

Volume

5



Wayside Software

Division of Wayside Marketing, Inc.

Netsocket/400

WAYSIDE MARKETING INC.

Netsocket/400 - Version 5.0.2

Wayside Software

Division of Wayside Marketing, Inc.
121 Grace Circle
Marlboro, MA 01752
Phone: 508-485-2203
Fax: 508-481-9015
www.wayssidemarketing.com



Table of Contents

OVERVIEW	1
PREREQUISITES	2
INSTALLATION STEPS.....	3
RUNTIME REQUIREMENTS.....	4
USING SSL FEATURE	5
NOTES ON SSL COMPATABILITY	6
BASIC ILE CONCEPTS.....	7
METHODS FOR CREATING SERVER PROGRAMS	8
NETSOCKET/400 PROCEDURES	13
PROGRAMMING EXAMPLES	34
RPG ILE EXAMPLES.....	36
COBOL ILE EXAMPLES.....	71
C ILE EXAMPLES.....	88
PROCEDURE PROTOTYPES	97
STATUS CODES AND MESSAGES.....	98
COMMUNICATING WITH OTHER PLATFORMS	108
DEFAULT HEXADECIMAL CONVERSION CHART	109

OVERVIEW

NETSOCKET/400 is a TCP/IP Programmer's Toolkit for the iSeries (AS/400) that was developed to create TCP/IP applications without having to learn to create socket applications in the C programming language.

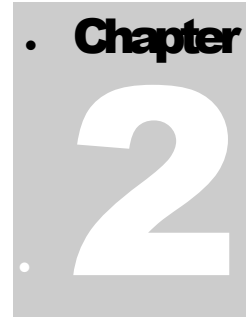
NETSOCKET/400 provides a simple, intuitive development application-programming interface (API). It includes a module and service program that users can bind into their RPGLE, CBLLE, or C applications to allocate and communicate with standard connection-oriented stream sockets. Users can also send and receive data over any IP network. In addition, the same module allows client, server, and daemon applications to be built simultaneously.

TCP/IP protocols are usually accessed via an application-programming interface known as sockets. IBM's implementation of sockets on the AS/400 is compatible with the Berkeley Software Distributions (BSD) 4.3 Sockets.

NETSOCKET/400 expects that TCP/IP is properly setup on the AS/400. For information on configuring TCP/IP, reference IBM manual on TCP/IP Configuration.

Some Features of NETSOCKET/400 include:

- Send and receive data over any IP Network
- Secure Socket Capabilities
- Allocate and communicate with stream-oriented sockets
- Design and build client and/or server applications
- Create clients, servers and daemons
- Daemon automatically monitors and manages the number of available server jobs
- Record transactions down to the millisecond
- Support for Prototyped calls
- Service from 1 to 512 clients simultaneously
- Pre-start as many server jobs as you want
- Complete ASCII and EBCDIC conversions
- Receive detailed status messages
- Trace all activities



PREREQUISITES

The AS/400 operating system must be at Version 5 Release 3 or higher. In addition, if you plan on using the SSL functionality of Netsocket you will require one of the cryptographic provider products (5769AC1[40 Bit], 5769AC2[56 Bit], 5769AC3[128 Bit]) and OS/400 option 34 - Digital Certificate Manager.

The TCP/IP Connectivity Utilities/400 licensed program. This software is shipped free with V5R3 and higher of OS/400.

RPGLE, CBLLE, or C must be used to access NETSOCKET/400

INSTALLATION STEPS

- Make certain the zip file has unzipped a file called "TCPIPxxx" (xxx is the version).
- Create a save file on your AS/400 in the QGPL library using the following command
CRTSAVF FILE(QGPL/TCPIPxxx)
- Create the Netsocket/400 library on your AS/400 in the QSYS library using the following command:
CRTLIB LIB(TCPIPxxx)
- Establish a connection to the AS/400 using FTP by going to the DOS prompt screen and typing in the following command: **ftp "IP address of your AS/400"**
- You will be prompted for your AS/400 user id and password as part of the connection process. Once connected you will get a "logged on" message on your screen.
- Use the FTP command **"cd QGPL"** to change your current AS/400 library to QGPL.
- Use the FTP command **"binary"** to put FTP into binary transfer mode.
- Use the FTP command **"put TCPIPxxx"** to send the save file to the AS/400.
- Use the FTP command **"quit"** after the file is transferred to exit FTP.
- To install Netsocket/400 on your AS/400 type the following command:
**RSTOBJ OBJ(*ALL) SAVLIB(TCPIPxxx) DEV(*SAVF)
SAVF(QGPL/TCPIPxxx) ALWOBJDIF(*ALL) RSTLIB(TCPIPxxx)**

Netsocket/400 is now installed!

- Use the following command to load the password for your system:
**CHGDTAARA DTAARA(TCPIPxxx/TCP_ENABLE)
VALUE(enter value provided)**

Note: Do not enter the dashes from the password provided into the data area. They are only there for readability purposes only .

RUNTIME REQUIREMENTS

Various objects will be installed that NETSOCKET/400 must be able to locate. By default, they are placed in library *TCPIP502*. They do not have to be kept in this library. They do, however, have to be in the job's library list. Below is a list of objects that will be installed.

Name	Type	Purpose
TCPIP502	*MODULE	The NETSOCKET/400 module. Either this module or the service program below should be bound into your ILE applications. It does not have to be distributed as a *MODULE if bound into your program.
TCPIP502	*SRVPGM	The NETSOCKET/400 service program. Either this service program or the module above should be bound into your ILE applications. If the service program is used within your applications it will have to be distributed along with your application programs.
TCP_ENABLE	*DTAARA	Contains a password that enables the toolkit to be used on a given AS/400. It must be in the library list.
TCP_SOURCE	*FILE	Members that contain source code for examples and TCPDMNLOG.
TCPDMNLOG	*FILE	File that stores daemon activity. It must be in the library list.
TCP_PRINTF	*FILE	Trace output is directed through this print file. It must be in the library list if the trace will be enabled.
TCP_CNTRL	*FILE	This file is used to help facilitate communication between, and management of, active Daemon and Server jobs.



USING SSL FEATURE

Make sure you have one of the Cryptographic provider products installed on your AS/400. (5769AC1[40 Bit], 5769AC2[56 Bit], 5769AC3[128 Bit])

Install OS/400 option 34 - Digital Certificate Manager

You need to know the path & password for the keyringfile or key database file you want to use.

The *SYSTEM certificate store path is
"/QIBM/USERDATA/ICSS/CERT/SERVER/DEFAULT.KDB"

Make sure your HTTP ADMIN server is running on your AS/400. If you need to configure the certificate store start a WEB browser and enter "http://(your AS/400 address here):2001 If your HTTP admin is secure use "https://(your AS/400 address here):2010

The following documents and links are available for more information on configuring AS/400 SSL:

HTTP Server for AS/400 Webmaster's Guide book, GC41-5434 Web Programming Guide available from URL: <http://www.ics.raleigh.ibm.com>

HTTP Server for AS/400 Quick Beginnings book, GC41-5433 *TCP/IP Configuration and Reference* book, SC41-5420 See the following URL for information related to securely configuring the HTTP server: <http://www.as400.ibm.com/tstudio/workshop/webbuild.htm>

NOTES ON SSL COMPATABILITY

AS/400 supports SSL v3 and SSL v2

The default cipher suite list for the Internet Connection Secure Server (US) 5769AC3 128 bit product in preference order is as follows:

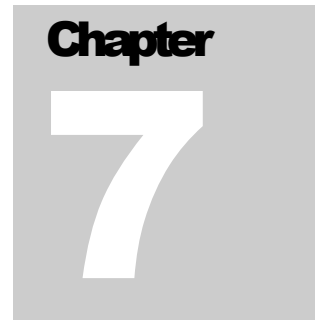
- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_DES_CBC_MD5
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_MD5
- SSL_RSA_WITH_RC2_CBC_128_MD5
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5

The default cipher suite list for the Internet Connection Secure Server (US) 5769AC2 56 bit product in preference order is as follows:

- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_DES_CBC_MD5
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5

The default cipher suite list for the Internet Connection Secure Server (International) 5769AC1 40 bit product in preference order is as follows:

- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5



BASIC ILE CONCEPTS

This chapter is provided to give a simple overview of the integrated Language Environment (ILE). If you need more information on the subject, please reference the IBM publication SC41-3606 entitled ILE Concepts. Wayside Marketing's technical support cannot shoulder the responsibility of teaching the end user these concepts. We hope you understand.

NETSOCKET/400 is a module and service program developed for the Integrated Language Environment (ILE). ILE allows single programs to be written with multiple programming languages. It also allows for the storage of commonly used procedures into libraries called modules.

To bind multiple modules into one application program, each source member must be compiled into a separate module with the CRTRPGMOD, CRTCLMOD, CRTCPGM, or CRTCLMOD command. Once each module exists they are bound together with the CRTCPGM command. If at any time one of the modules is changed, the whole program does not have to be recreated. The UPDTPGM command can be used to update one or more modules within the program.

External procedures are called from ILE RPG with the call bound procedure (CALLB), or call prototyped procedure (CALLP) statements. If using the CALLP method you will need to use the NETSOCKET/400 prototypes provided for your programming language. All the prototypes are stored in the source file TCP_SOURCE provided within the NETSOCKET/400 library. "LINKAGE TYPE IS PRC" is needed with COBOL/400. See the example source for specifics.

All ILE CL programs pass numeric values as decimal(15,5). For this reason, ILE CL is not directly supported for use with NETSOCKET/400. This would require that all numeric values passed to and from the toolkit be decimal(15,5). This was not acceptable. An intermediate module created with ILE RPG, ILE COBOL or ILE C would be required to accept the values from ILE CL and pass them correctly to the toolkit. It would also return the appropriate information back to the ILE CL module.

METHODS FOR CREATING SERVER PROGRAMS

NETSOCKET/400 provides three different methods for creating server programs on the AS/400. Each method has their pros and cons. In this section we will explain each option and how to best utilize them in your organization.

Option 1 - Using the Daemon approach

This option requires that you use the NETSOCKET/400 procedure `TCP_DAEMON` to create a Daemon program that the sole purpose is to detect incoming client connections and then pass that connection to a different server program (usually running in batch). The `TCP_SERVER` procedure would then be used to accept that connection. The daemon program would then go on to listen for the next client connection. With this approach each connecting client has their own server program handling all their needs.

Because of this the daemon approach is best suited for high volume applications that require a fast response time. This is typically the approach used when the client is running interactively with some user at the remote location driving the process. The downside to using this approach is the very fact that you have a separate program servicing each client. If you have the possibility of a large number of simultaneous client connections you could have a large number of programs running on your system to service them all.

The daemon procedure will monitor the number of currently active and available server programs at its disposal. If the minimum servers parameter has a non-zero value, the daemon procedure will startup that many server jobs when the first client connection is detected. From that point on the procedure will attempt to maintain that level of available server jobs without exceeding the maximum server value provided. The method that is used to logically link the daemon with the server jobs it manages is by way of the program ID. This ID is used for the job name of any submitted job from the daemon procedure. The ID is basically a base 41 representation of the address and port being monitored by the daemon procedure. Communication between the daemon procedure and the server jobs is accomplished using the `TCP_CNTRL` file. The `TCP_PGID` procedure is provided so that if you want to pre-start the server jobs yourself, you can submit these jobs with the proper program ID for the job name.

Option 2 - Using the multiplexing I/O approach

This option requires that you use the NETSOCKET/400 procedure `TCP_SERVER` with the address and host port parameters set to the address and host port on the system you want the server to start listening for incoming connections on. The `TCP_SERVER` procedure will then perform the task that the `TCP_DAEMON` procedure performed in the previous method. In addition, to run in multiplexing mode the maximum clients parameter on the `TCP_SERVER` procedure will have to be loaded with a value greater than one. Due to the fact that with this method the server program is servicing the needs of more than one client, you have to specify the maximum clients you want this server program to handle. All other client connections beyond the maximum value you specify will be rejected. A single server program will be allowed to manage up to 512 client connections

This approach is best suited for low-volume transactions with a large number of clients being serviced. Because the clients are all being serviced by a single server program all the inbound transactions and connection requests are basically queued on a first come first serve basis. Because of this fact this method is best suited to applications where the remote program is not driven by an interactive process because, based on the current depth in the queue, a user could wait a significant amount of time for a reply.

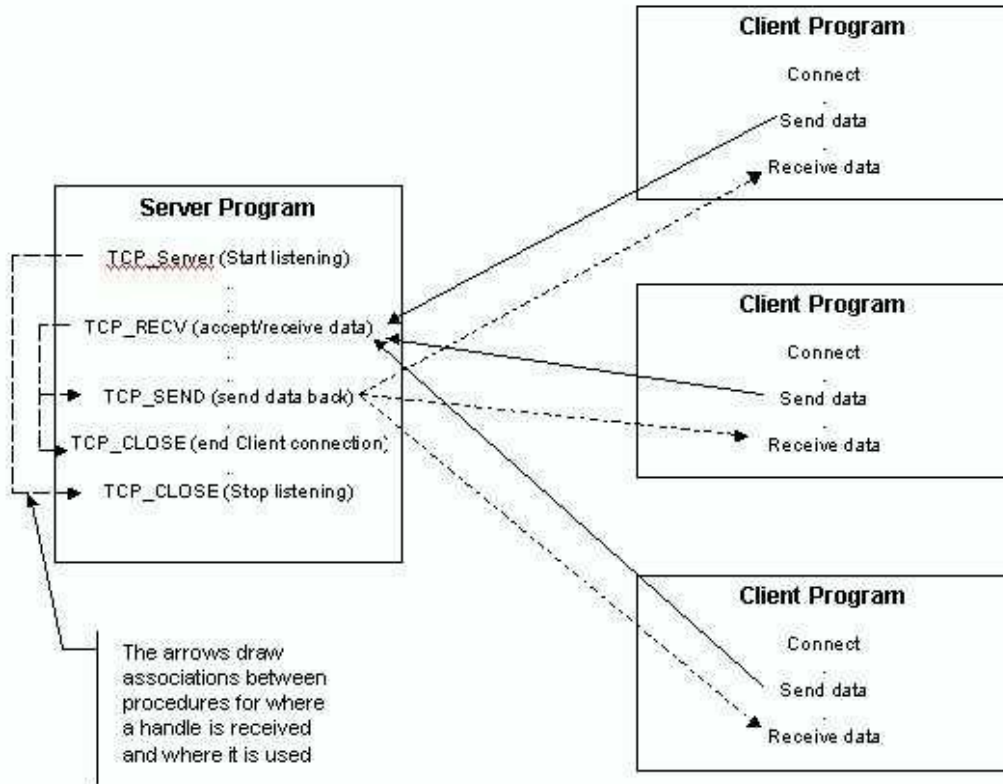
Using this method the `TCP_RECV` procedure plays a very significant role in processing and receiving incoming connection requests. The `TCP_SERVER` procedure is first used to identify and set up a listening connection on your system, which waits for all incoming client connections. But the handle that is returned is not used for communicating with any client connections like the other server methods.

The handle returned by the server is simply the handle of the socket listening for incoming connections. You should use this handle to close the socket just before your server program ends processing. After the `TCP_SERVER` procedure, the `TCP_RECV` routine would be the very first procedure you would use to check for incoming client connections and receive data from that connection. Four parameters of the `TCP_RECV` procedure will then be populated. The handle will be loaded with the handle of the client connection you are receiving data from. The data and length parameters will be populated as specified in the parameter descriptions. And the IP address/host port of the connecting client will be loaded for purposes of further identifying these client connections. You can then use the handle provided to respond to the client using the `TCP_SEND` procedure or immediately receive more data from this or other clients using the `TCP_RECV` procedure again.

It is your responsibility to keep track of all the different handles provided to you so you can later close them down as desired. Typically your application will receive a text message indicating the desire of the remote program to close a connection. Additionally, you should monitor for the fact that the remote connection closed their socket. In either case you should immediately close the socket using the current handle with the `TCP_CLOSE` procedure.

See the diagram below for a further understanding of how the multiplexing server process works.

Multiplexing I/O server diagram



Option 3 – Single Client connection approach

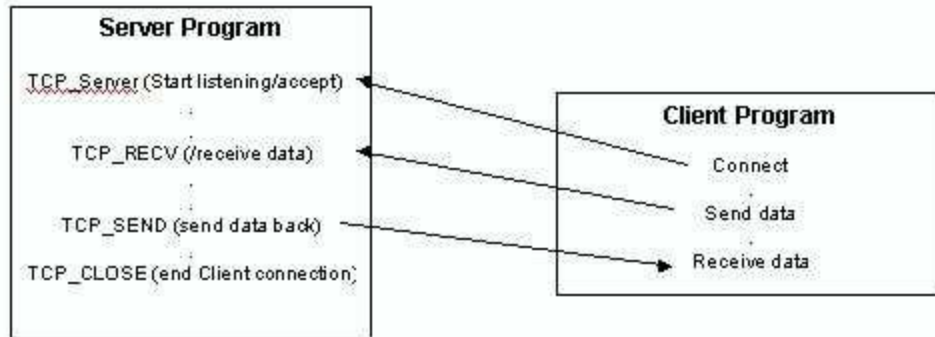
This option requires that you use the NETSOCKET/400 procedure TCP_SERVER with the address and host port parameters set to the address and host port on the system you want the server to start listening for an incoming connection. To run in single client mode the maximum clients parameter on the TCP_SERVER procedure will have to be loaded with a value of one or zero.

This approach is best suited for high volume applications where only a single client connection is required. This is typically the case in peer to peer connections where you want to communicate between two host applications.

Using this method the TCP_SERVER procedure is first used to identify and set up a listening connection on your system which waits for the incoming client connection. When the client connects, control is returned back to your application program with the handle needed to respond to the client using the TCP_SEND or TCP_RECV procedures. When communication is no longer required, you should close the socket using the handle and the TCP_CLOSE procedure.

See the diagram below for a further understanding of how the single client server process works.

Single-client connection server diagram





NETSOCKET/400 PROCEDURES

This chapter provides a description of the procedures available to your applications. They are available for use from within the NETSOCKET/400 module. Each procedures description provides a list of the parameters along with an explanation of the purpose and allowed values for these parameters.

TCP_CLIENT Initiate a connection with a remote host

The TCP_CLIENT procedure will attempt to establish a communications link with a remote host. If this procedure is successful, a handle will be passed back which is used for working with this link. This handle will be used with the TCP_SEND and TCP_RECV procedures to send and receive data via this conversation. Once the application is finished communicating over this link, it should be closed with the TCP_CLOSE procedure to ensure that all system resources are freed. If this procedure is unsuccessful, the error variables will be set.

Secured socket mode is automatically initiated if the last four (4) parameters in this specification are used. Otherwise the client connection and all subsequent send/receive operations will be in non-secured mode.

The TCP_CLIENT procedure expects the following parameters:

Parameters	Attributes	Description
Host	Character (up to 256)	The host to establish communication with. This field can either contain a dotted decimal IP address such as "128.1.1.1", or a host name such as "host1.esd.com".
Host port	decimal(5)	Enter the host port where the remote server/daemon is listening for inbound connection attempts.
Handle	decimal(10)	If the TCP_CLIENT procedure is successful, it will then populate this variable with a handle to keep track of the established communications link. A handle with a value of 0 may be valid. To determine if a handle is valid, check the error variables. This handle cannot be passed to other programs. Other modules within the same program may however, use it. This is provided that your application makes these variables publicly accessible to the other modules. This handle should be closed with the TCP_CLOSE procedure when it is not needed any longer or upon application termination.
KeyRingFile	character(up to 256)	Path to the key ring file to be used for this job's SSL processing. The path must be a fully qualified integrated file system (IFS) file name.
KeyRingPassword	Character (up to 256)	Password for the key ring file named in the <i>keyringFileName</i> field. If this parameter's value is equal to BLANKS, then SSL support will attempt to extract a password from a key-ring password file.
SSLsocket	character(1)	A flag "Y" or "N" will be returned to indicate if the SSL connect was successful.
TimeOutWait	decimal(5)	The number of seconds to wait before giving up on the attempt at a SSL connection. If this parameter is omitted, 120 seconds is the default.

If your application is using TCP_CLIENT to start a server through a NETSOCKET/400 daemon, it must then send 21 characters indicating the name in ASCII of the server and then receive 7 ASCII bytes for the status.

If the status is “SUCCESS” then the server has been initiated and the application can continue as designed. I

f the status is “FAILURE” then there was some sort of problem. More information can be obtained by looking in the daemon log or by checking a trace that was turned on by the daemon with TCP_TRACE.

If the status is “MAX_SVR” then the maximum number of servers is already running. The maximum number of servers was defined in the TCP_DAEMON procedure when the daemon was initiated. You may perform whatever retries logic you wish. If you receive “MAX_SVR” you may wish to delay for a specified period of time and then try to connect again.

The ability to start a server using this method is NOT supported in SSL mode

TCP_CLOSE Shutdown a client or server

The TCP_CLOSE procedure is used to reallocate the resources and shut down a communications link. This procedure should be executed for all conversations that were established with the TCP_CLIENT and TCP_SERVER procedures to ensure that all system resources are freed.

The TCP_CLOSE procedure expects the following parameter:

Parameters	Attributes	Description
Handle	decimal(10)	The handle of a communications link that was established with either the TCP_CLIENT, TCP_RECV, or TCP_SERVER procedures.

TCP_DAEMON **Start servers for incoming clients**

The TCP_DAEMON procedure allocates resources for and initiates the listening on a particular host port for incoming clients.

Once a client is found, it will establish communications with it. If the hard-coded server parameter is not blank, then that server will be initiated and the client will be passed to it. Otherwise, the daemon expects the client to transfer a 21-character string in ASCII defining the name and location of the server. The daemon will then attempt to start the server and pass the client conversation to it. If this fails, the daemon will send the text string "FAILURE" in ASCII to the client and drop the conversation. It will then continue listening for more clients. If it is successful, then the server has received the client and the daemon has to do nothing more than search for more clients.

The TCP_DAEMON procedure expects the following parameters.

Parameters	Attributes	Description
Address	character(up to 256)	The local IP address to scan for incoming clients. The address can be in dotted decimal format such as "128.1.1.1" or a host name such as "host1.esd.com". If the first character is an asterisk (*), the daemon will monitor all configured IP address interfaces for that AS/400.
Host port	decimal(5)	The daemon will scan a certain host port for incoming clients. Host ports are a means of allowing multiple simultaneous TCP/IP conversations. The host port can range from 1 to 65535; although the first 2047 ports are generally reserved.
Maximum servers	decimal(10)	The maximum number of servers to allow to be run at any given time. If this many servers are active when a client connects to the daemon, the daemon will refuse the connection.
Hard-coded server	character(21)	If this field is passed in as <i>blanks</i> , then the daemon will expect the client to send an initial 21 bytes determining the name of the server to start. It will then transmit back "SUCCESS", "MAX_SVR" or "FAILURE". If this field is <i>not blank</i> , it then describes the name of the server that the daemon should start for any incoming clients. The 21 bytes represent up to a 10 character library name a '/' and up to a 10 character program name.
Job description	character(21)	The job description name that the server should run under. It can have a library preceding it such as "TCPIPxxx/TCPJOB" or just the job description such as "TCPJOB". Note the spacing. This field must be 21 characters in size. If the field is entirely blank, the job description that the daemon is running under

Job desc (cont)		is used. The toolkit does not install a job description. You must create one yourself if it is required.
Named Port	character(30)	If a value is entered for this parameter it will be used to automatically add a service table entry that will be assigned to the host port specified above. The purpose of using this parameter is for documentation. Using this method you can document what a given Daemon/Server program is providing as a service and it will be displayed as such on the working with TCP/IP Connection Status screen.
Minimum Servers	decimal(10)	If a value is entered on this parameter the Daemon program will automatically pre-start this number of Server jobs. After a Client connection is detected the Daemon will check for any available Server jobs and pass the connection to it on a first come first serve basis. The Daemon procedure will periodically monitor the AS/400 for the current number of active/available Server jobs and start new Server jobs as needed. The use of this parameter is only supported if there is a value entered for the hard-coded server parameter above.

When a client contacts the daemon, and the hard-coded server parameter was left blank, then it must send a 21-character string. This string defines the server to initiate.

The client must receive the string within two minutes or the daemon will drop communications with the client. **The string must be sent in ASCII.** The string can have library information or not. If not, then the daemon’s library list is searched.

Examples are “TCPIPxxx/RPG_SERVER ” and also “RPG_SERVER “. The client must then wait for a seven-character status. The daemon will either send “FAILURE” indicating a problem initiating the server, or it will send “MAX_SVR” indicating that there are currently too many servers active. It will never send “SUCCESS”. “SUCCESS” is only sent by a server that has been successfully initiated. So if a client receives “SUCCESS” it can assume that there has not been a problem and can continue on normally.

The daemon will continue on listening for clients and initiating servers, if needed, indefinitely. The only way to **properly shutdown a Daemon** is to use the TCP_ENDDMN procedure. If the daemon is stopped in any other fashion, the port(s) may remain open. The status of open ports can be obtained by typing **NETSTAT** and selecting option 3.

Note: The TCP_DAEMON procedure does not return control back to the calling application unless there was an error or the daemon was ended by calling

TCP_ENDDMN.TCP_DMNLOG **Start or stop logging for daemon activity**

All daemons capture activity information. This information can be recorded in a file named TCPDMNLOG. This log is for your use. It allows you to record the amount of activity a daemon has received and who is accessing your servers. If you wish for a given daemon to record this information, your application should call the TCP_DMNLOG procedure prior to TCP_DAEMON to turn on activity logging to the file.

The TCP_DMNLOG procedure expects the following parameter.

Parameters	Attributes	Description
Value	decimal(1)	A value of zero turns logging off. A value of one turns logging on.

TCP_ENDDMN **Shutdown a running daemon**

The TCP_ENDDMN procedure is used to shut down a currently running daemon.

The TCP_ENDDMN procedure expects the following parameters:

Parameters	Attributes	Description
Address	character (up to 256)	Enter the same address here that was entered on the TCP_DAEMON address parameter
Host Pot	Decimal (5)	Enter the same port number here that was entered on the TCP_DAEMON host port parameter

TCP_ERRVARS **Define variables that receive status data**

The TCP_ERRVARS procedure provides a means of letting your application know about various errors and statuses that may occur. NETSOCKET/400 remembers the variables that are passed to this routine and all error information is placed into those variables.

This routine needs only to be called once. All procedures in NETSOCKET/400 with the exception of TCP_TRACE initialize these variables and then set them if there is an error.

Your application should query them for errors after calling each procedure.

The TCP_ERRVARS procedure expects the following parameters:

Parameters	Attributes	Description
Error number	Decimal (2)	The number of the error. If this value is zero , your application can assume success. For a list of errors, see the <i>Status Codes and Messages section</i>
Error string	Character (70)	A textual description of the error. If there is no error, it will be blank. If an error occurs and the trace is on, this description is displayed in the trace

TCP_OVRCVN Override default translation tables

The TCP_OVRCVN procedure can be used to override the default translation tables used when the TCP_SEND and TCP_RECV procedures are used and the convert parameter is on. The default translation tables used are CCSID 00437 for ASCII and CCSID 00037 for EBCDIC.

The TCP_OVRCVN procedure expects the following parameters:

Parameters	Attributes	Description
Remote CCSID	Decimal (5)	Coded character set identifier for the translation table to be used for translating the data before sending it to the remote connection. (<i>See CCSID table below to determine correct identifier.</i>)
Local CCSID	Decimal (5)	Coded character set identifier for the translation table to be used for translating the data after receiving it from the remote connection. (<i>See CCSID table below to determine correct identifier.</i>)

CCSID Table

CCSID	Description
00037	US, Canada, Netherlands, Portugal, Brazil, New Zealand, Australia

00256	Netherlands
00273	Austria, Germany
00277	Denmark, Norway
00278	Finland, Sweden
00280	Italy
00284	Spanish, Latin America
00285	United Kingdom
00290	Japan Katakana (extended range)
00297	France
00300	Japan English
00301	Japanese PC Data
00367	ANSI X3.4 ASCII standard; USA
00420	Arabic-speaking countries
00423	Greece
00424	Hebrew
00437	PC Data; PC Base; USA
00500	Belgium, Canada, Switzerland, International Latin-1
00813	ISO 8859-7; Greek/Latin
00819	ISO 8859-1; Latin Alphabet
00833	Korea (extended range)
00834	Korea host double byte (including 1880 UDC)
00835	Traditional Chinese host double byte (including 6204 UDC)
00836	Simplified Chinese (extended range)
00837	Simplified Chinese
00838	Thailand (extended range)
00850	PC Data; MLP 222 Latin Alphabet 1
00852	PC Data; Latin-2 Multilingual
00855	PC Data; ROECE Cyrillic
00857	PC Data; Turkey Latin #5
00860	PC Data; Portugal
00861	PC Data; Iceland
00862	PC Data; Hebrew
00863	PC Data; Canada
00864	PC Data; Arabic
00865	PC Data; Denmark, Norway
00866	PC Data; Cyrillic #2 – Personal Computer
00869	PC Data; Greek

00870	Latin-2 Multilingual
00871	Iceland
00874	Thai PC Data
00875	Greece
00880	Cyrillic Multilingual
00891	Korean PC Data (non-extended)
00897	Japanese PC Data (non-extended)
00903	Simplified Chinese PC Data (non-extended)
00904	Traditional Chinese PC Data
00905	Turkey Latin-3
00912	ISO 8859-2; ROECE Latin-2 Multilingual
00915	ISO 8859-5; Cyrillic; 8-bit ISO
00916	ISO 8859-8; Hebrew
00920	ISO 8859-9; Latin 5
00926	Korean PC Data DBCS, UDC 1880
00927	Traditional Chinese PC Data DBCS, UDC 6204
00928	Simplified Chinese PC Data DBCS, UDC 1880
00930	Japan Katakana (extended range) 4370 UDC (User Defined Characters)
00932	Japan PC Data Mixed
00933	Korea (extended range), 1880 UDC
00934	Korean PC Data
00935	Simplified Chinese (extended range)
00936	Simplified Chinese (non-extended)
00937	Traditional Chinese (extended range)
00938	Traditional Chinese (non-extended)
00939	Japan English (extended range) 4370 UDC
00942	Japanese PC Data Mixed
00944	Korean PC Data Mixed
00946	Simplified Chinese PC Data Mixed
00947	ASCII Double-byte
00948	Traditional Chinese PC Data Mixed 6204 UDC (User Defined Characters)
00949	Republic of Korea National Standard Graphic Character Set (KS) PC Data mixed-byte including 1800 UDC
00950	Traditional Chinese PC Data Mixed for Big5
00951	Republic of Korea National Standard Graphic Character Set (KS) PC Data double-byte including 1800 UDC
00956	JIS X201 Roman for CP 00895; JIS X208-1983 for CP 00952

00957	JIS X201 Roman for CP 00895; JIS X208-1978 for CP 00955
00958	ASCII for CP 00367; JIS X208-1983 for CP 00952
00959	ASCII for CP 00367; JIS X208-1978 for CP 00955
00964	G0 – ASCII for CP 00367; G1 – CNS 11643 plane 1 for CP 960
00965	ASCII for CP 00367; CNS 11643 plane 1 for CP 960
00970	G0 ASCII for CP 00367; G1 KSC X5601-1989 (including 188 UDCs) for CP 971
01008	Arabic 8-bit ISO/ASCII
01010	ISO-7; France
01011	ISO-7; Germany
01012	ISO-7; Italy
01013	ISO-7; United Kingdom
01014	ISO-7; Spain
01015	ISO-7; Portugal
01016	ISO-7; Norway
01017	ISO-7; Denmark
01018	ISO-7; Finland and Sweden
01019	ISO-7; Belgium and Netherlands
01025	Cyrillic Multilingual
01026	Turkey Latin 5 CECP
01027	Japan English (extended range)
01040	Korean Latin PC Data extended
01041	Japanese PC Data extended
01042	Simplified Chinese PC Data extended
01043	Traditional Chinese PC Data extended
01046	PC Data – Arabic Extended
01088	Korean PC Data single-byte
01097	Farsi
01098	Farsi (IBM-PC)
01114	Traditional Chinese, Taiwan Industry Graphic Character Set (Big5)
01115	Simplified Chinese, People’s Republic of China National Standard (GB), personal computer SBCS
01380	Simplified Chinese, People’s Republic of China National Standard (GB), personal computer DBCS
01381	Simplified Chinese, People’s Republic of China National Standard (GB) personal computer mixed SBCS and DBCS
04396	Japanese Host DB including 1880
04948	Latin 2 PC Data Multilingual

04951	Cyrillic PC Data Multilingual
04952	Hebrew PC Data
04953	Turkey PC Data Latin 5
04960	Arabic PC Data
04965	Greek PC Data
05026	Japan Katakana (extended range) 1880 UDC
05035	Japan English (extended range) 1880 UDC
05050	G0 – JIS X201 Roman for CP 895; G1 JIS X208-1990 for CP 952
05052	JIS X210 Roman for CP 895; JIS X208-1983 for CP 952
05053	JIS X201 Roman for CP 895; JIS X208-1983 for CP 955
05054	ASCII for CP 367; JIS X208-1983 for CP 952
05055	ASCII for CP 367; JIS X208-1978 for CP 955
17354	G0 – ASCII for CP 00367; G1 – KSC X5601-1989 (including 188 UDCs) for CP 00971
28709	Traditional Chinese (extended range)
57345	All Japanese 2022 characters
61952	ISO/IEC 10646 Universal Character Set Level 2 (UCS-2)
65534	Look at lower level CCSID
65535	Special Value indicating data is hex and should not be converted. This is the default for the QCCSID system value

TCP_PGID Calculate a program ID for Address/Host port to be monitored

The TCP_PGID procedure is used to calculate a unique program ID that will be used for all job names associated with the Daemon program monitoring the selected IP address/Host port. This procedure returns a 10 character parameter (*see prototype provided*) that is a base 41 representation of the address/host port entered below.

The program ID is the mechanism used to associate one or more server programs with their respective managing Daemon program. The reason this procedure is provided is to allow you to manage the number of available server programs for use by a certain Daemon. The value returned here should be used for the submitted job name for any server program to be associated with a Daemon.

The TCP_PGID procedure expects the following parameters:

Parameters	Attributes	Description
Host	Character (up to 256)	The IP address the Daemon program is listening to for incoming client connections. This field can either contain a dotted decimal IP address such as “128.1.1.1”, or a host name such as

		“host1.esd.com”.
Host Port	Decimal (5)	The host port where the Daemon is listening for inbound connection attempts.

TCP_RCVBND Receive a string of bytes under boundary control

The TCP_RCVBND is used to receive data from a remote server or client up to a detected registered boundary sequence. A registered boundary sequence is any one or two character boundary sequence registered with Netsocket using the TCP_REGBND procedure.

The communications link must be already established with either the TCP_CLIENT or TCP_SERVER procedures. **This procedure cannot be used in conjunction with the multiplexing I/O method of server operations.** The remote application must have sent or be sending the data you are trying to receive.

The TCP_RCVBND procedure expects the following parameters:

Parameters	Attributes	Description
Handle	Decimal (10)	The value entered here will be the one that was provided using either the TCP_CLIENT or TCP_SERVER procedures.
Data	Character (up to 32,000 bytes)	A character field large enough to hold the data to be received. The data received in this parameter will contain all the data up to, but not including, the first registered boundary character detected in the received buffer. Any subsequent receive attempts may retrieve data from the holding buffer leftover from the last receive attempt or may trigger another actual TCP receive from the TCP buffer of the socket..
Length	Decimal (10)	The value of this parameter should be set to the size of the data field above. When the procedure is complete, this variable will be populated with how many bytes were actually placed in the data parameter

Time out	Decimal (5)	<p>This variable allows your application to specify how many one-second increments to wait for the data to be received and then scanned for the registered boundary characters.</p> <p>The TCP_RCVBND procedure will attempt to process the data in 1/10th second bursts up to the number of seconds specified. This value may be adjusted based on traffic volume through the AS/400 TCP/IP stack and the amount of traffic on the network.</p> <p>If this value is set to '0' the procedure will complete immediately after any bytes are received regardless if any boundary characters were detected.</p>
Convert	Decimal (1)	<p>This variable should have a value of either one or zero. If the value is one, then the TCP_RECV procedure will convert the data before it is returned to your application based on the current from/to CCSID values. The default setting if not overridden with the TCP_OVRCVN procedure is EBCDIC and ASCII</p>
Flush buffer	Character (1)	<p>The value entered here should be either 'Y' or 'N'. If this value is 'Y' the TCP_RCVBND procedure will flush the contents of the current holding buffer containing all data received to-date from the socket. This will include any boundary characters that may be contained within. This option should be used as the last step performed before closing a socket.</p>
Boundary	Character (2)	<p>This parameter will be populated with whatever registered boundary sequence was detected that caused data to be returned to your program.</p>

TCP_RECV Receive a string of bytes

The TCP_RCVBND is used to receive data from a remote server or client. The communications link must be already established with either the TCP_CLIENT or TCP_SERVER procedures. The remote application must have sent or be sending the data you are trying to receive.

Netsocket/400 utilizes connection oriented streaming sockets, which means you can read as much of the buffer as you want on every receive operation regardless of how much data was sent to your application.

The TCP_RECV procedure expects the following parameters:

Parameter	Attributes	Description
-----------	------------	-------------

Handle	Decimal (10)	<p>If not using multiplexing mode the value entered here will be the one that was provided using either the TCP_CLIENT or TCP_SERVER procedures.</p> <p>If using multiplexing mode any value entered here will first be cleared then populated with the handle of the client that you are receiving data from. The handle can then be used to send a response back to that particular client using the TCP_SEND procedure.</p>
Data	Character (up to 32,000 bytes)	A character field large enough to hold the data to be received.
Length	Decimal (10)	<p>The TCP_RECV procedure will attempt to read this many bytes of data and populate the variable passed in the previous parameter with the received data.</p> <p>The value of this parameter must not exceed the size of the data field or unpredictable results will occur. When the procedure is complete, this variable will be populated with how many bytes were actually read.</p>
Time-out	Decimal (5)	<p>This variable allows your application to specify how many one-second increments to wait for the data to be received. The TCP_RECV procedure will attempt to read the expected data in 1/10th second bursts up to the number of seconds specified. This value may be adjusted based on traffic volume through the AS/400 TCP/IP stack and the amount of traffic on the network.</p> <p>If this value is set to '0' and the fixed length parameter below is set to 'N' the procedure will complete immediately after any bytes are received. This option is especially valuable when you do not know how many bytes you are going to receive.</p>
Convert	Decimal (1)	This variable should have a value of either one or zero. If the value is one, then the TCP_RECV procedure will convert the data before it is returned to your application based on the current from/to CCSID values. The default setting if not overridden with the TCP_OVRVCVN procedure is EBCDIC and ASCII.
Fixed length	Character (1)	<p>The value entered here should be either 'Y' or 'N'. If this value is 'Y' the TCP_RECV procedure will wait indefinitely for the number of bytes specified in the length parameter. While waiting, the program will go to sleep and use minimal system resources. This also means that control will never be returned to your program until the length number of bytes is received.</p> <p>Make sure that if you enter a 'Y' for this parameter that you are expecting fixed length strings.</p>
Client address	Character (up	The dotted decimal address of the remote client you are currently

	to 256)	receiving data from will be returned in this parameter when running in multiplexing mode. This will assist you in identifying the current client you are conversing with.
Client host port	Decimal (5)	The host port of the remote client you are currently receiving data from will be returned in this parameter when running in multiplexing mode. This will assist you in identifying the current client you are conversing with.

TCP_REGBND Register a boundary sequence

The TCP_RCVBND procedure is used to register up to 50 boundary sequences that will be used when scanning received character strings on the TCP_RCVBND procedure. A boundary sequence is any one or two character string that is used to distinguish the boundary between fields or logical records.

Typically boundary sequences are used whenever variable length strings are sent to help distinguish between the beginning and the end of a transaction set. For example, a carriage return (CR) - line feed (LF) sequence is often used to distinguish logical record boundaries.

The TCP_REGBND procedure expects the following parameter.

Parameter	Attributes	Description
Boundary	Character (2)	This parameter should be populated with whatever boundary sequence is to be registered. If the boundary sequence is a single character then it should be left justified.

TCP_SEND Send a string of bytes

The TCP_SEND procedure is used to send data to a remote server or client. The communications link must be already established with either the TCP_CLIENT or TCP_SERVER procedures. The remote application must be expecting to receive this data.

The TCP_SEND procedure expects the following parameters

Parameter	Attributes	Description
Handle	decimal(10)	The handle that was provided as a result of using either the TCP_CLIENT or TCP_SERVER procedures.
Data	character(up to 32,000 bytes)	A character field containing the data to be sent
Length	decimal(10)	The TCP_SEND procedure will attempt to send

		<p>this many bytes of the data that was passed in the previous parameter.</p> <p>The value of this parameter must not exceed the size of the data field or unpredictable results will occur. When the procedure is complete, this variable will be populated with how many bytes were actually sent.</p>
Time-out	decimal(5)	<p>The TCP_SEND procedure will attempt to send the data every 1/10th second. This variable allows your application to specify how many 1/10th second increments to wait for the data to be sent. This value may be adjusted based on traffic volume through the AS/400 TCP/IP stack and the amount of traffic on the network.</p>
Convert	decimal(1)	<p>This variable should have a value of either one or zero. If the value is one, then the TCP_SEND procedure will convert the data before it is sent to the remote application based on the current from/to CCSID values. The default setting if not overridden with the TCP_OVRCVN procedure is EBCDIC and ASCII.</p>

TCP_SERVER Receive client connection

The TCP_SERVER procedure can be used in many ways to develop server programs. The available methods and how to use them are explained in detail in the previous chapter.

Secured socket mode is automatically initiated if the last four (4) parameters in this specification are used. Otherwise the server connection and all subsequent send/receive operations will be in non-secured mode.

The TCP_SERVER procedure expects the following parameter.

Parameter	Attributes	Description
Handle	decimal(10)	<p>If successful, the TCP_SERVER procedure will populate this variable with a handle. If not in multiplexing mode, this handle would then be used to communicate with the client via the TCP_SEND and TCP_RECV procedures.</p> <p>If running in multiplexing mode, the handle returned here is for the socket associated with listening for inbound client connections. It should be saved so it can later be used with the TCP_CLOSE procedure to free up socket resources just before</p>

		<p>the server program is ended. When in multiplexing mode this handle should never be used to send or receive data using the TCP_SEND and TCP_RECV procedures.</p> <p>A handle with a value of 0 may be valid This handle should be closed with the TCP_CLOSE procedure when it is not needed any longer or upon application termination.</p>
Address	character(up to 256)	<p>The local IP address to scan for incoming clients. The address can be in dotted decimal format such as “128.1.1.1” or a host name such as “host1.esd.com”.</p> <p>If the first character is an asterisk (*), the daemon will monitor all configured IP address interfaces for that AS/400.</p> <p>A value in this parameter is only needed if not using the Daemon program to pass inbound clients.</p>
Host port	decimal(5)	<p>The daemon will scan a certain host port for incoming clients. Host ports are a means of allowing multiple simultaneous TCP/IP conversations.</p> <p>The host port can range from 1 to 65535; although the first 2047 ports are generally reserved.</p> <p>A value in this parameter is only needed if not using the Daemon program to pass inbound clients and will be ignored if no value is entered for the address parameter.</p>
Maximum clients	decimal(10)	<p>The maximum number of client connections allowed to be accepted by this server. This value can range from 0 to 512.</p> <p>If zero or one is entered and a value is entered for the address parameter the server procedure will allow a single client connection to be accepted.</p> <p>If a value greater than one is entered you will be in multiplexing mode for up to this number of client connections.</p> <p>A value in this parameter is only needed if not using the Daemon program to pass inbound clients and will be ignored if no value is entered for the address parameter.</p>
Named port	character(30)	<p>If a value is entered for this parameter it will be used to automatically add a service table entry that will be assigned to the host port specified above.</p> <p>The purpose of using this parameter is for documentation. Using this method you can document what a given</p>

		<p>Daemon/Server program is providing as a service and it will be displayed as such on the when working with TCP/IP Connection Status screen.</p> <p>A value in this parameter is only needed if not using the Daemon program to pass inbound clients and will be ignored if no value is entered for the address parameter.</p>
KeyRingFile	character(up to 256)	Path to the key ring file to be used for this jobs SSL processing. The path must be a fully qualified integrated file system (IFS) file name.
KeyRingPassword	character(up to 256)	Password for the key ring file named in the KeyRingFile name field. If this parameter's value is equal to BLANKS, then SSL support will attempt to extract a password from a key-ring password file.
SSLsocket	character(1)	A flag "Y" or "N" will be returned to indicate if the SSL connect was successful.
TimeOutWait	decimal(5)	The number of seconds to wait before giving up on the attempt at a SSL connection. If this parameter is omitted, 120 seconds is the default.

TCP_TRACE Turn on or off tracing functionality

The TCP_TRACE procedure can be used to turn on or off an internal trace mechanism. The trace data will be printed out using the TCP_PRINTF print file.

Every time it is turned on, a new spooled file is generated holding trace data from that instance forward. It can be turned back off at any time by setting the value to 0. The trace is off by default. Trace entries are time stamped to the millisecond.

The TCP_TRACE procedure expects the following parameter.

Parameter	Attributes	Description
Value	decimal(1)	<p>The trace level as listed below.</p> <ol style="list-style-type: none"> 1. Trace level 1 documents each API command as it is used. Each API may have supplemental information that will be shown such as TCP_SEND will show how many bytes attempted and actually sent, et cetera. 2. Trace level 2, outputs all trace information of level 1

		<p>along with many intermediate steps.</p> <ol style="list-style-type: none"> 3. Trace level 3 is a diagnostic trace. A dump of the actual data sent back and forth. The data is displayed with the hexadecimal equivalent. Non-displayable data is displayed with a dot. The data is displayed before any conversion with the TCP_SEND procedure and after any conversion with the TCP_RECV procedure. This trace can grow rather large. 4. Trace level 4 is the most complete diagnostic trace. It outputs all the data that levels 1, 2 and 3 output along with information related to passing socket descriptors. In addition, all the converted data (in hexadecimal form) is shown for both inbound and outbound data. This trace can grow rather large.
--	--	--

TCP_TRCTXT Output user defined text to the trace

The TCP_TRCTXT procedure is used if a trace has been initiated with the TCP_TRACE. This procedure may be used to output user defined text into the trace.

The TCP_TRCTCT procedure expects the following parameter.

Parameter	Attributes	Description
Data	Character (up to 132 bytes)	Trace data to be displayed in the trace.

TCP_WAIT Amount of time to sleep

The TCP_WAIT procedure can be used to put your program in a sleep state and not use system resources for a specified amount of time

The TCP_WAIT procedure expects the following parameter.

Parameter	Attributes	Description
Seconds	Decimal (10)	Enter the number of seconds to sleep. Valid values range from

		0 to 2147483647.
Millisec	Decimal (10)	Enter the number of milliseconds to sleep. Valid values range from 0 to 2147483647

Chapter 10

PROGRAMMING EXAMPLES

This chapter contains listings of example programs. Examples of each of the supported ILE languages are provided. They are provided to assist you in developing client, server and daemon programs in the language of your choice. These examples are for demonstration purposes only. They do not interrogate and react to specific errors, as more robust TCP/IP applications should.

About the Daemon Examples

The daemon examples receive two command line parameters. The IP address and the host port. The IP address can be a dotted decimal address of a local IOP such as “128.1.1.1”, or a valid host name of an IOP such as “host1.esd.com”. This is the IP address that the client must access your AS/400 through to reference the daemon. TCP_DAEMON has the added functionality to specify an asterisk “*” for the IP address that tells the daemon that it doesn’t care what IP address the client entered the system through. The second parameter is a host port. It can range from 0 to 65,535.

The examples take the command line arguments and pass them to TCP_DAEMON. Control does not return back to the example unless there was an error or the daemon was shut down. See the TCP_DAEMON procedure for more details. The examples do monitor for possible errors, but they do not react to them. They simply end the program.

The examples leave the hard-coded server parameter of the TCP_DAEMON procedure blank. Therefore, the daemon requires the client to transmit the name of the server to initiate. The server name should be transmitted as a 21-byte ASCII string. Then, a 7-byte reply will be sent back to the client. The reply will be transmitted automatically. If there was a problem or the maximum amount of servers is currently running the TCP_DAEMON procedure will transmit “FAILURE” or “MAX_SVR” respectively. If the server is successfully initiated, then the TCP_SERVER procedure will transmit “SUCCESS”. Either way, the client should be waiting for a 7-byte reply.

There are also examples of programs that shut down daemons. This is accomplished by passing a port number to TCP_ENDDMN.

About the Client Examples

The client examples take the same two command line parameters as the daemon examples. These represent the IP address and host port on the remote system. Of course, the IP address cannot be an asterisk. This would not make sense because you have to specify where the remote system is. If you plan on running these example applications all on the same AS/400 for testing purposes, just use an IP address of a your AS/400. The communications will never leave the AS/400, but it will appear to work the same as if they were remote. These parameters are passed to `TCP_CLIENT` and a conversation handle is obtained. This handle is then used to communicate back and forth with the remote host. The example client applications are designed to communicate with a `NETSOCKET/400` daemon. Therefore, after a good call to `TCP_CLIENT`, the client must then send a 21-character string. It indicates the name of the server that the daemon should initiate. It should then receive 7 characters for a status. See the `TCP_CLIENT` procedure for more details. Once the server is properly initiated, the client starts communicating with it. It uses a very simple application protocol. If the client sends "REQ1", it expects a 35 characters string back. If the client sends "DONE", it expects the server to shut down. This is where you, as the programmer, have ultimate flexibility. You can design your own application protocol as robust and feature-packed as desired. The examples do monitor for possible errors, but they do not react to them. They simply end the program.

About the Server Examples

The server examples are designed to be called by a daemon. They take no command line arguments. The server application simply calls `TCP_SERVER`. If the procedure is successful, it returns a conversation handle to be used for communication to the client. `TCP_SERVER` internally send the "SUCCESS" status message that the client expects. Once this is complete, the application protocol kicks in. The server handles what is known as transaction sets. It waits for a request of some sort from the client. It then carries it out and perhaps sends information or a confirmation back. The transaction set designed for the examples is extremely simple, as you will see. The examples do monitor for possible errors, but they do not react to them. They simply end the program.



RPG ILE EXAMPLES

Daemon

```

*-----*
* Example Daemon program using the TCP/IP Toolkit *
*-----*
F*
F* The following file is used to print status info and error conditions
FQSYSPRT O F 132 PRINTER
*
DTRACTEXT C CONST('Example RPG program establish-
D ing daemon')
*
C* The following entry parameters are used to establish the TCP/IP
C* address and host port of the daemon pgm.
C *ENTRY PLIST
C PARM PMTCPA 256
C PARM PMTCPH 15 5

C* Convert TCP host port to numeric for later use.
C Z-ADD PMTCPH XXTCPH 5 0
C*
* The following procedure call establishes the error variables that
* the Toolkit API's will use when any error conditions are encountered
C CALLB 'TCP_ERRVARS'
C PARM 00 PMERNO 2 0 TCP error number
C PARM PMERTX 70 TCP error text
*
C DO
*
* The following procedure call establishes trace data level.
C CALLB 'TCP_TRACE'
C PARM 4 PMTRLV 1 0 TCP trace level
*

C* Perform any desired error routines
C IF PMERNO <> 0

```

```

C          LEAVE
C          ENDIF
C*
C* Clear error fields prior to using "TCP_TRCTXT"
C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
* The following procedure call establishes trace data text.
C          CALLB  "TCP_TRCTXT"
C          PARM  TRACTEXT  PMTRTX      39      TCP trace text
*
C* Perform any desired error routines
C          IF    PMERNO <> 0
C          LEAVE
C          ENDIF
C*
C* Clear error fields prior to using "TCP_DAEMON"
C          CLEAR          PMERNO
C          CLEAR          PMERTX
*
* Establish a Daemon for the address/port passed
C          CALLB  "TCP_DAEMON"
C          PARM          PMTCPA      256      TCP/IP address
C          PARM          XXTCPH      5 0      TCP/IP host port
C          PARM  10          PMMAXS      10 0      Max Servers
C          PARM  *BLANKS    PMSRVN      21      Job Description
C          PARM  *BLANKS    PMJOBDB      21      Job Description
C          PARM  *BLANKS    PMNMPT      30      Named port
C          PARM  0          P          MMINS      10 0      Minimum servers
*
C* Exit client program.
C          ENDDO
C*
C* Error text present - Print.
C  PMERTX  IFNE  *BLANKS
C          EXCEPT  ERRTXT
C          ENDIF
*
C          MOVE  *ON          *INLR
C*
OQSYSPRT  E          ERRTXT      1
O          6 'Error:'
O          PMERNO      9
O          PMERTX      81

```

Shutdown Daemon

```

*-----*
* Example program to End TCP/IP Toolkit Daemons. *
*-----*
*
* The following file is used to print status info and error conditions
FQSYSPRT O F 132 PRINTER

C* The following entry parameters are used to establish the TCP/IP
C* address and host port of the daemon pgm that will be shutdown.
C *ENTRY PLIST
C PARM PMTCPA 256
C PARM PMTCPH 15 5
C*
C* Convert TCP host port to numeric for later use.
C Z-ADD PMTCPH XXTCPH 5 0
*
* The following procedure call establishes the error variables that
* the Toolkit API's will use when any error conditions are encountered
C CALLB 'TCP_ERRVARS'
C PARM PMERNO 2 0 TCP error number
C PARM PMERTX 70 TCP error text
*
* End the Daemon for the address/port passed
C CALLB 'TCP_ENDDMN'
C PARM PMTCPA
C PARM XXTCPH
*
*
C* Check result of end attempt by checking error fields.
C PMERTX IFNE *BLANKS
C* Perform any desired error routines
C EXCEPT ERRTXT
C ENDIF
*
C MOVE *ON *INLR
*
OQSYSPRT E ERRTXT 1
O 6 'Error:'
O PMERNO 9
O PMERTX 81

```

Client

```

*-----*
* Example client program using the TCP/IP Toolkit.      *
*-----*
F*
F* The following file is used to print status info and error conditions
FQSYSPRT O F 132 PRINTER
D*
D* The following constants are sent to server program as strings.
DTRACTEXT C CONST('Example client application in-
D RPG')
DREQUEST1 C CONST('REQ1')
DREQUEST2 C CONST('REQ2')
DREQUEST3 C CONST('RPG_SERVER ')
DDONE C CONST('DONE')
C*
C* The following entry parameters are used to establish the TCP/IP
C* address and host port of the daemon pgm this client wishes to talk
C* to.
C *ENTRY PLIST
C PARM PMTCPA 256
C PARM PMTCPH 15 5
C*
C* Convert TCP host port to numeric for later use.
C Z-ADD PMTCPH XXTCPH 5 0
C MOVE *OFF XXCONN
C*
C* The following procedure call establishes the error variables that
C* the Toolkit API's will use when any error conditions are encountered
C CALLB 'TCP_ERRVARS'
C PARM PMERNO 2 0
C PARM PMERTX 70
C*
C do
C*
C* The following procedure call turns on the trace for all of the
C* TCP/IP Toolkit API's.
C*
C CALLB 'TCP_TRACE'
C PARM 4 PMTRAC 1 0
C*
C* Perform any desired error routines
C IF PMERNO <> 0
C LEAVE
C ENDIF
C*

```

```

C* Clear error fields prior to using 'TCP_TRCTXT'
C      CLEAR      PMERNO
C      CLEAR      PMERTX
*
* The following procedure call establishes trace data text.
C      CALLB     'TCP_TRCTXT'
C      PARM     TRACTEXT  PMTRTX      33      TCP trace text
*
C* Perform any desired error routines
C      IF      PMERNO <> 0
C      LEAVE
C      ENDIF
C*
C* Clear error fields prior to using 'TCP_CLIENT'
C      CLEAR      PMERNO
C      CLEAR      PMERTX
*
* The following procedure call establishes connection and returns
* a valid handle.
C      CALLB     'TCP_CLIENT'
C      PARM      PMTCPA      256
C      PARM      XXTCPH      5 0
C      PARM      PMHAND      10 0
*
C* Perform any desired error routines
C      IF      PMERNO <> 0
C      LEAVE
C      ENDIF
C*
C      MOVE     *ON      XXCONN      1
C*
C* Process send/receive request for server.
C* Clear error fields prior to using 'TCP_SEND'
C      CLEAR      PMERNO
C      CLEAR      PMERTX
C*
C* The following procedure call sends the server program startup
C* request to the daemon program.
C      CALLB     'TCP_SEND'
C      PARM      PMHAND      10 0
C      PARM     REQUEST3  PMDATA      35
C      PARM     21      PMDLEN      10 0
C      PARM     120     PMTRY       5 0
C      PARM     1      PMCNVT      1 0
C*
C* Check result of send attempt by checking error fields.
C      IF      PMERNO <> 0

```

```

C          LEAVE
C          ENDIF
C*
C* Print sent value
C          EXCEPT  SNDLIN
C*
C*
C* Clear error fields prior to using 'TCP_RECV'
C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
C* The following procedure call receives the response from the daemon
C* program for the server program start request.
C          CALLB  'TCP_RECV'
C          PARM          PMHAND
C          PARM  *BLANKS  PMDATA      35
C          PARM  7        PMDLEN     10 0
C          PARM  120      PMTRY      5 0
C          PARM  1        PMCNVN     1 0
C          PARM  'N'      PMFIXD     1
C          PARM  *BLANKS  PMCLIP     15
C          PARM  0        PMCLPT     5 0
C*
C* Check result of connection attempt by checking error fields.
C          IF    PMERNO <> 0
C          LEAVE
C          ENDIF
C*
C* Check result of the server connection.
C          IF    %SUBST(PMDATA:1:7) <> 'SUCCESS'
C          LEAVE
C          ENDIF
C*
C* Print received value
C          EXCEPT  RCVLIN
C*-----
C* Process send/receive request one
C*-----
C*
C* Clear error fields prior to using 'TCP_SEND'
C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
C* The following procedure call sends the first string to the server
C* program.
C          CALLB  'TCP_SEND'
C          PARM          PMHAND

```

```

C          PARM  REQUEST1  PMDATA    35
C          PARM   4          PMDLEN   10 0
C          PARM  120        PMTRY     5 0
C          PARM   1          PMCNVT   1 0
C*
C* Check result of send attempt by checking error fields.
C          IF    PMERNO <> 0
C          LEAVE
C          ENDIF
C*
C* Print sent value
C          EXCEPT  SNDLIN
C*
C* Clear error fields prior to using 'TCP_RECV'
C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
C* The following procedure call receives the response from the server
C* program for the 'REQ1' send request.
C          CALLB  'TCP_RECV'
C          PARM          PMHAND
C          PARM  *BLANKS  PMDATA    35
C          PARM   35      PMDLEN   10 0
C          PARM  120      PMTRY     5 0
C          PARM   1       PMCNVT   1 0
C          PARM  'N'      PMFIXD    1
C          PARM  *BLANKS  PMCLIP    15
C          PARM   0       PMCLPT   5 0
C*
C* Check result of receive attempt by checking error fields.
C          IF    PMERNO <> 0
C          LEAVE
C          ENDIF
C*
C* Print received value
C          EXCEPT  RCVLIN
C*-----
C* Process send/receive request two
C*-----
C*
C* Clear error fields prior to using 'TCP_SEND'
C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
C* The following procedure call sends the second string to the server
C* program.
C          CALLB  'TCP_SEND'

```

```

C          PARM          PMHAND
C          PARM  REQUEST2  PMDATA    35
C          PARM    4          PMDLEN   10 0
C          PARM   120          PMTRY    5 0
C          PARM    1          PMCNVT    1 0
C*
C* Check result of send attempt by checking error fields.
C          IF    PMERNO <> 0
C          LEAVE
C          ENDIF
C*
C* Print sent value
C          EXCEPT  SNDLIN
C*
C* Clear error fields prior to using 'TCP_RECV'
C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
C* The following procedure call receives the response from the server
C* program for the 'REQ2' send request.
C          CALLB  'TCP_RECV'
C          PARM          PMHAND
C          PARM  *BLANKS  PMDATA    35
C          PARM    35          PMDLEN   10 0
C          PARM   120          PMTRY    5 0
C          PARM    1          PMCNVT    1 0
C          PARM  'N'          PMFIXD    1
C          PARM  *BLANKS  PMCLIP    15
C          PARM    0          PMCLPT    5 0
C*
C* Check result of receive attempt by checking error fields.
C          IF    PMERNO <> 0
C          LEAVE
C          ENDIF
C*
C* Print received value
C          EXCEPT  RCVLIN
C*
C* Clear error fields prior to using 'TCP_SEND'
C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
C* The following procedure call sends the done string to the server
C* program.
C          CALLB  'TCP_SEND'
C          PARM          PMHAND
C          PARM  DONE      PMDATA    35

```

```

C          PARM      4          PMDLEN      10 0
C          PARM     120          PMTRY       5 0
C          PARM      1          PMCNVF      1 0
C*
C* Check result of send attempt by checking error fields.
C          IF      PMERNO <> 0
C          LEAVE
C          ENDIF
C*
C* Print sent value
C          EXCEPT  SNDLIN
C*
C* Exit client program.
C          ENDDO
C*
C* The following procedure call shuts down the conversation to the
C* server program if the allocation request (TCP_CLIENT) was previously
C* successful.
C  XXCONN  IFEQ  *ON
C          CALLB  'TCP_CLOSE'
C          PARM          PMHAND
C          ENDIF
C*
C* Error text present - Print.
C  PMERTX  IFNE  *BLANKS
C          EXCEPT  ERRTXT
C          ENDIF
C*
C          MOVE  *ON          *INLR
C*
OQSYSPRT E          ERRTXT      1
O          6 'Error:'
O          PMERNO          9
O          PMERTX          81
O  E          SNDLIN      1
O          6 'Sent:'
O          PMDATA          42
O  E          RCVLIN      1
O          6 'Rcvd:'
O          PMDATA          42

```

SSL Client

```

*-----*
* Example SSL Client program using the TCP/IP To *
*-----*
F*
F* The following file is used to print status info and error conditions
FQSYSPRT O F 132 PRINTER
D*
D* The following constants are sent to server program as strings.
DTRACTEXT C CONST('Example SSL client application in-
D RPG')
DREQUEST3 C CONST('TCPIPSRC/BULKSRV0A')
D DONE C CONST('DONE')
C*
C MOVE '63.81.75.66' PMTCPA 256
C Z-ADD 8888 XXTCPH 5 0
C MOVE *OFF XXCONN
C*
C*-----C*
C* The following procedure call establishes the error variables that
C* the Toolkit API's will use when any error conditions are encountered
C CALLB 'TCP_ERRVARS'
C PARM PMERNO 2 0
C PARM PMERTX 70
C*
C*-----C*
C* The following procedure call turns on the trace for all of the
C* TCP/IP Toolkit API's.
C*
C* trace levels 0 thru 4
C* 0 = turn off trace
C* 1 = turn on trace (Documents each API command as it is used)
C* 2 = turn on trace (Outputs all trace information as in level 1
C* along with a dump of actual data sent)
C* 3 = turn on trace (Diagnostic trace, outputs same as level 1
C* along with the intermediate steps)
C* 4 = turn on trace (Outputs all data from levels 1,2,3. along
C* with the internal dump data)
C*
C CALLB 'TCP_TRACE'
C PARM 4 PMTRAC 1 0
C*
C*
C* Perform any desired error routines
C PMERNO CABNE *ZEROS EXIT
C*

```

```

C* Clear error fields prior to using 'TCP_OVRCVN'
C      CLEAR      PMERNO
C      CLEAR      PMERTX
C*
C* Change default character translation sets
C      callb  'TCP_OVRCVN'
C      parm   00437  pmascii   5 0
C      parm   00037  pmebcdec  5 0
C*
C* Perform any desired error routines
C  PMERNO  CABNE  *ZEROS  EXIT
C*
C* Clear error fields prior to using 'TCP_CLIENT'
C      CLEAR      PMERNO
C      CLEAR      PMERTX
*
* The following procedure call establishes connection and returns
* a valid handle.
C      Eval  KeyFileNm =
C              '/QIBM/USERDATA/ICSS/CERT/SERVER'+
C              '/DEFAULT.KDB'
C      Eval  KPWD = 'ABCDEFG'

C      CALLB  'TCP_CLIENT'
C      PARM      PMTCPA      256
C      PARM      XXTCPH      5 0
C      PARM      PMHAND      10 0
C*-----SSL PARMS-----
C      PARM      KeyFileNm  256
C      PARM      KPWD      256
C      PARM  'Y'  SSL_Sock   1
C      PARM  120  SSL_TimezOut 5 0
*
C* Perform any desired error routines
C  PMERNO  CABNE  *ZEROS  EXIT
C*
C      MOVE  *ON  XXCONN  1
C*
C* Loop until
C      clear      count

C  0      DOWEQ  0
C*
* Clear error fields prior to using 'TCP_RECV receive function
C      CLEAR      PMERNO
C      CLEAR      PMERTX
C*

```

```

C* The following procedure receives back request data.
C      CLEAR          BUFFER
C      Eval  buflen = 10
C      Eval  Timeout = 5
C*
C      CALLB  'TCP_RECV'
C      PARM          PMHAND      10 0
C      PARM          BUFFER      2024
C      PARM          BUFLLEN     10 0
C      PARM          TIMEOUT     5 0
C      PARM  1      CNVERT       1 0
C      PARM  'Y'    PMfixed      1
C      PARM          PMrclt      256
C      PARM          PMrhst      5 0
C*
C      if  (%subst(buffer:1:4) = 'DONE')
C      CALLB  'TCP_CLOSE'
C      PARM          PMHAND
C      LEAVE
C      endif
C*
C* Check result of receive function by checking error fields.
C      if  (pmerno <> 19) and (pmerno <> 0)
C      leave
C      endif
C*
C      add  1      count      3 0
C*
C      ENDDO
C*
C* Exit client program.
C  EXIT  TAG
C      EXCEPT  prtcnt
C*
C* The following procedure call shuts down the conversation to the
C* server program if the allocation request (TCP_CLIENT) was previously
C* successful.
C  XXCONN  IFEQ  *ON
C          CALLB  'TCP_CLOSE'
C          PARM          PMHAND
C          ENDIF
C*
C* Error text present - Print.
C  PMERTX  IFNE  *BLANKS
C          EXCEPT  ERRTXT
C          ENDIF
C*

```

```
C          MOVE  *ON      *INLR
C*
OQSYSPRT  E      ERRTXT   1
O          6 'Error:'
O          PMERNO     9
O          PMERTX     81
O  E      prtcnt     1
O          6 'Count:'
O          count     z 11
```

Server

```

*-----*
* Example server program using the TCP/IP Toolkit.      *
*-----*
F*
F* The following file is used to print status info and error conditions
FQSYSPRT O F 132 PRINTER
D*
D* The following constants are used to respond to client requests.
DTRACTEXT C CONST('Example server application in-
D RPG)
DRESPONSE1 C CONST('Server received the first request ')
DRESPONSE2 C CONST('Server received the second request')
DREQUEST1 C CONST('REQ1')
DREQUEST2 C CONST('REQ2')
DDONE C CONST('DONE')
C*
C* The following procedure call establishes the error variables that
C* the Toolkit API's will use when any error conditions are encountered
C CALLB 'TCP_ERRVARS'
C PARM PMERNO 2 0
C PARM PMERTX 70
C*
C* The following procedure call turns on the trace for all of the
C* TCP/IP Toolkit API's.
C*
C CALLB 'TCP_TRACE'
C PARM 4 PMTRAC 1 0
C*
c DO
C*
C* Clear error fields prior to using 'TCP_TRCTXT'
C CLEAR PMERNO
C CLEAR PMERTX
*
* The following procedure call establishes trace data text.
C CALLB 'TCP_TRCTXT'
C PARM TRACTEXT PMTRTX 33 CP trace text
*
C* Perform any desired error routines
C IF PMERNO <> 0
C LEAVE
C ENDIF
C*

C* Clear error fields prior to using 'TCP_SERVER'

```

```

C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
C* The following procedure call receives a connection from the
C* daemon program.
C
C          CALLB  'TCP_SERVER'
C          PARM          HANDLE          10 0
C          PARM  *BLANKS  ADDRESS        256
C          PARM  0        PORT           5 0
C          PARM  0        MAXCLNT        10 0
C          PARM  *BLANKS  NAMEDPRT       30
C*
C* Check result of connection attempt by checking error fields.
C          IF    PMERNO <> 0
C          LEAVE
C          ENDIF
C*
C*-----
C* Process client requests
C*-----
C* Loop until
C  0        DOWEQ  0
C*
C* Clear error fields prior to using 'TCP_RECV' receive function
C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
C* The following procedure receives back request data.
C          CLEAR          BUFFER
C*
C          CALLB  'TCP_RECV'
C          PARM          HANDLE          10 0
C          PARM          BUFFER          4
C          PARM  4        BUFLen        10 0
C          PARM  120      TIMEOUT        5 0
C          PARM  1        CNVERT         1 0
C          PARM  'N'      FIXEDLEN       1
C          PARM  *BLANKS  CLNTADR        15
C          PARM  0        CLNTPRT        5 0
C*
C* Check result of receive function by checking error fields.
C          IF    PMERNO <> 0
C          LEAVE
C          ENDIF
C* Send response based on information return from TCP_RECV.
C          SELECT

```

```

C*
C  BUFFER  WHENEQ  REQUEST1
C          CALLB  'TCP_SEND'
C          PARM   HANDLE      10 0
C          PARM   RESPONSE1  DATA  35
C          PARM   35    DATLEN  10 0
C          PARM   120   TIMEOUT  5 0
C          PARM   1    CNVERT   1 0
C*
C  BUFFER  WHENEQ  REQUEST2
C          CALLB  'TCP_SEND'
C          PARM   HANDLE      10 0
C          PARM   RESPONSE2  DATA  35
C          PARM   35    DATLEN  10 0
C          PARM   120   TIMEOUT  5 0
C          PARM   1    CNVERT   1 0
C*
C  BUFFER  WHENEQ  DONE
C          CALLB  'TCP_CLOSE'
C          PARM   HANDLE
C          LEAVE
C*
C          OTHER
C          CALLB  'TCP_CLOSE'
C          PARM   HANDLE
C          LEAVE
C          ENDSL
C*
C          ENDDO
C*
C          ENDDO
C*
C* Error text present - Print.
C  PMERTX  IFNE  *BLANKS
C          EXCEPT  ERRTXT
C          ENDIF
C*
C          MOVE  *ON      *INLR
C*
OQSYSPRT E          ERRTXT  1
O          6 'Error:'
O          PMERNO    9
O          PMERTX    81

```

SSL Server

```

*-----*
* Example of a SSL Multiport Server in ILE RPG      *
*-----*
F*
F* The following file is used to print status info and error conditions
FQSYSPRT O F 132 PRINTER
D*
D* The following constants are used to respond to client requests.
D TRACTEXT C          CONST('Test dropping x(0A) on Recv')
D DONE     C          CONST('DONE')
C*
C* The following procedure call establishes the error variables that
C* the Toolkit API's will use when any error conditions are encountered
C          CALLB 'TCP_ERRVARS'
C          PARM          PMERNO          2 0
C          PARM          PMERTX          7 0
C*
C* The following procedure call turns on the trace for all of the
C* TCP/IP Toolkit API's.
C*
C*TRACE Level
C*
C* 0 = turn off trace
C* 1 = turn on trace (Documents each API command as it is used)
C* 2 = turn on trace (Outputs all trace information as in level 1
C*          along with a dump of actual data sent)
C* 3 = turn on trace (Diagnostic trace, outputs same as level 1
C*          along with the intermediate steps)
C* 4 = turn on trace (Outputs all data from levels 1,2,3. along
C*          with the internal dump data)
C*
C          CALLB 'TCP_TRACE'
C          PARM 4          PMTRAC          1 0
C*
C* Clear error fields prior to using 'TCP_OVRCVN'
C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
C* Change default character translation sets
C          callb 'TCP_OVRCVN'
C          parm 00437      pmascii          5 0
C          parm 00037      pmebcdc          5 0
C*
C* Perform any desired error routines

```

```

C  PMERNO  CABNE  *ZEROS  EXIT
C*
C* Clear error fields prior to using 'TCP_SERVER'
C      CLEAR          PMERNO
C      CLEAR          PMERTX
C*
C* The following procedure call establishes a connection with the
C* server program.
C
C      eval  ipaddr = '63.81.75.66'
C      Eval  KeyFileNm =
C            '/QIBM/USERDATA/ICSS/CERT/SERVER' +
C            '/DEFAULT.KDB'
C      Eval  KPWD = 'ABCDEFG'
C*
C      CALLB  'TCP_SERVER'
C      PARM  0      HANDLE      10 0
C      PARM          IPAddr      256
C      PARM  8888   HostPort     5 0
C      PARM  1      MacClient    10 0
C      PARM  *Blanks  NamedPort  30
C      PARM          KeyFileNm   256
C      PARM          KPWD        256
C      PARM  'Y'    SSL_Sock     1
C      PARM  120   SSL_TimezOut  5 0
C*
C* Check result of connection attempt by checking error fields.
C  PMERNO  CABNE  *ZEROS  EXIT
C*
C*-----
C* Process client requests
C*-----
C* Process send request to client.
C      do  500
C* Clear error fields prior to using 'TCP_SEND'
C      CLEAR          PMERNO
C      CLEAR          PMERTX
C*
C* The following procedure call sends string to client pgm
C      eval  pmdata = '123456789' + x'25'
C*
C      CALLB  'TCP_SEND'
C      PARM          HANDLE      10 0
C      PARM          PMDATA      256
C      PARM  10      PMDLEN     10 0
C      PARM  5      PMTRY      5 0
C      PARM  1      PMCNVT     1 0

```

```

C*
C* Check result of send attempt by checking error fields.
C  PMERNO  CABNE  *ZEROS  EXIT
C          enddo
C*
C* Clear error fields prior to using 'TCP_SEND'
C          CLEAR          PMERNO
C          CLEAR          PMERTX
C*
C* The following procedure call sends the done string to the client
C* program.
C          CALLB  'TCP_SEND'
C          PARM          HANDLE
C          PARM  DONE    PMDATA
C          PARM  4        PMDLLEN    10 0
C          PARM  5        PMTRY      5 0
C          PARM  1        PMCNVT     1 0
C*
C* Check result of send attempt by checking error fields.
C  PMERNO  CABNE  *ZEROS  EXIT
C*
C* Exit client program.
C  EXIT    TAG
C*
C          CALLB  'TCP_CLOSE'
C          PARM          HANDLE
C*
C* Error text present - Print.
C  PMERTX  IFNE  *BLANKS
C          EXCEPT  ERRTXT
C          ENDIF
C*
C          MOVE  *ON    *INLR
C*
OQSYSPRT  E    ERRTXT    1
O          6 'Error:'
O          PMERNO    9
O          PMERTX    81

```

Peer to Peer Server sending Boundary Strings

```

*-----*
* Example server program sending boundary strings using the TCP/IP Toolkit. *
*-----*
*
* The following file is used to print status info and error conditions
FQSYSPRT O F 132 PRINTER
*
* Prototypes will be copied into source before compile with following COPY statement
/COPY *LIBL/TCP_SOURCE,RPG_PROTO
*
* Declared variables
d pmerno s 2 0
d pmertx s 70
d pmascii s 5 0 inz(437)
d pmebcdc s 5 0 inz(37)
d handle s 10 0
d address s 256 inz('63.81.75.66')
d hostport s 5 0 inz(8000)
d maxclnt s 10 0 inz(1)
d namedprt s 30 inz('BndyServer')
d pmdata s 14
d length s 10 0
d timeout s 5 0 inz(5)
d convert s 1 0 inz(1)
*
* Start mainline program.
c do
*
* The following procedure call establishes the error variables that
* the Toolkit API's will use when any error conditions are encountered
c callp tcp_errvars(pmerno : pmertx)
*
* The following procedure call turns on the trace for all of the
* TCP/IP Toolkit API's. In this case level 3.
c callp tcp_trace(3)
*
* Change default character translation sets
c callp tcp_ovrcvn(pascii : pmebcdc)
*
* Perform any desired error routines & exit
c if pmerno <> *zeros
c leave
c endif
*
* The following procedure call listens for a connection at the given address/port.

```

```

*
* Notice how parameters are initialized above when they are declared:
* - Address and hostport initialized with address and port to listen to. This means there is
* no Daemon involved in this process.

* - Maxclnt parameter initialized with a value of one which means it will not be in
* multiplexing mode and will accept only one connection. (like a peer to peer operation)

* - Namedprt is used and initialized with a value that matches the server program name.

c          callp  tcp_server(handle : address : hostport :
c                      maxclnt : namedprt)
*
* Check result of connection attempt by checking error fields.
c          if    pmemo <> *zeros
c          leave
c          endif

*-----
* Process client requests
*-----
*
* Process 10 send requests to client.
c          do    10
*
* The following procedure call sends a string containing 2 delimiter sets to client pgm.
* Notice the imbedded hex 0D/25 (CR/LF) and 0B (VT) in string. These will be registered
* as boundary strings in Client program.
c          eval  pmdata = *allx'F1F2F3F4F5F6F7F8F90D25F1F20B'
c          eval  length = 14
c          callp tcp_send(handle : pmdata : length : timeout :
c                      convert)
*
* Check result of send attempt by checking error fields.
c          if    pmemo <> *zeros
c          leave
c          endif
c          enddo
*
* The following procedure call sends the done string to the client
* program.
c          eval  pmdata = 'DONE' + x'0D25'
c          eval  length = 6
c          callp tcp_send(handle : pmdata : length : timeout :
c                      convert)
* Exit mainline program.
C          enddo

```

```
C      callp  tcp_close(handle)
*
* Error text present - Print.
C      if    pmertx <> *blanks
C      except errtxt
C      endif
C      eval  *inlr = *on
Oqsysprt e      errtxt      1
O              6 'Error:'
O      pmerno      9
O      pmertx      81
```

Client Receiving Strings Containing Boundaries

```

*-----*
* Example client receiving strings containing boundaries *
*-----*
*
* The following file is used to print status info and error conditions
FQSYSPRT O F 132 PRINTER
*
* Prototypes will be copied into source before compile with following COPY statement
/COPY *LIBL/SOURCE,RPG_PROTO

* Declared variables
d handle      s      10 0
d buffer      s      500
d length      s      10 0
d address     s      256 inz('as400.esdcomputers.com')
d hostport    s      5 0 inz(8000)
d boundary    s      2
d pmerno      s      2 0
d pmertx      s      70
d xxconn      s      1  inz('0')
*
* Start mainline program.
c          do
* The following procedure call establishes the error variables that
* the Toolkit API's will use when any error conditions are encountered
c          callp  tcp_errvars(pmerno : pmertx)
*
* The following procedure call turns on the trace for all of the
* TCP/IP Toolkit API's. In this case level 3.
c          callp  tcp_trace(3)
* The following procedure call establishes connection to server and returns
* a valid handle.
c          callp  tcp_client(address : hostport : handle)
*
* Check result of connection attempt by checking error fields.
c          if    pmerno <> *zeros
c          leave
c          endif
*
c          eval  xxconn = *on
*
* Register boundary characters
* register hex 0D0A = Carriage return/Line feed
c          callp  tcp_regbnd(x'0D0A')

```

```

c      if    pmerno <> *zeros
c      leave
c      endif
*
* register hex 0B = Vertical Tab and left justified
c      callp  tcp_regbnd(x'0B40')
c      if    pmerno <> *zeros
c      leave
c      endif
*
* Loop until 'DONE' text encountered
c      0      DOWEQ  0
*
* The following procedure receives data up to any registered boundary sequence
c      clear      buffer
c      eval      length = 500
c      callp  tcp_rcvbnnd(handle : buffer : length : 5 :
c              1 : 'N' : boundary)
C*
C* Check result of receive function by checking error fields.
C      if    (pmerno <> 19) and (pmerno <> 0)
c      leave
c      endif

C* Check if received DONE in data buffer
C      if    (%subst(buffer:1:4) = 'DONE')
C      leave
c      endif
c      enddo
C*
C* The following procedure flushes out remaining data
C      clear      buffer
c      eval      length = 500
c      callp  tcp_rcvbnnd(handle : buffer : length : 5 :
c              1 : 'Y' : boundary)
*
* Exit mainline program.
c      enddo
*
* The following procedure call shutdown the conversation to the
* server program if the allocation request (TCP_CLIENT) was previously
* successful.
c      if    xxconn = *ON
c      callp  tcp_close(handle)
c      endif
* Error text present - Print.
c      if    pmertx <> *blanks

```

```
c      except  errtxt
c      endif
*
c      eval   *inlr = *on
*
oqsysprt e      errtxt      1
o              6 'Error:'
o              pmerno       9
o              pmertx       81
```

Trace showing Client using TCP_RCVBND

(10.24.55.944) Scanning received string for any registered boundary characters.

(10.24.55.945) Boundary character detected

(10.24.55.973) After conversion...

| F1F2F3F4F5F6F7F8F9 | | | |

| 1 2 3 4 5 6 7 8 9 | | | |

(10.24.55.989) *** TCP_RCVBND (0,"<data>",500,5,1,N)

(10.24.55.991) Trying to receive data stream until boundary characters detected.

(10.24.55.992) Scanning received string for any registered boundary characters.

(10.24.55.993) Boundary character detected

(10.24.55.996) After conversion...

| F1F2 | | | |

| 1 2 | | | |

(10.24.56.012) *** TCP_RCVBND (0,"<data>",500,5,1,N)

(10.24.56.013) Trying to receive data stream until boundary characters detected.

(10.24.56.018) Scanning received string for any registered boundary characters.

(10.24.56.019) Boundary character detected

(10.24.56.022) After conversion...

| F1F2F3F4F5F6F7F8F9 | | | |

| 1 2 3 4 5 6 7 8 9 | | | |

Server using Multiplexing

```

*-----*
* Example server program in Multiplexing mode.      *
*-----*
*
* The following file is used to print status info and error conditions
Fqsysprt O F 132 printer
*
* Declared variables
d srvhnd      s      10 0
d handle      s      10 0
d buffer      s      500
d length      s      10 0
d address     s      256 inz('as400.esdcomputers.com')
d hostport    s      5 0 inz(8000)
d ipaddr      s      15
d pmerno      s      2 0
d pmertx      s      70
d maxclnt     s      10 0 inz(10)
d namedprt    s      30 inz('MplxServer')
*
* Prototypes will be copied into source before compile with following COPY statement
/COPY *LIBL/TCP_SOURCE,RPG_PROTO
*
* Start mainline program.
c      do
* The following procedure call establishes the error variables that
* the Toolkit API's will use when any error conditions are encountered
c      callp tcp_errvars(pmerno : pmertx)
*
* The following procedure call turns on the trace for all of the
* TCP/IP Toolkit API's. In this case level 3.
c      callp tcp_trace(3)
C*
C* Change default character translation sets
c      callp tcp_ovrcvn(437 : 37)
*
* Perform any desired error routines & exit
c      if pmerno <> *zeros
c      leave
c      endif
C*

* The following procedure call waits for a connection from a Client

```

```

* program.
* Notice how parameters are initialized above when they are declared:
* - Address and hostport initialized with address and port to listen to.
* This means there is no Daemon involved in this process.
* - Maxclnt parameter initialized with a value of 10 which means it will multiplex
* up to 10 clients at once.
* - Namedprt is used and initialized with a value that matches the server
* program name.
c      callp   tcp_server(srvhnd : address : hostport :
c                          maxclnt : namedprt)
*
* Check result of connection attempt by checking error fields.
c      if      pmerno <> *zeros
c      leave
c      endif
C*
C*-----
C* Process client requests
C*-----
*
* Wait on requests from clients and echo back. If no activity
* detected after 3 minutes expire - shutdown.
c      dow    0 = 0
*
* The following procedure receives Client data. Handle will
* be returned for Client connection being processed. Notice how
* this handle is not the same used for receiving connection.
C      clear      buffer
*
c      eval      length = 10
c      callp     tcp_rcv(handle : buffer : length : 180 :
c                  1 : 'Y' : ipaddr : hostport)

* Check result of receive attempt by checking error fields.
* Below are likely status conditions that will be received.
*
* Errno = 0 means received data
* Errno = 18 means timed out
* Errno = 19 means received data but less than 10 bytes
c      if      (pmerno <> 19) and (pmerno <> 0)
c      leave
c      endif
*
* Clear error fields prior to using 'TCP_SEND'
c      if      length > 0
* The following procedure call echos the string back to the client
* program.

```

```

c          callp  tcp_send(handle : buffer : length : 5 : 1)
*
* Check result of send attempt by checking error fields.
C          if    pmerno <> 0
c          leave
c          endif
*
c          endif
c          enddo
*
* Exit client program.
c          enddo
*
* Close socket resources for listening socket
c          callp  tcp_close(srvhnd)
*
* Error text present - Print.
c          if    pmertx <> *blanks
c          except  errtxt
c          endif
*
c          eval  *inlr = *on
*
Oqsysprt e          errtxt          1
O                  6 'Error:'
O          pmerno          9
O          pmertx          81

```

Trace from Multiplexing Server

```

(14.59.25.859) *** TCP_RECV (0,"<data>",10,180,1,Y,,)
(14.59.25.861) Trying to receive 10 bytes.
(15.00.12.149) Client connection detected from IP address 63.81.75.66 Port 7924
(15.00.12.150) Enabling socket blocking.
(15.00.12.203) Received 10 bytes.
(15.00.12.230) After conversion...
| D285A58995F1404040F1 |           |           |
| K e v i n 1   1 |           |           |
(15.00.12.262) *** TCP_SEND (1,"<data>",10,5,1)
(15.00.12.263) Before conversion...
| D285A58995F1404040F1 |           |           |
| K e v i n 1   1 |           |           |
(15.00.12.279) Disabling socket blocking.
(15.00.12.282) Sent 10 bytes.
(15.00.12.284) *** TCP_RECV (0,"<data>",10,180,1,Y,,)
(15.00.12.285) Trying to receive 10 bytes.
(15.00.12.333) Enabling socket blocking.
(15.00.12.335) Received 10 bytes.
(15.00.12.336) After conversion...
| D285A58995F1404040F2 |           |           |
| K e v i n 1   2 |           |           |
(15.00.12.352) *** TCP_SEND (1,"<data>",10,5,1)
(15.00.12.354) Before conversion...
| D285A58995F1404040F2 |           |           |
| K e v i n 1   2 |           |           |
(15.00.12.370) Disabling socket blocking.
(15.00.12.373) Sent 10 bytes.
(15.00.13.348) *** TCP_RECV (0,"<data>",10,180,1,Y,,)
(15.00.13.350) Trying to receive 10 bytes.
(15.00.19.239) Client connection detected from IP address 63.81.75.66 Port 7926
(15.00.19.240) Enabling socket blocking.
(15.00.19.242) Received 10 bytes.
(15.00.19.244) After conversion...
| D285A58995F2404040F1 |           |           |
| K e v i n 2   1 |           |           |
(15.00.19.260) *** TCP_SEND (1,"<data>",10,5,1)
(15.00.19.261) Before conversion...
| D285A58995F2404040F1 |           |           |
| K e v i n 2   1 |           |           |
(15.00.19.278) Disabling socket blocking.
(15.00.19.280) Sent 10 bytes.

```

Client talking to Server using Multiplexing

```

*-----*
* Example client program talking to Multiplexing server. *
*-----*
*
* The following file is used to print status info and error conditions
FQSYSPRT O F 132 PRINTER
* Declared variables
d handle s 10 0
d buffer s 500
d length s 10 0
d address s 256 inz('as400.esdcomputers.com')
d hostport s 5 0 inz(8000)
d ipaddr s 15
d ID s 7
d pmerno s 2 0
d pmertx s 70
d count s 3 0
d xxconn s 1 inz('0')
*
/COPY *LIBL/TCP_SOURCE,RPG_PROTO
*
* ID value received in entry parm is used to identify one occurrence
* of Client from another. Author used values of 'Client1' 'Client2'..
c *entry plist
c parm ID
*
* Start mainline program.
c do
*
* The following procedure call establishes the error variables that
* the Toolkit API's will use when any error conditions are encountered
c callp tcp_errvars(pmerno : pmertx)
*
* The following procedure call turns on the trace for all of the
* TCP/IP Toolkit API's. In this case level 3.
c callp tcp_trace(3)
*
* The following procedure call places the ID in trace data text.
c callp tcp_trctxt(ID)
C* PARM ID PMTRTX 39 TCP trace text
*
* Change default character translation sets
c* callp tcp_ovrcvn(pascii : pmebcd)
c callp tcp_ovrcvn(437 : 37)
* Perform any desired error routines & exit

```

```

c      if    pmerno <> *zeros
c      leave
c      endif
*
* The following procedure call establishes connection and returns
* a valid handle.
c      callp  tcp_client(address : hostport : handle)
*
* Check result of connection attempt by checking error fields.
c      if    pmerno <> *zeros
c      leave
c      endif
*
c      eval  xxconn = *on
*
* Send and then receive back echoed string 10 times
c      DO    10
* Catenating string to send from ID and adding a count
C      eval  count = count + 1
c      eval  buffer = ID + %editc(count : '3')
* Send data buffer
c      eval  length = 10
c      callp  tcp_send(handle : buffer : length : 5 : 1)
* Receive data buffer echoed back from server
c      clear  buffer
c      eval  length = 10
c      callp  tcp_rcv(handle : buffer : length : 5 :
c              1 : 'Y' : ipaddr : Hostport)
*
* Check result of receive function by checking error fields.
c      if    (pmerno <> 19) and (pmerno <> 0)
c      leave
c      endif

C      enddo
*
* Exit mainline program.
c      enddo
*
* The following procedure call shuts down the conversation to the
* server program if the allocation request (TCP_CLIENT) was previously
* successful.
c      if    xxconn = *ON
c      callp  tcp_close(handle)
c      endif
* Error text present - Print.
c      if    pmertx <> *blanks

```

```
c      except  errtxt
c      endif
*
c      eval   *inlr = *on
*
oqsysprt e      errtxt      1
o              6 'Error:'
o              pmerno       9
o              pmertx       81
```

Trace from Client #1 sending to Multiplexing Server

```

(15.00.12.154) *** TCP_SEND (0,"<data>",10,5,1)
(15.00.12.156) Before conversion...
| D285A58995F1404040F1 |
| K e v i n 1 1 |
(15.00.12.198) Disabling socket blocking.
(15.00.12.201) Sent 10 bytes.
(15.00.12.214) *** TCP_RECV (0,"<data>",10,5,1,Y,,)
(15.00.12.215) Trying to receive 10 bytes.
(15.00.12.217) Enabling socket blocking.
(15.00.12.288) Received 10 bytes.
(15.00.12.289) After conversion...
| D285A58995F1404040F1 |
| K e v i n 1 1 |
(15.00.12.305) *** TCP_SEND (0,"<data>",10,5,1)
(15.00.12.306) Before conversion...
| D285A58995F1404040F2 |
| K e v i n 1 2 |
(15.00.12.323) Disabling socket blocking.
(15.00.12.326) Sent 10 bytes.

```

Trace from Client #2 sending to Multiplexing Server

(15.00.19.210) *** TCP_SEND (0,"<data>",10,5,1)
(15.00.19.211) Before conversion...
| D285A58995F2404040F1 |
| K e v i n 2 1 |
(15.00.19.227) Disabling socket blocking.
(15.00.19.230) Sent 10 bytes.
(15.00.19.232) *** TCP_RECV (0,"<data>",10,5,1,Y,,)
(15.00.19.233) Trying to receive 10 bytes.
(15.00.19.234) Enabling socket blocking.
(15.00.19.286) Received 10 bytes.
(15.00.19.288) After conversion...
| D285A58995F2404040F1 |
| K e v i n 2 1 |
(15.00.19.303) *** TCP_SEND (0,"<data>",10,5,1)
(15.00.19.305) Before conversion...
| D285A58995F2404040F2 |
| K e v i n 2 2 |
(15.00.19.322) Disabling socket blocking.
(15.00.19.324) Sent 10 bytes.

COBOL ILE EXAMPLES

Daemon

```
*-----*
Example of Daemon using ILE COBOL
*-----*
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "CBL_DAEMON".
AUTHOR. Wayside Marketing, Inc..
```

```
*
```

```
*****
```

```
*remarks: This example program illustrates the use of
* "LINKAGE TYPE IS PRC" using Special Names paragraph
* in the Configuration Section of the Environment
* Division.
```

```
*
```

```
* The "USING ALL DESCRIBED" clause is required to
* pass operational descriptors for all parameters.
```

```
*
```

```
*remarks: This module must be compiled with the
* following OPTIONS:
```

```
* *NOMONOPRC
```

```
* *STDINZ
```

```
* LINKLIT
```

```
* *PGM
```

```
*****
```

```
*
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-AS400.
OBJECT-COMPUTER. IBM-AS400.
SPECIAL-NAMES. LINKAGE TYPE IS PRC FOR "TCP_ERRVARS"
USING ALL DESCRIBED
LINKAGE TYPE IS PRC FOR "TCP_TRCTXT"
```

USING ALL DESCRIBED
 LINKAGE TYPE IS PRC FOR "TCP_TRACE"
 USING ALL DESCRIBED
 LINKAGE TYPE IS PRC FOR "TCP_DAEMON"
 USING ALL DESCRIBED.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 WORK-AREAS.

```

* work TCP/IP trace text
  05 PMTRACETXT          PIC X(50)  VALUE
      "Example cobol program establishing daemon   ".
* TCP/IP error number
  05 PMERNO              PIC S9(2)  VALUE ZEROES
                          COMP-3.
* TCP/IP error text
  05 PMERTX              PIC X(70)  VALUE SPACES.
* TCP/IP trace level
  05 PMTRACE             PIC S9(1)  VALUE 4
                          COMP-3.
* work TCP/IP Ip address
  05 PMIPADDRESS-X      PIC X(256) VALUE SPACES.
* work TCP/IP host port
  05 PMPORT-X           PIC S9(5)  VALUE ZEROES
                          COMP-3.
* work TCP/IP host port
  05 PMMAXSERVERS      PIC S9(10) VALUE 10
                          COMP-3.
* work TCP/IP job description
  05 PMJOBDESCRIPTION   PIC X(21)  VALUE SPACES.
* work TCP/IP server name
  05 PMSERVERNAME      PIC X(21)  VALUE SPACES.
* Named Port
  05 PMNAMEDPORT       PIC X(30)  VALUE SPACES.
* Minimum Servers
  05 PMMINSERVERS      PIC S9(10) VALUE 1 COMP-3.
LINKAGE SECTION.
* work TCP/IP Ip address
  01 PMIPADDRESS        PIC X(256).
* work TCP/IP host port
  01 PMPORT             PIC S9(10)V99999 COMP-3.

```

PROCEDURE DIVISION USING PMIPADDRESS, PMPORT.

```

MAIN-RTN.
  MOVE PMIPADDRESS      TO PMIPADDRESS-X.
  MOVE PMPORT           TO PMPORT-X.
  PERFORM START-PROGRAM THRU EXIT-PROGRAM.
  STOP RUN.
START-PROGRAM.
  
```

```

* The following procedure call establishes the error variables *
* The toolkit api's will use when any error conditions are   *
* encountered.                                             *
  
```

```

  INITIALIZE PMERNO, PMERTX.
  CALL LINKAGE TYPE IS PRC "TCP_ERRVARS" USING PMERNO
  PMERTX.
  
```

```

* The following procedure call turns on the internal trace *
  
```

```

  INITIALIZE PMERNO, PMERTX.
  CALL LINKAGE TYPE IS PRC "TCP_TRACE" USING BY REFERENCE
  PMTRACE.
  CALL LINKAGE TYPE IS PRC "TCP_TRCTXT" USING BY REFERENCE
  PMTRACETXT.
  
```

```

* The following procedure call starts the daemon program. *
  
```

```

  INITIALIZE PMERNO, PMERTX.
  CALL LINKAGE TYPE IS PRC "TCP_DAEMON" USING BY REFERENCE
  PMIPADDRESS-X,
  PMPORT-X,
  PMMAXSERVERS,
  PMSERVERNAME,
  PMJOBDESCRIPTION,
  PMNAMEDPORT,
  PMMINSERVERS.
  
```

```

* Check result of connection attempt by checking error fields. *
  
```

```

  IF PMERNO NOT = ZEROES
  *   perform appropriate error handling procedures
  GO TO EXIT-PROGRAM.
EXIT-PROGRAM.
EXIT.
  
```

Shutdown Daemon

```
*-----*
Example to shut down Daemons in ILE COBOL
*-----*
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "CBL_ENDDMN".
AUTHOR. Wayside Marketing, Inc.
```

```
*
```

```
*****
```

```
*remarks: This example program illustrates the use of
* "LINKAGE TYPE IS PRC" using Special Names paragraph
* in the Configuration Section of the Environment
* Division.
```

```
*
```

```
* The "USING ALL DESCRIBED" clause is required to
* pass operational descriptors for all parameters.
```

```
*
```

```
*remarks: This module must be compiled with the
* following OPTIONS:
```

```
* *NOMONOPRC
* *STDINZ
* LINKLIT
* *PGM
```

```
*****
```

```
*
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-AS400.
OBJECT-COMPUTER. IBM-AS400.
SPECIAL-NAMES. LINKAGE TYPE IS PRC FOR "TCP_ERRVARS"
                USING ALL DESCRIBED
                LINKAGE TYPE IS PRC FOR "TCP_TRCTXT"
                USING ALL DESCRIBED
                LINKAGE TYPE IS PRC FOR "TCP_TRACE"
                USING ALL DESCRIBED
                LINKAGE TYPE IS PRC FOR "TCP_ENDDMN"
                USING ALL DESCRIBED.
```

```
INPUT-OUTPUT SECTION.
```

```
FILE-CONTROL.
```

```
DATA DIVISION.
```

```
FILE SECTION.
```

```
WORKING-STORAGE SECTION.
```

```
01 WORK-AREAS.
```

```
* work TCP/IP trace text
```

```

05 PMTRACETXT          PIC X(50)  VALUE
    "Example cobol program demonstrating daemon end ".
* TCP/IP error number
05 PMERNO              PIC S9(2)  VALUE ZEROES
                        COMP-3.
* TCP/IP error text
05 PMERTX              PIC X(70)  VALUE SPACES.
* TCP/IP trace level
05 PMTRACE             PIC S9(1)  VALUE 4
                        COMP-3.
05 PMPORT-X           PIC S9(5)  VALUE ZEROES
                        COMP-3.

```

LINKAGE SECTION.

```

* Address
01 PMADDRESS           PIC X(256).
* work TCP/IP host port
01 PMPORT              PIC S9(10)V99999
                        COMP-3.

```

PROCEDURE DIVISION USING PMADDRESS, PMPORT.

MAIN-RTN.

```

    MOVE PMPORT TO PMPORT-X.
    PERFORM START-PROGRAM THRU EXIT-PROGRAM.
    STOP RUN.

```

START-PROGRAM.

```

* The following procedure call establishes the error variables *
* The toolkit api's will use when any error conditions are *
* encountered. *

```

```

    INITIALIZE PMERNO, PMERTX.
    CALL LINKAGE TYPE IS PRC "TCP_ERRVARS" USING BY REFERENCE
        PMERNO, PMERTX.

```

```

* The following procedure call turns on the internal trace *

```

```

    INITIALIZE PMERNO, PMERTX.
    CALL LINKAGE TYPE IS PRC "TCP_TRACE" USING BY REFERENCE
        PMTRACE.
    INITIALIZE PMERNO, PMERTX.
    CALL LINKAGE TYPE IS PRC "TCP_TRCTXT" USING BY REFERENCE
        PMTRACETXT.

```

```

* The following procedure call ends the daemon program. *

```

```

    INITIALIZE PMERNO, PMERTX.

```

CALL LINKAGE TYPE IS PRC "TCP_ENDDMN" USING BY REFERENCE
PMADDRESS,
PMPORT-X.

* Check result of connection attempt by checking error fields. *

IF PMERNO NOT = ZEROES

* perform appropriate error handling procedures

GO TO EXIT-PROGRAM.

GO TO EXIT-PROGRAM.

EXIT-PROGRAM.

EXIT.

Client

```
*-----*
Example to be a Client in ILE COBOL
*-----*
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "CBL_CLIENT".
AUTHOR. Wayside Marketing, Inc.
```

```
*
*****
*remarks: This example program illustrates the use of
* "LINKAGE TYPE IS PRC" using Special Names paragraph
* in the Configuration Section of the Environment
* Division.
*
* The "USING ALL DESCRIBED" clause is required to
* pass operational descriptors for all parameters.
*
*remarks: This module must be compiled with the
* following OPTIONS:
* *NOMONOPRC
* *STDINZ
* LINKLIT
* *PGM
*****
```

```
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-AS400.
OBJECT-COMPUTER. IBM-AS400.
SPECIAL-NAMES. LINKAGE TYPE IS PRC FOR "TCP_ERRVARS"
                USING ALL DESCRIBED
                LINKAGE TYPE IS PRC FOR "TCP_CLIENT"
                USING ALL DESCRIBED
                LINKAGE TYPE IS PRC FOR "TCP_RECV"
                USING ALL DESCRIBED
                LINKAGE TYPE IS PRC FOR "TCP_SEND"
                USING ALL DESCRIBED
                LINKAGE TYPE IS PRC FOR "TCP_TRACE"
                USING ALL DESCRIBED
                LINKAGE TYPE IS PRC FOR "TCP_TRCTXT"
                USING ALL DESCRIBED
                LINKAGE TYPE IS PRC FOR "TCP_CLOSE"
                USING ALL DESCRIBED.
```

INPUT-OUTPUT SECTION.

FILE-CONTROL.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 WORK-AREAS.

```

05 PMTRTXT          PIC X(35)  VALUE
                    "Example client application in Cobol".
05 PMDATA           PIC X(21)  VALUE SPACES.
05 XXTCPH           PIC S9(5)  VALUE ZEROES
                    COMP-3.
05 PMERNO           PIC S9(2)  VALUE ZEROES
                    COMP-3.
05 PMERTX           PIC X(70)  VALUE SPACES.
05 PMHAND           PIC S9(10) VALUE ZEROES
                    COMP-3.
05 PMDLEN           PIC S9(10) VALUE ZEROES
                    COMP-3.
05 PMTIMEOUT        PIC S9(05) VALUE 120
                    COMP-3.
05 PMCNVT           PIC S9(01) VALUE 1
                    COMP-3.
05 PMTRACE          PIC S9(01) VALUE 4
                    COMP-3.
05 PMFIXEDLEN       PIC X      VALUE 'N'.
05 PMCLIENTADR      PIC X(15)  VALUE SPACES.
05 PMCLIENTPORT     PIC S9(5)  VALUE ZEROS
                    COMP-3.
05 PMRECEIVE        PIC X(35)  VALUE SPACES.
05 PMRECEIVE7       PIC X(7)   VALUE SPACES.
05 PMSEND           PIC X(4)   VALUE SPACES.
05 PMIPADDRESS-X    PIC X(256) VALUE SPACES.
05 PMPORT-X         PIC S9(5)  VALUE ZEROES
                    COMP-3.
    
```

LINKAGE SECTION.

```

01 PMIPADDRESS      PIC X(256).
01 PMPORT           PIC S9(10)V99999 COMP-3.
    
```

PROCEDURE DIVISION USING PMIPADDRESS, PMPORT.

MAIN-RTN.

```

MOVE PMIPADDRESS TO PMIPADDRESS-X.
MOVE PMPORT      TO PMPORT-X.
PERFORM START-PROGRAM THRU EXIT-PROGRAM.
STOP RUN.
START-PROGRAM.
    
```

```

* The following procedure call establishes the error variables *
* The toolkit api's will use when any error conditions are *
* encountered. *
*****
      INITIALIZE PMERNO, PMERTX.
      CALL LINKAGE TYPE IS PRC "TCP_ERRVARS" USING BY REFERENCE
            PMERNO,
            PMERTX.
*****
* The following procedure call turns trace on *
*****
      INITIALIZE PMHAND.
      CALL LINKAGE TYPE IS PRC "TCP_TRACE" USING BY REFERENCE
            PMTRACE.
      CALL LINKAGE TYPE IS PRC "TCP_TRCTXT" USING BY REFERENCE
            PMTRTXT.
*****
* The following procedure call establishes a connection with *
* the daemon program. *
*****
      CALL LINKAGE TYPE IS PRC "TCP_CLIENT" USING BY REFERENCE
            PMPADDRESS-X,
            PMPORT-X,
            PMHAND.
*****
* Check result of the daemon connection checking error fields. *
*****
      IF PMERNO NOT = ZEROES
      GO TO EXIT-PROGRAM.
      DISPLAY "CLIENT CONNECTION TO DAEMON COMPLETE ".
*****
* start client server conversation *
*****
      PROCESS-REQUESTS.
      INITIALIZE PMERNO,
            PMERTX,
            PMDATA.
*****
* The following procedure call starts server *
*****
      MOVE 21          TO PMDLN.
      MOVE "CBL_SERVER" TO PMDATA.
      CALL LINKAGE TYPE IS PRC "TCP_SEND" USING BY REFERENCE
            PMHAND
            PMDATA,
            PMDLN,
            PMTIMEOUT,

```

PMCNVT.

* Check result of the start server send *

```

PERFORM CHECK-FOR-ERRORS THRU CHECK-FOR-ERRORS-EXIT.
MOVE 7          TO PMDLEN.
INITIALIZE PMRECEIVE7, PMSSEND.
CALL LINKAGE TYPE IS PRC "TCP_RECV" USING BY REFERENCE
          PMHAND,
          PMRECEIVE7,
          PMDLEN,
          PMTIMEOUT,
          PMCNVT,
          PMFIXEDLEN,
          PMCLIENTADR,
          PMCLIENTPORT.
    
```

* Check result of the start server receive status *

```

PERFORM CHECK-FOR-ERRORS THRU CHECK-FOR-ERRORS-EXIT.
    
```

* Check result of the server connection *

```

IF PMRECEIVE7 NOT = "SUCCESS"
PERFORM CHECK-FOR-ERRORS THRU CHECK-FOR-ERRORS-EXIT.
DISPLAY "CLIENT AND SERVER CONNECTION COMPLETE".
    
```

* Send first request to server *

```

MOVE 4          TO PMDLEN.
MOVE "REQ1"     TO PMSSEND.
CALL LINKAGE TYPE IS PRC "TCP_SEND" USING BY REFERENCE
          PMHAND,
          PMSSEND,
          PMDLEN,
          PMTIMEOUT,
          PMCNVT.
PERFORM CHECK-FOR-ERRORS THRU CHECK-FOR-ERRORS-EXIT.
DISPLAY "REQUEST 1 " PMDLEN " BYTES SENT FROM CLIENT".
MOVE 35        TO PMDLEN.
INITIALIZE     PMRECEIVE.
CALL LINKAGE TYPE IS PRC "TCP_RECV" USING BY REFERENCE
          PMHAND,
          PMRECEIVE
          PMDLEN,
          PMTIMEOUT,
    
```

```

                                PMCNVT,
                                PMFIXEDLEN,
                                PMCLIENTADR,
                                PMCLIENTPORT.
    PERFORM CHECK-FOR-ERRORS THRU CHECK-FOR-ERRORS-EXIT.
    DISPLAY "REQUEST 1 " PMDLEN " RECEIVED FROM SERVER".
*****
* Send second request to server                                *
*****
    MOVE 4                      TO PMDLEN.
    MOVE "REQ2"                  TO PMSEND.
    CALL LINKAGE TYPE IS PRC "TCP_SEND" USING BY REFERENCE
                                PMHAND,
                                PMSEND,
                                PMDLEN,
                                PMTIMEOUT,
                                PMCNVT.

    PERFORM CHECK-FOR-ERRORS THRU CHECK-FOR-ERRORS-EXIT.
    DISPLAY "REQUEST 2 " PMDLEN " BYTES SENT FROM CLIENT".
    MOVE 35                      TO PMDLEN.
    INITIALIZE                    PMRECEIVE.
    CALL LINKAGE TYPE IS PRC "TCP_RECV" USING BY REFERENCE
                                PMHAND,
                                PMRECEIVE
                                PMDLEN,
                                PMTIMEOUT,
                                PMCNVT,
                                PMFIXEDLEN,
                                PMCLIENTADR,
                                PMCLIENTPORT.

    PERFORM CHECK-FOR-ERRORS THRU CHECK-FOR-ERRORS-EXIT.
    DISPLAY "REQUEST 2 " PMDLEN " RECEIVED FROM SERVER".
*****
* Send done request to server                                *
*****
    MOVE 4                      TO PMDLEN.
    MOVE "DONE"                  TO PMSEND.
    CALL LINKAGE TYPE IS PRC "TCP_SEND" USING BY REFERENCE
                                PMHAND,
                                PMSEND,
                                PMDLEN,
                                PMTIMEOUT,
                                PMCNVT.

    PERFORM CHECK-FOR-ERRORS THRU CHECK-FOR-ERRORS-EXIT.
    DISPLAY "DONE " PMDLEN " BYTES SENT".
    CALL LINKAGE TYPE IS PRC "TCP_CLOSE" USING BY REFERENCE
                                PMHAND.

```

```
GO TO EXIT-PROGRAM.
EXIT-PROGRAM.
EXIT.
CHECK-FOR-ERRORS.
*****
* Check result of send attempt by checking error fields.      *
*****
IF PMERNO NOT = ZEROES
*   perform appropriate error handling procedures
  CALL LINKAGE TYPE IS PRC "TCP_CLOSE" USING BY REFERENCE
    PMHAND
GO TO EXIT-PROGRAM.
CHECK-FOR-ERRORS-EXIT.
EXIT.
```

Server

```
*-----*
  Example to be a Server in ILE COBOL
*-----*
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  "CBL_SERVER".
AUTHOR.     Wayside Marketing, Inc.
```

```
*
*****
```

```
*remarks:  This example program illustrates the use of
*          "LINKAGE TYPE IS PRC" using Special Names paragraph
*          in the Configuration Section of the Environment
*          Division.
```

```
*          The "USING ALL DESCRIBED" clause is required to
*          pass operational descriptors for all parameters.
```

```
*remarks:  This module must be compiled with the
*          following OPTIONS:
```

```
*          *NOMONOPRC
*          *STDINZ
*          LINKLIT
*          *PGM
```

```
*****
*
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-AS400.
OBJECT-COMPUTER. IBM-AS400.
SPECIAL-NAMES.  LINKAGE TYPE IS PRC FOR "TCP_ERRVARS"
                  USING ALL DESCRIBED
                  LINKAGE TYPE IS PRC FOR "TCP_SERVER"
                  USING ALL DESCRIBED
                  LINKAGE TYPE IS PRC FOR "TCP_RECV"
                  USING ALL DESCRIBED
                  LINKAGE TYPE IS PRC FOR "TCP_SEND"
                  USING ALL DESCRIBED
                  LINKAGE TYPE IS PRC FOR "TCP_TRACE"
                  USING ALL DESCRIBED
                  LINKAGE TYPE IS PRC FOR "TCP_TRCTXT"
                  USING ALL DESCRIBED
                  LINKAGE TYPE IS PRC FOR "TCP_CLOSE"
                  USING ALL DESCRIBED.
```

```
INPUT-OUTPUT SECTION.
```

FILE-CONTROL.
 DATA DIVISION.
 FILE SECTION.
 WORKING-STORAGE SECTION.

* The following constants are sent to the server program as strings.

```

77 RESPONSE1          PIC X(35) VALUE
    "Server received the first request. ".
77 RESPONSE2          PIC X(35) VALUE
    "Server received the second request.".
01 WORK-AREAS.
  05 PMTRTXT          PIC X(35) VALUE
    "Example server application in Cobol".
  05 PMDATA           PIC X(35) VALUE SPACES.
* work TCP/IP host port
  05 XXTCPH           PIC S9(5) VALUE ZEROES
    COMP-3.
* TCP/IP error number
  05 PMERNO           PIC S9(2) VALUE ZEROES
    COMP-3.
* TCP/IP error text
  05 PMERTX           PIC X(70) VALUE SPACES.
* TCP handle
  05 PMHAND           PIC S9(10) VALUE ZEROES
    COMP-3.
* send data length
  05 PMDLEN           pic s9(10) VALUE ZEROES
    COMP-3.
* time out
  05 PMTIMEOUT        PIC S9(05) VALUE ZEROES
    COMP-3.
  05 PMCNVT           PIC S9(01) VALUE ZEROES
    COMP-3.
  05 PMTRACE          PIC S9(01) VALUE 4
    COMP-3.
  05 PMADDRESS        PIC X(256) VALUE SPACES.
  05 PMHOSTPORT       PIC S9(5) VALUE ZEROES
    COMP-3.
  05 PMMAXCLIENT      PIC S9(10) VALUE ZEROES
    COMP-3.
  05 PMNAMEDPORT      PIC X(30) VALUE SPACES.
  05 PMFIXEDLEN        PIC X VALUE IS 'N'.
  05 PMCLIENTADR      PIC X(15) VALUE SPACES.
  05 PMCLIENTPORT     PIC S9(5) VALUE ZEROS
    COMP-3.

```

LINKAGE SECTION.

```

*****
PROCEDURE DIVISION.
*****

MAIN-RTN.
    PERFORM START-PROGRAM THRU EXIT-PROGRAM.
    STOP RUN.
START-PROGRAM.
*****
* The following procedure call establishes the error variables *
* The toolkit api's will use when any error conditions are *
* encountered. *
*****
    INITIALIZE PMERNO, PMERTX.
    CALL LINKAGE TYPE IS PRC "TCP_ERRVARS" USING BY REFERENCE
        PMERNO,
        PMERTX.
*****
* The following procedure call establishes a connection with *
* the server program. *
*****
    INITIALIZE PMHAND.
    CALL LINKAGE TYPE IS PRC "TCP_TRACE" USING BY REFERENCE
        PMTRACE.
    CALL LINKAGE TYPE IS PRC "TCP_TRCTXT" USING BY REFERENCE
        PMTRTXT.
    CALL LINKAGE TYPE IS PRC "TCP_SERVER" USING BY REFERENCE
        PMHAND,
        PMADDRESS,
        PMHOSTPORT,
        PMMAXCLIENT,
        PMNAMEDPORT.
*****
* Check result of connection attempt by checking error fields. *
*****
    IF PMERNO NOT = ZEROES
*       perform appropriate error handling procedures
        CALL LINKAGE TYPE IS PRC "TCP_CLOSE" USING BY REFERENCE
            PMHAND
        GO TO EXIT-PROGRAM.
    PERFORM PROCESS-REQUESTS THRU
        PROCESS-REQUESTS-EXIT.
*****
* loop to send/recieve client data *
*****
PROCESS-REQUESTS.
    INITIALIZE PMERNO,
        PMERTX,

```

PMDATA.

* The following procedure call receives data from client *

```

MOVE 4      TO PMDLEN.
MOVE 120    TO PMTIMEOUT.
MOVE 1      TO PMCNVT.
CALL LINKAGE TYPE IS PRC "TCP_RECV" USING BY REFERENCE
                PMHAND,
                PMDATA,
                PMDLEN,
                PMTIMEOUT,
                PMCNVT,
                PMFIXEDLEN,
                PMCLIENTADR,
                PMCLIENTPORT.

```

IF PMERNO NOT = ZEROES

```

* perform error handler routine
CALL LINKAGE TYPE IS PRC "TCP_CLOSE" USING BY REFERENCE
                PMHAND
GO TO EXIT-PROGRAM.

```

IF PMDATA = "REQ1"

```

MOVE RESPONSE1 TO PMDATA
PERFORM SEND-CLIENT-DATA THRU SEND-CLIENT-DATA-EXIT
GO TO PROCESS-REQUESTS.

```

IF PMDATA = "REQ2"

```

MOVE RESPONSE2 TO PMDATA
PERFORM SEND-CLIENT-DATA THRU SEND-CLIENT-DATA-EXIT
GO TO PROCESS-REQUESTS.

```

IF PMDATA = "DONE"

```

CALL LINKAGE TYPE IS PRC "TCP_CLOSE" USING BY REFERENCE
                PMHAND
GO TO EXIT-PROGRAM.

```

```

* perform error handler routine
CALL LINKAGE TYPE IS PRC "TCP_CLOSE" USING BY REFERENCE
                PMHAND

```

GO TO EXIT-PROGRAM.

PROCESS-REQUESTS-EXIT.

EXIT.

SEND-CLIENT-DATA.

```

INITIALIZE PMERNO, PMERTX.
MOVE 35    TO PMDLEN.
MOVE 120   TO PMTIMEOUT.
MOVE 1     TO PMCNVT.
CALL LINKAGE TYPE IS PRC "TCP_SEND" USING BY REFERENCE
                PMHAND,

```

PMDATA,
PMDLEN,
PMTIMEOUT,
PMCNVT.

IF PMERNO NOT = ZEROES

* perform error handler routine
CALL LINKAGE TYPE IS PRC "TCP_CLOSE" USING BY REFERENCE
PMHAND

GO TO EXIT-PROGRAM.

SEND-CLIENT-DATA-EXIT.

EXIT.

EXIT-PROGRAM.

EXIT.

C ILE EXAMPLES

Daemon

```

*-----*
      Example to establish a Daemon in ILE C
*-----*

/*****
*****/
/* The TCP/IP Toolkit requires pointers to be passed for all expected parameters. ILE
COBOL & */
/* ILE RPG pass variables by reference so the variable can be passed normally. But, C of
course */
/* passes by value, so the C programmer must explicitly pass the variables' addresses as
shown. */
/*****
*****/
#include <stdio.h>
#include <decimal.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
/*PROTOTYPES*/
void TCP_ERRVARS(decimal(2) *, char *);
void TCP_TRACE(decimal(1) *);
void TCP_TRCXT(char *);
void TCP_DAEMON(char *, decimal(5) *, decimal(10) *, char *, char *, char *, decimal(10)
*);
int DisplayError(void);
decimal(2) ErrorNum=0; /* Variables to have error info placed */
char *ErrorMsg=" ";
char *ServerAddress; /* Retrieved from the first argument */
decimal(5) ServerHostPort; /* Retrieved from the second argument */
decimal(10) MaxServers=3; /* Don't allow more than 03 servers to be started */
decimal(5) Tries=120; /* Try up to 2 minutes */

```

```

decimal(1) Convert=TRUE;                                /* No Conversion */
decimal(1) Trace=4;                                    /* Diagnostic trace */
decimal(10) MinServ=1;                                  /* Pre-Start 1 server */
char    *NamedPrt="";
int main(int argc, char *argv??(??))
{ if(argc!=3)
  { printf("This program requires two strings as parameters. \n");
    printf("  parm 1 : IP address in dotted decimal format. \n");
    printf("  parm 2 : Host port to listen for clients upon. \n");
    return(TRUE);
  }
  ServerAddress=argv??(1??);                            /* first parm is the address */
  ServerHostPort= (decimal(5))*(decimal(15,5) *)argv??(2??); /* second parm is the port */
  printf("\n\nTCP/IP Programmer's Toolkit for the AS/400\n");
  printf("Example C program demonstrating how to establish a daemon. \n\n");
  printf("Address = %s, Port = %D(5) \n",ServerAddress,ServerHostPort);
  TCP_ERRVARS(&ErrorNum,ErrorMsg);                      /* Tell the toolkit the error variables */
  TCP_TRACE(&Trace);                                    /* Turn the internal trace on - Optional */
  TCP_TRCTXT("Example daemon application written in ILE C.");
  TCP_DAEMON(ServerAddress,&ServerHostPort,&MaxServers,"
,NamedPrt,
    &MinServ);
  return(DisplayError());
}
/***** */
int DisplayError(void)                                  /* If there was an error, print it and return true */
{ if(ErrorNum)
  { printf("ERR=%-70s'\n",ErrorMsg);                    /* Print the Error Message to
STDOUT */
    return(TRUE);                                       /* Tell main that there was an error */
  }
  return(FALSE);                                       /* Tell main that there was no error */
}

```

Shutdown Daemon

```

*-----*
      Example to shut down Daemons in ILE C
*-----*

/*****
*****/
/* The TCP/IP Toolkit requires pointers to be passed for all expected parameters. ILE
COBOL & */
/* ILE RPG pass variables by reference so the variable can be passed normally. But, C of
course */
/* passes by value, so the C programmer must explicitly pass the variables' addresses as
shown. */
/*****
*****/
#include <stdlib.h>
#include <stdio.h>
#include <decimal.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
/*PROTOTYPES*/
void TCP_ERRVARS(decimal(2) *, char *);
void TCP_ENDDMN(char *, decimal(5) *);
int DisplayError(void);
/*GLOBALS*/
decimal(2) ErrorNum=0;          /* Variables to have error info placed */
char *ErrorMsg="";
char *ServerAddress;          /* Retrieved from the argument */
decimal(5) ServerHostPort;    /* Retrieved from the argument */
int main(int argc,char *argv[])
{ if(argc!=3)
  { printf("This program requires two parameters. \n");
    printf("  parm 1 : Address of the Daemon to end. \n");
    printf("  parm 2 : Host Port of the Daemon to end. \n");
    return(TRUE);
  }
  ServerAddress= argv??(1??);
  ServerHostPort= (decimal(5))*(decimal(15,5) *argv??(2??));
  TCP_ERRVARS(&ErrorNum,ErrorMsg); /* Tell the toolkit the error variables */
  TCP_ENDDMN(ServerAddress,&ServerHostPort); /* Tell the daemon to shut down */
  return(DisplayError());
}

/*****
*****/

```

```
int DisplayError(void)
{ if(ErrorNum)
  { ErrorMessage='\0';          /* Add the NULL to print the error */
  printf("ERR='%-70s'\n",ErrorMessage); /* Print the Error Message to STDOUT */
return(TRUE);                  /* Tell main that there was an error */
  }
  return(FALSE);               /* Tell main that there was no error */
}

/*****/
```

Client

```

*-----*
      Example to be a Client in ILE C
*-----*

/*****
*****/
/* The TCP/IP Toolkit requires pointers to be passed for all expected parameters. ILE
COBOL & */
/* ILE RPG pass variables by reference so the variable can be passed normally. But, C of
course */
/* passes by value, so the C programmer must explicitly pass the variables' addresses as
shown. */
/*****
*****/
/* This program is hard-coded to start a server named C_SERVER located somewhere in the
library */
/* list. 21 bytes are sent containing the text: C_SERVER. The text MUST be converted to
ASCII */
/* because that is what the daemon expects. A reply of "SUCCESS" (also ASCII) confirms
that the */
/* server was properly started.
*/
/*****
*****/
#include <stdio.h>
#include <stdlib.h>
#include <decimal.h>
#include <string.h>
#define FALSE 0
#define TRUE 1
#define SEND 0
#define RECV 1
/* NetSocket V3.0.1 Prototypes */
void TCP_CLIENT(char *,decimal(5) *, decimal(10) *);
void TCP_CLOSE(decimal(10) *);
void TCP_TRACE(decimal(1) *);
void TCP_TRCTXT(char *);
void TCP_ERRVARS(decimal(2) *,char *);
void TCP_SEND(decimal(10) *,char *,decimal(10) *,decimal(5) *,decimal(1) *);
void TCP_RECV(decimal(10) *,char *,decimal(10) *,decimal(5) *,decimal(1) *,char *,char
*,decimal(5) *);
/*PROTOTYPES OF LOCAL FUNCTIONS*/
void TransferData(int,char *,int);
int DisplayError(void);

```

```

decimal(2) ErrorNum=0;          /* variables to have error info placed */
char   *ErrorMsg="";
decimal(1) Trace=4;           /* Diagnostic Trace */
decimal(10) Handle;           /* Each conversation has it's own handle */
char   *ServerAddress;        /* IP Address of the daemon & server */
decimal(5) ServerHostPort;    /* Port to the daemon is listening to */
char   *DataIn="";           /* Storage to receive all inbound. */
                                /* It must must also be initialized to non-NULLs so that */
                                /* strlen() correctly returns the length of the data field.*/

int main(int argc, char *argv??(??))
{ if(argc!=3)
  { printf("This program requires two parameters... \n");
    printf("  parm 1 : IP address as a dotted decimal string. \n");
    printf("  parm 2 : Numeric value of the host port to connect with. \n");
    return(TRUE);
  }
  ServerAddress=argv??(1??);
  ServerHostPort= (decimal(5))*(decimal(15,5) *argv??(2??)); /* Dereference and cast a
dec(5) */
  TCP_ERRVARS(&ErrorNum,ErrorMsg); /* Tell the toolkit the error variables */
  TCP_TRACE(&Trace); /* Turn the internal trace on - Optional */
  TCP_TRCTXT("Example client application written in ILE C."); /* User text in the trace.
*/
  TCP_CLIENT(ServerAddress,&ServerHostPort,&Handle); /* Connect to the server */
  if(DisplayError()) return(TRUE); /* Check for and display error */
  TransferData(SEND,"C_SERVER",21); /* Which server ??? */
  TransferData(RECV,DataIn,7); /* Get SUCCESS or FAILURE */
  if(memcmp(DataIn,"SUCCESS",7)) { TCP_CLOSE(&Handle); return; } /* Shutdown
on FAILURE */
  TransferData(SEND,"REQ1",4); /* Client/Server activity */
  TransferData(RECV,DataIn,35);
  TransferData(SEND,"REQ2",4);
  TransferData(RECV,DataIn,35);
  TransferData(SEND,"DONE",4);
  TCP_CLOSE(&Handle); /* Shut the conversation down */
  return(FALSE);
}
/*****
*****/
/* Numeric values cannot be used directly with the TCP_SEND and TCP_RECV because
C must pass a*/
/* pointer to the numeric values. This can be a pain. This can easily be overcome by using a
*/
/* routine like this one. This allows simplified data transfer as shown in main();
*/
/*****
*****/

```

```

void TransferData(int DIRECTION,char *DATA,int DATALENGTH)
{ decimal(10) DataLength;
  decimal(5) Timeout=120;
  decimal(1) Convert=TRUE;
  decimal(5) CIntPort;
  char    *CIntAddr=" ";
  char    *Fixedlen="N";
  DataLength=(decimal(10))DATALENGTH;

if(DIRECTION==SEND)
  { printf("Sending \%-*s\n",DATALENGTH,DATA);
    TCP_SEND(&Handle,DATA,&DataLength,&Timeout,&Convert);
  }
  else
  {
TCP_RECV(&Handle,DATA,&DataLength,&Timeout,&Convert,Fixedlen,CIntAddr,&CIntP
ort);
  printf("Received \%-*s\n",DATALENGTH,DATA);
  }
  if(DisplayError()) { TCP_CLOSE(&Handle); exit(TRUE); }
  return;
}
/*****
*****/
int DisplayError(void)      /* If there was an error, print it and return TRUE */
{ if(ErrorNum)
  { printf("ERR='%-70s'\n",ErrorMsg);    /* Print the Error Message to STDOUT */
    return(TRUE);                      /* Tell main that there was an error */
  }
  else return(FALSE);                /* Tell main that there was no error */
}

```

Server

```

*-----*
      Example to be a Server in ILE C
*-----*

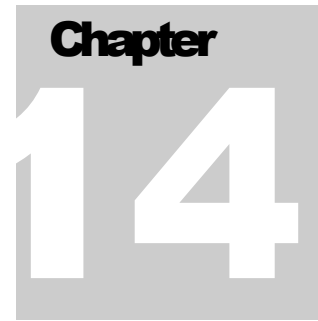
/*****
*****/
/* The TCP/IP Toolkit requires pointers to be passed for all expected parameters. ILE
COBOL & */
/* ILE RPG pass variables by reference so the variable can be passed normally. But, C of
course */
/* passes by value, so the C programmer must explicitly pass the variables' addresses as
shown. */
/*****
*****/
#include <stdio.h>
#include <decimal.h>
#include <milib.h>
#include <mipoces.h>
#include <string.h>
#include <xxdtaa.h>
#define TRUE 1
#define FALSE 0
/*PROTOTYPES*/
void TCP_SERVER(decimal(10) *, char *, decimal(5) *, decimal(10) *, char *);
void TCP_CLOSE(decimal(10) *);
void TCP_TRACE(decimal(1) *);
void TCP_TRCXT(char *);
void TCP_ERRVARS(decimal(2) *,char *);
void TCP_SEND(decimal(10) *,char *,decimal(10) *,decimal(5) *,decimal(1) *);
void TCP_RECV(decimal(10) *,char *,decimal(10) *,decimal(5) *,decimal(1) *,char *,char
*,decimal(5) *);
int DisplayError(void);
decimal(2) ErrorNum=0;          /* Variables to have error information placed */
char *ErrorMsg=" ";
decimal(10) Handle;           /* Conversation handle */
char *DataIn=" ";
decimal(10) DataInLength=4;
decimal(10) DataOutLength=35;
decimal(5) Timeout=120;       /* Try up to 2 minutes */
decimal(1) Convert=TRUE;
decimal(1) Trace=4;          /* Diagnostic Trace */
int Done=FALSE;
decimal(5) HostPort=0;        /* Host port */
decimal(10) MaxClient=0;      /* Maximum Clients (when using multiplex) */

```

```

char    *FixedLen="N";           /* Receive fixed length strings */
char    *ClientAddr;           /* IP addr of current client (Mplx) */
decimal(5) ClientPort=0;      /* Port of current client (Mplx) */
int main()                     /* The server expects no parameters */
{
    TCP_ERRVARS(&ErrorNum,ErrorMsg); /* Tell the toolkit the error variables */
    TCP_TRACE(&Trace);           /* Turn the internal trace on - Optional */
    TCP_TRCTXT("Example server application in ILE C."); /* User defined trace data */
    TCP_SERVER(&Handle,"",&HostPort,&MaxClient,""); /* Get the Client from the
Daemon*/
    if(DisplayError()) return(TRUE); /* Display any error. Shutdown if error */
    do
    {
        TCP_RECV(&Handle,DataIn,&DataInLength,&Timeout,&Convert,FixedLen,ClientAddr,&
ClientPort); /* Get the request. */
        if(DisplayError()) { Done=TRUE; continue;}
        if((memcmp(DataIn,"REQ1",4))) /* ...and act on it.*/
        { TCP_SEND(&Handle,"Server received the first
request",&DataOutLength,&Timeout,&Convert);
          if(DisplayError()) Done=TRUE;
          continue;
        }
        if((memcmp(DataIn,"REQ2",4)))
        { TCP_SEND(&Handle,"Server received the second
request",&DataOutLength,&Timeout,&Convert);
          if(DisplayError()) Done=TRUE;
          continue;
        }
        if(memcmp(DataIn,"DONE",4)) printf("An unknown request was sent from the
client.\n");
        Done=TRUE; /* If we get here, we either got DONE or something we didn't
expect */
    }while(!Done);
    TCP_CLOSE(&Handle); /* Free the TCP/IP resources */
    return(ErrorNum!=FALSE); /* Return TRUE if error or FALSE if not */
}
/*****
int DisplayError(void) /* If there was an error, print it and return TRUE */
{ if(ErrorNum)
  { printf("ERR='%-70s'\n",ErrorMsg); /* Print the Error Message to STDOUT */
    return(TRUE); /* Tell main that there was an error */
  }
  return(FALSE);
}

```



PROCEDURE PROTOTYPES

Procedure prototypes have been included with Netsocket/400. The prototypes for ILE RPG are located in a member called `RPG_PROTO4`, and can be copied using the `"/COPY *LIBL/TCP_SOURCE,RPG_PROTO4"` statement where you want the prototypes to be located in your program.

Prototypes have also been included for C programmers with the member name `C_PROTO`. This prototype can be included using the `#Include` directive if desired. ILE C programmers must also remember to include the decimal header file to work with the decimal type.

Chapter 15

STATUS CODES AND MESSAGES

Your application should define a decimal(2) numeric value and a 70 character string value prior to calling any Netsocket/400 procedure using the TCP_ERRVARS procedure. When a procedure needs to communicate information such as an error condition or normal status information to your application it will populate the variables specified by your application with the status codes below.

With the exception of TCP_TRACE, all Netsocket/400 procedures initialize these variables on entry. If there is no error, the numeric value will be 0. If there is an error, it will be set to one of the following values.

Status codes and their associated messages

Code	Status Description
00	Successful operation – no text returned
01	<p>Trial period has expired.</p> <p>Cause: Based on the NetSocket password found in data area TCP_ENABLE it has been determined that your license has expired.</p> <p>Solution: Contact your NetSocket sales representative for a new password.</p>
02	<p>Invalid license password detected.</p> <p>Cause: Based on the NetSocket password found in data area TCP_ENABLE it has been determined that your license doesn't match the AS/400 serial number/LPAR.</p> <p>Solution: Contact your techsupport@waysidemarketing.com for a new password.</p>

03	<p>Invalid fixed length parameter received – not Y or N.</p> <p>Cause: The fixed length parameter on either the TCP_RECV or TCP_RCVBND procedure was run with a value other than ‘Y’ or ‘N’.</p> <p>Solution: Fix parameter value and run again.</p>
04	<p>Invalid length parameter detected.</p> <p>Cause: The length parameter on either the TCP_RECV, TCP_RCVBND, Or TCP_SEND procedure was run with a value less than or equal to zero.</p> <p>Solution: Fix parameter value and run again.</p>
05	<p>Cannot output your trace text because the trace is not on.</p> <p>Cause: The TCP_TRCTXT procedure was run before the TCP_TRACE procedure was executed to turn the trace on.</p> <p>Solution: Execute the TCP_TRACE procedure with a logging level greater than zero before executing the TCP_TRCTXT procedure.</p>
06	<p>Not currently used.</p>
07	<p>Port values range from 0 to 65535.</p> <p>Cause: The TCP_CLIENT, TCP_SERVER, TCP_DAEMON, TCP_ENDDMN, or TCP_PGID procedure was executed with a length parameter value greater than 65535.</p> <p>Solution: Execute the procedure again with a length parameter value less than 65536.</p>
08	<p>Maximum number of boundaries (50) already registered.</p> <p>Cause: The TCP_REGBND procedure was executed to register a new boundary but the maximum number of boundaries (50) has already been registered.</p>

09	<p>Could not acquire any IP addresses for the host name given.</p> <p>Cause: A host name was provided as a value for the address parameter of a NetSocket procedure that was executed and the IP address could not be found in the DNS server.</p> <p>Solution: Correct the host name in the address parameter and try again.</p>
10	<p>No boundary characters detected in registry command.</p> <p>Cause: The TCP_REGBND procedure was just executed but no value was detected in the boundary parameter.</p> <p>Solution: Correct the boundary parameter and try again.</p>
11	<p>TCP_CLIENT cannot have an IP address value of '*'.</p> <p>Cause: A value of '*' was detected in the address parameter of the TCP_CLIENT procedure. The value '*' represents the fact you want to monitor all incoming IP addresses on the AS/400 but a specific address is required.</p> <p>Solution: Enter a valid IP address or host name in the address parameter.</p>
12	<p>An error occurred while passing the client to the server. See the job log.</p> <p>Cause: Somewhere during the process of passing the socket descriptor from the Daemon program to the Server program an error was detected. Check the job log of the Daemon and Server programs for specific error information.</p>
13	<p>Holding buffer size will be exceeded before boundary characters found.</p> <p>Cause: The holding buffer used by the TCP_RCVBND procedure is full and a boundary sequence has not been detected.</p> <p>Solution: Restart program and check for the fact that the correct boundary sequences are being registered before using TCP_RCVBND.</p>
14	<p>No data in holding buffer to flush.</p> <p>Cause: A flush of the holding buffer used by the TCP_RCVBND procedure was requested but no data currently resides in the buffer.</p> <p>Solution: This is not a problem and is a common status message.</p>

15	<p>Flush buffer parameter not Y or N.</p> <p>Cause: The flush buffer parameter of the TCP_RCVBND procedure has an incorrect value. Value should be 'Y' or 'N'.</p> <p>Solution: Correct parameter value to Y or N and run the procedure again.</p>
16	<p>Boundary character detected beyond size of data buffer provided.</p> <p>Cause: A registered boundary sequence was detected in the holding buffer but the data string cannot be returned because the length of the data buffer provided in the length parameter is too small.</p> <p>Solution: Increase the size of the data buffer and length parameter and try the procedure again.</p>
17	<p>No boundaries currently registered – this procedure aborted.</p> <p>Cause: The TCP_RCVBND procedure was executed but no boundary sequences were previously registered with the TCP_REGBND procedure.</p> <p>Solution: Make sure you register at least one valid boundary sequence before executing the TCP_RCVBND procedure.</p>
18	<p>Time out occurred while waiting for client connection in Multiplex mode.</p> <p>Cause: The TCP_RECV procedure has timed out while waiting for a Client request.</p> <p>Solution: This is a valid status message. To lessen the number of these status messages try increasing the time-out parameter. If this message is received, try reissuing the TCP_RECV procedure again.</p>
19	<p>All of the requested data was not received within the allocated time.</p> <p>Cause: The number of bytes requested in the length parameter have not been received within the time-out value time provided. This is a common status.</p> <p>Solution: This status message is common when receiving variable length strings. This should not be treated as an error unless expecting fixed length string. Check the length parameter value returned for the actual number of bytes received.</p>
20	<p>All of the data to send was not sent within the allocated time.</p>

21	<p>General error occurred – check job log for details.</p> <p>Cause: A non-specific error occurred – check the job log for further details.</p>
22	<p>Maximum clients managed by a single server cannot exceed 512 – Multiplex mode.</p> <p>Cause: The maximum clients parameter of the TCP_SERVER procedure has a value greater than 512.</p> <p>Solution: Make the maximum clients parameter value less than 513 and try again.</p>
23	<p>Receiving under boundary control is not allowed in Multiplexing mode.</p> <p>Cause: The TCP_RCVBND procedure has been executed but the program is running in multiplexing mode which is not allowed for TCP_RCVBND.</p> <p>Solution: Either use the TCP_RECV procedure to receive the data or change the maximum clients parameter on the TCP_SERVER procedure to 0 or 1.</p>
24	Max of 512 SSL sockets reached.
25	SSL not allowed with Dynamic Server.
26	Invalid number of parameters passed.
27	xxx SSL connection failed. (xxx is an SSL sub error code).
28	xxx No Keyring file path was passed. (xxx is an SSL sub error code).
29	xxx Invalid Keyring file path was passed. (xxx is an SSL sub error code).
30-49	Not currently used
50	E2BIG - Conversion stopped. Not enough destination buffer space.
51	EACCES - Permission denied. No privileges to the destination address.

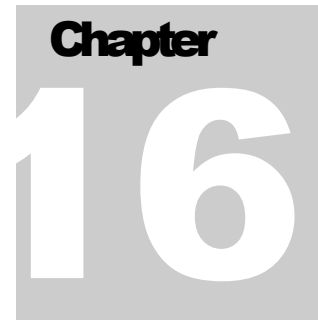
52	<p>EADDRINUSE - The requested TCP/IP address or port is already in use.</p> <p>Cause: This error is typically caused when trying to listen on a specified address and port that is currently in use.</p> <p>Solution: Either change the address or port you want to listen on or terminate the use of the port with the <i>NETSTAT</i> command.</p>
53	<p>EADDRNOTAVAIL - Address or port not available.</p> <p>Cause: This error is typically caused when trying to listen on a specified address and port that is currently in use.</p> <p>Solution: Either change the address or port you want to listen on or terminate the use of the port with the <i>NETSTAT</i> command.</p>
54	<p>EAFNOSUPPORT - The type of socket is not supported.</p>
55	<p>EALREADY - Socket is already connected.</p>
56	<p>EBADDATA - Could not convert the requested data.</p>
57	<p>EBADF - Socket descriptor not valid.</p>
58	<p>ECONNABORTED - Connection ended abnormally.</p> <p>Cause: This error is typically caused when the remote program ended the socket connection without first notifying your program. This is very common.</p> <p>Solution: Either terminate your program or try and reestablish communications with the <i>TCP_CLIENT</i> procedure if you are the Client.</p>
59	<p>ECONNREFUSED - Destination refused attempted connection.</p> <p>Cause: This error is typically caused when the <i>TCP_CLIENT</i> procedure tries to connect to a remote host that is either not listening at the specified address/port, or the network connection was not possible for some reason.</p>
60	<p>ECONNRESET - Remote socket reset the connection.</p>
61	<p>ECONVERT - Could not convert the requested data.</p>

62	EDESTADDRREQ - Destination address required.
63	EFAULT - Bad address.
64	EHOSTDOWN - The remote host is not available
65	EHOSTUNREACH -No available route to the remote host.
66	EINPROGRESS - Requested operation is being processed.
67	EINVAL - Invalid address, address length or buffer length or CCSID.
68	EIO - Input / Output error.
69	EISCONN - Connection already established.
70	EMFILE - Too many descriptions for this process.
71	EMSGSIZE - Data field too large.
72	ENETDOWN - The network is currently not available
73	ENETUNREACH - Cannot reach the destination network
74	ENFILE - Too many descriptions in system
75	ENOBUFS - There is not enough buffer space for the requested operation.
76	ENOMEM - There is not enough storage space available for the operation.
77	ENOTCONN – Requested operation requires a connection.
78	ENOTDIR - Not a directory.

79	ENOTSOCK - The handle does not reference a socket.
80	EOPNOTSUPP - Operation not supported.
81	EPIPE - Broken pipe. Cause: This error is caused when the connection broken with remote host. Solution: Either terminate your program or try and reestablish communications with the TCP_CLIENT procedure if you're the Client.
82	EPERM - Operation not permitted.
83	EPROTONOSUPPORT - TCP/IP protocol does not exist.
84	ESOCKTNOSUPPORT - Specified socket type not supported
85	ETIMEDOUT - Remote host did not respond within the timeout period.
86	EUNATCH - TCP/IP protocol not available at this time.
87	EUNKNOWN - An undetected error occurred. Contact service.
88	EWOULDBLOCK - Socket timeout
99	Error #00000 occurred – check job log for details. Cause: A non-specific error occurred – check the job log for further details.
-1	No ciphers available or specified.
-2	No certificate is available for SSL processing.
-4	Bad Certificate.
-6	OS/400 does not support the certificate type.
-10	Error occurred in SSL processing.

-11	Received badly formatted message.
-12	Bad message authentication code.
-13	SSL is not supported.
-14	Certificate Signature invalid.
-15	Bad Certificate.
-16	Peer system is not recognized.
-17	Permission was denied to access object.
-18	The certificate is self-signed.
-20	Unable to allocate SSL storage.
-21	SSL detected a bad state in the session.
-22	Socket has been closed.
-23	The certificate is not signed by a trusted certificate authority.
-24	The validity time period of the certificate is expired.
-25	Certificate has a bad date.
-26	Certificate key length is invalid.
-90	File specified is not a keyring file.
-91	Keyring password has expired.
-92	Certificate is not valid or was rejected.
-93	SSL is not available for use.
-94	SSL_Init() was not previously called for this job.

-95	No keyring file was specified.
-96	SSL is not enabled on this AS/400.
-97	Cipher suite is invalid.
-98	The SSL session ended.
-99	An unknown or unexpected error occurred during SSL processing

A gray square graphic containing the word "Chapter" in a bold, black, sans-serif font at the top. Below it, the number "16" is displayed in a very large, white, bold, sans-serif font, centered within the square.

COMMUNICATING WITH OTHER PLATFORMS

Netsocket/400 communicates using connection-oriented stream sockets of type `AF_INET`. It uses TCP (Transmission Control Protocol) sockets.

It does not support UDP or RAW sockets. Any application (on any platform) that communicates in this fashion will work with the Netsocket/400 and your application

Chapter 17

DEFAULT HEXADECIMAL CONVERSION CHART

The chart below details how characters get converted at the hexadecimal level when using the default conversion tables CCSID 00437 for ASCII and CCSID 00037 for EBCDIC

Hex value before conversion (37)	Hex value after conversion (437)	Hex value before conversion (37)	Hex value after conversion (437)	Hex value before conversion (37)	Hex value after conversion (437)
00	00	01	01	02	02
03	03	04	EC	05	09
06	CA	07	1C	08	E2
09	D2	0A	D3	0B	0B
0C	0C	0D	0D	0E	0E
0F	0F	10	10	11	11
12	12	13	13	14	EF
15	C5	16	08	17	CB
18	18	19	19	1A	DC
1B	D8	1C	1A	1D	1D
1E	1E	1F	1F	20	B7
21	B8	22	B9	23	BB
24	C4	25	0A	26	17
27	1B	28	CC	29	CD
2A	CF	2B	D0	2C	D1
2D	05	2E	06	2F	07
30	D9	31	DA	32	16
33	DD	34	DE	35	DF
36	E0	37	04	38	E3
39	E5	3A	E9	3B	EB

3C	B0	3D	B1	3E	9E
3F	7F	40	20	41	FF
42	83	43	84	44	85
45	A0	46	F2	47	86
48	87	49	A4	4A	9B
4B	2E	4C	3C	4D	28
4E	2B	4F	7C	50	26
51	82	52	88	53	89
54	8A	55	A1	56	8C
57	8B	58	8D	59	E1
5A	21	5B	24	5C	2A
5D	29	5E	3B	5F	AA
60	2D	61	2F	62	B2
63	8E	64	B4	65	B5
66	B6	67	8F	68	80
69	A5	6A	B3	6B	2C
6C	25	6D	5F	6E	3E
6F	3F	70	BA	71	90
72	BC	73	BD	74	BE
75	F3	76	C0	77	C1
78	C2	79	60	7A	3A
7B	23	7C	40	7D	27
7E	3D	7F	22	80	C3
81	61	82	62	83	63
84	64	85	65	86	66
87	67	88	68	89	69
8A	AE	8B	AF	8C	C6
8D	C7	8E	C8	8F	F1
90	F8	91	6A	92	6B
93	6C	94	6D	95	6E
96	6F	97	70	98	71
99	72	9A	A6	9B	A7
9C	91	9D	CE	9E	92
9F	A9	A0	E6	A1	7E
A2	73	A3	74	A4	75
A5	76	A6	77	A7	78

A8	79	A9	7A	AA	AD
AB	A8	AC	D4	AD	D5
AE	D6	AF	D7	B0	5E
B1	9C	B2	9D	B3	FA
B4	9F	B5	15	B6	14
B7	AC	B8	AB	B9	FC
BA	5B	BB	5D	BC	E4
BD	FE	BE	BF	BF	E7
C0	7B	C1	41	C2	42
C3	43	C4	44	C5	45
C6	46	C7	47	C8	48
C9	49	CA	E8	CB	93
CC	94	CD	95	CE	A2
CF	ED	D0	7D	D1	4A
D2	4B	D3	4C	D4	4D
D5	4E	D6	4F	D7	50
D8	51	D9	52	DA	EE
DB	96	DC	81	DD	97
DE	A3	DF	98	E0	5C
E1	F6	E2	53	E3	54
E4	55	E5	56	E6	57
E7	58	E8	59	E9	5A
EA	FD	EB	F5	EC	99
ED	F7	EE	F0	EF	F9
F0	30	F1	31	F2	32
F3	33	F4	34	F5	35
F6	36	F7	37	F8	38
F9	39	FA	DB	FB	FB
FC	9A	FD	F4	FE	EA
FF	C9				