7188E/843X/844X/883X/884X

TCP/IP Library User's Manual

Table of Contents

1. Introduction

1.1 Package List & Release Notes

Package List

In addition to this manual, all packages includes the following items:

- One 7188E hardware module
- One user's manual (this manual)
- One release note
- One software utility disk or CD
- One download cable, CA0915

Note: If any of these items are missing or damaged, contact the local distributors for more information. Save the shipping materials and cartons in case you want to ship in the future.

Release Note

It is recommended to read the **release note & README.TXT** first. The release note is given in the shipping. The README.TXT is given in the CD\README.TXT. Some important information are given in the release note & CD\README.TXT

Order Information

Call distributor for details.

1.2 Why! Ethernet Solutions

“Embedded Internet” and “Embedded Ethernet” are hot topics today. Nowadays, the Ethernet protocol has become the de-facto standard for local area networks. Via the Internet, connectivity is spreading into many different areas, from home appliances to vending machines, from testing equipment to UPS...etc. Many embedded designers now face dilemma of adding an Ethernet interface to their products, either for use with local networks or for connecting to the Internet. Solutions to this problem include both hardware and software. Connecting via Ethernet requires a software protocol called TCP/IP. The installed base of Ethernet networks is vast and growing quickly. Most office buildings, factories, and new homes have an installed Ethernet network. With Ethernet, the network is always available. Using Ethernet for network in industrial areas is appealing because the required cabling is already installed.

The 7188E series are a series of embedded controllers designed to meet the most common requirements of Internet/Ethernet applications. They can be used to replace the PC or PLC for the harsh environment.

The 7188E series provide one on-board 10BASE-T port that is directly driven by a NE2000 compatible Ethernet controller. The 10BASE-T port is equipped with a RJ-45 connector. The 10BASE-T interface supports the max. cable length (100 meters) between 7188E & a hub. To link the 7188E & the other device through a 10BASE_T hub, simply use two straight-through cables: one cable connects to 7188E; the other cable connects the hub to the other device.

1.3 The 7188EX, 7188EA & 7188EN Series

The I-7188EX is powered by a 80188-40 processor with 512K bytes of static RAM, and 512K bytes of Flash memory. One serial RS-232 port and one RS-485 port are provided. The Ethernet support is provided by a NE-2000 compatible controller with 16K bytes of on-chip buffer memory and 10Base-T media interface. The I-7188EX also provides 14 user defined I/O lines. A cost-effective I/O expansion board with A/D, D/A, relays drivers and protected inputs are available. The I-7188EX also supports a battery backup SRAM board and Flash-Rom board, providing non-volatile mass storage from 128K bytes megabytes to 64 megabytes. The 10BASE-T port is equipped with a RJ-45 connector. The 10BASE-T interface supports a max. of 100 meters of Cable length between the I-7188EX and the network hub.

Compared to the I-7188EX, the I-7188EA adds seven open-collector output channels and six digital Input channels. The I/O Expansion bus has been occupied by a D/I/O expansion board.

The I-7188EN, Embedded Internet/Ethernet Controller, focuses on embedded control applications while the I-7188EN, Internet Communication Controller, focuses on communication applications and applications dependant on different embedded firmware programs. The Internet Communication Controller can be used as Device Server, Addressable Ethernet to RS-232/485/422 Converter, or Embedded Internet/Ethernet Controller. The user should refer to the comparison table to choose the product that best suit their needs. Currently, we offer a wide range of Internet Communication Controllers, such as the I-7188E1/E2/E3/E4/E5/E8. Except the RTC circuitry, the basic hardware of the I-7188EN is similar to the I-7188EX. Since there are too many Configurations for the I-7188EN series product, an OEM or ODM version is welcomed.

1.4 TCP/IP 4-layer model

4 Application Layer	TELNET, FTP, SMTP, DNS
3 Transport Layer	TCP , UDP
2 Network Layer	IP, ICMP, ARP, RARP
1 Physical Layer	Packet radio, Ethernet

Figure 1: TCP/IP protocol suite using a 4-layer model

1.5 Internet Address

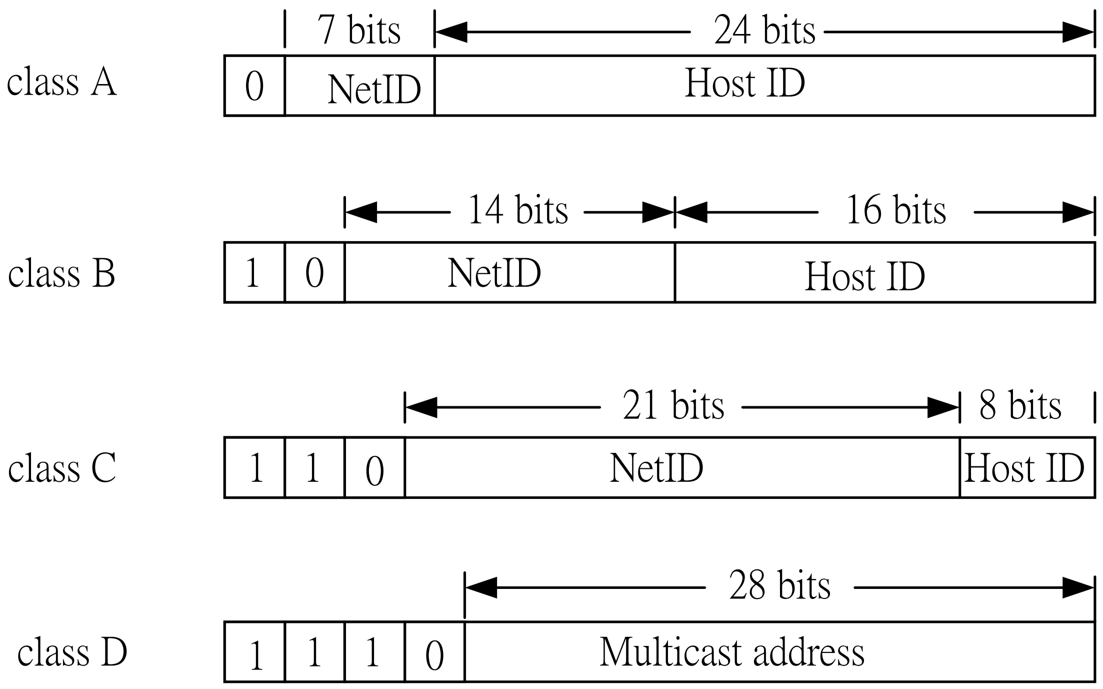


Figure 2: Internet Address.

- **Network Mask:**
  Class A: 255.0.0.0
  Class B: 255.255.0.0
  Class C: 255.255.255.0
  Class D: Multicast address

1.6 Connection- Oriented Protocol

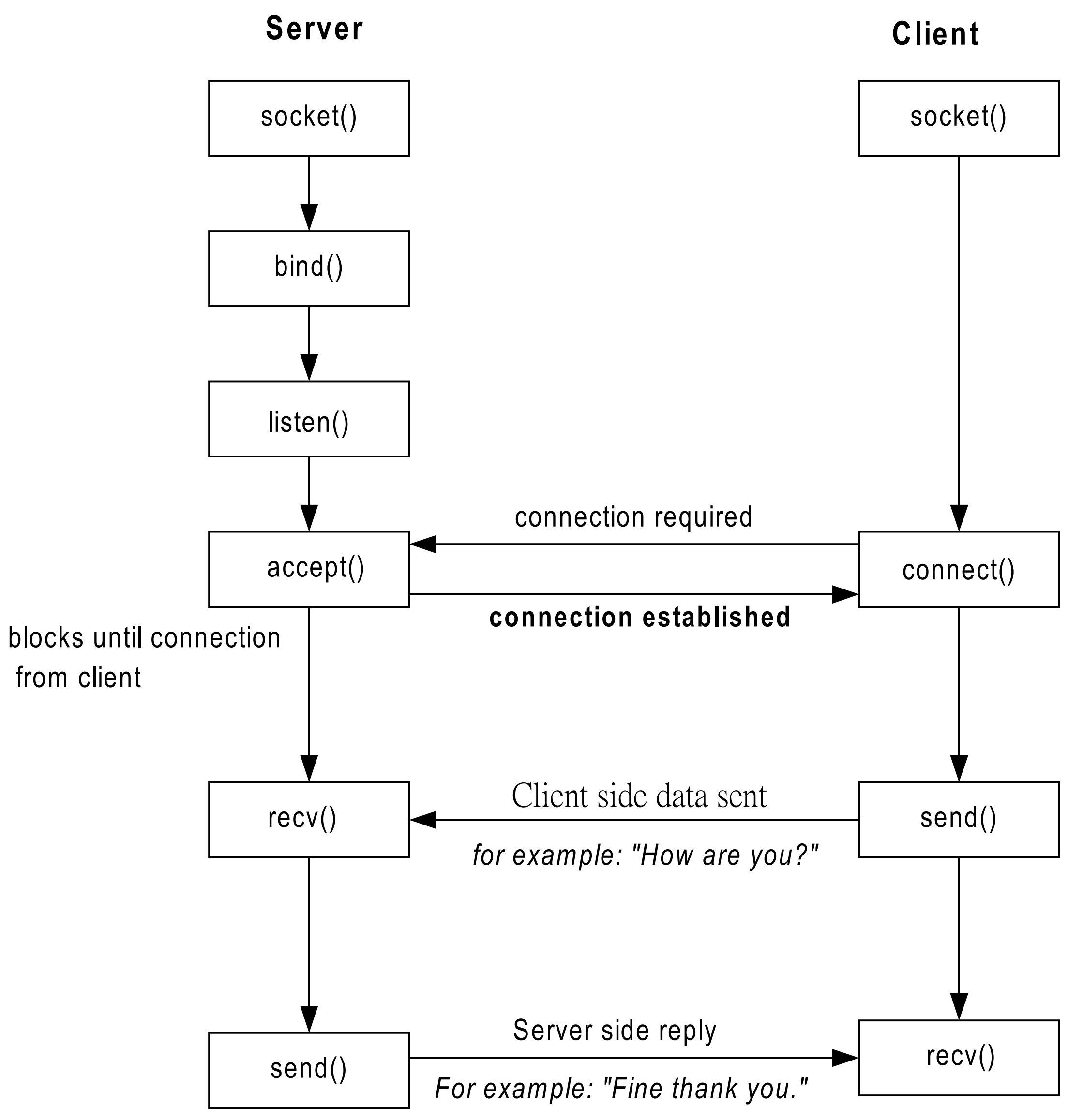


Figure 3: Connection-Oriented Protocol

1.7 Connectionless Protocol

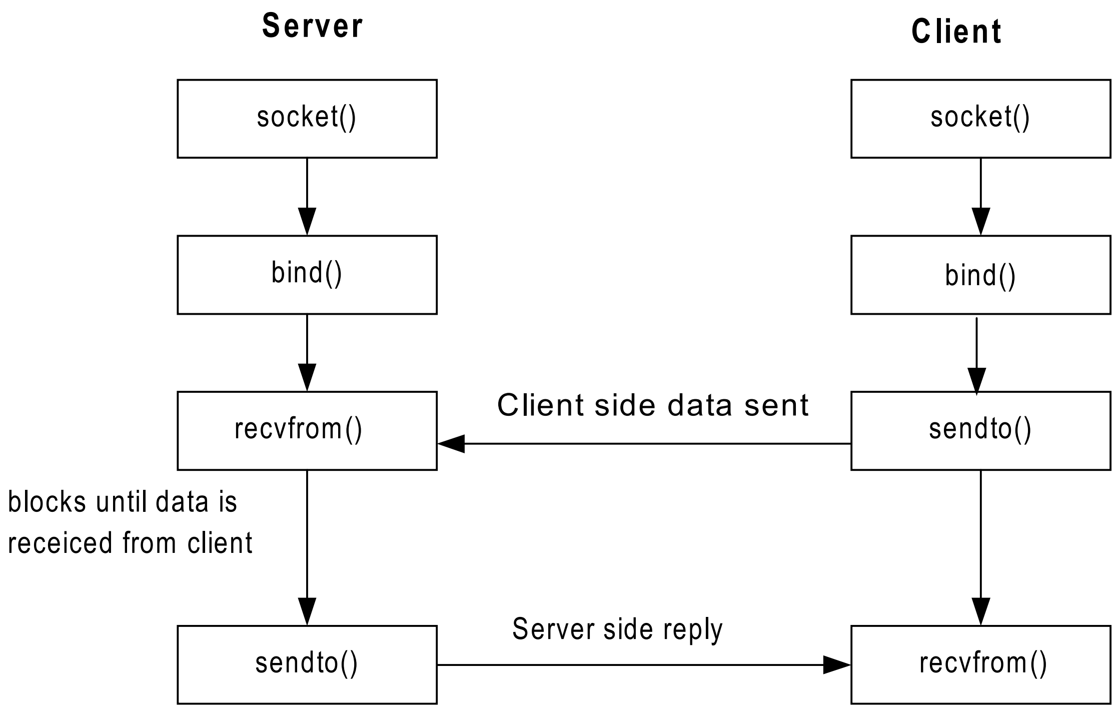


Figure 4: Connectionless Protocol

2. Software Installation

- ***For the 7188E/843X/883X/844X/884X***
  The software for the *7188E/843X/883X/844X/884X* consists of one CD-ROM.

  Directories of CD-ROM: Napdos\7188e\tcpip>
  Napdos\7188e\tcpip>xcopy/e/s   *.*   c:\tcpip\.

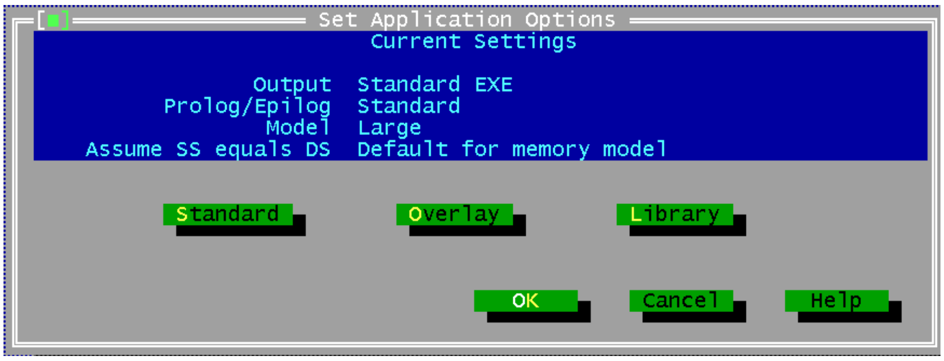
3. Compiling & linking

User must use C Language when writing programs. You can use TC++, BC++, MSC or MSVC++（before 1.52）. Please take care of the following items:

- Generate a standard DOS executable program.
- Select CPU=80188/80186.
- Select EMULATION if floating point computation is required.(can not select 8087)

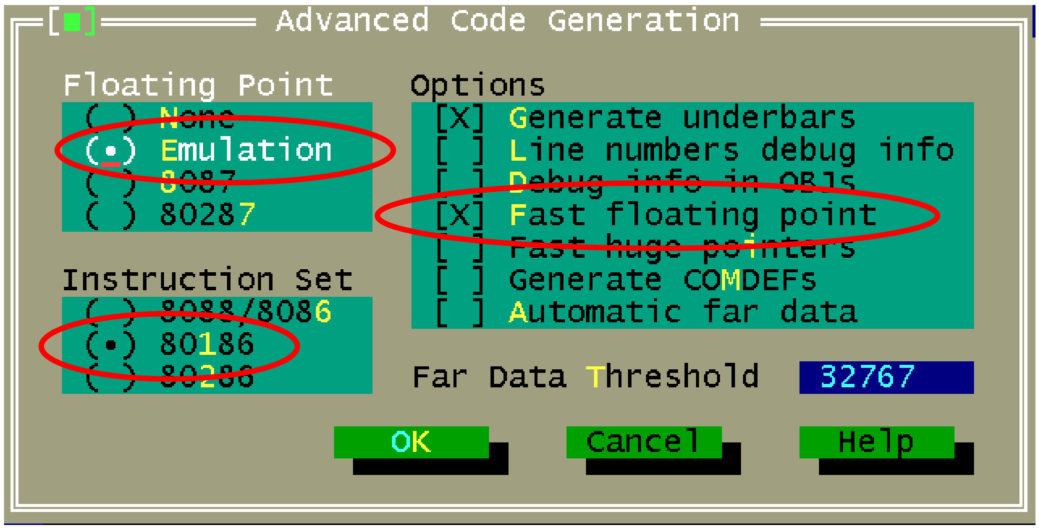
3.1 Using TC++

- Select standard code:
  Options → Application…→



```
Set Application Options
Current Settings

          Output  Standard EXE
    Prolog/Epilog  Standard
            Model  Large
Assume SS equals DS  Default for memory model
```

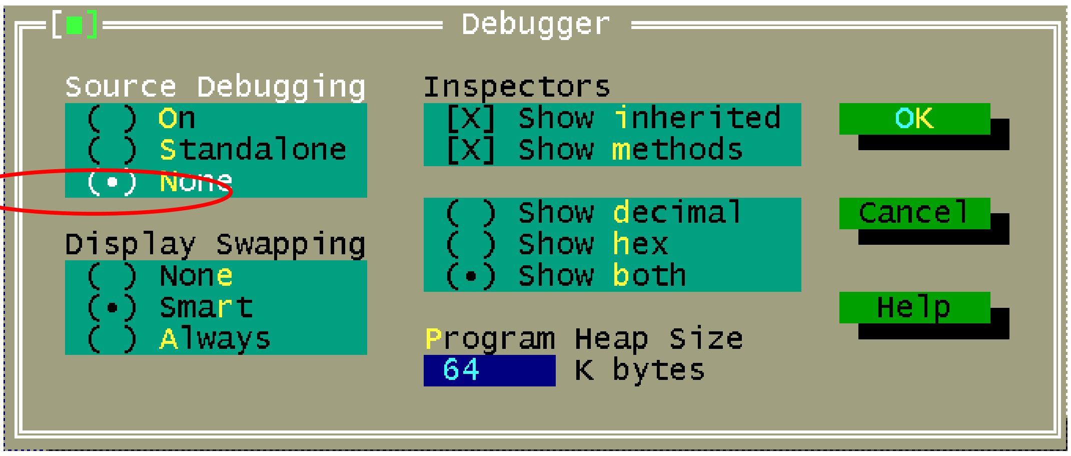
- Select EMULATION , 80186 , Debug info in OBJs cancel:
  Option → Compiler → Advanced Code Generation…→



```
Advanced Code Generation

Floating Point        Options
( ) None              [X] Generate underbars
(•) Emulation         [ ] Line numbers debug info
( ) 8087              [ ] Debug info in OBJs
( ) 80287             [X] Fast floating point
                      [ ] Fast huge pointers
Instruction Set       [ ] Generate COMDEFs
( ) 8088/8086         [ ] Automatic far data
(•) 80186
( ) 80286             Far Data Threshold   32767
```

- Select None Source Debugging

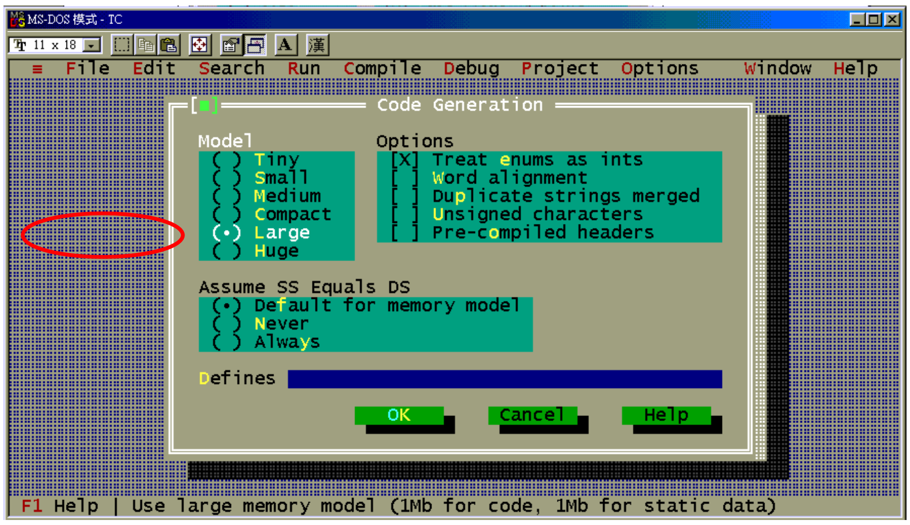
Option → `Debugger` →



```
[■]              Debugger

Source Debugging      Inspectors
 ( ) On                [X] Show inherited      OK
 ( ) Standalone        [X] Show methods
 (•) None
                       ( ) Show decimal        Cancel
Display Swapping       ( ) Show hex
 ( ) None              (•) Show both
 (•) Smart                                     Help
 ( ) Always           Program Heap Size
                       64     K bytes
```

- Select large mode:

Options → Compiler → Code generation…→

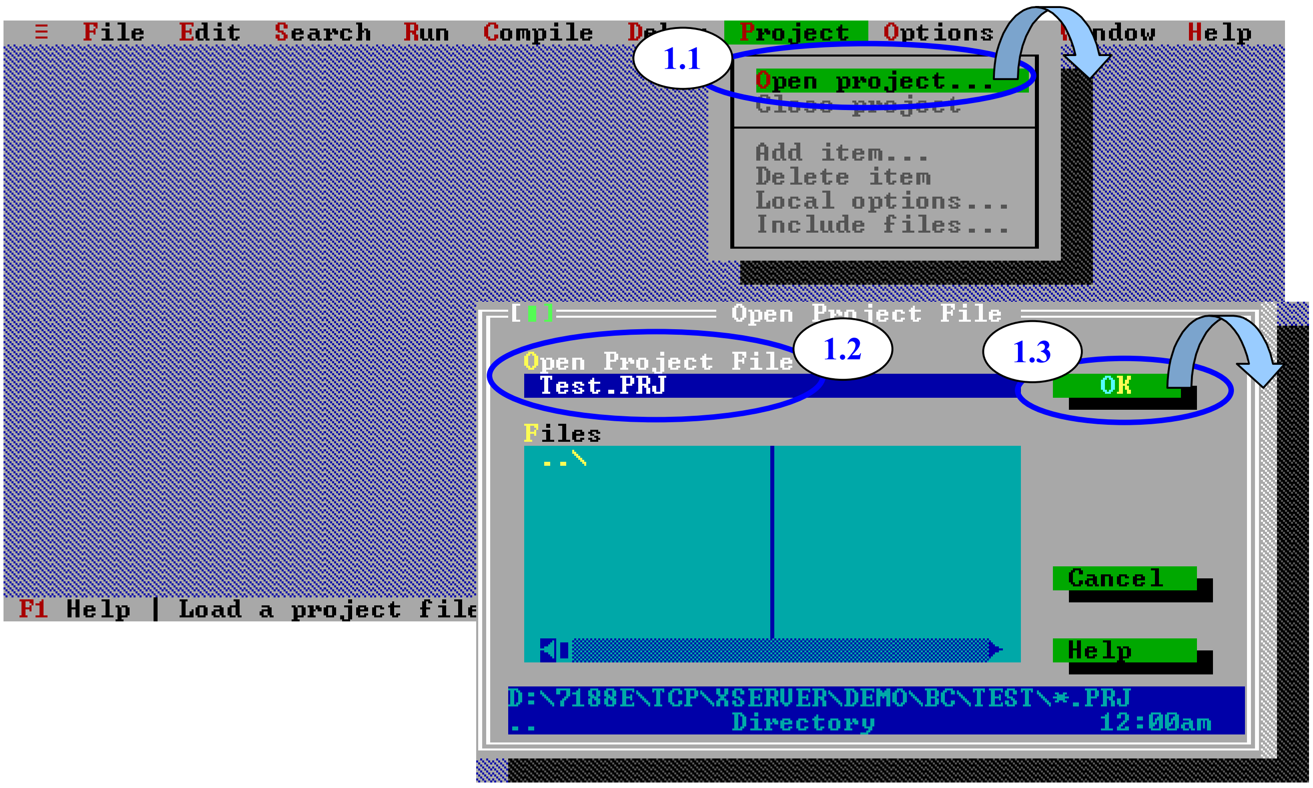


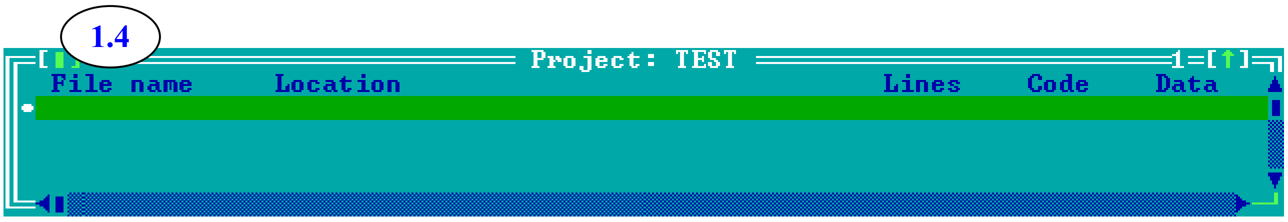
MS-DOS 模式 - TC
11 x 18
≡ File Edit Search Run Compile Debug Project Options Window Help

```
[■]            Code Generation

Model            Options
 ( ) Tiny         [X] Treat enums as ints
 ( ) Small        [ ] Word alignment
 ( ) Medium       [ ] Duplicate strings merged
 ( ) Compact      [ ] Unsigned characters
 (•) Large        [ ] Pre-compiled headers
 ( ) Huge

Assume SS Equals DS
 (•) Default for memory model
 ( ) Never
 ( ) Always

Defines

           OK        Cancel        Help

F1 Help | Use large memory model (1Mb for code, 1Mb for static data)
```

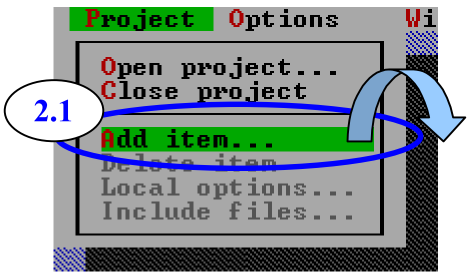
3.2 Using BC ++ 3.1

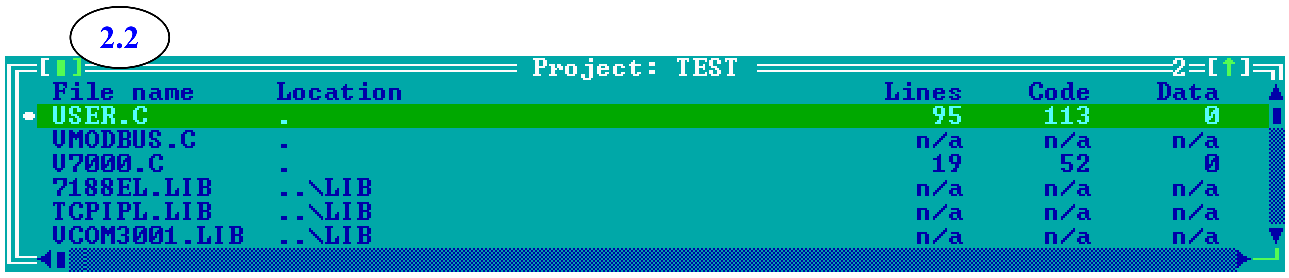
Step 1: Create a new project.





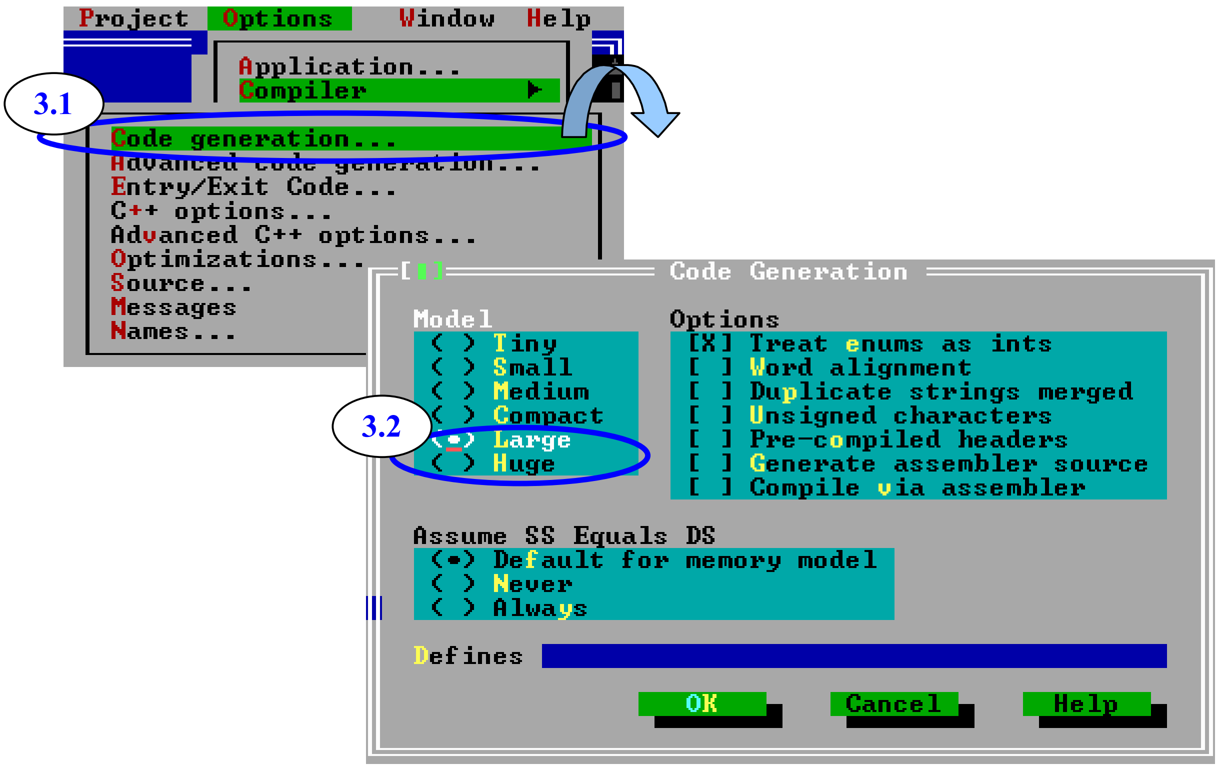
Step 2: Add all necessary files into the project.





```
[■]════════════════════ Project: TEST ══════════════════════════2=[↑]
 File name      Location                           Lines    Code    Data
• USER.C        .                                     95     113       0
 VMODBUS.C      .                                    n/a     n/a     n/a
 V7000.C        .                                     19      52       0
 7188EL.LIB     ..\LIB                               n/a     n/a     n/a
 TCPIPL.LIB     ..\LIB                               n/a     n/a     n/a
 VCOM3001.LIB   ..\LIB                               n/a     n/a     n/a
```

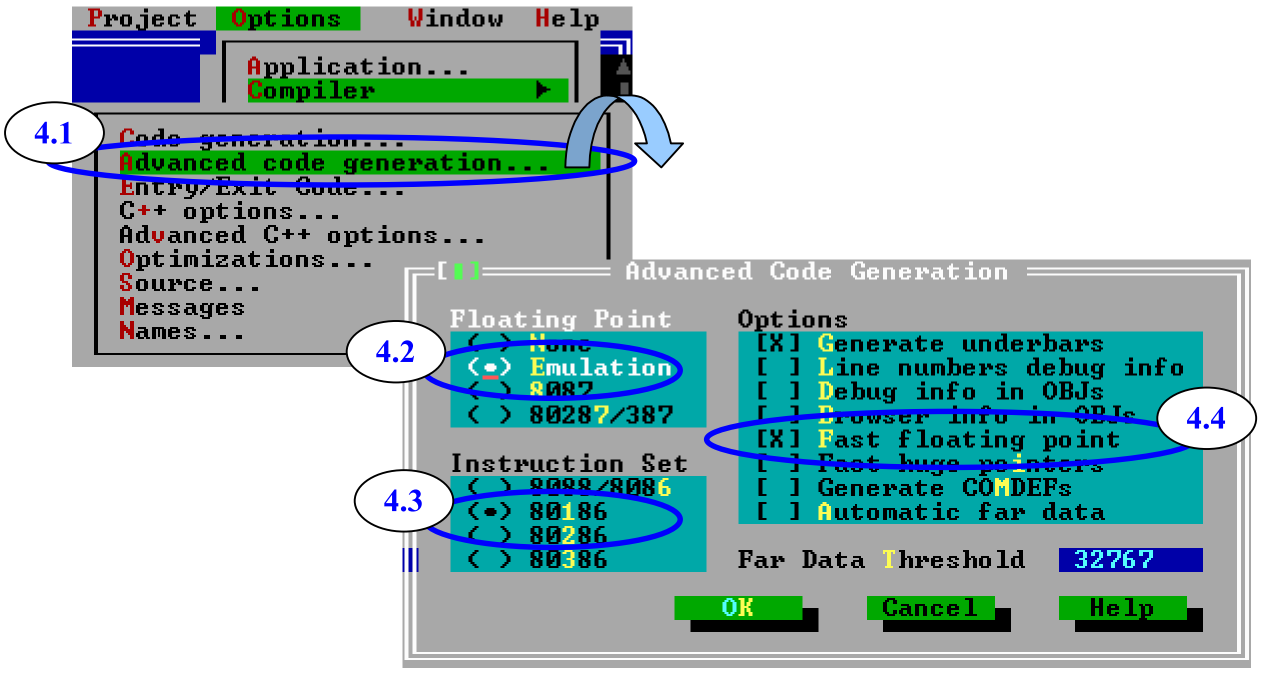
Step 3: Set Code generation options.



```
Project  Options   Window  Help
         Application...
         Compiler          ►
  Code generation...
  Advanced code generation...
  Entry/Exit Code...
  C++ options...
  Advanced C++ options...
  Optimizations...
  Source...
  Messages
  Names...
```

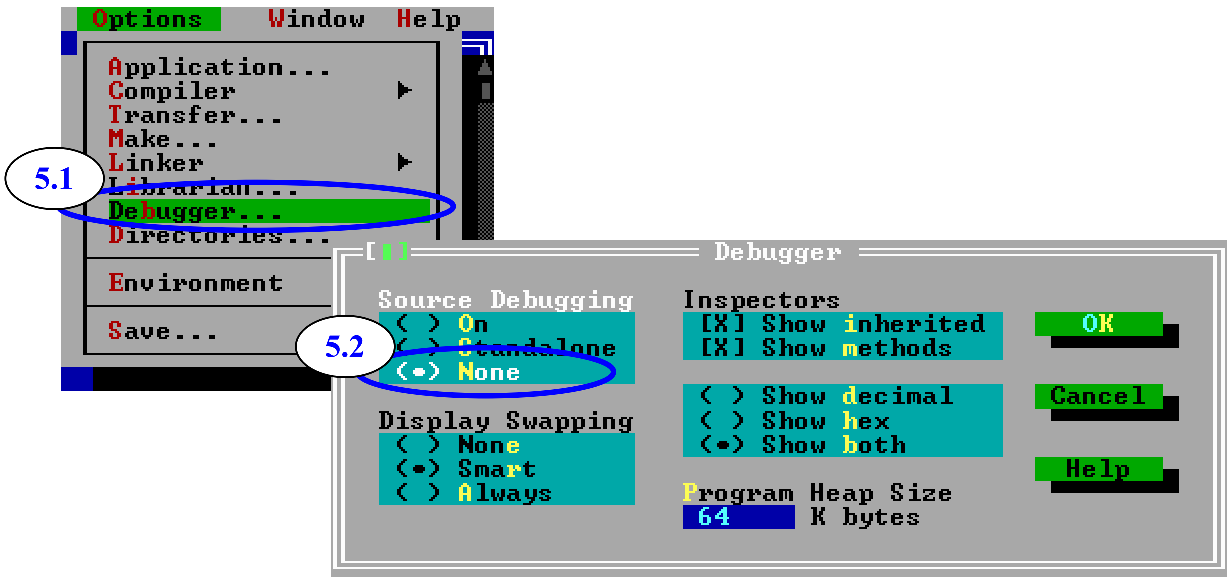
```
[■]═════════════ Code Generation ═══════════════
 Model                Options
  ( ) Tiny             [X] Treat enums as ints
  ( ) Small            [ ] Word alignment
  ( ) Medium           [ ] Duplicate strings merged
  ( ) Compact          [ ] Unsigned characters
  (•) Large            [ ] Pre-compiled headers
  ( ) Huge             [ ] Generate assembler source
                       [ ] Compile via assembler

 Assume SS Equals DS
  (•) Default for memory model
  ( ) Never
  ( ) Always

 Defines

              OK        Cancel       Help
```

Step 4: Set Advanced code generation options.



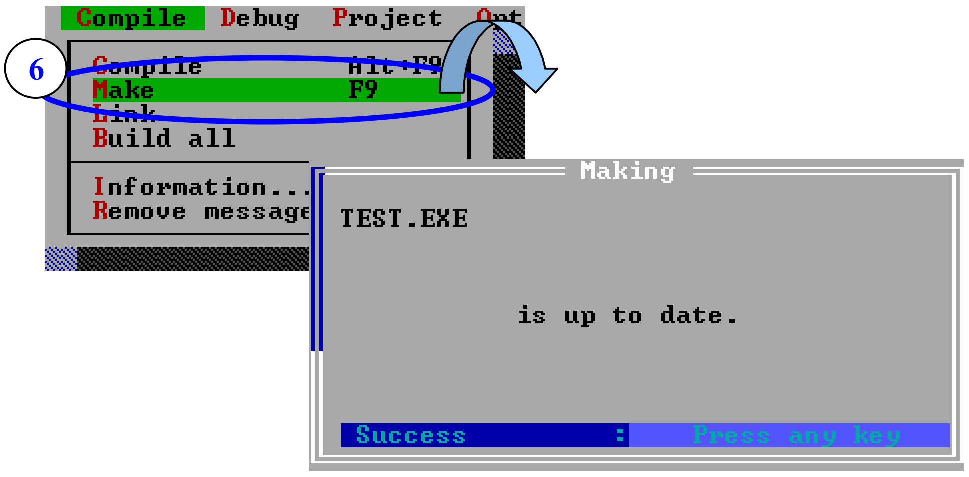
```
Advanced Code Generation

Floating Point              Options
( ) None                    [X] Generate underbars
(•) Emulation               [ ] Line numbers debug info
( ) 8087                    [ ] Debug info in OBJs
( ) 80287/387               [ ] Browser info in OBJs
                            [X] Fast floating point
Instruction Set             [ ] Fast huge pointers
( ) 8088/8086               [ ] Generate COMDEFs
(•) 80186                   [ ] Automatic far data
( ) 80286
( ) 80386                   Far Data Threshold  32767
```

Step 5: Set Debugger options.



```
Debugger

Source Debugging            Inspectors
( ) On                      [X] Show inherited
( ) Standalone              [X] Show methods
(•) None
                            ( ) Show decimal
Display Swapping            ( ) Show hex
( ) None                    (•) Show both
(•) Smart
( ) Always                  Program Heap Size
                            64    K bytes
```

Step 6: Make the project.



TEST.EXE

is up to date.

4. TCP/IP Library function

4.1 TCP/IP function

4.1.1 Ninit

- **Description:** Network initialization.
- **Syntax:** int Ninit (void);
- **Return Value:** > = 0 : successful
- **For example:**

```
main()
{
    if( Is7188e() == 0)  /* I-7188E detect */
    {
        Print("The Hardware is not I-7188E.\n\r");
        exit(1);
    }
    if( Ninit() < 0 )
    Print("Network initialization error\n\r");
}
```

4.1.2 Portinit

- **Description:** Network interface initialization.
- **Syntax:** int Portinit (char *name);
- **Input Parameter:** *: all network interfaces will be initialized.
- **Return Value:** > = 0: successful
- **For example:**

```
main()
{
    if( Is8000() == 0)  /* I-8000 detect */
    {
        Print("The Hardware is not I-8000.\n\r");
        exit(1);
    }
    if( Ninit() < 0 )
        Print("Network initialization error\n\r");
    if( Portinit( "*" ) < 0 )
        Print("Network initialization error\n\r");
}
```

4.1.3 Nterm

- **Description:** Network shutdown.
- **Syntax:** int Nterm (void);
- **Input Parameter:** none
- **Return Value:** 0: Always return 0.
- **For example:**

```
main()
{
    if( Is7188e() == 0)  /* I7188E detect */
    {
        Print("The Hardware is not I-7188E.\n\r");
        exit(1);
    }

    if( Ninit() < 0 )
        Print("Network initialization error\n\r");
        …
        …
    Nterm();
}
```

4.1.4 Portterm

- **Description:** Shutdown network interface.
- **Syntax:** int Portterm (char *name);
- **Input Parameter:** *: all network interfaces will be shutdown.
- **Return Value:** >=0: Always return 0.
- **For example:**

```
main()
{
    if( Is8000() == 0)  /* I-8000 detect */
    {
        Print("The Hardware is not I-8000.\n\r");
        exit(1);
    }

    if( Ninit() < 0 )
      Print("Network initialization error\n\r");
    if( Portinit( "*" ) < 0 )
    {
        Print("Network initialization error\n\r");
        Portterm("*");
    }
}
```

4.1.5 accept

- **Description:** Accepts a connection at a socket.

- **Syntax:** int accept (int s, struct sockaddr *name, int *namelen);

- **Input Parameter:**
  s: Socket ID.
  *name: The *name storage IP of client side.
  *namelen: The *namelen storage size of IP of client side

- **Return Value:** >=0: successful.
  -1: error.

- **Reference:** *socket, bind, listen*

- **For example:**

```
main()
{
    int s1, s2, socksize;
    struct sockaddr_in socka;
      …
    socksize = sizeof(socka);
    memset(&socka, 0, sizeof(socka) );
    socka.sin_family = AF_INET;
    s2 = accept( s1, (struct sockaddr *)&socka, &socksize );
    if( s2 < 0 )
        Print("Error in accept. \n\r");

}
```

4.1.6 bind

- **Description:** Binds a socket.

- **Syntax:** int bind(int s, struct sockaddr *name, int *namelen);

- **Input Parameter:**
  s: Socket ID.
  *name: The *name storage IP of client side.
  *namelen: The *namelen storage size of IP of client side

- **Return Value:** 0: successful.
  -1: error.
- **Reference:** *socket, listen, accept, closesocket*

- **For example:**

```
int s;                    /* socket */
struct sockaddr_in socka;     /* Local port */
…
memset( &socka, 0, sizeof( socka ) );   /* set zero */
socka.sin_family = AF_INET;
socka.sin_port = htons( 2000 );     /* set socket port */
if( bind( s, (struct sockaddr *) &socka, sizeof( spcka ) ) < 0 )
    Print("Error in bind \n\r");
```

4.1.7 closesocket

- **Description:** Closes a socket.

- **Syntax:** int closesocket(int s);

- **Input Parameter:**
  s: Socket ID.

- **Return Value:** 0: successful..
  -1: error.

4.1.8 connect

- **Description:** Initates a connection at a socket.

- **Syntax:** int connect(int s, struct sockaddr *name, int namelen);

- **Input Parameter:**
  s: Socket ID.
  name: structure that identifies the remote end of the connect.
  namelen: size of name.

- **Return Value:** 0 : successful.
  -1 : error.
- **Reference:** *closesocket*

- **For example:**

```
   …
if( connect( s, (struct sockaddr *) &socka, sizeof( socka ) ) < 0 )
     Print("Error connecting to remote server.\n\r");
```

4.1.9 fcntlsocket

- **Description:** socket control flags.

- **Syntax:** int fcntlsocket(int s, int cmd, int arg);

- **Input Parameter:**
  s: Socket ID.

- **Return Value:** 0: successful. .
  -1: Error.

4.1.10 gethostbyname

- **Description:** Returns the IP address which corresponds to a host name.
- **Syntax:** struct hostent *gethostbyname(char *name);

- **Input Parameter:**
  *name: the name of the host.

- **Return Value:** 0: error.
  !=0: successful.

4.1.11 getsockname

- **Description:** Gets the local address information of a socket.
- **Syntax:** int getsockname(int s, struct sockaddr *name, int *namelen);
- **Input Parameter:**
  s: socket ID.
  name: the name of the host.
  namelen: length of the name
- **Return Value:** -1: error.
  0: successful.
- **Reference:** gethostbyname_r
- **For example:**

```
int s;
…
s = socket( PF_INET, SOCK_DGRAM, 0 );
…
if( getsocketname( s, (struct sockaddr *)&socka, &socksize ) < 0 )
    Print("Error in getsockname.\n\r");
```

4.1.12 ioctlsocket

- **Description**: socket sets control parameters.

- **Syntax:** int ioctlsocket(int s, int request, char *arg);

- **Input Parameter:**
  s: socket ID.
  request: request type.
  arg: Optional argument.

- **Return Value:** 0: successful.
  -1: error.

4.1.13 listen

- **Description:** Listens to connections.

- **Syntax:** int listen(int s, int backlog);

- **Input Parameter:**
  s: socket ID.
  backlog: Set the maximum number of connections.

- **Return Value:** 0 : successful.
  -1: error.
- **Reference:** *recv, recvfrom, recvmsg*

- **For example:**

```
int s;
…
if( listen( s, 5 ) < 0 )
    Print("Error calling listen.\n\r");
```

4.1.14 readsocket

- **Description:** receives data from a socket ID.

- **Syntax:** int readsocket(int s, char *buf, int len);

- **Input Parameter:**
  s: socket ID.
  buf: received buffer.
  len: Maximum number.

- **Return Value:** >=0: successful.
  - 1: error.

- **Reference:** *recv, recvfrom, recvmsg*

4.1.15 recv

- **Description:** receives messages from a TCP connection.

- **Syntax:** int recv(int s, char *buf, int len, int flags);

- **Input Parameter:**
  s: socket ID.
  buf: receive buffer.
  len: Maximum number.
  flags: Always 0.

- **Return Value:** >=0: successful
  -1: error.
- **Reference:** *recv, recvfrom, recvmsg*

- **For example:**

```
int rc;           /* return code */
int s1, s2;       /* s1 is old socket, s2 is new socket */
unsigned char buff[9];   /* read buffer */
…
s2 = accept( s1, (struct socket socladdr *)&socka, &socksize );
…
rc = recv( s2, buff, 8, 0 );  /* set the receive maximum bytes are 8
                                 bytes */
if( rc < 0 )
    Print("Error receiving data. \n\r");
else if( rc == 8 )
    Print("Success: read 8 byes.\n\r");
else
    Print("Error: Did not retrieve 8 bytes.\n\r");
```

4.1.16 recvfrom

- **Description:** receives messags from a UDP connection .

- **Syntax:** int recvfrom(int s, char *buf, int len, int flags, struct
  sockaddr *from, int *fromlen);

- **Input Parameter:**
  s: socket ID.
  buf: receive buffer.
  len: Maximum number.
  flags: Always 0.
  from: The remote host to which the connection should be made.
  fromlen: Size of data structure.

- **Return Value:** >=0: successful.
  - 1: error.

- **Reference:** *recv, recvmsg*

- **For example:**

```
int s1, int s2;                   /* s1 is old socket, s2 is new socket */
int rc;                      /* return code */
unsigned char buff[64];
struct sockaddr_in socka;          /* remote host address */
…
socka.sin_port = htons( 2000 );      /* set the local host port */
rc = recvfrom( s2, buff, 8, 0, (struct sockaddr *)&socka, &socksize );
if( rc < 0 )
    Print("Error receiving data. \n\r");
else if( rc == 8 )
    Print("Success: read 8 byes.\n\r");
else
    Print("Error: Did not retrieve 8 bytes.\n\r");
```

4.1.17 recvmsg

- **Description:** receives data.

- **Syntax:** int recvmsg(int s, msghdr *msg, int flags);

- **Input Parameter:**
  s: socket ID.
  msg: received data should be stored.
  flags: Always 0.

- **Return Value:** >=0: successful.
  -1: error.
- **Reference:** *recv, recvfrom*

4.1.18selectsocket

- **Description:** Waits for activity of a set of sockets.

- **Syntax:** int selectsocket(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

- **Input Parameter:**
  nfds: socket ID.
  readfds: Socket ID which *selectsocket()* should return if data
  becomes
  available.
  writefds: Socket ID which *selectsocket()* should return if the socket
  can
  accept more data.
  exceptfds: Socket ID which *selectsocket()* should return if
  out-of-band data
  is available.
  timeout: set time out .

- **Return Value:** >0: this number of sockets are ready.
  0: timeout occurred.
  - 1: error.

- **For example:**

```
int s1 s2, s3;
int rc;
fd_set socket_set1, socket_set2;
…
FD_ZERO( &socket_set1 );
FD_ZERO( &socket_set2 );
FD_SET( s1, &socket_set1 );
FD_SET( s2, &socket_set1 );
FD_SET( s3, &socket_set2 );
rc= selectsocket( 3, socket_set1, socket_set2, 0, NULL );
if( rc < 0 )
```

```
     Print(“Error, no sockets ready.\n\r”);
else
     Print(“%d sockets ready.\n\r”,rc);
 if( FD_ISSET( s1, &socket_set )
     Print(“Socket 1 is ready to be read.\n\r”);
else if( FD_ISSET( s2, &socket_set2 )
     Print(“Socket 2 is ready to be written.\n\r”);
else if( FD_ISSET( s3, &socket_set3 )
     Print(“Socket 3 is ready to be read.\n\r”);
else
     Print(“Error.\n\r”);
```

4.1.19send

- **Description:** sends a message by an established TCP connection.

- **Syntax:** int send(int s, char *buf, int len, int flags);

- **Input Parameter:**
  s: socket ID.
  buf: data buffer.
  len: data of maximum number of bytes to be send.

- **flags:** Always 0.

- **Return Value:** >=0 : successful.
  - 1 : Error.

- **Reference:** *sendto, sendmsg*

- **For example:**

```
unsigned char buff[128];
int s1, s2;
…
if( send( s2, buff, sizeof(buff), 0 ) < 0 )
   Print("Error sending data.\n\r");
```

4.1.20 sendmsg

- **Description:** sends a message which can be split between buffers.
- **Syntax:** int sendmsg(int s, msghdr *msg, int flags);
- **Input Parameter:**
  s: socket ID.
  msg: data buffer.
- **flags:** Always 0.
- **Return Value:** >=0: successful.
  - 1: error.
- **Reference:** *send, sendto*

4.1.21 sendto

- **Description:** sends a message on an established UDP connection.

- **Syntax:** int sendto(int s, char *buf, int len, int flags, struct sockaddr *to, int tolen);
- **Input Parameter:**
  s: socket ID.
  buf: data buffer.
  len: data of maximum number of bytes to be send.
  flags: Always 0.
  to: Specifies the remote host to which the message will be sent.
  tolen: Size of the to data structure.

- **Return Value:** >=0: successful.
  - 1: Error.

- **Reference:** *sendto, sendmsg*

- **For example:**

```
int s1, s2;
…
if( sendto( s2, "HelloWorld", 10, 0, (struct sockaddr *) sizeof( socka ) )
< 0 )
   Print("Error sending data.\n\r");
```

4.1.22 shutdown

- **Description:** shuts down part of a connection.

- **Syntax:** int shutdown(int s, int how);

- **Input Parameter:**
  s: socket ID.
  how: 0 shutdown receive data path.
  1 shutdown send data path, TCP sends FIN.
  2 shutdown send and receive path.

- **Return Value:** 0 : successful.
  -1: error.
-
- **Reference:** *closesocket*

4.1.23 socket

- **Description:** creates a socket.

- **Syntax:** int socket(int domain, int type, int protocol);

- **Input Parameter:**
  domain: This should always be PF_INET.

type:	(1). SOCKET_STREAM	stream socket is TCP/IP.
	(2). SOCKET_DGRAM	data gram socket is UDP/IP.
	(3). SOCKET_RAW	raw-protocol interface.

  protocol: Always 0.

- **Return Value:** >=0: successful.
  -1 : error.

- **Reference:** *closesocket*

- **For example:**

```
int s1, s2;
…
if( Is7188e() == 0)  /* I7188E detect */
{
    Print("The Hardware is not I-7188E.\n\r");
    exit(1);
}
…
s1= socket( PF_INET, SOCKET_STREAM, 0);
if(  s1 < 0 )
    Print("Error opening socket.\n\r");
```