so that computers of different speeds and sizes can communicate. When TCP at the receiving end sends an acknowledgment, it also advertises how much data it is prepared to accept on the next transmission.

---

## Network layer

The Internet Protocol (IP) provides services at the network layer of the OSI model. IP is responsible for sending data across multiple networks. IP accepts segments of data from TCP or UDP, places the data in packets called datagrams, and determines the correct path for the datagram to take. The datagrams are sent across the internet, through as many gateways as needed, until they reach the destination host.

IP provides an addressing mechanism that allows routing between networks. The header of an IP datagram contains source and destination internet addresses so that any host in a network can route a packet to a destination, either directly or through a gateway.

IP has the ability to fragment a datagram as it is transmitted across a network. Since IP can be used with many different physical network implementations that specify different sizes for physical data frames, datagrams can be fragmented to fit into a small data frame. Fragments are reassembled as they arrive at the destination.

IP is often referred to as an unreliable delivery system because it makes a best-effort attempt to deliver all datagrams, but delivery is not guaranteed (TCP guarantees delivery). It is also called a connectionless delivery system because it routes each datagram separately. When IP receives a sequence of datagrams from TCP or UDP, it routes each datagram in the sequence individually and each datagram may travel over a different path to the destination.

---

### **Link access layer**

The AppleTalk® Link Access Protocol (ALAP) and Ethernet Link Access Protocol (ELAP) provide services at the link access layer of the OSI model. ALAP and ELAP provide best-effort delivery of information between devices. They provide the basic service of packet transmission between devices connected to a single physical network.

---

### **Physical layer**

MacTCP operates networks in which the physical layer uses the LocalTalk™ cable system or the Ethernet networking standard.

## Chapter 2 The MacTCP Driver

MACTCP IS A MACINTOSH Operating System driver that implements the Internet Protocol (IP), User Datagram Protocol (UDP), and Transmission Control Protocol (TCP). A part of the Macintosh Operating System called the Device Manager handles communication between applications and the MacTCP driver.

The MacTCP driver resides in the Macintosh System Folder. At startup, the MacTCP driver registers itself with the Device Manager as .IPP.

The MacTCP driver supports the following low-level Device Manager calls:

- PBOpen
- PBControl
- PBClose

---

## The PBOpen call

Before the application can exchange information with the MacTCP driver, the driver must be opened using the Device Manager PBOpen call, which opens the driver and returns its reference number. All subsequent calls to the driver use this reference number.

On machines with 1 megabyte (MB) of memory or more, the MacTCP driver uses the INIT-31 mechanism to automatically install itself in the device table and load itself into the system heap each time the machine is restarted. At load time, all internal buffers needed by MacTCP are also allocated on the system heap. These buffers are used (among other things) to hold incoming fragments awaiting reassembly and outgoing fragments following fragmentation. The amount of buffer space allocated is based on the amount of memory in the machine.

On machines with less than 1 MB of memory, MacTCP will not load itself at system startup; rather, the driver must be installed in the System file using the provided Installer script. When the PBOpen call is made, the driver is loaded into the application heap on a Macintosh 512K enhanced computer by the Device Manager.

The MacTCP driver performs its initialization sequence at PBOpen time. The initialization sequence includes initializing the local network handler, setting its local address and subnet mask, verifying that this address is reasonable and unused, and starting up the internal TCP/IP/UDP protocol machinery. PBOpen returns with an error code if any step in this process fails. In almost all cases, if PBOpen fails, the driver must be reconfigured and the system then restarted to clear the problem. Very few circumstances exist where transitory problems cause PBOpen to fail.

The reference number returned from the PBOpen call should be used in all subsequent PBControl calls to this drive. The ioCRefNum field should be assigned this value.

Because of the complexity of initializing MacTCP, PBOpen can return errors from many parts of the system including the Resource Manager, Device Manager, File Manager, Slot Manager, and AppleTalk driver.

---

## The PBControl call

The Device Manager PBControl call sends control information in the following parameter block to the driver:

```
struct CntrlParam {
    struct QElem *qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    ProcPtr ioCompletion;
    OSErr ioResult;
    char *ioNamePtr;
    short ioVRefNum;
    short ioCRefNum;
    short csCode;
    short csParam[11];
}
```

The type of information sent is specified by `csCode` in this parameter block. The MacTCP driver decides what to do based on the procedure number specified by the `csCode` (the Appendix lists procedure numbers).

All UDP and TCP routines documented in this guide are implemented as Device Manager Control calls. All MacTCP routines support both synchronous and asynchronous modes.

If a synchronous call is made, the application can't continue until the call is completed. When control is returned to the application, the result code in the parameter block is set indicating the success or failure of the call.

If the call is asynchronous, the I/O request is placed on the driver I/O queue, and control is returned to the calling program almost immediately. While requests are taken from the driver I/O queue one at a time and processed, the application is free to perform other tasks. The application has two ways of knowing when the call has been completed:

- By polling the `ioResult` field in the parameter block. When this value changes from `commandInProgress` to some other value, the call has been completed.
- By specifying an I/O completion routine to be called when the requested operation is complete.

The `ioCompletion` field in the parameter block contains a pointer to a completion routine to be executed at the end of an asynchronous call. The `ioCompletion` field should be `NIL` for asynchronous calls with no completion routine, and is automatically set to `NIL` for all synchronous calls.

The `ioNamePtr` and `ioVRefNum` fields are used internally by the driver.

Each routine description includes a parameter block format. The number next to each field name indicates the byte offset of the field from the start of the parameter block pointed to by A0. An arrow **next** to each parameter name indicates whether it's an input, output, or input/output parameter:

<b>Arrow</b>	<b>Meaning</b>
→	Parameter is passed to the routine.
←	Parameter is returned by the routine.
↔	Parameter is passed to and returned by the routine.

All Device Manager Control calls return an integer result code of type OSErr in the ioResult field of the I/O parameter block. Each routine description lists all of the applicable result codes generated by MacTCP, along with a short description of what the result code means.

---

## The PBClose call

The application does not need to issue a PBClose call to MacTCP. Because the driver can be shared by a variety of applications, once initialized, it never stops operation until the machine is restarted. PBClose returns an error code of closErr.

---

## Implementation notes

This section describes internal algorithms and decisions made by MacTCP.

---

## Breakpoints

The MacTCP driver contains several breakpoints that should never be encountered if the application is operating correctly. When one of these breakpoints occurs, it can be proceeded from and the text string that is printed should give some indication of the programmer-level error.

---

## Fragmentation and reassembly

The Internet Protocol (IP) has the ability to fragment a packet so that it can be sent across a network with a small maximum transfer unit (MTU). The fragments are reassembled as they arrive at the destination. Since the application developer must select the size of UDP datagrams (unlike TCP, which negotiates packet size), you should be aware of implementation limitations imposed on the size of these datagrams.

MacTCP internally allocates a buffer memory to temporarily hold all incoming fragments waiting to be reassembled, and all outgoing fragments waiting to be sent. The amount of space allocated varies depending on whether 1, 2, or 4 megabytes (MB) of memory is installed. The maximum size of the packet being fragmented or reassembled depends solely on the available free space in this buffer. Because the memory pool is shared by all MacTCP users, the allowance for the maximum size of a UDPWrite packet will vary, depending on competing demands on the memory pool. A given destination can only guarantee the reassembly of a packet that is 576 bytes (including IP and UDP headers); therefore, it is possible to send a packet from MacTCP that cannot be reassembled by the destination host.

---

## Routing

MacTCP supports routing through the Routing Information Protocol (RIP). Because MacTCP is internally listening to RIP broadcasts, attempting to create a stream on local port 520 will return a duplicateSocket error.

A default gateway can be configured using the Control Panel.



## Chapter 3 User Datagram Protocol

THE USER DATAGRAM PROTOCOL (UDP) provides a low-overhead transaction service to allow upper-layer protocols (ULP) to send datagrams between one another. UDP is datagram oriented with best-effort delivery, but it does not use acknowledgments to make sure messages are delivered to the destination, does not order incoming messages, and does not provide feedback to control the rate at which information flows between machines.

Like the Transmission Control Protocol (TCP), UDP uses port fields to specify the source and destination processes of each transaction. An optional checksum is also used. ■

---

## Data structures

The UDP packet is composed of an 8-byte header, followed by up to 65,507 bytes of data. The UDP header has the following structure:

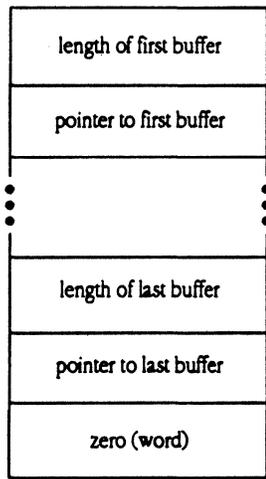
source port      word  
destination port word  
length            word  
checksum         word

---

## Write Data Structures

To send a UDP datagram, you must format a Write Data Structure (WDS), which has the format shown in Figure 3-1. A WDS specifies a list of buffers to write in a single operation.

■ **Figure 3-1** UDP Write Data Structure



The simplest WDS describes a single buffer and is 8 bytes in length: a word length, a long pointer, followed by a terminating word of 0. The most complex WDS that can be used to send a UDP datagram describes 6 buffers and is 38 bytes in length.

---

## Receive buffer area

The User Datagram Protocol (UDP) does not allocate memory for storing UDP stream databases or for buffering received datagrams. Instead, you must pass UDP enough memory for these purposes in the `UDPCreate` call. This has two advantages:

- The buffer memory can be allocated off the application heap instead of the system heap, which is very limited.
- You have control over the buffering provided by UDP and can allocate the appropriate amount of memory for the type of application and performance level desired.

This buffer area for incoming datagrams belongs to the `MacTCP` driver as long as the UDP stream is opened. When `UDPRelease` is called, this memory is returned to you and you can then reuse it or return it to the system.

---

## Using UDP

To send or receive UDP datagrams, you must first call `UDPCreate` to create a UDP stream, creating a port listener for the given UDP port and passing a memory block to the UDP driver to use in buffering incoming datagrams. `UDPCreate` also registers a routine that UDP uses to notify you of asynchronous events associated with this UDP stream. `MacTCP` can support 64 open UDP streams simultaneously.

To receive a UDP datagram, call `UDPRead`. Then, when you finish with the buffer that holds the received datagram, call `UDPBfrReturn`. To send a UDP datagram, call `UDPWrite`. When you finish using a UDP port, call `UDPRelease` to close the UDP stream.

- ◆ *Note:* Use of link-level packets larger than the maximum transfer unit (MTU) on LocalTalk networks is not advised because there are several problems with these packets on Datagram Delivery Protocol-Internet Protocol (DDP-IP) gateways. The packet MTU size should be negotiated by upper-layer protocols.

---

## UDP routines

This section describes calls to the UDP driver. The C parameter block definitions are presented at the end of the chapter. Table 3-1 lists each UDP routine and its function.

■ **Table 3-1** UDP routines

<b>Routine</b>	<b>Function</b>
UDPCreate	Opens a UDP stream
UDPRead	Retrieves a datagram received on a UDP stream
UDPBfrReturn	Returns a receive buffer to the UDP driver
UDPWrite	Sends a datagram on a UDP stream
UDPRelease	Closes a UDP stream
UDPMTU	Returns the maximum size of UDP data that can be sent in a datagram without IP fragmentation

---

---

## UDPCreate

<b>Parameter block</b>	→	26	word	csCode = UDPCreate
	←	28	long	stream pointer
	→	32	long	pointer to receive buffer area
	→	36	long	length of receive buffer area
	→	40	long	pointer to asynchronous notification routine (ASR)
	↔	44	word	local UDP port
	→	46	long	user data pointer

UDPCreate opens a UDP stream. It must be called before any UDP datagrams can be sent or received on a particular UDP port. UDPCreate returns a stream pointer that must be used in all subsequent UDP calls that operate on this UDP stream.

The receive buffer area is a block of memory that you must give to the UDP driver. UDP uses this memory to buffer incoming datagrams. This block of memory belongs to UDP while the stream is open; it cannot be modified or relocated until UDPRelease is called. The minimum allowed size of the receive buffer area is 2048 bytes, but it should be at least  $2N + 256$  bytes in length, where N is the size in bytes of the largest UDP datagram you expect to receive.

The ASR is called by UDP to notify the user of asynchronous events such as data arrival and Internet Control Message Protocol (ICMP) messages. If the routine is 0, you are not notified of asynchronous events. See the section "UDP Asynchronous Notification Routine" later in this chapter for more information.

If the local port is 0, UDP assigns an unused local port.

The user data pointer is returned in all ASRs for the created UDP stream.

<b>Result codes</b>	noErr	no error
	streamAlreadyOpen	there is already an open stream using this receive buffer area
	invalidLength	the receive buffer area is too small
	invalidBufPtr	the receive buffer area pointer is 0
	duplicateSocket	a stream is already open using this local UDP port
	insufficientResources	there are already 64 UDP streams open

---

## UDPRead

Parameter block	→	26	word	csCode = UDPRead
	→	28	long	stream pointer
	→	32	word	command time-out value in seconds (0 = infinite)
	←	34	long	remote IP address
	←	38	word	remote UDP port
	←	40	long	pointer to UDP data
	←	44	word	length of UDP data
	→	46	word	reserved; must be set to 0
	→	48	long	user data pointer

UDPRead retrieves a datagram that has been received on the UDP stream defined by the stream pointer. Some number of datagrams are buffered internally within UDP even when no UDPRead commands are outstanding, so it is not necessary to keep a UDPRead command outstanding at all times. The exact number of datagrams that can be buffered within MacTCP depends on the size of the receive buffer area given to MacTCP in the UDPCreate call and the size of datagrams received.

If a UDP datagram arrives on an open UDP stream and no UDPRead commands are outstanding, you are given a data arrival notification as a hint that a UDPRead command should be issued. See the section "UDP Asynchronous Notification Routine" later in this chapter for more information.

The command time-out period is specified in seconds. If no datagram arrives within the time-out period, the UDPRead command is completed in error. The minimum allowed value for the command time-out is 2 seconds. A zero command time-out means infinite; the UDPRead command will not be completed until a datagram arrives.

The remote IP address and remote UDP port specify the source of the datagram.

UDPRead can return successfully even though the length of UDP data is 0. This happens when a UDP packet arrives that has the passed-in value of the UDP stream's local UDP port, but contains no data. Since the UDPSend command permits sending zero-length UDP datagrams, the UDPRead command must pass up zero-length datagrams for symmetry.

For every UDPRead command that is completed successfully and returns a nonzero amount of data, you must call UDPBfrReturn with the same stream pointer and UDP data pointer, to return the receive buffer to the UDP driver for reuse.

<b>Result codes</b>	noErr	no error
	invalidStreamPtr	the specified UDP stream is not open
	commandTimeout	no data arrived within the specified period
	connectionTerminated	the UDP stream was closed by a UDPRelease command

---

## UDPbfrReturn

<b>Parameter block</b>	→	26	word	csCode = UDPbfrReturn
	→	28	long	stream pointer
	→	40	long	pointer to UDP data
	→	48	long	user data pointer

UDPbfrReturn returns a receive buffer to the UDP driver that had been passed to you because of a successful UDPRead call that returned a nonzero amount of data.

<b>Result codes</b>	noErr	no error
	invalidStreamPtr	the specified UDP stream is not open
	invalidBufPtr	the UDP receive buffer is not owned by the user

---

## UDPWrite

<b>Parameter block</b>	→	26	word	csCode = UDPWrite
	→	28	long	stream pointer
	→	34	long	remote IP address
	→	38	word	remote UDP port
	→	40	long	pointer to WDS
	→	44	byte	checksum flag
	→	46	word	reserved
	→	48	long	user data pointer

UDPWrite sends a datagram on a UDP stream.

The datagram's destination is specified by the remote IP address and remote UDP parameter ports. The buffers described by the WDS must not be modified or relocated until the command is completed. There can be up to 6 buffers described by the WDS. The total length of the UDP data described by the WDS must be between 0 and 8,192 inclusive. If the Checksum flag is nonzero, UDP computes and transmits a checksum; otherwise, the checksum is transmitted as 0. The reserved field must be set to 0.

In an Ethernet environment, the size of UDPWrite packets should be restricted to less than or equal to 8,192 bytes. Packets of this size can usually be reassembled by computers operating in Ethernet environments. In a LocalTalk environment, the size of UDPWrite packets should be restricted to less than or equal to 1,458 bytes because of Datagram Delivery Protocol-Internet Protocol (DDP-IP) gateway performance considerations.

- ◆ *Note:* Unless you have prior knowledge of the capabilities of the destination machine, the UDPWrite size should be limited to the value returned by the UDPMTU call for maximum interoperability.

<b>Result codes</b>	noErr	no error
	invalidStreamPtr	the specified UDP stream is not open
	invalidWDS	too many buffers in WDS or WDS pointer is NIL
	invalidLength	the total amount of data described by the WDS was greater than 65,535 bytes
	insufficientResources	too many datagrams are outstanding in the transmit queue
	ipNoFragMemErr	insufficient internal memory was available to fragment the packet
	ipRouteErr	unable to send the packet to an off-network destination because all gateways are down

---

## UDPRelease

<b>Parameter block</b>	→	26	word	csCode = UDPRelease
	→	28	long	stream pointer
	←	32	long	pointer to receive buffer area
	←	36	long	length of receive buffer area
	→	46	long	user data pointer

UDPRelease closes a UDP stream. Any outstanding commands on that stream are terminated with an error. The ownership of the receive buffer area used to create the UDP stream passes back to you.

- ◆ *Note:* Before UDPRelease is called, you must make sure that all pending UDPWrite commands have been completed. There is no way to abort a UDPWrite command in progress.

<b>Result codes</b>	noErr	no error
	invalidStreamPtr	the specified UDP stream is not open

---

## UDPMTU

<b>Parameter block</b>	→	26	word	csCode = UDPMaxMTUSize
	←	32	word	maximum datagram size
	→	34	long	remote address
	→	38	long	user data pointer

UDPMTU returns the maximum size of UDP data that can be sent in a single datagram without fragmentation. This number does not include the IP and UDP headers. The value is relative to the destination address. If the address is on the local network, the network MTU size is returned; otherwise, a value of 548 is returned.

<b>Result codes</b>	noErr	no error
---------------------	-------	----------

---

## UDP asynchronous notification routine

The asynchronous notification routine (ASR), which was registered with UDP in the UDPCreate call, is called by UDP to notify you of asynchronous events relevant to a particular UDP stream.

Since this routine is called from the interrupt level, you must not allocate or return memory to the system. Also, you are not allowed to make further synchronous MacTCP calls from an ASR. The values of all registers must be preserved except registers A0-A2 and D0-D2.

The C description of the ASR is as follows:

```
pascal void UDPNotifyProc (StreamPtr udpStream, unsigned short eventCode,
Ptr userDataPtr, struct ICMPReport *icmpMsg);
```

NotifyProc is passed to the UDPCreate call for use on notification of data arrival and ICMP message reception. If this method of notification is not desired, no procedure should be passed to the UDPCreate call. Asynchronous notification will be used with the UDPRead command only; all other commands are completed in a finite amount of time and can be called synchronously.

At entry, A0 contains the stream pointer, A1 contains a pointer to the ICMP report structure if the event code in D0 is ICMP received, A2 contains the user data pointer, A5 is already set up to point to application globals, and D0 (word) contains an event code.

<b>Event codes</b>	UDPDDataArrival	a UDP datagram has arrived on this stream but no UDPRead commands are outstanding
	UDPICMPReceived	an ICMP has been received on this stream; register A1 points to the ICMP report

An ICMP message reports an error in the processing of a datagram that was sent on a UDP stream. When an ICMP message is received, a data structure is passed up by UDP to the client to describe the received message. This data structure, called an ICMP report, has the following format:

0	long	stream pointer
4	long	local IP address of stream
8	word	local UDP port of stream
10	long	remote IP address (destination of original datagram)
14	word	remote UDP port
16	word	ICMP message type
18	word	optional additional information
20	long	optional additional information pointer

The values for the ICMP message type are as follows:

0	net unreachable
1	host unreachable
2	protocol unreachable
3	port unreachable
4	fragmentation required
5	source route failed
6	time exceeded
7	parameter problem
8	missing required option

Codes 0-3 are defined as follows:

- Net unreachable indicates that, according to the information in a gateway routing table, the network specified in the IP destination field of a UDP datagram is unreachable.
- Host unreachable indicates that a gateway determined that the host specified in the IP destination field of a UDP datagram is unreachable.
- Protocol unreachable indicates that a UDP datagram was delivered to the destination host, but UDP was not ready to receive any datagrams.
- Port unreachable indicates that a UDP datagram was delivered to the destination host, but no UDP client was listening on that particular port.

These ICMP messages may be received occasionally when the topology of the internet changes. A single destination unreachable message should not be taken too seriously; however, if several successive UDPsend commands each result in an ICMP report indicating that the destination is unreachable, the UDP client should assume that the remote host has either crashed or is no longer accessible.

The remaining codes (4-8) indicate problems in the format of the IP header on a UDP datagram. They are informational only. Since the UDP client has no access to the IP header, you cannot correct the error.

---

## C parameter block definitions

The following C type definitions are used with parameter-block Device Manager calls to UDP:

```
#define UDPCreate      20
#define UDPRead       21
#define UDPBfrReturn  22
#define UDPWrite      23
#define UDPRelease    24
#define UDPMaxMTUSize 25
#define UDPctlMax     29

typedef enum UDPEventCode {
    UDPDataArrival = 1,
    UDPICMPReceived,
    lastUDPEvent = 65535
};

typedef pascal void (*UDPNotifyProc) (
    StreamPtr udpStream,
    unsigned short eventCode,
    Ptr userDataPtr,
    struct ICMPReport *icmpMsg);

typedef void (*UDPIOCompletionProc) (struct UDPIOpb *iopb);

typedef unsigned short udp_port;

typedef struct UDPCreatePB { /* for create and release calls */
    Ptr          rcvBuff;
    unsigned long rcvBuffLen;
    UDPNotifyProc notifyProc;
    unsigned short localPort;
    Ptr          userDataPtr;
} UDPCreatePB;

typedef struct UDPSendPB {
    unsigned short reserved;
    ip_addr       remoteHost;
    udp_port      remotePort;
    Ptr          wdsPtr;
    Boolean       checksum;
    unsigned short sendLength;
    Ptr          userDataPtr;
} UDPSendPB;
```

The following parameter block is used with the UDPRead and UDPBfrReturn commands:

```
typedef struct UDPReceivePB
    unsigned short    timeOut;
    ip_addr           remoteHost;
    udp_port          remotePort;
    Ptr               rcvBuff;
    unsigned short    rcvBuffLen;
    unsigned short    secondTimeStamp;
    Ptr               userDataPtr;
} UDPReceivePB;

typedef struct UDPMTUPB {
    unsigned short    mtuSize;
    ip_addr           remoteHost;
    Ptr               userDataPtr;
} UDPMTUPB;

typedef struct UDPIopb {
    char              fill12[12];
    UDPIOCompletionProc ioCompletion;
    short             ioResult;
    char              *ioNamePtr;
    short             ioVRefNum;
    short             ioCRefNum;
    short             csCode;
    StreamPtr         udpStream;
    union {
        struct UDPCreatePB    create;
        struct UDPSendPB      send;
        struct UDPReceivePB   receive;
        struct UDPMTUPB       mtu;
    } csParam;
} UDPIopb;
```

## Chapter 4 **Transmission Control Protocol**

THE TRANSMISSION CONTROL PROTOCOL (TCP) is a highly reliable, connection-oriented byte-stream protocol. It is designed to operate over a wide variety of networks and to provide virtual circuit service with orderly transmission of user data. TCP serves as the basis for a reliable interprocess communication mechanism on top of the IP layer where loss, damage, duplication, delay, or misordering of packets can occur. ■

---

## Data structures

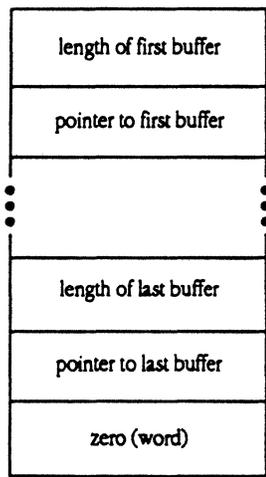
This section describes TCP Read and Write Data Structures and the buffer area.

---

### Read and Write Data Structures

The MacTCP driver uses Read Data Structures and Write Data Structures (RDS/WDS) to pass data between the user and TCP. These structures allow a single read or write operation to handle multiple blocks of data; that is, they allow scatter-read and gather-write capability. Figure 4-1 shows TCP Read and Write Data Structures.

- **Figure 4-1** TCP Read and Write Data Structures



The simplest possible Read and Write Data Structures are 8 bytes in length: a word length, a long pointer, and a terminating word of 0. No maximum number of buffers can be described by a RDS/WDS used to transfer data between TCP and the user.

---

## Receive buffer area

TCP does not allocate memory for storing TCP stream databases or for buffering received datagrams. Instead, you are required to pass TCP enough memory for these purposes in the `TCPCreate` call. This has two advantages:

- The buffer memory can be allocated off the application heap instead of the system heap, which is very limited.
- You have control over the buffering provided by TCP and can allocate the appropriate amount of memory for the type of application and performance level desired.

This buffer area for incoming datagrams belongs to the `MacTCP` driver as long as the TCP stream is open. When `TCPRelease` is called, this memory is returned to you and you can then reuse it or return it to the system.

---

## Using TCP

This section describes how to send and receive TCP segments.

---

## Streams and connections

A TCP stream supports one connection at a time. But a TCP connection on a stream can be closed and another connection opened without releasing the TCP stream. `MacTCP` can support 64 open TCP streams simultaneously.

---

## Asynchronous notification routine

When a TCP stream is created, a routine can be registered that TCP uses to notify you of asynchronous events relevant to that TCP stream. This routine is called the asynchronous notification routine (ASR).

Examples of events that `MacTCP` communicates to the user by means of an ASR include Time-out expired, Data arrived, and Connection terminated. Since the ASR is called from interrupt level, it cannot allocate or release memory. In addition, synchronous `MacTCP` driver commands cannot be issued from an ASR.

---

## Connection opening

To listen for an incoming connection, use the `TCPPassiveOpen` command. This command can be used to specify whether any incoming connection will be accepted or only connections from a particular remote IP address and TCP port.

To initiate outgoing connections, use the `TCPActiveOpen` command.

---

## Receiving data

Two methods are available for receiving data: a high-performance method and a simple method. You can choose the method that best suits the application or combine the two methods if desired.

Using the `TCPNoCopyRcv` routine is the high-performance method. Data is delivered to the user directly from the internal TCP receive buffers and no copy is required. An RDS is formatted to allow many received segments to be delivered to the user in one `TCPNoCopyRcv` command.

`TCPBfrReturn` must be called for every `TCPNoCopyRcv` that returns a nonzero amount of data in order to return the internal receive buffers to the `MacTCP` driver.

Using the `TCPRcv` routine is the simple method of receiving data. Data is copied from the TCP internal receive buffers into the user's buffer, which can be of arbitrary length and location. No RDS is used and no `TCPBfrReturn` call is required.

---

## Sending data

To send data on a connection that is already established, use the `TCPSend` command.

Both commands use WDSs to implement a gather-write capability. This allows you to send several noncontiguous chunks of data in one operation.

---

## Time-outs

TCP normally provides some time-out services to the upper-layer protocol (ULP). This time-out service is known as the ULP time-out. If there is data to send on a connection, but for some specified period of time either the data cannot be sent because the remote TCP is not set to receive any data or the data has been sent but the remote TCP has not acknowledged it, then the ULP time-out expires.

When the ULP time-out expires, TCP executes the specified ULP time-out action. There are two possible actions: abort the connection or report the timer expiration by means of an ASR and restart the timer.

In addition to this normal TCP timer, `MacTCP` provides command time-outs on commands that are not subject to the ULP timer. These commands are `TCPPassiveOpen`, `TCPNoCopyRcv`, and `TCPRcv`.

A TCPPassiveOpen command instructs TCP to wait for an incoming connection. The ULP timer is not started until the first connection initiation segment arrives. To prevent a TCPPassiveOpen command from waiting indefinitely for a connection initiation segment to arrive, a command time-out is provided. If no connection initiation segment is received within the specified time-out period, the TCPPassiveOpen command is completed with an error code.

Similarly, the TCPNoCopyRcv and TCPRcv commands are not protected by the ULP timer. In the absence of command time-outs, both commands could wait indefinitely for data to arrive. If a command time-out is specified on a TCPNoCopyRcv or TCPRcv command, the command is always completed within the specified time period. If no data has arrived at that point, the command is completed with an error. If some data has arrived, the command is completed successfully, returning the data that has arrived so far.

---

### **Pushed data**

Normally, TCP is allowed to collect data sent by means of TCPSend commands and to send that data in segments, as convenient. By setting the Push flag in the TCPSend command, TCP can be instructed to send all unsent data at once.

Similarly, TCP is allowed to collect received data internally and deliver it with reasonable promptness to the user. However, when pushed data arrives, TCP immediately delivers all received data to the user.

Note that there is no necessary relationship between the Push flag in a TCPSend command and segment boundaries. The push function does not provide a method of marking message boundaries.

---

### **Urgent mode**

TCP's urgent mechanism allows the sending user to prompt the receiving user to accept some urgent data and permits the receiving TCP to indicate to the receiving user when all currently known urgent data has been received.

TCP does not define what the user is required to do when in urgent mode, but the general practice is that the receiving user takes action to process the arriving data quickly.

A receiving user can be put in urgent mode in two ways: by means of an Urgent flag in a TCPNoCopyRcv or TCPRcv command, or by an urgent ASR notification. The user is always taken out of urgent mode by a Mark flag in the TCPNoCopyRcv or TCPRcv command that contains the last byte of urgent data.

To send data as urgent, you must set the Urgent flag in the TCPSend command.

---

## Connection closing

TCP closes communications gracefully. All outstanding Send requests are transmitted and acknowledged before the connection is allowed to close. You can issue several TCPSend commands followed by a TCPClose command and expect that all the data will be sent successfully to the remote TCP.

A TCPClose command means "I have no more data to send," but it does not mean "I will receive no more data" or "shut down this connection immediately." A connection may remain open indefinitely after a TCPClose command is completed as the remote TCP continues to send data. When the remote TCP also issues a close command—and only then—the connection is closed. A TCPRecv command should be issued after a TCPClose command is completed to make sure that all the data is received.

If the desired effect is to break the connection without any assurance that all data in transit is delivered, use the TCPAbort command.

---

## Network management information

TCP keeps two types of network management information: global TCP information and stream-specific information, which is relevant only to a particular stream.

The TCPGlobalInfo command makes global TCP information accessible. This command returns pointers to the actual structures where TCP stores this information. Thus, the user has read-write access to this information.

The TCPStatus command makes stream-specific information accessible. Most stream-specific information is copied into the TCPStatus I/O parameter block, giving the user read-only access. But a direct pointer to the traffic statistics allows the user read-write access to those counters (see the section "TCPGlobalInfo" later in this chapter).

---

## Formatting MacTCP commands

In most cases, a 0 value for a parameter in an I/O parameter block means that TCP should use its default value. Thus, you could initialize I/O parameter blocks and then fill in the required parameters, plus optional parameters.



---

## TCP routines

This section presents calls to the TCP driver. Table 4-1 lists each TCP routine and its function. The C parameter-block definitions are presented at the end of the chapter.

■ **Table 4-1** TCP routines

<b>Routine</b>	<b>Function</b>
TCPCreate	Opens a TCP stream
TCPPassiveOpen	Listens for an incoming connection
TCPActiveOpen	Initiates an outgoing call to a remote TCP
TCPSend	Sends specified data on a connection
TCPNoCopyRcv	Retrieves data that has been received on a connection
TCPBfrReturn	Returns a set of receive buffers to the TCP driver
TCPRcv	Retrieves data that has been received on a connection
TCPClose	Signals that the user has no more data to send on this connection
TCPAbort	Terminates the connection without trying to send outstanding data or deliver received data
TCPStatus	Extracts information from TCP regarding a particular connection
TCPRelease	Closes a TCP stream
TCPGlobalInfo	Allows the user access to global statistics and parameters that affect the operation of TCP

---



## TCPCreate

<b>Parameter block</b>	→	26	word	csCode = TCPCreate
	←	28	long	stream pointer
	→	32	long	pointer to receive buffer area
	→	36	long	length of receive buffer area
	→	40	long	pointer to ASR
	→	44	long	user data pointer

TCPCreate opens a TCP stream. A TCP stream is not equivalent to a TCP connection. MacTCP returns a pointer to a stream database. The stream pointer is an input parameter in all subsequent commands affecting the stream.

The receive buffer area is a block of memory that TCP uses to buffer incoming segments. Ownership of this block of memory passes to TCP. The memory—a minimum of 4096 bytes—cannot be modified or relocated until TCPRelease is called. The size of the receive window that TCP offers is based on the size of the receive buffer area passed to TCP in the TCPCreate call. High-performance and block-oriented applications should provide TCP with a large receive buffer area: 16 KB (kilobytes) is recommended and up to 128 KB can be useful in certain applications. Character-oriented applications can use the minimum value of 4096 bytes; however, at least 8192 bytes is recommended.

An ASR may be provided. The ASR is called by TCP to notify you of asynchronous events such as Data arrival, Urgent data outstanding, and Connection terminated. If the routine is 0, you are not notified of asynchronous events.

<b>Result codes</b>	noErr	no error
	streamAlreadyOpen	an open stream is already using this receive buffer area
	invalidLength	the receive buffer area is too small
	invalidBufPtr	the receive buffer area pointer is 0
	insufficientResources	there are already 64 TCP streams open



## TCPPassiveOpen

Parameter block	→	26	word	csCode = TCPPassiveOpen
	→	28	long	stream pointer
	→	32	byte	ULP time-out value in seconds; 0 = use default
	→	33	byte	ULP time-out action; 0 = report, nonzero = abort
	→	34	byte	validity bits for optional parameters
	→	35	byte	command time-out in seconds; 0 = infinity
	↔	36	long	remote IP address; can be 0
	↔	40	word	remote TCP port; can be 0
	←	42	long	local IP address
	↔	46	word	local TCP port; if 0, TCP assigns an unused port
	→	48	byte	type of service
	→	49	byte	precedence
	→	50	byte	don't fragment flag
	→	51	byte	time to live
	→	52	byte	security flag
	→	53	byte	IP option count
	→	54	var.	IP options
	→	94	long	user data pointer

TCPPassiveOpen listens for an incoming connection. The command is completed when a connection is established or when an error occurs.

Several fields in the TCPPassiveOpen command are optional. To indicate whether the user is including these optional parameters, a set of validity bits is defined as follows:

- bit 4    precedence parameter is valid
- bit 5    type of service parameter is valid
- bit 6    ULP time-out action parameter is valid
- bit 7    ULP time-out value parameter is valid

For example, a value of 0xC0 would mean that the ULP time-out value and action parameters are valid. If a validity bit is 0, TCP uses its default value for that parameter.

If the remote IP address and remote TCP port are 0, a connection is accepted from any remote TCP. If they are nonzero, a connection is accepted only from that particular remote TCP. If the local TCP port is 0, TCP assigns an unused port value.

If a connection is partly established but cannot be completed within the ULP time-out period after the first connection opening segment arrives, the ULP action is taken. If the ULP time-out action is abort (the default value), the connection is broken and the TCPPassiveOpen command is completed in error. If the ULP time-out action is report, an ASR call informs the ULP, and the ULP timer is restarted. The minimum value of the ULP time-out is 2 seconds; 0 means that TCP should use its default value of 2 minutes.



If no connection opening packet arrives within the specified command time-out period after the TCPPassiveOpen command is issued, the command is completed in error. The minimum value of the command time-out is 2 seconds; 0 means infinite.

You have control over many fields in the IP header of all segments sent on this connection. These fields can be set only when the connection is opened. They stay fixed during the connection.

The type of service is a 3-bit field:

bit 0     set for low delay  
bit 1     set for high throughput  
bit 2     set for high reliability

For example, a value of 0x02 means high throughput. The default value is 0.

Precedence has the following values:

0     routine  
1     priority  
2     immediate  
3     flash  
4     flash-override  
5     CRITIC/ECP  
6     internetwork control  
7     network control

The default value for precedence is 0.

If the Don't Fragment flag is nonzero, all segments sent on this connection are prohibited from being fragmented by the local IP or any intermediate IP. If a segment cannot be delivered without fragmentation, it is discarded.

The Time to live indicates the maximum time that segments on this connection are allowed to remain in the internet system. This value is decreased by every IP module that processes the segment; thus, it is effectively a maximum hop count (the number of times a segment can pass through a module). The minimum value is 2; 0 means TCP should use its default value (the default value is 60).

If the Security flag is nonzero, TCP inserts its configured default IP security option into all segments sent on this connection. In addition, for all arriving segments that contain a security option, TCP verifies that the security matches the configured default security. Note that this flag is relevant only if no security option is present in the user-specified IP options.

Finally, you can specify additional IP options to be sent with every segment. The option count is the number of long words in the IP option field. Pad bytes of 0 should be appended to the IP options so that the options are an integral number of long words. The maximum value of the option count field is 10 unless the Security flag is also nonzero, in which case the maximum value is 9.



TCP does not perform any verification on the user-specified IP options, but simply inserts them into the IP header of every segment sent on the connection. If you specify an invalid list of IP options, the result cannot be predicted. See Request for Comment (RFC) 894 for the proper format of IP options.

<b>Result codes</b>		
noErr		no error
invalidStreamPtr		the specified TCP stream is not open
connectionExists		this TCP stream already has an open connection
duplicateSocket		a connection already exists between this local IP address and TCP port, and the specified remote IP address and TCP port
commandTimeout		no connection attempt was received in the specified time-out period
openFailed		the connection came halfway up and then failed



## TCPActiveOpen

<b>Parameter block</b>	→	26	word	csCode = TCPActiveOpen
	→	28	long	stream pointer
	→	32	byte	ULP time-out value in seconds; 0 = use default
	→	33	byte	ULP time-out action; 0 = report, 1 = abort
	→	34	byte	validity bits
	→	36	long	remote IP address; cannot be 0
	→	40	word	remote TCP port; cannot be 0
	←	42	long	local IP address
	↔	46	word	local TCP port; if 0, TCP assigns an unused port
	→	48	byte	type of service
	→	49	byte	precedence
	→	50	byte	don't fragment flag
	→	51	byte	time to live
	→	52	byte	security flag
	→	53	byte	IP option count
	→	54	var.	IP options
	→	94	long	user data pointer

TCPActiveOpen initiates an outgoing call to a remote TCP. The command is completed when a connection is established or when an error occurs.

TCPActiveOpen accepts the same parameters as TCPPassiveOpen, except that the remote IP address and remote TCP port must be specified. In addition, no command time-out is provided; if the connection cannot be established within the ULP time-out period, the command is completed in error.

See TCPPassiveOpen for a description of other parameters.

<b>Result codes</b>	noErr	no error
	invalidStreamPtr	the specified TCP stream is not open
	connectionExists	this TCP stream already has an open connection
	duplicateSocket	a connection already exists between this local IP address and TCP port, and the specified remote IP address and TCP port
	openFailed	the connection came halfway up and then failed



## TCPSend

<b>Parameter block</b>	→	26	word	csCode = TCPSend
	→	28	long	stream pointer
	→	32	byte	ULP time-out value in seconds
	→	33	byte	ULP time-out action; 0 = abort, nonzero = report
	→	34	byte	validity bits
	→	35	byte	push flag
	→	36	byte	urgent flag
	→	38	long	WDS pointer
	→	42	long	reserved
	→	46	word	reserved
	→	48	long	user data pointer

TCPSend sends the specified data over the connection. The command is completed when all data has been sent and acknowledged, or when an error occurs.

If all data cannot be sent and acknowledged within the ULP time-out period, then the ULP action is taken. If the ULP action is abort, the connection is broken, all pending commands are returned, and a terminate notification is given. If the ULP action is report, a ULP time-out notification is given. See TCPPassiveOpen for a description of the validity bits.

If the Push flag is nonzero, TCP sends the data immediately without waiting to see if more TCPSend commands are issued. If the Urgent flag is nonzero, TCP sends the data in urgent mode. In this case, all data described by the WDS is assumed to be urgent data.

The WDS can be arbitrarily complex; that is, there is no limit to the number of buffers that can be sent in a TCPSend command. However, the total number of data bytes described by the WDS must be between 1 and 65,535, inclusive. You must not modify or relocate the WDS and the buffers it describes until the TCPSend command has been completed.

<b>Result codes</b>	noErr	no error
	invalidStreamPtr	the specified TCP stream is not open
	invalidLength	the total amount of data described by the WDS was either 0 or greater than 65,535 bytes
	invalidWDS	the WDS pointer was 0
	connectionDoesntExist	there is no open connection on this TCP stream
	connectionClosing	a TCPClose command was already issued, so there is no more data to send on this connection
	connectionTerminated	the connection went down; the reason will be given in a terminate ASR



## TCPNoCopyRcv

Parameter block	→	26	word	csCode = TCPNoCopyRcv
	→	28	long	stream pointer
	→	32	byte	command time-out in seconds
	←	34	byte	urgent flag
	←	35	byte	mark flag
	→	42	long	RDS pointer; RDS is modified by TCP
	→	46	word	number of entries in RDS excluding terminating 0; not modified by TCP
	→	48	word	reserved
	→	50	long	user data pointer

TCPNoCopyRcv retrieves data that has been received on a connection. Data is not copied out of TCP's internal buffers; rather, an RDS is formatted to allow the user access to the TCP data in place. This command offers significant performance improvements over TCPRecv commands.

The command is completed when

- pushed data arrives
- urgent data is outstanding
- some reasonable period passes after the arrival of nonpush, nonurgent data
- the RDS is full; that is, the received data is in more noncontiguous chunks than the RDS can describe
- the amount of data received is greater than or equal to 25 percent of the total receive buffering for this stream
- the command time-out expires

You must allocate memory for the RDS, which can contain an unlimited number of elements. The specified number of entries in the RDS does not include the terminating 0. For example, the simplest possible RDS has one element and is 8 bytes in length: a word length field, a long pointer field, and a terminating word length field of 0.

TCP formats the RDS to point to the received TCP data. The RDS may not be completely filled by TCP. For example, if you pass an RDS three entries, TCP may complete the TCPNoCopyRcv command with the RDS describing only one buffer. TCP does not modify the field in the TCPNoCopyRcv command that indicates the number of entries in the RDS; rather, it places a terminating zero in the RDS itself.



The command time-out period starts when the receive command is issued, not when the first byte of data arrives. If no data arrives within this time-out period, the TCPNoCopyRcv command is completed in error. If some data has arrived when the command time-out expires, the TCPNoCopyRcv command is completed successfully, returning the data that has arrived so far. A command time-out of 0 indicates an infinite time-out period. In this case, the TCPNoCopyRcv command is not completed until some data has arrived.

If the Urgent flag is nonzero, the data returned by this command is the beginning of the outstanding urgent data. This flag is one of two mechanisms that puts you in urgent mode. The other mechanism is by means of an urgent ASR notification, which is used when there are no outstanding TCPNoCopyRcv or TCPRcv commands.

If the Mark flag is nonzero, the data returned by this command ends the urgent data. Since TCP does not deliver urgent and nonurgent data together, the last byte of data described by the RDS is the last byte of urgent data. The Mark flag is the only mechanism for taking you out of urgent mode.

The Urgent flag is set only on the first TCPNoCopy Rcv or TCPRcv command that contains urgent data. The Mark flag will be set only on the last TCPNoCopyRcv or TCPRcv command that contains urgent data. For example, if three TCPNoCopy Rcv or TCPRcv commands are required to deliver all urgent data, the settings of the Urgent and Mark flags will be as follows:

	<b>Urgent flag</b>	<b>Mark flag</b>
First TCPNoCopyRcv/TCPRcv command	1	0
Second TCPNoCopyRcv/TCPRcv command	0	0
Third TCPNoCopyRcv/TCPRcv command	0	1

Both the Urgent flag and the Mark flag can be set in a single TCPNoCopyRcv or TCPRcv command if all outstanding urgent data can be delivered in a single command.

You are responsible for calling TCPBfrReturn after every TCPNoCopyRcv command that is completed successfully, in order to return the receive buffers owned by the TCP driver. The RDS must be returned unmodified so that the TCP driver can correctly recover the appropriate receive buffers.

**Result codes**

noErr	no error
invalidStreamPtr	the specified TCP stream is not open
connectionDoesntExist	this TCP stream has no open connection
invalidLength	the RDS has 0 entries
invalidBfrPtr	the RDS pointer was 0
commandTimeout	no data arrived within the specified time-out period
connectionClosing	all data on this connection has already been delivered
connectionTerminated	the connection went down; the reason will be given in a terminate ASR



---

## TCPBfrReturn

<b>Parameter block</b>	→	26	word	csCode = TCPBfrReturn
	→	28	long	stream pointer
	→	42	long	RDS pointer
	→	50	long	user data pointer

TCPBfrReturn returns to the TCP driver a set of receive buffers that a successfully completed TCPNoCopyRcv command passed directly to the user. The RDS must be identical to the RDS given to the user when the TCPNoCopyRcv command is completed. TCPBfrReturn returns an error if you attempt to return a set of buffers more than once.

<b>Result codes</b>	noErr	no error
	invalidStreamPtr	the specified TCP stream is not open
	connectionDoesntExist	this TCP stream has no open connection
	invalidBufPtr	the RDS pointer was 0
	invalidRDS	the RDS refers to receive buffers not owned by the user



## TCPRcv

Parameter block	→	26	word	csCode = TCPRcv
	→	28	long	stream pointer
	→	32	byte	command time-out in seconds
	←	34	byte	urgent flag
	←	35	byte	mark flag
	→	36	long	receive buffer pointer
	↔	40	word	receive buffer length
	→	48	word	reserved
	→	50	long	user data pointer

TCPRcv retrieves data that has been received on a connection. Data is copied out of the TCP internal buffers into the user's buffer. The command is completed when

- enough data has arrived to fill the receive buffer
- pushed data arrives
- urgent data is outstanding
- some reasonable period passes after the arrival of nonpushed, nonurgent data
- the amount of data received is greater than or equal to 25 percent of the total receive buffering for this stream
- the command time-out expires

The command time-out period starts when the receive command is issued, not when the first byte of data arrives. If no data arrives within this command time-out period, the TCPRcv command is completed in error. If some data has arrived when the command time-out expires, the TCPRcv command is completed successfully. A command time-out of 0 means infinite; the TCPRcv command will not be completed until some data has arrived.

If the Urgent flag is nonzero, the data returned by this command is the beginning of the outstanding urgent data. This flag is one of two mechanisms that puts you in urgent mode. The other mechanism is by means of an urgent ASR notification, which is used when there are no outstanding TCPNoCopyRcv or TCPRcv commands.

If the Mark flag is nonzero, the data returned by this command ends the urgent data. Since TCP does not deliver urgent and nonurgent data together, the last byte of data described by the RDS is the last byte of urgent data. The Mark flag is the only mechanism for taking you out of urgent mode.



The Urgent flag is set only on the first TCPNoCopyRcv or TCPRcv command that contains urgent data. The Mark flag is set only on the last TCPNoCopyRcv or TCPRcv command that contains urgent data. For example, if three TCPNoCopy Rcv or TCPRcv commands are required to deliver all urgent data, the settings of the urgent and Mark flags will be as follows:

	Urgent flag	Mark flag
First TCPNoCopyRcv/TCPRcv command	1	0
Second TCPNoCopyRcv/TCPRcv command	0	0
Third TCPNoCopyRcv/TCPRcv command	0	1

Both the Urgent flag and the Mark flag can be set in a single TCPNoCopyRcv or TCPRcv command if all outstanding urgent data can be delivered in a single command.

If the TCPRcv command is completed in error, the receive buffer length is not modified by TCP, but no data is returned.

**Result codes**

noErr	no error
invalidStreamPtr	the specified TCP stream is not open
invalidLength	the receive buffer length was 0
invalidBufPtr	the receive buffer pointer was 0
commandTimeout	no data arrived within the specified time-out period
connectionDoesntExist	this TCP stream has no open connection
connectionClosing	all data on this connection has already been delivered
connectionTerminated	the connection went down; the reason will be given in a terminate ASR



## TCPClose

<b>Parameter block</b>	→	26	word	csCode = TCPClose
	→	28	long	stream pointer
	→	32	byte	ULP time-out value in seconds
	→	33	byte	ULP time-out action; 0 = abort, nonzero = report
	→	34	byte	validity bits
	→	35	long	user data pointer

TCPClose signals that the user has no more data to send on the connection. It does not mean that the connection should be broken. The remote TCP must also issue a close before the connection can be gracefully closed, so a connection may stay up an arbitrary amount of time after you have issued a TCPClose. To break a connection without ensuring that all data has been sent and acknowledged, use the TCPAbort command.

The command is completed when the FIN flag has been sent and acknowledged. If the FIN is not acknowledged within the ULP time-out period, the ULP time-out action is taken. If the ULP action is abort, the connection is broken, all pending commands are returned, and a terminate notification is given. If the ULP action is report, a ULP time-out notification is given. See TCPPassiveOpen for a description of the validity bits.

<b>Result codes</b>	noErr	no error
	invalidStreamPtr	the specified TCP stream is not open
	connectionDoesntExist	this TCP stream has no open connection
	connectionClosing	one TCPClose command was already issued for this connection
	connectionTerminated	the connection went down; the reason will be given in a terminate ASR



---

## TCPAbort

<b>Parameter block</b>	→	26	word	csCode = TCPAbort
	→	28	long	stream pointer
	→	32	long	user data pointer

TCPAbort terminates the connection without attempting to send all outstanding data or to deliver all received data. TCPAbort returns the TCP stream to its initial state. You are also given a terminate notification.

<b>Result codes</b>	noErr	no error
	invalidStreamPtr	the specified TCP stream is not open
	connectionDoesntExist	this TCP stream has no open connection



## TCPStatus

Parameter block	→	26	word	csCode = TCPStatus
	→	28	long	stream pointer
	←	32	byte	ULP time-out value in seconds
	←	33	byte	ULP time-out action; 0 = abort, nonzero = report
	←	38	long	remote IP address
	←	42	word	remote TCP port
	←	44	long	local IP address
	←	48	word	local TCP port
	←	50	byte	type of service
	←	51	byte	precedence
	←	52	byte	TCP connection state
	←	54	word	send window
	←	56	word	receive window
	←	58	word	amount of unacknowledged data
	←	60	word	amount of unread data
	←	62	long	security option pointer
	←	66	long	send unacknowledged
	←	70	long	send next
	←	74	long	congestion window
	←	78	long	receive next
	←	82	long	smoothed round-trip time in milliseconds
	←	86	long	last round-trip time in milliseconds
	←	90	long	maximum segment size that can be sent
	←	94	long	pointer to statistics block
	→	98	long	user data pointer

TCPStatus allows you to extract information from TCP regarding a particular connection.

See the TCPPassiveOpen command for a description of the usual open parameters (bytes 32-51).

The TCP connection state has the following values:

0	Closed	no connection exists on this stream
2	Listen	listening for an incoming connection
4	SYN received	incoming connection is being established
6	SYN sent	outgoing connection is being established
8	Established	connection is up
10	FIN Wait 1	connection is up; close has been issued
12	FIN Wait 2	connection is up; close has been completed



14	Close Wait	connection is up; close has been received
16	Closing	connection is up; close has been issued and received
18	Last Ack	connection is up; close has been issued and received
20	Time Wait	connection is going down

The send window is the amount of data the remote TCP is currently willing to accept from the local TCP. The receive window is the amount of data the local TCP is currently willing to accept from the remote TCP.

The statistics block has the following structure:

0	long	number of data segments received
4	long	number of data segments sent
8	long	number of data segments retransmitted
12	long	number of data bytes received
16	long	number of duplicate data bytes received
20	long	number of data bytes received beyond receive window
24	long	number of data bytes sent
28	long	number of data bytes retransmitted
32	struct	size histogram of sent segments

The histogram is a variable-length structure of the following format:

0	word	number of size buckets
2	word	value: smallest segment size
4	long	counter number: number of segments sent between this size and the next larger size
8	word	value: second smallest segment size
10	long	counter number: number of segments sent between this size and the next larger size

You are free to update the statistics block. Only the counters in the segment-size histogram should be modified. The number of size buckets and the segment size value for each bucket cannot be changed.

**Result codes**

noErr	no error
invalidStreamPtr	the specified TCP stream is not open
connectionDoesntExist	this TCP stream has no open connection



---

## TCPRelease

<b>Parameter block</b>	→	26	word	csCode = TCPRelease
	→	28	long	stream pointer
	←	32	long	pointer to receive buffer area
	←	36	long	length of receive buffer area

TCPRelease closes a TCP stream. If there is an open connection on the stream, the connection is first broken as though a TCPAbort command had been issued.

The receive buffer area passed to MacTCP in the TCPCreate call is returned to the user. You are now free to reuse or release this buffer area.

<b>Result codes</b>	noErr	no error
	invalidStreamPtr	the specified TCP stream is not open



## TCPGlobalInfo

Parameter block	→	26	word	csCode = TCPGlobalInfo
	←	32	long	pointer to TCP parameters
	←	36	long	pointer to TCP statistics
	←	40	long	pointer to CDB table
	→	44	long	user data pointer

The TCPGlobalInfo command allows you access to

- parameters that affect the operation of TCP
- global statistics collected by TCP

You should not modify the TCP parameters. The structure has the following format:

0	long	pointer to Pascal string describing retransmission time-out algorithm
4	long	minimum value of retransmission time-out (ms)
8	long	maximum value of retransmission time-out (ms)
12	long	maximum segment size this TCP can receive
16	long	maximum number of streams MacTCP can support
20	long	maximum value of receive window MacTCP can offer

The TCP statistics are the sum of traffic information for all streams that have been opened since system startup. The counters will wrap around to 0 when incremented past their maximum value. You can modify these statistics. The TCP statistics structure has the following format:

0	long	number of outgoing connections attempted
4	long	number of outgoing connections opened
8	long	number of incoming connections accepted
12	long	number of connections that closed gracefully
16	long	number of connections that were aborted
20	long	number of data bytes received
24	long	number of data bytes sent
28	long	number of duplicate data bytes received
32	long	number of data bytes retransmitted
36	long	total number of segments received (includes acknowledgments)
40	long	total number of segments sent (includes acknowledgments)
44	long	number of segments received that contained all duplicate data



	48	long	number of segments retransmitted
<b>Result codes</b>	noErr		no error



## TCP asynchronous notification routine

The TCP asynchronous notification routine (ASR) is a user-supplied routine called by TCP to notify you of asynchronous events relevant to a particular TCP stream. You register this routine with TCP in the `TCPCreate` call.

Since this routine is called at interrupt level, it cannot release or allocate memory. In addition, an ASR routine can issue additional asynchronous MacTCP driver calls.

The C description of the ASR is as follows:

```
pascal void TCPNotifyProc (StreamPtr tcpStream, unsigned short
eventCode, Ptr userDataPtr, unsigned short terminReason,
struct ICMPReport *icmpMsg);
```

Register A1 contains a pointer to the Internet Control Message Protocol (ICMP) report structure if the event code in D0 is ICMP received, A2 contains the user data pointer, A5 is already set up to point to application globals, D0 (word) contains an event code, and D1 contains a reason for termination.

<b>Event codes</b>	closing	all data on this connection has been received and delivered
	ULP time-out	no response from the remote TCP; reported only if TCP is configured to report a time-out instead of aborting the connection
	terminate	connection no longer exists
	data arrival	data arrived, no receive commands outstanding
	urgent data outstanding	user should go into urgent mode
	ICMP message received	an Internet Control Message has been received on the stream; register A1 points to the ICMP report

For the terminate event, a reason for connection termination is given in D1 (word):

remote abort	the remote TCP aborted the connection
network failure	to be supplied in a later draft of this guide
security/precedence mismatch	invalid security option or precedence level
ULP time-out	the ULP time-out expired; ULP time-out action is abort
ULP abort	the user issued a <code>TCPAbort</code> command
ULP close	the connection closed gracefully
service failure	unexpected connection initiation segment read

◆ *Note:* Refer to the Appendix for the values of these types.

A closing notification means that the remote TCP has sent all the data it intends to send on this connection and that all data has been delivered to the user. Closing notification does not mean that the connection has been broken. You can continue to send data for an arbitrary length of time after a closing notification is given. Only when you issue a `TCPClose` command will the connection be terminated.



The ULP time-out notification is given if the configured ULP time-out action for this connection is report, and the local TCP cannot get some data sent or acknowledged (or both), within the specified ULP time-out period. The ULP timer is restarted after notification and expires again in another ULP time-out period if nothing changes.

For each connection, TCP issues exactly one terminate notification when the connection is broken. This rule applies both when the connection closes gracefully and when the connection terminates in error.

Data arrival notification is given if a segment arrives and no receive commands are outstanding. Even if more data arrives later, TCP does not issue another data arrival notification until a receive command has been issued and completed. In other words, a data arrival notification is not given with every segment that arrives, but instead is intended to prompt you to issue a receive command.

Urgent notification is given only if no receive commands are outstanding when TCP learns that there is outstanding urgent information on the connection. If there are outstanding receive commands, they are completed with the Urgent flag set to get you into urgent mode.

An ICMP message reports an error in the processing of a datagram that was sent on a TCP stream. When an ICMP message is received, a data structure is passed up by TCP to the client to describe the received message. This data structure, called an ICMP report, has the following format:

0	word	stream pointer
2	long	local IP address of stream
6	word	local TCP port of stream
8	long	remote IP address (destination of original datagram)
12	word	remote TCP port
14	word	ICMP message type
16	word	optional additional information
18	long	optional additional information pointer

The values for the ICMP message type are as follows:

0	net unreachable
1	host unreachable
2	protocol unreachable
3	port unreachable
4	fragmentation required
5	source route failed
6	time exceeded
7	parameter problem
8	missing required option



Codes 0-3 are defined as follows:

- Net unreachable indicates that, according to the information in a gateway routing table, the network specified in the IP destination field of a TCP segment is unreachable.
- Host unreachable indicates that a gateway determined that the host specified in the IP destination field of a TCP segment is unreachable.
- Protocol unreachable indicates that a TCP segment was delivered to the destination host, but there was no process on that host that would receive TCP segments.
- Port unreachable indicates that a TCP segment was delivered to the destination host, but there was no client of TCP listening on that particular TCP port.

If the TCP stream is configured for the ULP abort time-out action, the TCP client need not take any action in response to destination unreachable messages (they are informational only).

TCP breaks the connection if there is data to send, but it is not acknowledged within the ULP time-out period. If, however, this TCP stream has been configured for the ULP report time-out action, then the TCP user has taken responsibility for deciding when and if the remote host is no longer available, and the connection should be broken.

A single destination unreachable message should not be taken too seriously since such messages may be received occasionally when the topology of the internet changes. But if several successive TCPSend commands each result in an ICMP report indicating destination unreachable, the TCP client should assume that the remote host has either crashed or is no longer accessible, and should break the connection.

Codes 4-8 are explained in the following paragraphs.

Fragmentation required indicates that the TCP user has set the Don't Fragment flag in a TCPOpen command, yet a segment on that TCP connection could not be delivered to its destination without fragmentation. To avoid this, don't set the Don't Fragment flag on TCPOpen commands.

Source route failed indicates that the TCP user has specified the route this datagram should take in the IP options but that particular route was not available.

Time exceeded indicates that the Time to Live specified in a TCPOpen command was too short to allow a TCP segment on this TCP stream to be delivered through all the necessary gateways on the way to its destination. A longer Time to Live value should be specified in the TCPOpen command.

Parameter problem indicates that the IP header used on a TCP segment was not acceptable by either an intermediate gateway or the final destination. The additional information pointer in the ICMP report structure points to a static copy of the IP header used for sending segments on this TCP connection. The additional information value



indicates a byte offset in the IP header where the parameter problem exists. For example, an offset of 1 indicates that the Type of Service field was invalid, and an offset of 20 indicates that the first option present is invalid.

Missing required option means that the remote TCP requires a specific IP option to be present in the IP header. The IP option type code that is required by the remote TCP is passed in the additional information field.

Much of the information that asynchronous notifications pass to you is also available in other ways. For example, you can tell that the remote TCP has closed a connection either by waiting for a closing notification or by submitting TCPRcv commands until one is returned with a connectionClosing error.





```
typedef unsigned char byte;

enum ( /* ValidityFlags */
    timeoutValue = 0x80,
    timeoutAction = 0x40,
    typeOfService = 0x20,
    precedence = 0x10
);

enum ( /* TOSFlags */
    lowDelay = 0x01,
    throughPut = 0x02,
    reliability = 0x04
);

typedef struct TCPCreatePB {
    Ptr rcvBuff;
    unsigned long rcvBuffLen;
    TCPNotifyProc notifyProc;
    Ptr userDataPtr;
}TCPCreatePB;
```

The following parameter block is used with the TCPPassiveOpen and TCPActiveOpen commands:

```
typedef struct TCPOpenPB {
    byte ulpTimeoutValue;
    byte ulpTimeoutAction;
    byte validityFlags;
    byte commandTimeoutValue;
    ip_addr remoteHost;
    tcp_port remotePort;
    ip_addr localHost;
    tcp_port localPort;
    byte tosFlags;
    byte precedence;
    Boolean dontFrag;
    byte timeToLive;
    byte security;
    byte optionCnt;
    byte options[40];
    Ptr userDataPtr;
}TCPOpenPB;

typedef struct TCPSendPB {
    byte ulpTimeoutValue;
    byte ulpTimeoutAction;
    byte validityFlags;
    Boolean pushFlag;
    Boolean urgentFlag;
    Ptr wdsPtr;
    unsigned long sendFree;
    unsigned short sendLength;
    Ptr userDataPtr;
}TCPSendPB;
```



The following parameter block is used with the TCPrcv, TCPrcvBfrReturn, and TCPNoCopyRcv commands:

```
typedef struct TCPReceivePB
    byte commandTimeoutValue;
    byte filler;
    Boolean markFlag;
    Boolean urgentFlag;
    Ptr rcvBuff;
    unsigned short rcvBuffLen;
    Ptr rdsPtr;
    unsigned short rdsLength;
    unsigned short secondTimeStamp;
    Ptr userDataPtr;
}TCPReceivePB;

typedef struct TCPClosePB {
    byte ulpTimeoutValue;
    byte ulpTimeoutAction;
    byte validityFlags;
    Ptr userDataPtr;
}TCPClosePB;

typedef struct HistoBucket {
    unsigned short value;
    unsigned long counter;
};

#define NumOfHistoBuckets 7

typedef struct TCPConnectionStats {
    unsigned long dataPktsRcvd;
    unsigned long dataPktsSent;
    unsigned long dataPktsResent;
    unsigned long bytesRcvd;
    unsigned long bytesRcvdDup;
    unsigned long bytesRcvdPastWindow;
    unsigned long bytesSent;
    unsigned long bytesResent;
    unsigned short numHistoBuckets;
    struct HistoBucket sentSizeHisto[NumOfHistoBuckets];
    unsigned short lastRTT;
    unsigned short tmrSRTT;
    unsigned short rttVariance;
    unsigned short tmrRTO;
    byte sendTries;
    byte souchQuenchRcvd;
}TCPConnectionStats;
```



```
typedef struct TCPStatusPB {
    byte ulpTimeoutValue;
    byte ulpTimeoutAction;
    long unused;
    ip_addr remoteHost;
    tcp_port remotePort;
    ip_addr localHost;
    tcp_port localPort;
    byte tosFlags;
    byte precedence;
    byte connectionState;
    unsigned short sendWindow;
    unsigned short rcvWindow;
    unsigned short amtUnackedData;
    unsigned short amtUnreadData;
    Ptr securityLevelPtr;
    /* HEMS-HEMP stats */
    unsigned long sendUnacked;
    unsigned long sendNext;
    unsigned long congestionWindow;
    unsigned long rcvNext;
    unsigned long srtt;
    unsigned long lastRTT;
    unsigned long sendMaxSegSize;
    struct TCPConnectionStats *connStatPtr;
    Ptr userDataPtr;
}TCPStatusPB;

typedef struct TCPAbortPB {
    Ptr userDataPtr;
}TCPAbortPB;

typedef struct TCPParam {
    unsigned long tcpRtoA;
    unsigned long tcpRtoMin;
    unsigned long tcpRtoMax;
    unsigned long tcpMaxSegSize;
    unsigned long tcpMaxConn;
    unsigned long tcpMaxWindow;
}TCPParam;
```



```
typedef struct TCPStats {
    unsigned long tcpConnAttempts;
    unsigned long tcpConnOpened;
    unsigned long tcpConnAccepted;
    unsigned long tcpConnClosed;
    unsigned long tcpConnAborted;
    unsigned long tcpOctetsIn;
    unsigned long tcpOctetsOut;
    unsigned long tcpOctetsInDup;
    unsigned long tcpOctetsRetrans;
    unsigned long tcpInputPkts;
    unsigned long tcpOutputPkts;
    unsigned long tcpDupPkts;
    unsigned long tcpRetransPkts;
}TCPStats;

typedef struct TCPGlobalInfoPB {
    struct TCPParam *tcpParamPtr;
    struct TCPStats *tcpStatsPtr;
    unsigned long *tcpCDBTable;
    Ptr userDataPtr;
}TCPGlobalInfoPB;

typedef struct TCPIopb {
    char                fill12[12];
    TCPIOCompletionProc ioCompletion;
    short               ioResult;
    char                *ioNamePtr;
    short               ioVRefNum;
    short               ioCRefNum;
    short               csCode;
    StreamPtr           tcpStream;
    union {
        struct TCPCreatePB    create;
        struct TCPOpenPB      open;
        struct TCPSendPB      send;
        struct TCPReceivePB   receive;
        struct TCPClosePB     close;
        struct TCPAbortPB     abort;
        struct TCPStatusPB    status;
        struct TCPGlobalInfoPB globalInfo;
    } csParam;
}TCPIopb;
```



## Chapter 5 Name-to-Address Resolution

TEXTUAL NAMES are resolved to IP addresses using internal caches and the domain name server. The AddressXlation interface accomplishes this task by searching an internal table originally derived from the static file Hosts, queries made to domain name servers, and information found in the internal cache of domain name server responses. ■



---

## The AddressXlation.h header file

The AddressXlation.h interface supports the domain name resolver (DNR) function using the procedure calls described in the following sections.

---

### The OpenResolver call

```
extern OSErr OpenResolver(char *fileName);
```

The OpenResolver call must be made before any resolver queries are attempted. The full path name of the default Hosts file must be passed in the fileName field. If the value NIL is passed, the resolver uses the filename Hosts in the default start up folder, which is typically the System Folder. The Hosts file is used to load the internal cache with name-to-address mappings and domain name server values. The syntax of this file is outlined on page 33 of RFC1035.

#### Result codes

Error result codes can be returned by the Resource Manager

---

### The StrToAddr call

```
#define NUM_ALT_ADDRS 4

typedef struct hostInfo {
    int rtnCode;
    char cname[255];
    unsigned long addr[NUM_ALT_ADDRS];
};

typedef pascal void (*ResultProcPtr)(
    struct hostInfo *hostInfoPtr,
    char *userDataPtr);

extern OSErr StrToAddr(
    char *hostName,
    struct hostInfo *hostInfoPtr,
    ResultProcPtr ResultProc,
    char *userDataPtr);
```

The hostInfo record is passed into the address translation routine in the StrToAddr call. The rtnCode indicates whether the result fields are valid for this call; a value of noErr indicates that the call has been successful. The cname is the official name of the hostName that was specified. The addr array is a list of addresses for the hostName specified. Multiple addresses are returned for hosts that are multihomed.



StrToAddr takes a string in one of two forms and translates it into a 32-bit IP address. The string can be in IP dot notation (that is, W.X.Y.Z) or in a valid domain name syntax. The translated address is returned immediately if the passed in host is in IP dot notation or if the matching address is contained in the local cache. If the address is not contained in the local cache, the rtnCode is cacheFault and the domain name server is contacted to resolve the address. When the response has been returned from the domain name server or the domain name query has completed unsuccessfully, the ResultProc is called with the appropriate rtnCode and return information. The StrToAddr procedure can be called with a userDataPtr, which is user-defined and not modified by the resolver. The userDataPtr is returned when the ResultProc is called. Domain names that contain no domain name delimiters, that is, no dots ("."), are terminated with the domain name suffix specified for the default domain name server in the Control Panel or a dot if no default is specified.

**Result codes**

nameSyntaxErr	the hostName field had a syntax error
cacheFault	the name specified cannot be found in the cache
noResultProc	no result procedure is passed to the address translation call when the resolver must be used to find the address
noNameServer	no name server can be found for the specified name string
authNameErr	this domain name does not exist
noAnsErr	none of the known name servers are responding
dnrErr	the domain name server has returned an error
outOfMemory	not enough memory is available to issue the needed DNR query or to build the DNR cache
notOpenErr	the driver isn't open

---

**The AddrToStr call**

```
extern OSErr AddrToStr(unsigned long addr, char *addrStr);
```

The AddrToStr call takes an IP address and returns a string with the ASCII equivalent of the form W.X.Y.Z where W, X, Y, and Z are decimal numbers. The application must provide the storage for the return string. The string can have a maximum length of 16 bytes.

**Result codes**

notOpenErr	the driver isn't open
------------	-----------------------



---

## The EnumCache call

```
typedef enum AddrClasses {
    A = 1,
    NS,
    CNAME = 5,
    lastClass = 65535
};

typedef struct cacheEntryRecord {
    char *cname;
    unsigned short type;
    enum AddrClasses class;
    unsigned long ttl;
    union {
        char *name;
        ip_addr *addr;
    } rdata;
};

typedef pascal void (*EnumResultProcPtr)(
    struct cacheEntryRecord, *cacheEntryRecordPtr,
    char *userDataPtr);

extern OSErr EnumCache(
    EnumResultProcPtr enumResultProc,
    char *userDataPtr);
```

If the application wants to enumerate all the entries in the cache, EnumCache should be called. This procedure returns no errors and has completed enumerating the cache when it returns. For every entry in the cache, enumResultProc is called with a pointer to a cache entry. The fields and values returned in the CacheEntryRecord are as follows:

name	the name of the entry.
type	the type of the entry, where <ul style="list-style-type: none"><li><input type="checkbox"/> A (value = 1) is an address. The value of rdata is an addr.</li><li><input type="checkbox"/> NS (value = 2) is a name server. The value of rdata is a name.</li><li><input type="checkbox"/> CNAME (value = 5) is an alias for the canonical name (cname) found in the rdata field.</li></ul>
class	the class of the entry. The only class allowed is IN (value 1).
ttl	the time that the entry has to live in the cache relative to the current time (GetCurrent).
rdata	the rdata field as determined by the type of entry specified in the type field.

### Result codes

notOpenErr	the driver isn't open
------------	-----------------------



---

## The CloseResolver call

```
extern OSErr CloseResolver();
```

Before the application exits, the CloseResolver call must be made to release memory structures and terminate all outstanding domain name server calls. CloseResolver should not be called until all outstanding resolver calls have been completed.

### Result codes

notOpenErr	the driver isn't open
------------	-----------------------

---

## The AddrToName call

```
extern OSErr AddrToName(ip_addr addr, struct hostInfo  
*hostInfoPtr, ResultProcPtr ResultProc, char *userDataPtr);
```

The AddrToName call is used to acquire the canonical name of a host given its IP address. The domain name server is queried using an IN-ADDR query. The application passes to the AddrToName call the IP address of the host in question, a pointer to a hostInfo record, the result procedure to be notified with the result, and an optional user data pointer. The AddrToName call always returns cacheFault. It may return noNameServer if no name server can be found in the internal name server lists that can resolve the DNR query. The passed-in result procedure is called with the appropriate result code. If the result code is noErr, the cname field will contain the canonical name for the IP address passed to the AddrToName call.

### Result codes

cacheFault	the name specified cannot be found in the cache
noNameServer	no name server can be found for the specified name string
authNameErr	this domain name does not exist
noAnsErr	none of the known name servers are responding
dnrErr	the domain name server has returned an error
outOfMemory	not enough memory is available to issue the needed DNR query or to build the DNR cache
notOpenErr	the driver isn't open



---

## Binding the DNR to the application

The DNR in MacTCP is implemented as a code resource in the MacTCP driver file.

A file called DNR.c in the Libraries Folder of the MacTCP release disk provides a working example of how to open the DNR. This file should be compiled and linked to your application. Procedural access to the DNR is provided by a file produced by MPW 2.0.2 called DNR.o in the Libraries Folder of the MacTCP release disk. Sources have also been provided if you need to port the functionality to another development system.

To use the DNR, the application must first find the procedure pointers that are part of the DNR resource. The DNR resource, named *dnrp*, is attached to the driver file, which is in the System Folder of the startup disk. The resource can be opened using the Resource Manager routines in the Toolbox. Once the resource is opened, the first long word of the resource is a pointer to a procedure that jumps to the correct DNR procedure. Calls to this procedure should be made using the procedure index value as the first argument of the call, followed by the arguments for the procedure as they are specified in the AddressXlation.h interface. The procedure assignments are as follows:

OpenResolver	1
CloseResolver	2
StrToAddr	3
AddrToStr	4
EnumCache	5
AddrToName	6

---

## DNR operation

This section describes how the domain name server (DNS) list is used when making DNS queries.

If a default name extension and server are identified in the Control Panel, they are used for all nonqualified requests. For instance, if the name `homer` is passed to the DNR and the default extension is `pundit.edu`, the name `homer.pundit.edu` is used in the query; however, if the name `homer.drama.pundit.edu` is passed to the DNR, the extension is not appended.



The extension of the name passed to the DNR determines which name servers are chosen. Servers that match the full extension are found first, followed by servers that serve the ancestor of the full extension (for example, for the name `homer.drama.pundit.edu`, the server that serves `drama.pundit.edu` would be found first followed by the server that serves `pundit.edu`). If no servers are found, the default server is used. If no default is set, the DNR returns `noNameServer`. In the MacTCP Control Panel, you should enter a default domain and select the Default button.

Once a list of servers that support the domain is found, those servers are queried in the order of their distance from the querying host. First servers on the local network are queried, followed by servers on other networks. When you use the `AddrToName` query, you must select a default server.





## Chapter 6 **Miscellaneous Interfaces**

THIS CHAPTER describes types that are found throughout the programmatic interfaces supplied with MacTCP. ■



---

## MacTCPCommonTypes

This file defines result code name-to-number mapping, Internet Control Message Protocol (ICMP) message report structures, and other miscellaneous types throughout MacTCP.

---

### Result codes

The result codes in MacTCPCommonTypes are described as follows. Chapters 3, 4, and 5 contain specific occurrences of the result codes described in this section.

<code>#define inProgress</code>	1	When an IOPB is still pending, ioResult is set to inProgress.
<code>#define ipBadLapErr</code>	-23000	Unable to initialize the local network handler.
<code>#define ipBadCnfgErr</code>	-23001	The manually set address is configured improperly.
<code>#define ipNoCnfgErr</code>	-23002	A configuration resource is missing.
<code>#define ipLoadErr</code>	-23003	Not enough room in the application heap (Macintosh 512K enhanced only).
<code>#define ipBadAddr</code>	-23004	Error in getting address from a server or the address is already in use by another machine.
<code>#define connectionClosing</code>	-23005	A TCPclose command was already issued so there is no more data to send on this connection.
<code>#define invalidLength</code>	-23006	The total amount of data described by the WDS was either 0 or greater than 65,535 bytes.
<code>#define connectionExists</code>	-23007	The TCP or UDP stream already has an open connection.
<code>#define connectionDoesntExist</code>	-23008	This TCP stream has no open connection.
<code>#define insufficientResources</code>	-23009	There are already 64 TCP or UDP streams open.
<code>#define invalidStreamPtr</code>	-23010	The specified TCP or UDP stream is not open.
<code>#define streamAlreadyOpen</code>	-23011	An open stream is already using this receive buffer area.
<code>#define connectionTerminated</code>	-23012	The TCP connection went down; the reason will be given in a terminate ASR.
<code>#define invalidBufPtr</code>	-23013	The receive buffer area pointer is 0.
<code>#define invalidRDS</code>	-23014	The RDS refers to receive buffers not owned by the user.



<code>#define invalidWDS</code>	-23014	The WDS pointer was 0.
<code>#define openFailed</code>	-23015	The connection came up halfway and then failed.
<code>#define commandTimeout</code>	-23016	The specified command action was not completed in the specified time period.
<code>#define duplicateSocket</code>	-23017	A stream is already open using this local UDP port or a TCP connection already exists between this local IP address and TCP port, and the specified remote IP address and TCP port.
<code>#define ipDontFragErr</code>	-23032	The packet is too large to send without fragmenting and the Don't Fragment flag is set.
<code>#define ipDestDeadErr</code>	-23033	The destination host is not responding to address resolution requests.
<code>#define ipNoFragMemErr</code>	-23036	Insufficient internal driver buffers available to fragment this packet on send.
<code>#define ipRouteErr</code>	-23037	No gateway available to handle routing of packets to off-network destinations.
<code>#define nameSyntaxErr</code>	-23041	The <code>hostName</code> field had a syntax error. The address was given in dot notation (that is, W.X.Y.Z) and did not conform to the syntax for an IP address.
<code>#define cacheFault</code>	-23042	The name specified cannot be found in the cache. The domain name resolver will now query the domain name server and return the answer in the call-back procedure.
<code>#define noResultProc</code>	-23043	No result procedure is passed to the address translation call when the resolver must be used to find the address.
<code>#define noNameServer</code>	-23044	No name server can be found for the specified name string.
<code>#define authNameErr</code>	-23045	This domain name does not exist.
<code>#define noAnsErr</code>	-23046	None of the known name servers are responding.
<code>#define dnrErr</code>	-23047	The domain name server has returned an error.
<code>#define outOfMemory</code>	-23048	Not enough memory is available to issue the needed DNR query or to build the DNR cache.



---

## Miscellaneous types

This section describes types that are common to all the programmatic interfaces in MacTCP.

```
#define BYTES_16WORD      2      /* bytes per 16 bit ip word */
#define BYTES_32WORD      4      /* bytes per 32 bit ip word */
#define BYTES_64WORD      8      /* bytes per 64 bit ip word */

typedef unsigned char b_8;      /* 8-bit quantity */
typedef unsigned short b_16;    /* 16-bit quantity */
typedef unsigned long b_32;     /* 32-bit quantity */

typedef b_32 ip_addr;          /* IP address is 32-bits */
typedef b_16 ip_port

typedef struct ip_addrbytes {
    union {
        b_32 addr;
        char byte[4];
    } a;
} ip_addrbytes;

typedef struct wdsEntry {
    unsigned short length;      /* length of buffer */
    char * ptr;                 /* pointer to buffer */
} wdsEntry;

typedef struct rdsEntry {
    unsigned short length;      /* length of buffer */
    char * ptr;                 /* pointer to buffer */
} rdsEntry;

typedef unsigned long BufferPtr;
typedef unsigned long StreamPtr;
```



## Appendix Constants



---

## Command codes

UDPCreate	20
UDPRead	21
UDPBfrReturn	22
UDPWrite	23
UDPRelease	24
UDPMaxMTUSize	25
TCPCreate	30
TCPPassiveOpen	31
TCPActiveOpen	32
TCPSend	34
TCPNoCopyRcv	35
TCPBfrReturn	36
TCPRcv	37
TCPClose	38
TCPAbort	39
TCPStatus	40
TCPRelease	42
TCPGlobalInfo	43

---

## UDP asynchronous event codes

data arrival	1
ICMP message received	2

---

## TCP asynchronous event codes

closing	1
ULP timeout	2
terminate	3
data arrival	4
urgent data outstanding	5
ICMP message received	6



---

## Reasons for TCP termination

remote abort	2
network failure	3
security/precedence mismatch	4
ULP timeout	5
ULP abort	6
ULP close	7
service failure	8



---

## Internet Control Message Protocol report structures

In TCP and UDP, the ASR routine can be called with an Internet Control Message Protocol (ICMP) message. This section describes the types and structures of the ICMP messages.

```
typedef enum ICMPMsgType {
    netUnreach, hostUnreach, protocolUnreach, portUnreach, fragReqd,
    sourceRouteFailed, timeExceeded, parmProblem, missingOption,
    lastICMPMsgType = 65535
} ICMPMsgType;

typedef struct ICMPReport {
    StreamPtr streamPtr;
    ip_addr localHost;
    ip_port localPort;
    ip_addr remoteHost;
    ip_port remotePort;
    enum ICMPMsgType reportType;
    unsigned short optionalAddlInfo;
    unsigned long optionalAddlInfoPtr;
} ICMPReport;
```

Refer to the section "UDP Asynchronous Notification Routine" in Chapter 3 and "TCP Asynchronous Notification Routine" in Chapter 4 for details on how the ICMP report structure is used.



---

## GetMyIPAddr

This section describes how an application obtains the IP address of the machine on which it is running. GetMyIPAddr describes the parameter block that makes the PBControl call that returns the IP address and subnet mask of the local host. The csCode for this call is 15 and the driver reference number is returned from the Open Driver call.

```
#define ipctlGetAddr      15      /* csCode to get our IP address */

#define ParamBlockHeader \
    struct QElem *qLink;      \
    short qType;             \
    short ioTrap;           \
    Ptr ioCmdAddr;          \
    ProcPtr ioCompletion;    \
    OSErr ioResult;        \
    StringPtr ioNamePtr;    \
    short ioVRefNum;       \
    short ioCRefNum;       \
    short csCode

struct IPParamBlock {
    ParamBlockHeader; /* standard I/O header */
    ip_addr ourAddress; /* our IP address */
    long ourNetMask; /* our IP net mask */
};
```

The IP address is returned in the field ourAddress and the subnet mask is returned in the field ourNetMask.

