TTC Socket Custom Control v1.51a

Enhancements

The Socket Custom Control

The Socket Custom Control encapsulates the low-level functions required to perform communications on the Internet. The control allows you to open a socket, connect to a remote host, send and receive information, and wait for connections from remote hosts.

The purpose of developing this control is : 1) To provide a means of developing Internet connectivity applications 2) To provide a means of implementing many of the standard Internet application protocols 3) To provide Internet utilities for users to plug into their 32-bit container/controlling applications.

To use the control you must first incorporate the control into a 32-bit container's environment. Once added the custom controls functionality is available to your container environment. You have access to the following members for the control :

SOCKET.ErrorNum

Member containing the last integer error encountered.

SOCKET.AboutBox()

Display the about box containing copyright and version information.

SOCKET.OpenSocket(*AdrsFmt As Integer, SocketType As Integer, Protocol As Integer*)

This member function establishes a communications socket of the type specified in its parameters. The *AdrsFmt* is the format of the sockets address, *SocketType* defines whether the socket is a stream (TCP) or a datagram (UDP), *Protocol* is the protocol used for communications. For information on these parameters refer to Socket Parameters .

This method returns 1 upon success and 0 upon failure. If a failure occurs it places an error code in the **ErrorNum** member.

SOCKET.RecvRemote(*TimeOut As Integer*)

This method receives data from a remote host. The *TimeOut* argument is the number of milliseconds of timeout that you allow for the receive operation. A value of 0 for *TimeOut* is usually acceptable for most operations.

This function will place an error code in **ErrorNum** and return a NULL string if errors occur. If successful the function returns the received data as a string.

SOCKET. SendRemote(*Buffer As String, TimeOut As Integer*)

This method sends data to a remote host. The *Buffer* argument contains the data to be sent. The *TimeOut* argument is the number of milliseconds of timeout that you allow for the receive operation to take place. A value of 0 for *TimeOut* is usually acceptable for most operations.

This function will place an error code in **ErrorNum** and return 0 if errors occur. If successful the function returns 1.

SOCKET. ConnectRemote(*AdrsFmt As Integer, IPPort As Integer, IPAdrs As String*)

This method connects to a remote host. The *AdrsFmt* is the format of the sockets address, *IPPort* is the service port to which you wish to connect, *IPAdrs* is the address of the remote host to which you wish to connect. For information on these parameters refer to Socket Parameters . This function will place an error code in **ErrorNum**

and return 0 if errors occur. If successful the function returns 1.

SOCKET.CreateListenSocket(*IPPort As Integer, AdrsFmt As Integer, SocketType As Integer, Protocol As Integer*)

This method creates a socket to listen for connections from a remote host. The *IPPort* is the port to create to which remote hosts will connect, *AdrsFmt* is the format of the sockets address, *SocketType* defines whether the socket is a stream or a datagram, *Protocol* is the protocol used for communications. For information on these parameters refer to Socket Parameters . This function will return the name of the listen socket if successful. If unsuccessful, the function will return a NULL string and place an error code in **ErrorNum**.

SOCKET.WaitForConnect()

This function is called after a call to **SOCKET.CreateListenSocket()** is made in order to wait for a connection to a listening socket from a remote host.. This function will place an error code in **ErrorNum** and return 0 if errors occur. If successful the function returns 1.

SOCKET.CloseSocket()

Closes the socket.

Sample SMTP Code

In this set of sample code the Socket Custom Control is being used as an email client to allow you to send email using the Simple Mail Transfer Protocol (SMTP). This protocol handles handshaking with your mail server to submit mail for distribution to your email-pals on the Internet.

```
Private Sub Command1_Click()
```

We first use the GetHost Custom Control to determine the IP address of our mail server.

```
Address_String = GetAdrs1.GetHostAdrs("MYMAIL.SRVR.NET")
```

Once the IP address is received, we open a socket and connect to the remote node designated in Address_String.

```
retval = Socket1.OpenSocket(2, 1, 6)
retval = Socket1.ConnectRemote(2, 25, Address_String)
Text1.Text = Socket1.RecvRemote(0)
```

Once connected we begin communications with the remote mail server. Since this is merely a demo of the Socket Custom Control, the details of SMTP will not be detailed here.

```
msgbuffer = "HELO TOUPIN" + Chr(13) + Chr(10)
retval = Socket1.SendRemote(msgbuffer, 0)

msgbuffer = "MAIL FROM:<etoupin@toupin.com>" + Chr(13) + Chr(10)
retval = Socket1.SendRemote(msgbuffer, 0)

msgbuffer = "RCPT TO:<etoupin@toupin.com>" + Chr(13) + Chr(10)
retval = Socket1.SendRemote(msgbuffer, 0)

msgbuffer = "DATA" + Chr(13) + Chr(10)
retval = Socket1.SendRemote(msgbuffer, 0)

msgbuffer = "Date: " + Date$ + " " + Time$ + Chr(13) + Chr(10)
retval = Socket1.SendRemote(msgbuffer, 0)

msgbuffer = "From: etoupin@toupin.com " + Chr(13) + Chr(10)
retval = Socket1.SendRemote(msgbuffer, 0)
```

```
        msgbuffer = "To: etoupin@toupin.com" + Chr(13) + Chr(10)
        retval = Socket1.SendRemote(msgbuffer, 0)

        msgbuffer = "Subject: Test" + Chr(13) + Chr(10)
        retval = Socket1.SendRemote(msgbuffer, 0)

        msgbuffer = "Mailer: Test from Visual Basic" + Chr(13) + Chr(10)
        retval = Socket1.SendRemote(msgbuffer, 0)

        msgbuffer = "Test of  the Socket Control" + Chr(13) + Chr(10)
        retval = Socket1.SendRemote(msgbuffer, 0)

        msgbuffer = "." + Chr(13) + Chr(10)
        retval = Socket1.SendRemote(msgbuffer, 0)

        msgbuffer = "QUIT" + Chr(13) + Chr(10)
        retval = Socket1.SendRemote(msgbuffer, 0)
```

Once the data is sent as required the socket is closed.

```
        retval = Socket1.CloseSocket()
    End Sub
```

Sample Client / Server Code

The following code uses a small server and a small client to merely transfer messages between two Internet nodes.

Server Application

```
Private Sub Command1_Click()
```

We first establish a listening socket used to wait for a connection from a remote node.

```
    retstr$ = Socket1.CreateListenSocket(43, 2, 1, 6)
```

The application then sits and waits for a remote connection on our listening socket.

```
    retval = Socket1.WaitForConnect()
    If retval = 0 Then
        MsgBox "Server Wait : " + Str(Socket1.ErrorNum)
    End If
```

Whenever we receive a connection we send information to the remote client application then wait for information to arrive from the remote application.

```
    retval = Socket1.SendRemote("Howdy From the Server", 0)
    Text1.Text = Socket1.RecvRemote(0)

    retval = Socket1.CloseSocket()
End Sub
```

Client Application

```
Private Sub Command1_Click()
```

We first establish a socket through which we will communicate.

```
    retval = Socket1.OpenSocket(2, 1, 6)
```

We then connect to a remote node, specified by its IP address. This IP address can be resolved using the GetHost

Custom Control.

```
        retval = Socket1.ConnectRemote(2, 43, "204.132.123.26")
        If retval = 0 Then
             MsgBox "Client Connect : " + Str(Socket1.ErrorNum)
        End If
```

If the connection is successful we wait for information from the remote server then send information back to it.

```
        Text1.Text = Socket1.RecvRemote(0)
        retval = Socket1.SendRemote("Howdy From the Client", 0)

        retval = Socket1.CloseSocket()
    End Sub
```

Socket Parameters

The following lists are valid parameters for several of the member functions of the control.

Socket Types

STREAM	1	Stream socket
DGRAM	2	Datagram socket
RAW	3	Raw-protocol interface
RDM	4	Reliably-delivered message
SEQPACKET	5	Sequenced packet stream

IP Port Definitions

NETSTAT	15	Network Statistics
FTP	21	File Transfer Protocol
TELNET	23	Telnet
SMTP	25	Simple Mail Transfer Protocol
TIMESERVER	37	Network Time
NAMESERVER	42	Domain Name Service
WHOIS	43	WHOIS
FINGER	79	Finger
POP3	110	Post Office Protocol v3
NEWS	144	Newsgroups
SNMP	161	Simple Network Management Protocol
TALK	517	TALK
NTALK	518	NTALK

Address Format Definitions

UNSPEC	0	Unspecified
UNIX	1	Local communications (i.e. pipes, portals, etc.)
INET	2	Internet UDP, TCP, etc.
IMPLINK	3	Arpanet
PUP	4	Pup protocols
CHAOS	5	MIT PUP protocols

IPX	6	IPX and SPX
NS	6	XEROX NS protocols
ISO	7	ISO protocols
ECMA	8	European computer manufacturers
DATAKIT	9	Datakit protocols
CCITT	10	CCITT protocols, X.25 etc
SNA	11	IBM SNA
DECnet	12	DECnet
DLI	13	Direct data link interface
LAT	14	Local Area Transport
HYLINK	15	NSC Hyperchannel
APPLETALK	16	AppleTalk
NETBIOS	17	NetBios-style addresses

Protocol Definitions

IP	0	Dummy for IP
ICMP	1	Control message protocol
GGP	2	Gateway^2
TCP	6	TCP
PUP	12	PUP
UDP	17	User Datagram Protocol
IDP	22	XNS IDP
RAW	255	Raw IP packet

What Is Sockets

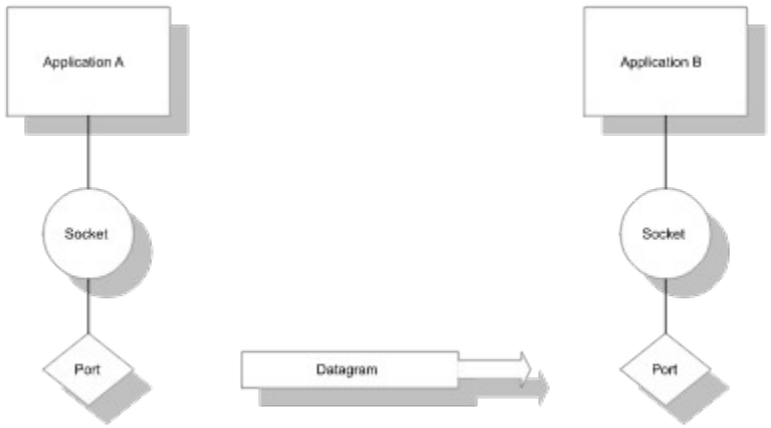
A socket is the basic building block for communications and represents endpoints of communications for processes on the Internet. Sockets can be of two specific types and provide a means of communicating with a specific protocols for a particular service. The two types of sockets currently available for users are a *stream socket* and a *datagram socket*. The primary purpose of the Socket Custom Control is to encapsulate the functionality of sockets so that you can easily develop applications that communicate across the Internet.

Stream sockets provides a connection oriented path that allows two processes to establish a *logical link* for communications. When a connection oriented link is established, the processes are aware of each other and can send information to each other until one of the processes closes the link. The advantage to stream sockets is that they provide bi-directional, reliable, sequenced, and unduplicated flow of data.

It might be easiest to think of a stream as a water hose pipe. You can connect one end of the hose pipe to a faucet and the other to a sprinkler. When you turn on the water at the faucet, the water flows from the faucet, through the pipe, and out the sprinkler. You can turn off the water and move the sprinkler to sprinkle in a different area of your yard. The logical link is represented by the hose pipe and the data stream is the water flowing to a predetermined destination.

Connection Oriented Stream Communications

Datagram sockets are connectionless and do not require a logical link to a remote socket in order to communicate. Datagrams are submitted to the Internet and are forwarded to remote hosts which in turn are accepted and processed at the remote node. Datagram sockets support bi-directional flow of data which is not promised to be sequenced, reliable, or unduplicated. Instead of packets being sent across an established connection, as with a stream, there is no permanent connection.



Connectionless Datagram Communications

Client / Server

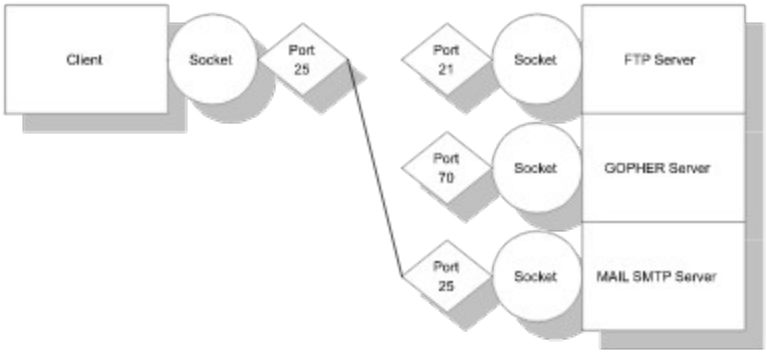
In constructing a system using sockets, the most common paradigm used is the *client/server model*. In this model, client applications request the services of server applications. In order to allow the clients and the servers to communicate they require a common set of conventions before service may be requested or rendered. This set of conventions is known as a *protocol* and must be known at both ends of a connection.

In socket communications there are several important protocol layers available : *communications layer, network layer,* and *application layer.* The network layer is the lowest level of communications available that you need to be aware of for this level of socket application development. This layer allows you to connect to the Internet and consists of the protocols used for the network including Ethernet and Token Ring as well as modem to modem communications.

Three Important Layers of Socket Programming

The next layer up is the network layer and consists of the protocol used to communicate across the network. This layer consists of protocols such as TCP/IP, NETBIOS, DECNet, etc. The topmost layer is the application layer and consists of the method by which applications communicate. This layer includes FTP, GOPHER, HTTP, WHOIS, NNTP, NTP, etc.

At the uppermost application layer applications determine how they talk to a remote server by way of a well-known service port. Service ports define the service access point at which applications exchange information. Lets look at an example. If someone were to call you they must dial your phone number. For applications, if an application wishes to talk to an FTP server on a remote node, they must connect to the service port for FTP on that remote node. A server application normally listens at the service port for service requests and remains dormant until a connection is requested by a client. When a request is made the server services the client to perform whatever actions the client requests.



Clients and Servers connect over a common service port.

Agreement

This software, hereafter referred to as **the software**, Copyright © 1995 Edward B. Toupin has been developed by Edward B. Toupin, hereafter referred to as **the author**, and remains the property of Edward B. Toupin, Denver, CO.

The software and accompanying documentation are protected by a United States Copyright. Any use of the software in violation of copyright laws or the terms of this agreement will be prosecuted to the best of our ability. Under no circumstances will the user remove the copyright notices made part of this software or documentation.

The author makes no warranty of any kind, express or implied, including without limitation, any warranties of merchantability and/or fitness for a particular purpose. The author shall not be held liable for any damages, whether direct, indirect, special or consequential arising from a failure of the software to operate in the manner desired by the user.

The author does not assume any liability for the use of the software. In no event will the author be liable for any damages including lost profits, lost savings, damage to property or other incidental or consequential damages arising out of your use or inability to use the software, or any claim by any other party.

The software is warranted to operate in the manner as described in the software documentation. If you should encounter an error in the documentation or the software, a description of the problem should be sent directly to the author. The error will be fixed and an update of the documentation and/or software will be returned to you.

By using the software, including any of the associated files, you agree to the terms of this agreement. If you do not agree, you should immediately return the software and documentation.

Please complete the included registration and feedback and include any comments or suggestions for future enhancements regarding the software. This registration will help us stay in touch with our customers and their needs. Email the feedback and US mail the registration as well as any comments or suggestions to Edward B. Toupin at etoupin@toupin.com or Compuserve 75051,1160.

Connectivity Custom Controls

Plug-n-Play for the Highway

Plug-n-Play

With the evolution of component software users have come to appreciate the ability to simply connect applications together to create a seamless application that meets their specific needs. The Connectivity Custom Control Pack provides you with that plug-n-play capability for network communications across the Information Superhighway.

No Programming Knowledge Necessary

The advantage to the Connectivity Custom Control Pack is that it can be used by anyone with any level of programming knowledge. The management of communications and application protocols are encapsulated in several small control libraries which allow you to connect to and manage information on the Information Superhighway.

All you need to use the Connectivity Custom Control Pack is Windows Sockets v1.1, a container/controlling application for 32bit operating systems, and a SLIP/PPP connection to the Internet.

Connectivity Custom Controls

The Connectivity Custom Control Pack comes with the following controls :

1. **ARCHIE CLIENT** Utilize ARCHIE services.
2. **CHATTER** Create applications to talk with other users on the Internet.
3. **CONNECT** Connect to providers and remote servers using dialup PPP.
4. **FINGER** Query hosts for user information.
5. **FTP** File transfer via File Transfer Protocol.
6. **GETHOST** Address and name resolution using the Domain Name System.
7. **GOPHER CLIENT** Utilize GOPHER services.
8. **IRC** Internet Relay Chat
9. **MAIL** Send (SMTP) and receive (POPv3) electronic mail.
10. **NEWS** Retrieve, read, and submit to newsgroups.
11. **SNMP** Network management via Simple Network Management Protocol.
12. **TELNET** Telnet terminal emulation.
13. **TIME** Set your computers time from a remote host.
14. **WAIS CLIENT** Utilize Wide Are Information Server services.
15. **WEB CLIENT** Retrieve and view WWW home pages.
16. **WEB SERVER** Create and administer a WWW server.
17. **WHOIS** Locate hosts and users on the Internet.
18. **WSOCK** Develop low-level socket applications.

Error Codes

When an error occurs the member functions will return a NULL string or a 0 and an error value will be placed into the **ErrorNum** member variable. The following list are possible error numbers that will occur and can only be handled within your container :

9	SOCK_BAD	Generic error for invalid format, bad format.
13	SOCK_ACCESS	Generic error for access violation.
14	SOCK_FAULT	Generic error for fault.
22	SOCK_INVALID	Generic error for invalid format, entry, etc.
24	SOCK_FILE	Generic error for file error.
25	SOCK_BADDR	The IP address provided is not valid or the host specified by the IP does not exist.
48	SOCK_ADDRINUSE	The specified address is already in use.
49	SOCK_ADDRNOTAVAIL	The specified address is not available.
50	SOCK_NETDOWN	The connected network is not available.
51	SOCK_NETUNREACH	The connected network is not reachable.
52	SOCK_NETRESET	The connected network connection has been reset.
53	SOCK_CONNABORTED	The current connection has been aborted by the network or intermediate services.
54	SOCK_CONNRESET	The current socket connection has been reset.
57	SOCK_NOTCONN	The current socket has not been connected.
58	SOCK_SHUTDOWN	The connection has been shutdown.
60	SOCK_TIMEDOUT	The current connection has timedout.
61	SOCK_CONNREFUSED	The requested connection has been refused by the remote host.
63	SOCK_NAMETOOLONG	Specified host name is too long.
64	SOCK_HOSTDOWN	Remote host is currently unavailable.
65	SOCK_HOSTUNREACH	Remote host is currently unreachable.
91	SOCK_SYSNOTREADY	Remote system is not ready.
92	SOCK_VERNOTSUPPORTED	Current socket version not supported

		by application.
93	SOCK_NOTINITIALISED	Socket API is not initialized.
101	SOCK_DISCON	Socket has been disconnected.
110	SOCK_FTP_RESTART	A restart marker has been received from the remote host.
120	SOCK_FTP_SYS_DELAY	FTP service ready in *nnn* minutes.
125	SOCK_FTP_CONN_X	FTP data connection already open.
150	SOCK_FTP_FILEOK	FTP File status is okay.
200	SOCK_FTP_OK	FTP command is okay.
202	SOCK_FTP_CMD_IMP1	FTP command is not implemented
210	MAIL_NOCNCT	No mail connection available.
211	SOCK_FTP_SYS_STAT	Current FTP reply is a system status or system help reply.
211	MAIL_BADORIG	Originating email address is invalid.
212	SOCK_FTP_DIR_STAT	Current message is a directory status.
212	MAIL_BADDEST	Destination email address is invalid.
213	SOCK_FTP_FILE_STAT	Current message is an FTP file status.
213	MAIL_BADDOM	Domain name provided to SMTP is invalid.
214	SOCK_FTP_HELP_MSG	Current message is a help message.
214	MAIL_NODATA	Cannot send email data.
215	SOCK_FTP_SYS_NAME	FTP system type.
215	MAIL_NOEND	Cannot end the transmission of email.
216	MAIL_NOQUIT	Unable to end the email session.
217	MAIL_BADCC	CC address(es) are invalid.
220	SOCK_FTP_SVC_READY	FTP service ready for new user.
220	POP_NOCNCT	POP connection not available.
221	SOCK_FTP_SVC_CLOSING	FTP server closing control connection.
221	POP_NOSTAT	POP statistics unavailable.
222	POP_NOLIST	POP email list unavailable.
223	POP_INVUSER	Invalid POP account username.
224	POP_INVPASS	Invalid POP account password.
225	SOCK_FTP_CONN_NOX	No FTP connection.
225	POP_NOOFF	Unable to logoff of POP.
226	SOCK_FTP_CLOSE_CONN	FTP server closing data connection.
226	POP_NODEL	Unable to delete mail.
227	SOCK_FTP_PASSIVE	Passive Mode (h1,h2,h3,h4,p1,p2).

227	POP_NOGET	Unable to get mail.
228	POP_NOUNDEL	Unable to undelete mail.
230	SOCK_FTP_LOGIN	Unable to log into FTP server.
250	SOCK_FTP_FILEACT_OK	Requested file action okay, completed.
257	SOCK_FTP_PATH_CREATE	"PATHNAME" created.
331	SOCK_FTP_PASS	Invalid FTP password.
332	SOCK_FTP_ACNT1	Invalid FTP account.
350	SOCK_FTP_FILE_WAIT	File action pending further information.
421	SOCK_FTP_SVC_NOT_AVAIL	Service not available.
425	SOCK_FTP_CONN_CLOSED	Cant open data connection.
426	SOCK_FTP_XFR_ABORT	Connection closed; transfer aborted.
450	SOCK_FTP_FILE_BUSY	Requested file action not taken.
451	SOCK_FTP_LOCAL_ERROR	Local error in processing.
452	SOCK_FTP_INSUFF_SPC	Insufficient storage space in system.
500	SOCK_FTP_CMD_SYNTAX	Syntax error, command unrecognized.
501	SOCK_FTP_ARG_SYNTAX	Syntax error in arguments.
502	SOCK_FTP_CMD_IMP2	FTP command not implemented.
503	SOCK_FTP_BAD_SEQ	Bad FTP sequence.
504	SOCK_FTP_CMD_IMP3	FTP command not implemented.
530	SOCK_FTP_NOLOGIN	Not logged into remote host.
532	SOCK_FTP_ACNT2	Need account on remote host to store files.
550	SOCK_FTP_FNOT_FOUND	Requested action not taken.
551	SOCK_FTP_PAGE_UNK	Page type unknown.
552	SOCK_FTP_XEED_STOR	Exceeded storage allocation
553	SOCK_FTP_NAME_NOTALL	File name not allowed.
926	SOCK_FTP_INVFILE_HANDLE	Local file is invalid
927	SOCK_FTP_INVFILE	Local file is invalid
1001	SOCK_HOST_NOT_FOUND	Host not found
1002	SOCK_TRY_AGAIN	Not authorized, host not found
1003	SOCK_NO_RECOVERY	Non recoverable error
1004	SOCK_NO_DATA	No data available for request
1010	SOCK_GPH_NOFNAME	Query requires an output file.
1011	SOCK_GPH_NOEXEC	Unable to execute application.
1012	SOCK_NO_SND_DATA	Unable to send data
1013	SOCK_NO_CNCT_DATA	Unable to connect to remote

The Toupin Corporation

An Internet Services Company

A port is doorway through which a connection to a remote Internet host passes. Ports are accessed by their number where each type of service on the Internet utilizes a particular port number. For Gopher, the port number is 70.

An IP Address is much like your own address for your home. Everyone of your friends knows your house by your name where, for instance, a house is the Smiths house. The post office knows that the Smiths house is located at 1122 South Street. On the Internet, all nodes have a name (e.g., toupin.com) and an associated address (e.g., 199.117.41.151). The address, being the IP Address, is used to send information to remote nodes. The address can be resolved for a known host name using the GetHost custom control.

Installation

To install the custom control you should perform the following steps :

Copy the custom control and the help file into a directory of your liking. Normally this directory is the Windows system subdirectory

The control must be registered into the system registry before it can be used by any application. Using the included REGSVR32.EXE application, type the following command line :

REGSRVR32 <*directory*>\ FNAME.OCX

From within your container application, include the custom control and access the methods and members of the control for your custom Internet application.

NOTE : If you move the control you must reregister the control to take into account the new directory.

Registration

Make checks payable to and mail to :

Edward B. Toupin
448 East Arden Circle
Highlands Ranch, CO 80126

Order Form

NAME : _____
COMPANY : _____
ADDRESS : _____
CITY : _____
STATE : _____ ZIP : _____
COUNTRY : _____

PHONE : _____
FAX :

EMAIL : _____
COMPUSERVE : _____

CONTROL	QTY		UNIT	TOTAL
ARCHIE CLNT v1.00a	_____	$	25.00	_____
CHATTER v1.00a	_____	$	20.00	_____
CONNECT v1.00a	_____	$	55.00	_____
FINGER v2.00a	_____	$	10.00	_____
FTP v2.05a	_____	$	35.00	_____
GOPHER CLNT v1.00a	_____	$	35.00	_____
GETHOST v2.50a	_____	$	10.00	_____
IRC v1.00a	_____	$	40.00	_____
MAIL v2.00a	_____	$	35.00	_____
NEWS v1.00a	_____	$	35.00	_____
SNMP v1.00a	_____	$	250.00	_____
TELNET v1.00a	_____	$	50.00	_____
TIME v2.00a	_____	$	10.00	_____
WAIS v1.00a	_____	$	25.00	_____
WEB CLIENT v1.00a	_____	$	275.00	_____
WEB SERVER v1.00a	_____	$	425.00	_____
WHOIS v1.00a	_____	$	10.00	_____
WSOCK v1.51a	_____	$	35.00	_____
			TOTAL	_____

Feedback

CONTACT

Name :
Company :
Position :
Address :
City : State : Zip :
Country :

Phone :
Fax :
Email :

Name and version of custom control :

Where did you acquire the application ?

Purpose of your Company :

How is this system going to be used :

Type of Computer and Peripherals :

Suggestions for future systems or enhancements to this system :

EXPERIENCE

LEVEL OF INTERNET USER EXPERIENCE : Beg Int Adv

LEVEL OF INTERNET CONNECTIVITY KNOWLEDGE : Beg Int Adv

LEVEL OF INTERNET APPLICATION DEVELOPMENT EXPERIENCE : Beg Int Adv

LEVEL OF CUSTOM CONTROL USER EXPERIENCE : Beg Int Adv

LEVEL OF CUSTOM CONTROL DEVELOPMENT EXPERIENCE : Beg Int Adv

SYSTEM DESCRIPTION

WINDOWS VERSION : Windows NT Windows 95 Windows 3.11 Windows 3.1

32-BIT CONTAINER APPLICATION SUPPORTING VBA OR VB :

INTERNET PROVIDER :

VISUAL C++ 2.0 OR LATER :

MEMORY, HARD DRIVE(S), MODEM (TYPE, BAUD, ETC.), ETC :

DO YOU WANT INFORMATION ON UPDATES EMAILED TO YOU : Y N

DO YOU SUPPORT UUENCODING AND PKZIP 2.X ?

Requirements and Information

The custom control was designed and developed under Windows NT 3.5 as a 32-bit custom control (OCX). To utilize the functionality of the system you will require the following :

Windows NT 3.5 or greater or Windows 95

Windows Sockets v1.1 or compliant socket interface

Connection to the Internet (e.g. SLIP/PPP via Internet provider)

For dial-up services, Remote Access (or equivalent) for communications

A 32-bit container application or development environment

Shareware

This software, hereafter referred to as **the software**, is **NOT** a public domain or free program. It is being distributed as Shareware. The Association of Shareware Professionals (ASP) offers the following definition of Shareware:

> Shareware distribution gives users a chance to try software before buying it. If you try a Shareware program and continue using it, you are expected to register. Individual programs differ on details some request registration while others require it, some specify a maximum trial period. With registration, you get anything from the simple right to continue using the software to an updated program with printed manual.
>
> Copyright laws apply to both Shareware and commercial software and the copyright holder retains all rights, with a few specific exceptions as stated below. Shareware authors are accomplished programmers, just like commercial authors, and the programs are of comparable quality. The main difference is in the method of distribution. The author specifically grants the right to copy and distribute the software, either to all or to a specific group. For example, some authors require written permission before a commercial disk vendor may copy their Shareware.
>
> Shareware is a distribution method, not a type of software. You should find software that suits your needs and pocketbook, whether it's commercial or Shareware. The Shareware system makes fitting your needs easier, because you can try before you buy. And because the overhead is low, prices are low as well. Shareware has the ultimate money-back guarantee -- *if you don't use the product, you don't pay for it.*

Registration licenses you to use the software. Any unregistered use other than trial use to determine if the software meets your needs is a violation of our license agreement and is forbidden.

The registered software license authorizes use of one copy of the software on one machine at a time. If you have multiple computers, either standalone, networked, or multi-user, you need to register one copy of the software for each workstation on which it will be used.

Commercial or government organizations should contact the author for site-licensing or distribution options.

Internet Communications Overview

What is TCP/IP?

TCP/IP (Transmission Control Protocol / Internet Protocol) is a suite of protocols used to allow cooperating computers to share resources across a network. In actuality, the more accurate description of the set of protocols is the *Internet protocol suite*. A few of the protocols of TCP/IP (e.g., IP, TCP, and UDP) provide low level functions needed for many applications. These low level functions provide a means of encapsulating data and sending it out onto the network.

Other higher level protocols are used for performing specific tasks such as transferring files between computers, sending mail, or finding out who is logged in on another computer. These high level protocols are known as *application protocols* because they are specific to an application and reside above the lower level TCP/IP network level protocols. The more important of these protocols and TCP/IP services are:

File Transfer The file transfer protocol (FTP) allows a user on any computer to send or retrieve files from another computer. Security is handled by requiring the user to specify a user name and password acceptable on the other computer.

Remote Login The network terminal protocol (TELNET) allows a user to log in to any other computer on the network that supports TELNET. You start a remote session by specifying a computer to connect to thus allowing you to send your keystrokes to the other computer. When you type you are still communicating with your own computer but the TELNET application makes the remote computer look as though it were your local computer.

Electronic Mail Electronic mail allows you to send messages to users on other computers. When you send mail, the mail software expects to be able to open a connection to the addressee's computer in order to send the mail. In many cases, users have an intermediate mail server that accepts mail for users. A user can then log onto the mail server at their leisure and accept their mail from the server.

Even though many people can perform the above operations from a single computer, that computer will occasionally call on the resources of other computers located on the network. This method of resource sharing has led to the *client/server model* of network services. A server is a system that provides specific services to client systems that reside on a network. The following is a list of typical servers available within the framework of TCP/IP on many of todays networks :

File Systems File servers allow computers on the network to be more closely coupled to remote applications than otherwise available through standard FTP. A network file server provides a means of allowing the client computers to view the servers disk as though it were local to the client computer.

Printing Print servers allows you to access printers on other computers as if they were directly attached to yours.

Remote Execution This method of resource sharing allows you to request that a particular program be executed on a different computer on the network. There are two primary types of remote execution : *command basis* and *remote procedure call.* The command based execution allows you to request that a specific command or set of commands execute on a remote computer. Remote procedure calls allow a program to call a subroutine that resides on a remote computer.

Name Servers Name servers provide a means of associating host names to host addresses. In large network installations there are a number of different collections of names that have to be managed including users, names and network addresses for computers, and accounts. Computers requesting access to other computers on a network use the name server protocol used to keep track of host names and addresses on the Internet.

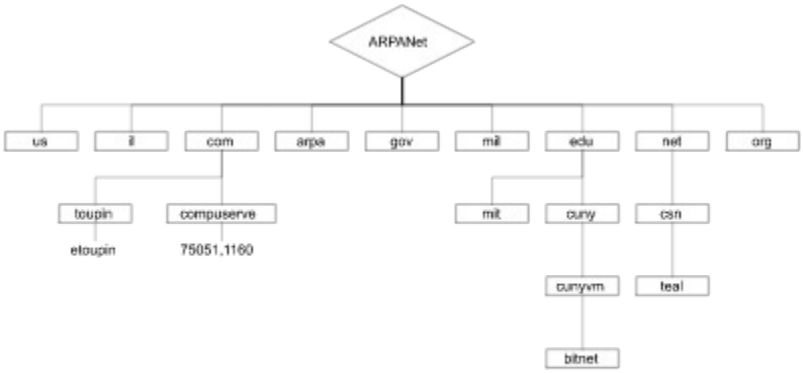
Terminal Servers Terminal servers are generally small computers that use the TELNET protocol and handle incoming calls and accesses from remote TELNET clients.

Network Based Window Systems Network window systems allow applications to use a display on a remote computer as opposed to having to execute high-performance graphics programs on a graphics screen directly attached to the computer. Network window systems allows you to distribute jobs out to remote systems but manage a graphically-based user interface.

What is the Internet?

The Internet is not just one network but is instead a collection of networks including Arpanet, NSFnet, regional networks such as NYsernet, local networks at a number of University and research institutions, and a number of military networks. The term Internet applies to this entire set of networks that constitute the world-wide communications paths available today.

The subset of the available networks that are managed by the Department of Defense is referred to as the Defense Data Network (DDN) which includes research-oriented networks, and specific military networks. All of these networks are connected to each other allowing users to send messages from one network to another quite transparently.

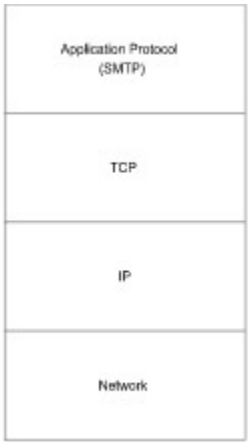


One portion of the Internet -- the ARPANet Gateway

TCP/IP Protocols and Communications

The TCP/IP suite of protocols is known as a *layered* set of protocols. Essentially, a layered set of protocols involves the hierarchical association of functionality for communications between applications on the Internet. To understand what this hierarchy consists of let us take a look at the process of sending email across the Internet.

The first level of the protocol for email is the Simple Mail Transfer Protocol (SMTP). This protocol defines a set of commands used for sending mail from one machine to another. These commands allow the identification of the sender and the recipient as well as the body of the message. This protocol, however, is not all that is required to simply send the information to another node. SMTP assumes that a mechanism exists that allows the SMTP information to be sent reliably between the two computers. Being a standard *application protocol* SMTP merely defines a common set of commands and messages. This protocol resides on top of a communications protocol like TCP and IP.



The basic communications stack

The TCP communications protocol is responsible for guaranteeing reliable transfer of information between the machines. TCP monitors the information that is being sent and retransmits any information that was not received at the other end. TCP also manages the *segmentation* of messages. If messages are too large for a particular packet, or *datagram*, TCP segments the information into several datagrams and makes sure that they all arrive in order and intact.

TCP resides above and calls on the services provided by IP. IP, which provides the basic service of routing datagrams to their destination, allows you to communicate in the protocol specific to the available network. The layered approach allows applications such as mail to call on the services of the layers immediately below.

Routing

TCP/IP assumes that there are a large number of independent networks connected together by gateways allowing users access to resources on any of the networks that make up the Internet. Because of these numerous gateways, datagrams usually pass through several networks before reaching their destination. In order to accomplishing the *routing* through the numerous network, each gateway must know the path to each consecutive network for the ultimate destination. As far as the user is concerned, all they need to know in order to access a remote system is the *IP Address* of the remote system. This address looks like 199.41.151.74 and is actually a 32-bit representation of the remote nodes address, however, it is normally represent with the 4 decimal numbers each representing 8 bits of the address. The address structure can provide you with some information about how to get to the remote system. For example, 199.41 is a network number assigned by a central authority to an organization. This organization uses the following value octet to indicate which of the organizations networks is involved in the communication. 199.41.151 could be the primary network for a research facility of the

organization. The final value allows for up to 254 systems on each network.

As you probably have seen we refer to systems by name rather than by a systems Internet address. When a name is specified the network communications software locates the name in a central database. The database, containing all network names and associated addresses, returns the corresponding Internet address of the computers name.

Communications

TCP/IP is built on a *connectionless* technology where information is transferred as a sequence of *datagrams*. *Datagrams* are a collection of information that is sent as a single message with each datagram being sent through the network individually and independently of the others. Even for those conversations that are established and continue for some time, information from those connections is broken up into datagrams, and those datagrams are treated by the network as completely separate. The protocols involved in the communication will break up the information into separate datagrams and send them to the other end. Once they arrive at the other end they will be put back together into the original message.

One point to note is that while those datagrams are in transit the network is not aware that there is any relation between the intransit datagrams. In certain circumstances it is possible for datagram 10 to arrive before datagram 9. In another circumstance it is also possible that an error will occur on the network thus keeping a datagram from arriving. In this situation that datagram would have to be resent.

The TCP Layer

The top layer of the two layer TCP/IP stack is TCP (Transmission Control Protocol) and is responsible for breaking up messages into datagrams, reassembling them at the other end, resending anything that does not arrive at the remote end of a connection, and reassembling segmented messages. The IP (Internet Protocol) handles all of the routing of individual datagrams.

On a small network, IP is not really utilized to its fullest, however, on the Internet, simply getting a datagram to its destination can be a complex job. A remote connection sometimes requires that a datagram route through several networks and network media. Managing the routes to all of the destinations and handling incompatibilities among different transport media turns out to be a complex job. Note that the interface between TCP and IP is fairly simple where TCP simply hands IP a datagram with a destination address associated with it.

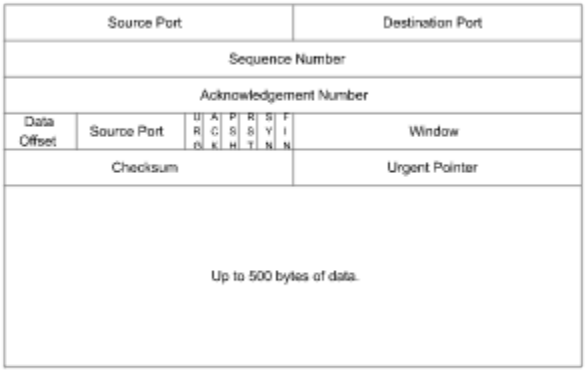
If you have multiple connections to a single host, TCP must has to know which connection is associated with which inbound datagram. The task of managing datagram-connection associations is known as *demultiplexing*. The information required to perform the demultiplexing is located in a series of *headers* which are simply a few extra bytes preceding the data of a datagram.

Lets say you want to send a file to another computer. This file starts out as a stream of bytes that is submitted to TCP for submission onto the network. In order to properly handle the data on the network, TCP segments the stream into smaller pieces of information, or datagrams.

On the front of each of the datagrams TCP attaches a header containing at least 20 bytes including a source and destination *port number* and a *sequence number.* The port numbers are used to keep track of different conversations. Suppose 3 different people are transferring files. Your TCP might allocate port numbers 1000, 1001, and 1002 to these transfers. When you are sending a datagram, this becomes the *source port* number since you are the source of the datagram while TCP on the receiving computer has assigned a port number of its own for the conversation. The local TCP has to know the port number used by the other end as well to place it into the *destination port* field.

Every datagram has a unique sequence number that is used so that the other end can make sure that the datagrams are received in the correct order and that no datagrams are missed. The computation for determining the sequence number is not based on the numeric order of the datagram but on the number of bytes being sent for the entire stream of data. For instance, if there are 50 octets of data in each datagram, the first datagram might be numbered 0, the second 50, the next 100, the next 150, etc.

Within the header of each datagram is a calculated checksum value. This is a numeric representation of the data contained in the datagram and is computed by adding up the values of all bytes in the datagram. The receiving end computes the checksum again and, if the two computed and transmitted checksums disagree, an error occurred in the transmission and that datagram is discarded.



With regards to other fields of the header, these are involved with managing the connection between the remote and local nodes. For instance, the *acknowledgment number* is used to inform the remote TCP to respond with an acknowledgment of receipt of the specific datagram. When the remote TCP responds with an acknowledgment of receipt that field contains a numeric response value. For instance, sending a packet with an acknowledgment number of 500 indicates that you have received all the data up to byte number 500. If the sender does not receive an acknowledgment within a reasonable amount of time, the data is sent again to the recipient.

The *window* field indicates the number of bytes of data a machine is ready to accept and is used to control how much data can be in transit at any one time. It is not efficient to wait for each datagram to be acknowledged before sending the next one nor can you just send data otherwise a high-speed machine might overrun the capacity of a slower machine. As the recipient receives data, the value in *window* decreases until it reaches 0. At 0 the sender has to stop sending until the recipient processes the data it currently has received. As more processing space becomes available the window value increases indicating that it is ready to accept additional data.

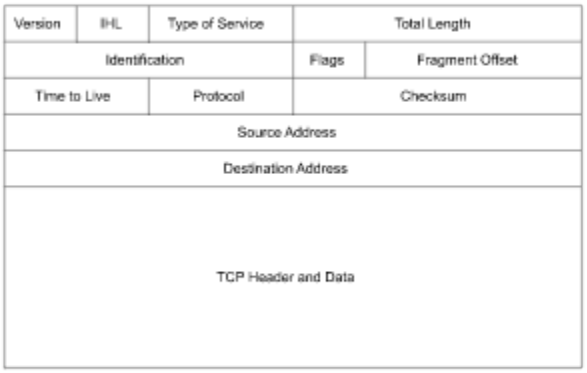
The *urgent* field allows one end to tell the other to skip ahead in its processing to a particular byte. This type of functionality is often useful in handling asynchronous events such as when you type a control character or other command that interrupts output.

The IP Layer

Once TCP has completed processing of the datagrams, it submits each datagram to the IP layer. With each of these datagrams is the IP address of the computer for which the datagram is destined. The IP layer is not concerned about the contents of the datagram but is instead concerned about determining the best route for the datagram to reach the remote computer.

To properly route the datagram and allow intermediate systems to forward the datagram, IP adds its own header to the datagram. This header contains the source and destination IP address, the *protocol identifier*, and another checksum. The source address is simply the address of the originating machine while the destination address is the address of the recipient machine. The informs the IP layer at the recipient to send the datagram to the appropriate protocol handler. Even though much of the common IP traffic uses TCP the stack allows other protocols to use IP so you must inform IP of the destination protocol handler.

As with TCP, the checksum allows IP to determine if the header was damaged during it travel through the network. IP maintains its own checksum in order to verify that its header did not get damaged in transit.



IP Header and Data

Some of the other fields of the header contain information and flags to keep track of the datagram if it should need to be split up. In many cases, this additional segmentation can occur if the size of a datagram exceeds the maximum size allowable on a given network.

The *time to live* field is a number that is decremented whenever the datagram passes through a network system. When this values is decremented down to 0 the datagram is discarded. This field has been implemented so that the datagram does not continue to travel aimlessly through a system should a loop condition occur somewhere in the network.

The Network Layer

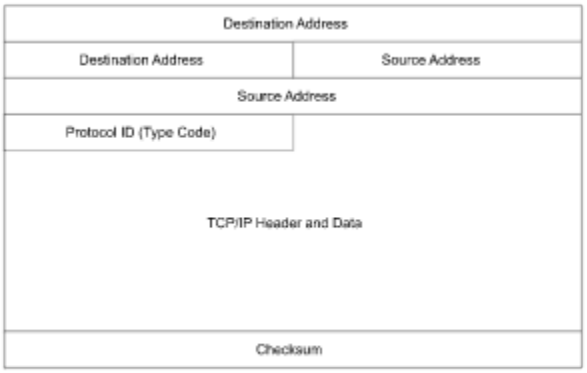
Lets examine the bottom layer of the communications stack, the *network layer*, and how information travels across one of the more common network architectures -- the Ethernet. This layer is important in that, regardless of the network medium, it too maintains a header that is applied to a TCP/IP datagram above and beyond the TCP and IP headers.

In addition to the IP addresses discussed in the IP layer we also have to contend with Ethernet addresses. Ethernet addresses are maintained by a 48 bit (6 byte) address that is intended to keep multiple machines from obtaining the same address across the board. Ethernet equipment vendors have to register with a central address management group to make sure that the numbers they assign do not overlap with any other manufacturer.

The Ethernet uses a *multiple access*, or shared, medium for communications in that every machine on a local network sees every packet on that local network. As with TCP/IP a header is added so that the Ethernet frame is accepted by the correct destination node. The Ethernet packet contains a 14-byte header that includes the source and destination Ethernet address, and a protocol identifier. The machines on the network only accept those packets with the appropriate Ethernet address in the destination field. There is no direct correlation between the Ethernet address and the IP address so there exists a table on the local network that relates Ethernet addresses with IP addresses.

The protocol identifier allows for several different protocol families to be used on the same network including TCP/IP, DECnet, etc. simultaneously. Each of the different types of protocols maintain a different protocol identifier so that the Ethernet stack can properly route the Ethernet frame to the proper protocol stack.

The Ethernet physical layer drivers compute a checksum of the entire packet, just as the TCP and IP layers calculate their checksums. In the case of Ethernet checksums, the checksum is appended to the end of the Ethernet frame. If the computed and original checksums do not agree at the receiving end, the packet is merely discarded and no request for a resend is made. This request for resend is the responsibility of the higher level protocols.

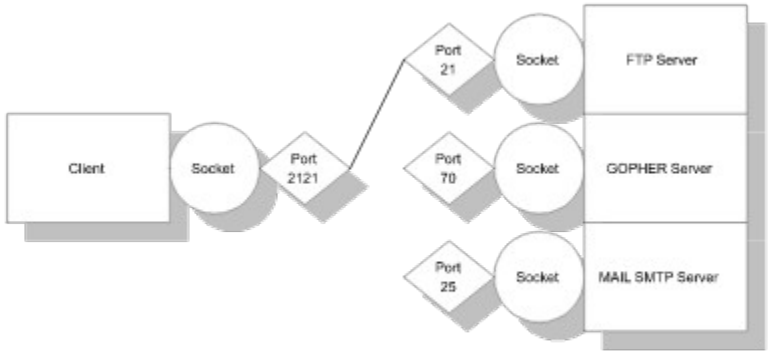


Ethernet Header and Data

The Application Layer

Over a network, there are methods available that allow you to open a connection to a specified computer, log into it, request files or resources, and control the resources or the transmission of a file. All of the aforementioned operations are handled by *application protocols*. These application protocols reside above and utilize the resources of TCP/IP.

Lets assume that you want to send a file to a machine whose IP address is 199.117.41.151. In order to send the file you must not only connect to the node at the IP address but you must also connect to the FTP (File Transfer Protocol) server on that node. Network applications on network nodes perform specialized tasks so connecting to a nodes address is not enough to handle all of the functionality required on the network. In order to specify that you wish to communicate with the FTP server you must connect to the specific *port* for the server. This port is the port that TCP uses in its header to keep track of individual conversations. Specific port numbers are assigned to the applications, or servers, that wait for requests from client applications. If you want to send a file, you will open a port with a random number on your end which connects to port number 21 on the remote machine -- the official port number for the FTP server.



Clients and Servers connect over a common service port.

Once we have established this connection using TCP and IP we can continue in establishing communications between the two applications. This next step of communications is in establishing an agreement on what set of commands the application will understand, and the format in which they are to be sent. For example, the mail protocol operates as follows :

A mail application establishes a connection with a mail server.

The mail application sends the host machine's name, the sender of the message, and the recipients of the message.

The mail application sends a command to initiate the message.

The mail server starts accepting the message.

The mail application sends the text of the message.

The mail application sends and end-of-message character.

The mail application and the mail server end the communications and close the connection.

The file transfer protocol, on the other hand, involves two separate connections : one for data and one for commands. Communications begins just like mail, however, once the command to transfer data is sent, a second connection is opened for the data itself. It is possible to send data on the same connection, however, file transfers

often take a long period of time and the originators of FTP decided it was best to allow users to issue commands while the transfer is going on (e.g., abort transfer, etc.).

Other Protocols

Recall that TCP is responsible for taking a message stream and segmenting it into datagrams for transmission on the network. In some situations however you may need to send a message that fits wholly into one datagram. One such example is that of a name lookup where an IP address is located for a given host name. In this situation a user attempts to make a connection to another system by specifying the system by name. The users system has to translate the name into an IP address before the connection can be transmitted. The user's system will send a query to one of the systems that maintains the name/address database and then wait for a response. This response will consist of one datagram therefore the full utilization of TCP would be overkill. In this situation the application could utilize UDP (User Datagram Protocol). UDP is designed specifically for situations where sequencing and reassembly of datagrams is not necessary. When sending data, UDP prepends a header to the data and submits it to the IP layer. Unlike TCP, UDP does not segment data into multiple datagrams, nor does it track what it has sent in case of a resend. The one similarity between UDP and TCP is that it does provide for port number assignment information for proper routing.

Another alternative protocol is ICMP (Internet Control Message Protocol) which is primarily used for error messages, and other messages intended for the TCP/IP software itself. ICMP is used for error messages sent back from remote hosts or intermediate systems (e.g., Host Unreachable, Network Down, etc.) and can also be used to find out information about the network. ICMP is similar to UDP in that it manages information in a single datagram however it is even simpler than UDP since it does not rely on port numbers nor contain port numbers in its header. Since all ICMP messages are interpreted by the network software itself, no port numbers are needed to route messages to any user application.

The Domain Name System

Users generally prefer to refer to computers on a network by name than by address. TCP/IP network software requires a 32-bit Internet address in order to establish a connection between to nodes or send a datagram to a remote node. To associate the user required node names and the node addresses these exists a database that allows network software to look up a host name and find the corresponding address.

For a small network this operation is simple since each system on the network would maintain a file containing all other systems names and addresses on the network. For the Internet, things are not that simple so to remedy this situation these name / address files are replaced by *name servers* that manage host names and their corresponding addresses. The host names stored on the name servers follow a hierarchical structure such as PEGASUS.TOUPIN.COM. In order to find its Internet address, you might potentially have to consult 3 different servers. First, you would ask a central, *root*, server of the location of the COM server. COM is a server that keeps track of for-profit organizations on the Internet. The root server would give you the names and Internet addresses of several servers for COM.

You would then ask COM for the server for TOUPIN. The COM server would give you names and Internet addresses of several servers for TOUPIN. Finally you would ask TOUPIN where the server for PEGASUS is located. The final result would be the Internet address for PEGASUS.TOUPIN.COM. Each of these levels is referred to as a *domain* while the entire name, PEGASUS.TOUPIN.COM , is called a *domain name*.

A few things to note. First, the root name servers also happen to be the name servers for the top-level domains such as COM, EDU, ORG, etc. therefore a single query to a root server will get you to TOUPIN. Second, *address resolution* software caches prior look-ups therefore once you look up a name at PEGASUS.TOUPIN.COM the software remembers where to find servers for PEGASUS.TOUPIN.COM, TOUPIN.COM, and COM.

Routing

The IP layer is responsible for the task of passing datagrams to a destination indicated by the destination address. This task of determining the best path to get a datagram to its destination is referred to as *routing*.

Routing is based entirely upon the network number of the destination address. For each computer there exists a table of network numbers. For each network number there exists a gateway that is used to access that network. The gateway does not have to be physically connected to the network, however, it is the best path to follow in order to access the desired network for the destination address.

Before sending a datagram, the IP of the originating computer checks to see if the destination address is on the computers own local network. If on the local network, the datagram can be sent directly to the destination address. If the destination node is not on the local network the system attempts to locate and an entry for the network on which the destination address is located. The IP submits the datagram on the network to the gateway listed in the entry associated with the destination address entry.

For the gateways along the path each gateway determines the next path over which the datagram should pass. This operation occurs on the Internet until a gateway to the destination nodes network is found or until the *time to live* field of the frame reaches 0.

Datagrams

TCP/IP was originally designed for use with many different networks, however, not all network vendors have agreed to a maximum packet size for their particular network implementations. Ethernet packets can be 1500 bytes and Arpanet packets have a maximum of around 1000 bytes while some of the faster network allow larger packet sizes.

Settling on the smallest possible packet size seems legitimate, however, this would introduce performance problems for end to end communications. To pass over many different networks we need to reach some sort of compromise. The transfer of large pieces of information is quite efficient with large packets, however, since the maximum packet sizes vary, we cannot establish a standard size. To manage these variable packet sizes TCP *negotiates* datagram size when a TCP connection first opens with a remote computer.

When negotiating packet sizes, TCP segments datagrams in order to accommodate the network whose maximum packet size is the smallest. The IP header contains fields indicating the a datagram has been split and enough information to let the pieces be put back together. If one gateway connects an Ethernet to the Arpanet, the Ethernet based TCP layer must be prepared to take 1500-byte Ethernet packets and segment them into packets that will fit onto the Arpanet. The destination node must then *reassemble* the segmented packets into the original message.

Support

Support for the Connectivity Custom Controls can be obtained by emailing the problem report below to Edward B. Toupin at :

Internet	:	etoupin@toupin.com
Compuserve	:	75051,1160

An online problem report can be filled out at :

WWW	:	http://www.toupin.com/~etoupin/cccprob.html
		http://www.toupin.com/~etoupin/welcome.html

Email is checked regularly so a response will be emailed as soon as the mail is received and examined. If you are a registered user, the problem will be investigated and a problem resolution will be emailed back to you. If the problem exists with the custom control a new custom control will be emailed to you otherwise an explanation on how to remedy the problem will be sent.

Problem Report

Name :

Company :

Area Code & Phone :

CompuServe ID :

Internet Address :

Problem Title :

Product and Version :

Operating System:

32bit Container App :

Network type :

Network protocol : version :

Network cabling :

LAN or WAN :

Modem : baud :

Service Provider :

What type of application were you developing :

How was the control being used :

Detailed problem description :

Steps to reproduce the problem (please explain step by step):