New Technical Notes
Macintosh

Developer Support

MacTCP Q&As

Networking **M.NW.MacTCP.Q&As**

Revised by: Developer Support Center October 1992
Written by: Developer Support Center October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

New Q&As and Q&As revised this month are marked with a bar in the side margin.

How to cancel transactions under MacTCP

Written: 12/7/90
Last reviewed: 8/1/92

How do I cancel transactions (specifically connect requests) under MacTCP? I know that I can specify a timeout; however, I would like the user to be able to hit 'Command-.' if the transaction's tired of waiting. I tried KillIO but that seems to crash.

—

Since MacTCP uses the Control call instead of reads and writes, the killio call is not supported to abort a transaction. You should use Abort and Release. Abort terminates the connection and Release releases the stream. As an alternative, a TCPPassiveOpen could be cancelled by issuing an active open from the same machine on the listening port.

Porting a TCP/IP application to the Macintosh

Written: 3/19/91
Last reviewed: 8/1/92

We need the following to port our mainframe front-end application to the Macintosh:

- a C++ compiler compatible with AT&T 2.0 (and ANSI C)
- a TCP/IP library with an Application Programming Interface (API) similar to sockets
- an Ethernet card

—

MPW C++ is based on AT&T CFront 2.0. For information about CFront 2.0, please refer to the UNIX System V AT&T C++ Language System Release 2.0 Product Reference Manual.

Sockets is the general UNIX (Berkeley) API for TCP/IP programming. Apple currently offers some libraries that allow you to work with MacTCP but not APIs similar to sockets. Third-party socket libraries for MacTCP are available, however. Also, universities such as the University of Toronto provide public domain socket wrappers for MacTCP.

Ethernet cards are available from a number of sources, including Apple's Ethernet cards.

MacTCP gateway address and subnet mask fields

Written: 5/7/91
Last reviewed: 8/1/92

What are the MacTCP Subnet Mask and Gateway Address fields used for?

—

In the TCP/IP protocol, there are a couple of provisions for simplifying the addressing scheme for smaller local networks. One of these is the subnet mask. It allows the sender to identify the destination machine by a shorter address, which is long enough to distinguish all the machines on the local network. This mask is made up of four octets, and is used as a bitwise mask to show how many bits are being used to specify the network number, subnet number, and node. The gateway address field specifies the IP address of the local network's router. Packets destined for another network are sent to this router, which then transmits the packets to other networks.

Communicating different processes on the same Mac with MacTCP

Written: 6/5/91
Last reviewed: 8/1/92

Is it possible to have both the source and the destination application on the same machine if they are using MacTCP to communicate? If so, what should I do to make sure both applications share the processor so this works properly?

—

It is possible to use MacTCP to communicate to different processes on the same Macintosh computer. Unlike PPC, MacTCP does not bypass the network for same-machine connections, so there is still some "network traffic" involved.

As far as "sharing" the processor goes, this is VERY important in what you are doing. Here's a brief outline of what you need to do:

• Make asynchronous MacTCP driver calls;
• If you want to use polled I/O, you can wait for ioResult < 1 to tell when the call completes. Otherwise, you may want to have a completion routine which queues completed calls for processing at event time; or
• Call WaitNextEvent or EventAvail while waiting for driver calls to complete (this gives time to the other MacTCP applications).

For a good reference on using "idle procs" to give time to background applications when using MacTCP, refer to the article, "MacTCP Cookbook" in develop magazine, Issue 6 and get the "NewsWatcher" source code from the Developer CD Series.

How to tell how many of StrToAddr's network addresses are valid

Written: 8/30/91
Last reviewed: 8/1/92

Using StrToAddr, you can receive up to four IP network addresses returned by the service for the host requested, but the MacTCP Programmer's Guide does not say how to tell how many of the addresses are valid. According to the name resolver code, MacTCP zeros out 4 long words before the StrToAddr lookup and then fills in the addresses it finds. This means that the array contains *at most* 4 addresses, and is terminated by a zero address (0.0.0.0) if fewer than four addresses are returned.

MacTCP hardware requirements

Written: 10/8/91
Last reviewed: 8/1/92

To use MacTCP, do I need just the software or do I need to have any extra hardware on the Macintosh, like an Ethernet card? Can I use MacTCP to set up a socket connection from LocalTalk to LocalTalk or I need to have some sort of an IP gateway between two Macintosh systems?

—

You have four choices: First, you can run MacTCP with an Ethernet card and directly connect to any other machine running IP (including other Macintosh systems). Another choice is to use your LocalTalk port. This requires an additional piece of hardware to do DDP-to-UDP encapsulation (also called KIP encapsulation). One example of this is the Cayman Gatorbox. You can't do LocalTalk to LocalTalk without the hardware. Your third

choice is to use EtherTalk in combination with a KIP gateway such as Fastpath or Gatorbox. Finally, MacTCP 1.1 provides a MacTCP extension mechanism whereby you can plug in alternate link layer protocolors by writing an 'mdev'. Documentation is available on the Developer CD Series disc. Examples include Apple's Token Ring extension and several third-party SLIP extensions.

MacTCP resolver code is in domain name resolver

Written: 9/17/91
Last reviewed: 8/1/92

Is the MacTCP resolver code contained in the domain name resolver (DNR)?

—

The resolver code is contained in the MacTCP DNR, as well as in the MacTCP control panel. The DNR code is stored as a resource file in the System folder and is read into memory as necessary in order to convert host names into addresses. It is used in conjunction with a domain name server or with the local hosts file.

MacTCP Hosts file location and function

Written: 9/17/91
Last reviewed: 8/1/92

Where should the MacTCP Hosts file be located, and who looks for that file?

—

According to the MacTCP 1.0 Documentation Kit (APDA #M0217LL/A), the Hosts file should be located in the user's System Folder. The file maps machine names to internet addresses, providing the same service as the domain name server. The Hosts file can be used if you have no domain server on your network. It is also suggested that this file be used for frequently used name-to-address mappings. Instructions on setting up the file are included in the MacTCP documentation.

MacTCP "obtain address" options

Written: 9/24/91
Last reviewed: 8/1/92

Under MacTCP, what is the significance of the "obtain address" options, with respect to (1) manually, (2) server, and (3) dynamically?

—

According to the MacTCP 1.0 Documentation Kit (APDA #M0217LL/A), "obtain address" means the following:

• Manually: You will need to fill in some or all of the fields in the IP Address box;
• Server: The Internet Protocol (IP) address for the user Macintosh is automatically obtained from a server every time the user Macintosh boots up. This option requires
 - a Reverse Address Resolution Protocol (RARP) or Bootstrap Protocol (BootP) server on an Ethernet, or
 - a Datagram Delivery Protocol–to–Internet Protocol (DDP-IP) gateway on an AppleTalk network that's compatible with the Kinetics Internet Protocol (KIP) developed at Stanford (also called "IPTalk"), such as the Shiva Fastpath or Cayman Gatorbox

• Dynamically: The node portion of the IP address for the user Macintosh is set dynamically every time the user Macintosh boots up. With this option you still need to set some of the fields in the IP Address Box. All these processes are described in further detail in the MacTCP documentation.

MacTCP and SLIP

Written: 11/6/91
Last reviewed: 8/1/92

Does MacTCP actually support SLIP? If not, does Apple plan to provide SLIP support in the future. Does anyone know of any company that does a SLIP for the Macintosh?

—

SLIP is an asynchronous, serial line protocol developed for running TCP/IP over serial communications lines in a point-to-point configuration. SLIP was developed to transmit IP packets over low-speed, sometimes noisy, asynchronous communications lines where error recovery and an efficient line protocol are needed. The SLIP protocol is now being replaced with a new serial line protocol named "PPP," which uses a more efficient means of establishing a point-to-point IP connection.

MacTCP includes hooks that let you write different link-layer modules. This makes possible the development of interfaces to SLIP, PPP, and to any other link layer that someone may need, like broadband, X.25, and FDDI. Apple does not currently provide support in MacTCP for SLIP or any other serial line protocol. There are, however, several third-party SLIP extensions available for use with MacTCP.

MacTCP StrToAddr and header's hostInfo record structure

Written: 12/12/91
Last reviewed: 8/1/92

I'm using MacTCP v1.1 with Think C, and when calling StrToAddr with hostname = "90.25.3.240", the returned addr field has the following:

```
addr [0] = 0X00005A19
addr [1] = 0X03F00000
```

The correct answer should be addr [0] = 0X5A1903F0. What's the problem?

—

The StrToAddr problem you are experiencing is related to a shortcoming in the header files. The structure for the hostInfo record you are accessing (from AddressXlation.h)is as follows:

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

The problem with the way the structure is defined is that rtnCode is defined as an "int." The size of an int is dependent on your development environment. In MPW, ints are 4 bytes, while Think C uses 2 byte ints as its default. If you're using Think C, you will see the addresses shifted forward in the storage by 2 bytes, since the rtnCode structure causes the rest of the struct to be shifted forward in memory.

If you're using Think C 5.0, you can either check the "Use 4 byte ints" check-box in the preferences, or you can change the header files to use "long" in place of "int" (a Think C long is the same size as an MPW int). A set of header diffs for Think C compatibility with MacTCP is on the latest Apple Developers CD Series disc in "Tools & Apps:Networking & Communication:MacTCP 1.1 DTS Header Changes."

MacTCP EnumCache data enumeration sequence

Written: 12/12/91
Last reviewed: 8/1/92

Does MacTCP's EnumCache name resolver call return the data in a particular order or randomly?

—

Since the EnumCache documentation doesn't specify any particular sequence for name resolver cache enumeration, it's best to play it safe and not assume any particular order for the data returned by the call. Since the internals of the name resolver may change in the future, you probably don't want to rely on any undocumented data ordering assumptions. If you want the entries separated, the best way probably would be to have a dynamically-sized array for each of the entry types (such as addresses, HInfo, and name servers) and fill them in the EnumCache callback.

MacTCP addressing modes and Shiva Fastpath

Written: 1/6/92
Last reviewed: 8/1/92

When I attempt to connect to a LocalTalk net using MacTCP 1.1, after opening the MacTCP driver, this error occurs: “Error opening driver / Error in getting address (-23004)” I‘m running System 7.0 on a Macintosh IIci with Shiva Fastpath 4.0, and I’ve configured TCP Admin as follows:

Indicate Resident Zone
Click More
Dynamic addressing
Range 1 to 65534
Enter Gateway Address (address of Fastpath for resident zone)
Enter Domain Name & Address

—

The Fastpath won’t work with MacTCP configured to use dynamic addressing. You need to set MacTCP to use server-based addressing. If you change this setting, and the FastPath is configured correctly, you should be up and running.

MacTCP 1.1 and sending urgent data per RFC 1122 or 793

Written: 4/7/92
Last reviewed: 8/1/92

MacTCP 1.1 packets containing “urgent data” have the so-called “urgent pointer” set according to RFC 1122. All our TCP/IP hosts expect a behavior corresponding to RFC 793, however. Is there a possibility to configure MacTCP to this effect?

—

MacTCP can send urgent data according to either RFC 1122 or RFC 793; however, it is not a setting in the driver that determines this. When a programmer makes the TCPSend call, he sets a flag in his parameter block that indicates whether or not to use urgent data, and which version of urgent data to use. Therefore, you would have to modify the program that you are using to set up the parameter blocks appropriately.

To send urgent data using the non-compliant RFC 793 method, set the urgent flag in the TCPSend parameter block to 2. Any other nonzero value in the urgent flag indicates the RFC 1122 method should be used.

MacTCP 1.1 ICMP echo request documentation bug

Written: 2/13/92
Last reviewed: 8/1/92

I’m trying to use the ICMP echo request function described on page 88 of the MacTCP Developer’s Kit 1.1 manual to 'ping' a TCP host before trying to open a connection. The problems I’m having seem centered around the ICMPEchoNotifyProc. If I declare the proc as

```
pascal void MyEchoNotifyProc (struct ICMPParamBlock *iopb)
```

the proc gets called and the ICMPParamBlockPtr is valid. But after the proc returns, my Macintosh crashes. Can you give me any suggestions as to how to make this puppy work?

—

You've found a "bug" in the manual. The ICMPEchoNotifyProc is actually defined as:

```
typedef void (*ICMPEchoNotifyProc) (struct ICMPParamBlock *iopb);
```

The definition is correct in <MiscIPPB.h>. Note that in this definition, the proc uses C, not Pascal calling conventions. In C, the caller removes the parameters from the stack, not the callee, so since your routine has already removed the parameters before returning, they are removed twice, corrupting the stack. Changing your procedure to use C calling conventions should fix your problem.

MacTCP 1.1 & 1.0.1 compatibility

Written: 2/13/92
Last reviewed: 8/1/92

We have implemented a TCP/IP product using MacTCP Development Kit version 1.1. If MacTCP version 1.0.1 is installed, will there be any compatibility problems with our implementation.?

—

There are only a very few new calls in MacTCP 1.1 that aren't in the 1.0.1 driver. Among these are:

- Name resolver HInfo and MXInfo calls
- UDP multi-port sends and receives
- ICMP echo protocol support

As long as you're not using any of these features, any program developed for MacTCP 1.1 will work with 1.0.1.

MacTCP 1.1 header file incompatibilities and fixes

Written: 9/30/91
Last reviewed: 8/1/92

MacTCP 1.1, Apple's System 7-compatible version of MacTCP, fixes several problems in earlier releases, but doesn't address several header file incompatibilities. This short note outlines the MacTCP 1.1 header file problems, but it does not include the fixes. Patches in the form of MPW "compare" output are available on Apple's latest Developer CD Series disc. The new headers are available as part of the "MacTCP Developers' Kit," available from APDA (part #M0704/C).

Each header problem is described below, organized by file(s) containing the problem and classified as a general bug or shortcoming, a Think C incompatibility, or a C++ incompatibility. NOTE: The changes have not been thoroughly tested, so use them cautiously.

Problems affecting all header files:

• General problem: Each header file is missing the

```
#ifdef __cplusplus
extern "C" {
#endif
```

and

```
#ifdef __cplusplus
extern "C" {
#endif
```

which are necessary for C++ unmangling.

• General problem: There are no #ifdefs to prevent including a file multiple times, as in other Apple headers. This was fixed by adding the following to each of the header files:

```
#ifndef __MACTCPCOMMONTYPES__
#define __MACTCPCOMMONTYPES__
...
#endif
```

where “MACTCPCOMMONTYPES” is replaced by the name of the header file.

• Think C problem: Think C before 5.0 does not support pascal typedefs, which MacTCP makes use of, so #ifdef THINK_C was added in several instances to declare these pascal typedef functions as type ProcPtr. This fixes the problem.

GetMyIPAddr.h problem:

• General problem: ParamBlockHeader is used as a #define here, conflicting with its use in <Files.h>. The solution is to change the name of the #define to IPParamBlockHeader.

MiscIPPB.h problems:

• General problem: ParamBlockHeader is used as a #define here, conflicting with its use in <Files.h>. The solution is to change the name of the #define to GetIPParamBlockHeader.

• General problem: The structure IPParamBlock is defined in this file with a different definition from the one in GetMyIPAddr.h. The solution is to change the name of the struct.

• General problem: AppleTalk.h is required for successful compilation. Code was added to include this if it has not already been included.

• General problem: icmpEchoTimeoutErr is defined in MacTCPCommonTypes in addition to being defined in MiscIPPB.h. Solution is to remove its definition here.

AddressXLation.h problems:

Think C incompatibility: The “int” type is used within the “hostInfo” structure definition. The default size of an int is different from Think to MPW. In all cases, int in the MacTCP headers should be changed to long.

• Think C incompatibility: The returnRec struct uses an int. Fix is to change to a long as above.

• Think C incompatibility: The CloseResolver() prototype is defined with an empty parameter list, not with a void parameter list. Think C requires the void to consider the definition a prototype. The fix is to add the void keyword as the function parameter list.

• C++ incompatibility: The cacheEntryRecord struct uses a variable named “class”. This causes major problems with C++ compilers, since the class keyword takes on a different meaning in this instance. The fix is to change this to “cacheClass.”

dnr.c problems:

• General/Think C problem: The typedef for OSErrProcPtr was incomplete, causing ambiguity for argument casting. This was changed to "typedef OSErr (*OSErrProcPtr)(long,...);" to make sure the first parameter to the name resolver calls is passed as a long.

• Think C incompatibility: Defines in Think default to short, not long, so all name resolver calls are made incorrectly. This is fixed by the typedef in above, but Think 4.0.5 does not recognize partial prototypes correctly. The fix is to add a “L” to the end of each of the name resolver command #defines.

• Think C incompatibility: The MXInfo() procedure has an extra semicolon after the trailing brace “};” which causes a syntax error with the Think C compiler. The semicolon was removed to fix the problem.

No Pascal headers for MacTCP

Written: 3/19/92
Last reviewed: 8/1/92

I want to access MacTCP in a Pascal program. Do you have the C parameter block definitions available as Pascal records or will I have to do my own translation?

—

Unfortunately, Pascal header files and interfaces are not available for MacTCP. If you want to use Pascal, you'll have to translate the interfaces.

One problem you will run into is that all MacTCP routines and callbacks (those in dnr.c/dnr.o, for example) use C calling conventions. You'll still be able to call the dnr routines by declaring them as external C functions, as described in the MPW Pascal documentation, but you'll still have to write any callback functions in C or Assembler, since the Pascal compiler can only call a function using C conventions, but you can't declare a Pascal function to have C conventions.

X-Ref:
MacTCP Programmer's Guide, Chapter 4, page 47

MacTCP Type of Service documentation fix

Written: 4/21/92
Last reviewed: 7/13/92

Are MacTCP's Type Of Service (Reliability, Low Delay and Thruput) bit settings swapped in the documentation?

—

The bit settings in the MacTCP programmer's documentation are reversed for the Type of Service constants. This will be fixed in a future version of the manual. The bit settings should be as follows:

Type of Service Byte (3-bit field)
Bit 0 set for low delay
Bit 1 set for high reliability
Bit 2 set for high throughput

If you wish to correct these settings they can be found on page 35 of the MacTCP Programmer's Guide.

MacTCP ULP timeout documentation fix

Written: 5/5/92
Last reviewed: 7/13/92

In the MacTCP Developer's Kit (Version 1.1) there is a ULP timeout action field which is used in several PowerBook calls to MacTCP. For some calls, zero indicates an abort and nonzero indicates a report. TCPSend, TCPClose and TCPStatus are this way. For other calls, the reverse is documented (0 indictes report and 1 indicates abort). TCPActiveOpen and

TCPPassiveOpen are this way. Can I believe the documentation or are there errors in it ? If there are errors, what is the correct way it should be ?

—

According to the MacTCP source code, and you're right, the documentation is incorrect. For each of the calls with a ULP timeout action, the correct values are:

1 = abort
0 = report only

Macintosh Communications Toolbox MacTCP tools

Written: 6/10/92
Last reviewed: 9/15/92

My application uses the Macintosh Communications Toolbox and ADSP to connect to network services. What software do I need to get to make the same connections using TCP/IP? MacTCP or other recommendations?

—

Apple only makes one MacTCP Communications Toolbox tool that works specifically with MacX and TCP/IP (the TCP Tool). This tool is not supported for use by third party developers. For CTB/MacTCP connection tools which are supported, you need to get a third-party CTB MacTCP tool. These are available from several vendors. For example, TCPack from ASC (Advanced Software Concepts) is a set of Connection Tools designed to allow simultaneous TCP/IP connections using Apple's MacTCP driver. All Apple's terminal tools (asc3270, asc5250, …), the Apple standard terminal tools (TTY, VT100, VT320), as well as several third-party terminal tools can be used to enter terminal sessions with TCPack. TCP/Tools from InterCon Systems is another package which allows you to use existing CTB applications to log onto Unix workstations using your LocalTalk or EtherTalk network.