DM&P DOS Socket Library

DSock Programmer's Manual

Version 2007-08-08

Table of Content

2004-12-09

What DSock Is

DSock is a TCP/IP library for DOS real mode, which is used by our Internet applications. It provides simple C functions for programmer to write Internet applications. We also provide Internet examples using DSock: BOOTP / DHCP, FTP server, SMTP client / server, HTTP server, TELNET server, Talk client / server, etc.

DSock provides a lot of example source code. Programmer can add Internet functions to their project easily and save a lot of time. We also provide a utility "MakeROM" (http://www.dmp.com.tw/tech/makerom) which can make a ROM image for programmer to fit their application, DOS and BIOS into one 512KB flash ROM like Mity-Mite Module Demo Box (http://www.dmp.com.tw/app/mitymite) do.

DSock is free for DM&P products using M6117D/Vortex86 CPU and business version is also available for other x86 CPUs. If you want to use DSock on other x86 CPUs, mail to info@icop.com.tw please.

Just running DSock examples, our single board computers with Ethernet interface can be a Web / FTP server. You can get benefits from DSock:

- It's free for DM&P signal board computers using M6117D/Vortex86.
- Easily to use with BSD-like C functions.
- A lot of useful examples source code to save time.
- Can be fitted into one flash ROM with our X-DOS like Mity-Mite Module Demo Box.
- Technical support.

DSock is compiled with Turbo C 2.0 and we recommend you to use Borland C++ / Turbo C++ to build your program. Download Turbo C++ 1.01 form http://community.borland.com/article/images/21751/tcpp101.zip (2.63 MB)

Update List

Version	Date	Update
0.61	2007/08/08	● Support Ethernet in Vortex86SX SoC.
0.60	2004/12/09	● Support AX88796 100M LAN chipset for M6117D series. ● Fix checksum bug for odd-bytes packets.
0.51	2002/07/24	Time out check added in SocketGetString() to avoid halt.
0.50	2002/06/11	Support Vortex86.
0.42	2002/05/20	● DSock programming guide and library reference updated. ● Function SocketFlush() and SocketFlushNext() added.
0.41	2002/05/16	● DSock programming guide updated. ● Function SocketIsTcp/UdpPortUsed() and SocketFindFreeTcp/UdpPort() added. ● SocketIsConnected() fixed.
0.40	2002/04/30	● DSock programming guide updated. ● Re-build DSock library with Turbo C 2.0. ● Function SocketSendTo(), SocketRecvFrom() and DSock_Version added. ● UDP Talk example updated.
0.33	2002/04/15	● Rebuild DSock library to be compatible with Turbo C++ 1.01. ● Rebuild examples to compile okay with Turbo C++ 1.01. ● DSock programming guide added.
0.32	2002/04/01	● SocketSend2() is added. It will read data in buffer and return immediately. ● Library reference document updated.
0.31	2002/01/17	Time out check added in SocketRecv() to avoid halt.
0.30	2002/01/15	PING command added.
0.21	2002/01/09	Time out check added in SocketSend() to avoid halt.
0.20	2001/12/11	● SocketPutString() improved. ● SMTP server example added. ● POST and GET method supported in HTTP server example. ● FTP/HTTP/SMTP/TELNET server functions are charged to non-blocking. Programmer can call those functions more easily. ● Start-up project supports FTP/HTTP/SMTP/TELNET server for programmer. Programmers can use this project to development their Internet application with DSock quickly. It already supports FTP/HTTP/SMTP/TELNET server function, and use BOOT/DHCP to get network setup. Add their job in MY_JOB.C and find remark with "@CODE" to add/improve service function. It is for people who want to create Internet application quickly and easily. This project is also used by our demo program of Mity-Mite Module demo box.
0.13	2001/11/22	Talk server example added. It will show you how to write a server program to accept multiple clients.

0.12	2001/09/19	SocketRecv2() is added. It will read data in buffer and return immediately.
0.11	2001/08/06	Add packet driver at \lib & \demo\exe.
0.10	2001/08/02	Original Release.

Specifications

Hardware Requirement	**Operating System**
- DM&P M6117D/Vortex86/Vortex86SX signal board computers - Ethernet interface. - 512 KB RAM - 64 KB Disk space - Realtek 8019/8139/AX88796 10/100 Base-T.	- X-DOS - DR-DOS - MS-DOS - FreeDOS
Packet Driver Requirement	**Protocols**
- RTL8019/8139/NE2000 compatible - AX88796 - R6040 in Vortex86SX	- IP / ICMP / ARP - TCP / UDP - BOOTP / DHCP - SMTP / HTTP / FTP / TELNET (Example)

Examples in DSock Library

Example Name	Description
DEMO\BOOTP	Use BOOTP/DHCP to get network setting from DHCP server.
DEMO\DNS	To get IP address from domain name.
DEMO\FTPD	FTP server example, default user name is "dmp" and password is "dmp".
DEMO\HTTPD	Web server example.
DEMO\SMTP	A simple program to send mail.
DEMO\SMTPD	SMTP server example.
DEMO\TELNETD	Simple TELNET server example, default user name is "dmp" and password is "dmp".
DEMO\TALK_S	Talk server example with multiple TCP connections.
DEMO\TALK_TCP	Talk example with TCP.
DEMO\talk_tcp_win	Talk (TCP) Windows version written with WinSock.
DEMO\TALK_UB	Talk example with broadcast.
DEMO\talk_ub_win	Talk (UDP) Windows version written with WinSock.
DEMO\TALK_UDP	Talk example with UDP.
DEMO\talk_udp_win	Talk (UDP) Windows version written with WinSock.
DEMO\SOCK_AP	A start-up project for programmer that support FTP/HTTP/SMTP/TELNET.

How to Run Example

Before running examples, you should load packet driver first.

RTL8019

The default I/O address and IRQ of DMP M6117D series are 0x320 and 5. Run NE2000.COM to install packet driver:

```
C:\DSOCK\DEMO\EXE>ne2000 0x62 5 0x320
```

Where software interrupt 62H can be any number between 60H and 70H. Just run pktdrv.bat in directory "EXE" to load packet driver:

```
C:\DSOCK\DEMO\EXE>pktdrv (Load packet driver for RTL8019)
C:\DSOCK\DEMO\EXE>ftpd   (Run any DSock example program)
```

RTL8100/8139

For Vortex86, you should use another packet driver:

```
C:\DSOCK\DEMO\EXE>rtspkt 0x62
```

Where software interrupt 62H can be any number between 60H and 70H. Or run pktdrv2.bat in directory "EXE" to load packet driver:

```
C:\DSOCK\DEMO\EXE>pktdrv2 (Load packet driver for RTL8139)
C:\DSOCK\DEMO\EXE>ftpd    (Run any DSock example program)
```

AX88796

For M6117D with 100M LAN, you should use AX88796 packet driver:

```
C:\DSOCK\DEMO\EXE>796pkt 0x62 5 0x320
```

where software interrupt 62H can be any number between 60H and 70H. Or run pktdrv3.bat in directory "EXE" to load packet driver:

```
C:\DSOCK\DEMO\EXE>pktdrv3 (Load packet driver for AX88796)
C:\DSOCK\DEMO\EXE>ftpd    (Run any DSock example program)
```

Vortex86SX

For Vortex86SX with 100M LAN, you should use R6040 packet driver:

```
C:\DSOCK\DEMO\EXE>r6040pd 0x62
```

where software interrupt 62H can be any number between 60H and 70H. Or run pktdrv4.bat in directory "EXE" to load packet driver:

```
C:\DSOCK\DEMO\EXE>pktdrv4 (Load packet driver for Vortex86SX)
C:\DSOCK\DEMO\EXE>ftpd    (Run any DSock example program)
```

Note: The default setting of network is on DSOCK.CFG. You should modify the default setup upon your network. After changing the DSOCK.CFG, you can run examples.

Update Utility for DSock

Update utility is a Windows program to upload files by FTP protocol. Use DSock to develop your application and support FTP service. You can use this utility to auto-upload files to update your firmware. Our auto-update program for WebCamera (http://www.dmp.com.tw/app/webcamera) and RSIP (http://www.dmp.com.tw/app/rsip) use Update utility to do firmware updating. If you use DSock to build your application with FTP service, package your new firmware with Update utility will be very useful for programmer to update firmware of his products.

For example, user can download RSIP auto-update form our web site to update firmware. RSIP uses a 512KB flash disk contains BIOS, X-DOS and RSIP main program like Mity-Mite Module (http://www.dmp.com.tw/app/mitymite).

Update Utility Page: http://www.dmp.com.tw/tech/dsock/update

DSock Programming Guide

DSock library file download from our web site is a ZIP file. Unzip it and will create a directory "DSock". Find a lot of examples in "DSock\Demo". All examples are compiled and put in "DSock\Demo\EXE".

DSOCK.LIB is for DOS large memory mode. It needs 128KB RAM to load DSock. Add DSOCK.LIB to your project and include DSOCK.H to start your software development. DSock is compiled with Turbo C 2.0 and demo programs are compiled with Turbo C++ 1.01. For this, we recommend programmer to use Borland C++/Turbo C++ to build his program and DSock is not compatible with Microsoft Visual C++ 1.5. If programmer do not have Borland or Turbo C/C++ compiler, download it form http://community.borland.com/article/images/21751/tcpp101.zip (2.63 MB)

Before reading programming guide, programmer must understand basic TCP/IP protocol.

Add DSock Library into Your Project

Find dsock.h and dsock.lib at dsock\lib. Copy those two files to the directory your application are for easy use. Add dsock.lib into project and put dsock.h to head of your source code. The fast way to start DSock project is to open start-up project of DSock example - "DSock\Demo\DSock_AP."

Before testing your program, load packet driver first. Packet driver can be found at "DSock\Lib\". The DSock library files:

File Name	Description
LIB\DSOCK.H	DSock library head file for C.
LIB\DSOCK.LIB	DSock library for DOS, large mode.
LIB\DSOCK.CFG	Default network configuration file.
LIB\LIB.HTM	DSock library reference.
LIB\PKTDRV\NE2000.COM	Packet driver for RTL8019.
LIB\PKTDRV\PKTDRV.BAT	DOS batch file to load RTL8019 packet driver.
LIB\PKTDRV\RTSPKT.COM	Packet driver for RTL8139/RTL8100.
LIB\PKTDRV\PKTDRV2.BAT	DOS batch file to load RTL8139 packet driver.
LIB\PKTDRV\796PKT.COM	Packet driver for AX88796.
LIB\PKTDRV\PKTDRV3.BAT	DOS batch file to load AX88796 packet driver.

DSock Functions Overview

System Functions

Those functions can initialize and close DSock library:

Function	Description
DSock_Open()	Open the socket library.
DSock_Close()	Close the socket library.
DSock_DoBootp()	Get Network setup from BOOTP/DHCP server.
DSock_LoadConfigFile()	Load configuration file.

Use DSock functions to set/get network configuration when you start DSock. There are:

Function	Description
DSock_GetMacAddr()	Get MAC address from network card.
DSock_GetHostIp()	Get host IP address.
DSock_SetHostIp()	Set host IP address.
DSock_GetNetmask()	Get netmask of local host.
DSock_SetNetmask()	Set netmask of local host.
DSock_GetGateway()	Get gateway IP address.
DSock_AddGateway()	Add gateway to DSock.
DSock_GetDomainNameServer()	Get domain name server from DSock.
DSock_AddDomainNameServer()	Add domain name server to DSock.
DSock_Resolve()	Convert domain name to IP address.

Help Functions

Those functions help programmer to convert host to network byte ordering and get IP address:

Function	Description
inet_ntoa()	Convert a network address into a string in dot notation.
inet_addr()	Convert a string containing a dotted IP address into a DWORD.
ntohs()	Convert a word from network to host byte order.
ntohl()	Convert a DWORD word from network to host byte order.
htons()	Convert a word from host to network byte order.
htonl()	Convert a double word from host to network byte order.

Socket Functions

Use those function to establish a connection. See server & client section to get more:

Function	Description
SocketCreate()	Create a socket.
SocketDestory()	Release a socket.

SocketClose()	Close a socket.
SocketAbort()	Abort a socket.
SocketBind()	Associate a local address with a socket.
SocketListen()	Establish a socket to listen for incoming connection.
SocketAccept()	Accept a connection on a socket. It's a non-blocking function.
SocketConnect()	Establish a connection with a peer.
SocketIsConnected()	Check connection of a socket.
SocketIsTcpPortUsed()	Is this port number used by DSock TCP sockets?
SocketIsUdpPortUsed()	Is this port number used by DSock UDP sockets?
SocketFindFreeTcpPort()	Find a free TCP port.
SocketFindFreeUdpPort()	Find a free UDP port.
SocketFlush()	Send pending data.
SocketFlushNext()	Cause next transmission to have a flush.

When socket connection is established (UDP is not need connection), use those functions to transmit data:

Function	Description
SocketSend()	Send data to a connected socket.
SocketSend2()	Non-blocking version of SocketSend().
SocketRecv()	Receive data from a socket.
SocketRecv2()	Non-blocking function of SocketRecv().
SocketPutChar()	Write a character to a socket.
SocketGetChar()	Read a character from a socket.
SocketGetString()	Read a string from a socket
SocketPutString()	Write a string to a socket.
SocketDataReady()	Check incoming data of a socket.

Start DSock

Packet driver is used by DSock to send/receive TCP/IP packets. DSock will check packet driver and CPU when it is started. Use **DSock_Open()** to initialize DSock and **DSock_Close()** to free it. Loading network setup is the second job programmer should do. Use **DSock_LoadConfigFile()** to load setup file or use **DSock_DoBootp()** to get network setup if BOOT/DHCP servers exist. "DSOCK.CFG" is a text configuration file used by DSock. See DSOCK.CFG section to get more information. Here is a simple example to start DSock:

```
#include "dsock.h"
#include <stdio.h>

int main()
```

```
{
  /* Initialize DSock library */
  if(DSock_Open()==FALSE)
  {
    printf("Unable to initialize socket library\n");
    return 1;
  }

  /* Use BOOTP/DHCP to get setup */
  if(DSock_DoBootp()==TRUE)
  {
    printf("Load network setup from BOOTP/DHCP\n");
  }
  else /* Load setup from config file */
  {
    printf("Unable to load setup from BOOTP/DHCP server\n");
    DSock_LoadConfigFile("dsock.cfg");
    printf("Load network setup from DSOCK.CFG\n");
  }

  /* You code here */

  /* Close DSock library */
  DSock_Close();
  return 0;
}
```

Use DNS

If "nameserver" is added in DSOCK.CFG or use DSock_AddDomainNameServer() to add domain name server, Resolve() can be used to get IP address from domain name.

```
char szBuf[32];
DWORD dwIp = DSock_Resolve("www.dmp.com.tw");
printf("IP of www.dmp.com.tw is %s\n",inet_ntoa(dwIp));
```

Use TCP

The Transmission Control Protocol provides a reliable, sequenced, connection-oriented service on top of IP, and on an end-to-end basis. It supports multiple connections through the use of ports number. Use TCP socket is easy: pass TCP_SOCKET to SocketCreate().

```
SOCKET s = SocketCreate(TCP_SOCKET);
```

Use UDP

The User datagram Protocol provides un-sequenced, unreliable, connectionless service. Pass UDP_SOCKET to SocketCreate() to create UDP socket:

```
SOCKET s = SocketCreate(UDP_SOCKET);
```

Server/Client Mode of TCP

TCP is a connection-oriented service that needs server & client mode. Server should listen for client and client has to connect to server. It is the table to show this:

Server	Client
SocketCreate()	SocketCreate()
SocketBind()	
SocketListen()	
	SocketConnect()
SocketAccept()	
SocketSend()/SocketRecv()	SocketRecv()/SocketSend()
SocketClose()	SocketClose()
SocketDestory()	SocketDestory()

There is a example "TALK_TCP" from DSock example to show the server-client mode by source code:

Server	Client

```
s = SocketCreate(TCP_SOCKET);
if(s==INVALID_SOCKET)
{
  printf("SocketCreate () error\n");
  DSock_Close();
  return 1;
}

if(nArgCnt==1) /* Server mode */
  TalkServer(s);
else         /* Client mode */
  TalkClient(s,pszArg[1]);
```

if(SocketBind(s,0L,TALK_PORT)==FALSE)
{
  printf("SocketBind() error\n");
  return FALSE;
}	

if(SocketListen(s)==FALSE)
{
  printf("SocketListen() error\n");
  return FALSE;
}
printf("Talk server mode,
listening...\n");	

	printf("Talk client mode, connecting to
	server...\n");
	
	/* Connect to server */
	if(SocketConnect(s,inet_addr(szServer),
	  TALK_PORT)==FALSE)
	{
	  printf("SocketConnect() error\n");
	  return FALSE;
	}

/* Wait for client */
while(TRUE)
{
  if(kbhit())
  {
  printf("Break by user\n");
  break;
  }
 	
  if(SocketAccept(s,&dwIp))
  {
  bConnected = TRUE;
  inet_ntoa(szBuf,dwIp);
  printf("Connected with %s\n",szBuf);
  break;
  }
}	

/* Is connected with client ? */	printf("Connected to

```
if(bConnected)
{
  printf("Start to talk...\n");
  while(TRUE)
  {
    /* Check key press */
    if(kbhit())
    {
      char c = getch();
      SocketPutChar(s,c);
      if(c==27)
      {
        printf("\nProgram
terminated\n");
        break;
      }
    }
    /* Check message sent by client */
    if(SocketDataReady(s))
    {
      char c;
      SocketGetChar(s,&c);
      printf("%c",c);
      if(c==27)
      {
        printf("\nProgram
terminated\n");
        break;
      }
    }
  }
}
```

```
%s:%d\n",szServer,TALK_PORT);
while(TRUE)
{
  /* Check key press and send it out */
  if(kbhit())
  {
    char c = getch();
    SocketPutChar(s,c);
    if(c==27)
    {
      printf("\nProgram terminated\n");
      break;
    }
  }
  /* Check key press sent by server */
  if(SocketDataReady(s))
  {
    char c;
    SocketGetChar(s,&c);
    printf("%c",c);
    if(c==27)
    {
      printf("\nProgram terminated\n");
      break;
    }
  }
}
```

```
  SocketClose(s);
  SocketDestory(s);
```

Server/Client Mode of UDP

Because UDP is connectionless, there is no server or client. Use UDP to send/receive data is easier than TCP. See the example "TALK_UDP" of DSock, it will receive all key press message sent to it and send back the message:

```
s = SocketCreate(UDP_SOCKET);
```

```
if(s==INVALID_SOCKET)
{
  printf("SocketCreate() error\n");
  DSock_Close();
  return 1;
}

if(SocketBind(s,0xFFFFFFFFL,1234)==FALSE)
{
  printf("SocketBind() error\n");
  DSock_Close();
  return 1;
}

printf("Start to talk, any press will be broadcast...\n");
while(TRUE)
{
  /* User press keyboard ? */
  if(kbhit())
  {
    char  c = getch();
    if(c == 27)
    {
      printf("\nProgram terminated\n");
      break;
    }
  }

  /* Is there any broadcast message ? */
  if(SocketDataReady(s))
  {
    char  c;
    /* Save remote IP and port */
    SocketRecvFrom(s, &dwAddr, &wPort,&c,1);
    printf("From %s:%d, send '%c' back.\n", inet_ntoa(szBuf, dwAddr), wPort, c);
    /* Send key press back to original port */
    SocketSendTo(s, dwAddr, wPort, &c, 1);
    if(c == 27)
    {
      printf("\nProgram terminated\n");
      break;
```

```
    }
  }
}

SocketClose(s);
SocketDestory(s);
```

UDP Broadcast

Those codes are from TALK_UB to show you how to use UDP to broadcast. It will receive broadcast message and broadcast its key press message:

```
s = SocketCreate(UDP_SOCKET);
if(s==INVALID_SOCKET)
{
  printf("SocketCreate() error\n");
  DSock_Close();
  return 1;
}

if(SocketBind(s,0xFFFFFFFFL,1234)==FALSE)
{
  printf("SocketBind() error\n");
  DSock_Close();
  return 1;
}

printf("Start to talk, any press will be broadcast...\n");
while(TRUE)
{
  /* User press keyboard ? */
  if(kbhit())
  {
    char c = getch();
    SocketPutChar(s,c);
    if(c==27)
    {
      printf("\nProgram terminated\n");
      break;
    }
```

```
  }
  /* Is there any broadcast message ? */
  if(SocketDataReady(s))
  {
    char c;
    SocketGetChar(s,&c);
    printf("%c",c);
    if(c==27)
    {
      printf("\nProgram terminated\n");
      break;
    }
  }
}

SocketClose(s);
SocketDestory(s);
```

Blocking & Non-blocking

Blocking function will return when job finish. Non-blocking function will return immediately. For example: SocketAccept() is non-blocking function. So, we have use a while loop to check client connection. The benefit is that we can do more jobs via polling method under DOS single-task environment. You can see "SOCK_AP" example of DSock to understand how to use DSock to accept FTP/HTTP/TELNET/SMTP connections. Non-blocking functions of DSock are: SocketAccept(), SocketSend2(), SocketRecv2().

Multiple Connections

In order to save system resource, SocketAccept() will not return a socket descriptor. If programmer want accept multiple connections, he should declare more sockets. See the example code:

```
#define MAX_SOCKET 5

SOCKET s[MAX_SOCKET];

for(i=0;i<MAX_SOCKET;i++)
{
  s[i] = SocketCreate(TCP_SOCKET);
  if(s==INVALID_SOCKET)
```

```
    return;
  if(SocketBind(s[i],0L,nPort)==FALSE)
    return;
  if(SocketListen(s[i])==FALSE)
    return;
}

while(TRUE)
{
  for(i=0;i<MAX_SOCKET;i++)
  if(SocketAccept(s[i],&dwIp))
  {
    /* Do Job Here */
    break;
  }
}
```

DSOCK.CFG

DSOCK.CFG is a text file to save network information for DSock. There are two way for DSock to get network setup: BOOT/DHCP or DSOCK.CFG. Call DSock_DoBootp() to use BOOT/DHCP protocol to get network setting when BOOT / DHCP servers exist. Or write network setting into DSOCK.CFG that DSock will read setting form it. Use DSOCK.CFG can guarantee IP address is fixed. There are four main statement of DSOCK.CFG: **ip**, **netmask**, **gateway**, **nameserver**,

Typical setup of DSOCK.CFG:

```
ip=192.168.0.234
netmask=255.255.255.0
gateway=192.168.0.1
nameserver=192.168.0.1
```

If you have no DNS server, you can do this: "nameserver=". If you have no gateway, the same way as nameserver tag.

Packet Driver

Because of DSock is designed for M6117D which is a 386 SX CPU, Realtek 8019AS chip will used by M6117D series. The default IRQ is set to 5 and I/O address is 320H. So just add those into your autoexec.bat:

```
ne2000 0x62 5 0x320
```

For Vortex86, the LAN chipset is 8100/8139. It needs other packet driver:

```
C:\DSOCK\DEMO\EXE>rtspkt 0x62
```

Where software interrupt 62H can be any number between 60H and 70H. Or run pktdrv2.bat in directory "EXE" to load packet driver:

Install Your Program

When finish your program, it will be installed into single board PC. For DOS, autoexec.bat can start your program when boot. Because packet driver is needed by DSock, packet driver should be installed. Necessary files are:

File Name	Description
X-DOS	Programmer can freely use X-DOS on DM&P single board PC.
AUTOEXEC.BAT	Load packet driver and start your program.
NE2000.COM	Packet driver for 8019AS on M6117D series.
DSOCK.CFG	DSock configuration file.
Your App	Your program uses DSock.

If device can boot with X-DOS, just copy those files into your storage (Flash Disk/DOC/DOM). Or install X-DOS first. Programmer can use "MakeROM" utility to make a ROM image for Mity-Mite Module to save DOC/DOM if Mity-Mite Module is uesed. Now, your device will have Internet function when it power on.

More Information

DM&P have experience to support OEM/ODM projects since 1989. Owing our kernel technology as below that we guarantee short lead-time from scratch to physical sample:

- Proprietary DM&P M6117D 386SX CPU kernel technology.
- Build-in GPIO function in CPU.
- Capable of supporting DM&P BIOS and O/S (X-DOS, Embedded Linux, etc.) modifications.
- Capable of integrating application software, O/S and BIOS into a single chip (Flash, EPROM or ASIC).
- Provide consultant DSP/ASIC design.
- All-in-one design and guarantee for smallest PCB design upon request.
- Supports operating temperature approximately from -20 to +60 Degree C.
- Flexibility of custom specifications.
- Guarantees afford ability and long-term support of all products.

Jan Yin Chan Electronics Co., LTD.
TEL: +886-2-22980770
FAX: +886-2-22991883
Address: 8F, No.12 Wu-Quan 7 Rd., Wu-Gu Industrial Park (PostCode:248) Wu Gu Xiang, Taipei Hsien, Taiwan, R.O.C.

Visit DM&P group web site: http://www.dmp.com.tw to get more information and resource on-line.

Technical Support

For more technical support, please visit http://www.dmp.com.tw/tech or mail to tech@dmp.com.tw.

Library Reference

Library Function	Socket Function	Help Function
DSock_Open()	SocketCreate()	inet_ntoa()
DSock_Close()	SocketDestory()	inet_addr()
DSock_Version()	SocketClose()	ntohs()
DSock_DoBootp()	SocketAbort()	htons()
DSock_LoadConfigFile()	SocketBind()	ntohl()
DSock_GetMacAddr()	SocketListen()	htonl()
DSock_AddGateway()	SocketAccept()	
DSock_GetGateway()	SocketConnect()	
DSock_AddDomainNameServer()	SocketRecv()	
DSock_GetDomainNameServer()	SocketRecv2()	
DSock_Resolve()	SocketRecvFrom()	
DSock_GetNetmask()	SocketSend()	
DSock_SetNetmask()	SocketSend2()	
DSock_GetHostIp()	SocketSendTo()	
DSock_SetHostIp()	SocketDataReady()	
	SocketPutChar()	
	SocketGetChar()	
	SocketPutString()	
	SocketGetString()	
	SocketIsConnected()	
	SocketIsTcpPortUsed()	
	SocketIsUdpPortUsed()	
	SocketFindFreeTcpPort()	
	SocketFindFreeUdpPort()	
	SocketFlush()	
	SocketFlushNext()	

BOOL DSock_Open()	
Description:	Open DSock socket library. Programmer has to initialize DSock after all socket operations.
Arguments:	**Return** - TRUE is success and FALSE is error.

Example:	`/* Initialize DSock library */`
	`if(DSock_Open()==FALSE)`
	`{`
	`  printf("Unable to initialize socket library\n");`
	`  return 1;`
	`}`

void DSock_Close()	
Description:	Close DSock socket library.
Arguments:	N/A
Example:	`/* Uninitialize library */` `DSock_Close();`

char *DSock_Version()	
Description:	Get DSock version information.
Arguments:	**Return** - Pointer to DSock version buffer.
Example:	`/* Show DSock version */` `printf("DSock Version %s\n",DSock_Version();`

BOOL DSock_DoBootp()	
Description:	Load network setup from BOOTP/DHCP server.
Arguments:	**Return** - TRUE is success and FALSE is error.

Example:	`/* Use BOOTP/DHCP to get setup */`
	`if(DSock_DoBootp()==TRUE)`
	`{`
	`  printf("Load network setup from BOOTP/DHCP\n");`
	`}`
	`else /* Load setup from config file */`
	`{`
	`  printf("Unable to load setup from BOOTP/DHCP server\n");`
	`  DSock_LoadConfigFile("dsock.cfg");`
	`  printf("Load network setup from DSOCK.CFG\n");`
	`}`

BOOL DSock_LoadConfigFile(char *szFile)	
Description:	Load DSock configuration file.

Arguments:	**Return** - TRUE is success and FALSE is error. **szFile** - Configuration file name.

Example:	`/* Use BOOTP/DHCP to get setup */`
	`if(DSock_DoBootp()==TRUE)`
	`{`
	`  printf("Load network setup from BOOTP/DHCP\n");`
	`}`
	`else /* Load setup from config file */`
	`{`
	`  printf("Unable to load setup from BOOTP/DHCP server\n");`
	`  DSock_LoadConfigFile("dsock.cfg");`
	`  printf("Load network setup from DSOCK.CFG\n");`
	`}`

void DSock_AddGateway(DWORD dwIp)	
Description:	Add gateway to DSock.
Arguments:	**dwIp** - IP address of gateway.
Example:	`/* Set gateway to 192.168.0.1 */` `DSock_AddGateway(inet_addr("192.168.0.1"));`

DWORD DSock_GetGateway()	
Description:	Get gateway IP address.
Arguments:	**Return** - IP address of gateway.

Example:	`char szBuf[32];`
	`DWORD dwGateway = DSock_GetGateway();`
	`inet_ntoa(szBuf,dwGateway);`
	`printf("Gateway = %s\n",szBuf);`

void DSock_AddDomainNameServer(DWORD dwIp)	
Description:	Add domain name server to DSock.
Arguments:	**dwIp** - IP address of DNS.
Example:	`/* Set DNS to 192.168.0.1 */` `DSock_AddDomainNameServer(inet_addr("192.168.0.1"));`

DWORD DSock_GetDomainNameServer()	
Description:	Get domain name server from DSock.
Arguments:	**Return** - IP address of DNS.

Example:	`char szBuf[32];`
	`DWORD dwDNS = DSock_GetDomainNameServer();`
	`inet_ntoa(szBuf,dwDNS);`

	`printf("DNS = %s\n",szBuf);`

DWORD DSock_Resolve(char *szName)	
Description:	Convert domain name to IP address. You have to set your domain namer server first.
Arguments:	**Return** - IP address of the domain name. **szName** - Domain name to be resolved.

Example:	`char szBuf[32];`
	`DWORD dwIp = DSock_Resolve("www.dmp.com.tw");`
	`printf("IP of www.dmp.com.tw is %s\n",inet_ntoa(dwIp));`

BYTE *DSock_GetMacAddr()	
Description:	Get MAC address from network card.
Arguments:	**Return** - A byte pointer to MAC address of network card.

Example:	`BYTE *pbyMac = DSock_GetMacAddr();`
	`printf("MAC Address is %02X-%02X-%02X-%02X-%02X-%02X\n",`
	`pbyMac[0],pbyMac[1],pbyMac[2],`
	`pbyMac[3],pbyMac[4],pbyMac[5]);`

DWORD DSock_GetHostIp()	
Description:	Get host IP address.
Arguments:	**Return** - Host IP address.

Example:	`char szBuf[32];`
	`DWORD dwIp = DSock_GetHostIp();`
	`inet_ntoa(szBuf,dwIp);`
	`printf("My IP is %s\n",szBuf);`

void DSock_SetHostIp(DWORD dwIp)	
Description:	Set local IP address.
Arguments:	**dwIp** - The local IP address you want to assign.
Example:	`/* Set local IP to 192.168.0.50 */` `DSock_SetHostIp(inet_addr("192.168.0.50"));`

DWORD DSock_GetNetmask()	
Description:	Get netmask of local host.
Arguments:	**Return** - Netmask of local TCP/IP.

Example:	`char szBuf[32];`
	`DWORD dwNetmask = DSock_GetNetmask();`
	`inet_ntoa(szBuf,dwNetmask);`
	`printf("Netmask : %s\n",szBuf);`

void DSock_SetNetmask(DWORD dwNetmask)	
Description:	Set netmask of local host.
Arguments:	**dwNetmask** - Set netmask of local TCP/IP.
Example:	`/* Set netmask to 255.255.255.0 */` `DSock_SetNetmask(inet_addr("255.255.255.0"));`

char *inet_ntoa(char *sz,DWORD dw)	
Description:	Convert a network address into a string in dot notation.
Arguments:	**Return** - Pointer to sz. **sz** - Buffer for ASCII string of IP address. **dw** – IP address want to be converted.

Example:	`char szBuf[32];`
	`DWORD dwIp = DSock_GetHostIp();`
	`inet_ntoa(szBuf,dwIp);`
	`printf("My IP is %s\n",szBuf);`

DWORD inet_addr(char *sz)	
Description:	Convert a string containing a dotted IP address into a DWORD.
Arguments:	**Return** - Converted IP address. **sz** - ASCIIZ string of IP address.
Example:	`/* Set local IP to 192.168.0.50 */` `DSock_SetHostIp(inet_addr("192.168.0.50"));`

WORD ntohs(WORD w)	
Description:	Convert a word from network to host byte order.
Arguments:	**Return** - Host byte order data. **w** - Network byte order data.
Example:	`WORD w = 0x1234;` `printf("w=%04x, ntohs(w)=%04x\n",w,ntohs(w));`

DWORD ntohl(DWORD dw)	
Description:	Convert a DWORD word from network to host byte order.
Arguments:	**Return** - Host byte order data. **dw** - Network byte order data.
Example:	`DWORD dw = 0x12345678L;` `printf("dw=%08lx, ntohl(dw)=%08lx\n",dw,ntohl(dw));`

WORD htons(WORD w)	
Description:	Convert a word from host to network byte order.
Arguments:	**Return** - Network byte order data.

	w - Host byte order data.
Example:	`WORD w = 0x1234;` `printf("w=%04x, htons(w)=%04x\n",w,htons(w));`

DWORD htonl(DWORD dw)	
Description:	Convert a double word from host to network byte order.
Arguments:	**Return** - Network byte order data. **dw** - Host byte order data.
Example:	`DWORD dw = 0x12345678L;` `printf("dw=%08lx, htonl(dw)=%08lx\n",dw,ntonl(dw));`

SOCKET SocketCreate(int nType)	
Description:	Create a socket.
Arguments:	**Return** - Socket descriptor, return INVALID_SOCKET if error. **nType** - TCP_SOCKET or UDP_SOCKET(defined on DSOCK.H).

Example:	`SOCKET s = SocketCreate(TCP_SOCKET);`
	`if(s==INVALID_SOCKET)`
	`{`
	`  printf("SocketCreate() error\n");`
	`  return;`
	`}`

BOOL SocketDestory(SOCKET s)	
Description:	Release a socket.
Arguments:	**Return** - TRUE is success and FALSE is error. **s** - Socket descriptor you want to release.
Example:	`/* Release socket */` `SocketDestory(s);`

BOOL SocketClose(SOCKET s)	
Description:	Close a socket.
Arguments:	**Return** - TRUE is success and FALSE is error. **s** - Socket descriptor you want to close.
Example:	`/* Close a socket */` `SocketClose(s);`

void SocketAbort(SOCKET s)	
Description:	Abort a socket.
Arguments:	**Return** - N/A **s** - Socket descriptor you want to abort.

Example:	`/* Abort a socket, like a fast close function */` `SocketAbort(s);`

BOOL SocketBind(SOCKET s,DWORD dwAddr,WORD wPort)	
Description:	Associate a local address with a socket.
Arguments:	**Return** - TRUE is success and FALSE is error. **s** - Socket descriptor you want to bind. **dwAddr** - Local IP address. **wPort** - Local port, DSock will find a free port when wPort=0.

Example:	`/* Assign local port to 80 */`
	`if(SocketBind(s,DWORD(0),80)==FALSE)`
	`  printf("SocketBind() error\n");`

BOOL SocketListen(SOCKET s)	
Description:	Establish a socket to listen for incoming connection.
Arguments:	**Return** - TRUE is success and FALSE is error. **s** - Socket descriptor you want to bind.

Example:	`/* Let socket start to listen */`
	`if(SocketListen(s)==FALSE)`
	`  printf("SocketListen() error\n);`

BOOL SocketConnect(SOCKET s,DWORD dwAddr,WORD wPort)	
Description:	Establish a connection with a peer.
Arguments:	**Return** - TRUE is success and FALSE is error. **s** - Socket descriptor you want to connect to. **dwAddr** - Remote IP. **wPort** - Remote port Remote port.
Example:	`/* Connect to www.dmp.com.tw:80 */` `SocketConnect(s,DSock_Resolve("www.dmp.com.tw"),80);`

int SocketRecv(SOCKET s,BYTE *pby,int nLen)	
Description:	Receive data from a socket.
Arguments:	**Return** - Bytes received, return -1 is error. **s** - Socket descriptor you want to receive from. **pby** - Buffer for received data. **nLen** - The size of the pby buffer.

Example:	`BYTE aby[64];`
	`if(SocketRecv(s,aby,64)<0)`
	`  printf("SocketRecv() error\n");`

int SocketRecv2(SOCKET s,BYTE *pby,int nLen)	
Description:	Receive data from a socket. It will receive data in buffer only.
Arguments:	**Return** - Bytes received, return -1 is error. **s** - Socket descriptor you want to receive from. **pby** - Buffer for received data. **nLen** - The size of the pby buffer.

Example:	`BYTE aby[64];`
	`int  nSize;`
	`nSize = SocketRecv2(s,aby,64);if(nSize<0)`
	`  printf("SocketRecv2() error\n");`
	`else`
	`  printf("%d bytes received\n",nSize);`

int SocketRecvFrom(SOCKET s,DWORD *pdwAddr,WORD *pwPort,BYTE *pby,int nLen)	
Description:	Receive data from UDP socket.
Arguments:	**Return** - Bytes received, return -1 is error. **s** - Socket descriptor you want to receive from. **pdwAddre** - Pointer of DWORD buffer to save remote IP address. **pwPort** - Pointer of WORD buffer to save remote port. **pby** - Buffer for received data. **nLen** - The size of the pby buffer.

Example:	`DWORD dwAddr;`
	`WORD   wPort;`
	`char   c;`
	`SocketRecvFrom(s, &dwAddr, &wPort,&c,1);`
	`printf("From %s:%d, send '%c' back.\n", inet_ntoa(szBuf, dwAddr), wPort, c);`

int SocketSend(SOCKET s,BYTE *pby,int nLen)	
Description:	Send data to a connected socket.
Arguments:	**Return** - Bytes sent. **s** - Socket descriptor you want to send to. **pby** - Buffer to send. **nLen** - The size of the pby buffer.
Example:	`if(SocketSend(s,aby,64)!=64)` `  printf("SocketSend() does not send all data out\n");`

int SocketSend2(SOCKET s,BYTE *pby,int nLen)	
Description:	Send data to a connected socket. It is a non-blocking function.
Arguments:	**Return** - Bytes sent. **s** - Socket descriptor you want to send to.

	pby - Buffer to send. **nLen** - The size of the pby buffer.
Example:	`if(SocketSend(s,aby,64)!=64)` `  printf("SocketSend() does not send all data out\n");`

int SocketSendTo(SOCKET s,DWORD dwAddr,WORD wPort,BYTE *pby,int nLen)	
Description:	Use UDP socket to send data.
Arguments:	**Return** - TRUE is success and FALSE is error. **s** - Socket descriptor you want to send to. **dwAddr** - Remote IP. **wPort** - Remote port. **pby** - Buffer to send. **nLen** - The size of the pby buffer.
Example:	`SocketSendTo(inet_addr("192.168.0.19"),1234,&c,1);`

int SocketDataReady(SOCKET s)	
Description:	Check incoming data of a socket.
Arguments:	**Return** - Size of data that socket has received. **s** - Socket descriptor you want to check.
Example:	`printf("There are %d bytes of socket internal buffer\n",SocketDataReady(s));`

void SocketPutChar(SOCKET s,char c)	
Description:	Write a character to a socket.
Arguments:	**s** - Socket descriptor. **c** - Character want to send.
Example:	`SocketPutChar(s,'x');`

void SocketGetChar(SOCKET s,char *pc)	
Description:	Read a character from a socket.
Arguments:	**s** - Socket descriptor. **pc** - Character pointer for character read.
Example:	`SocketGetChar(s,&c);`

void SocketPutString(SOCKET s,char *szFmt,...)	
Description:	Write a string to a socket.
Arguments:	**s** - Socket descriptor. **szBuf** - The format string.
Example:	`char szBuf[32];` `SocketPutString(s,"My IP is %s\r\n",inet_ntoa(szBuf,DSock_GetHostIp()));`

int SocketGetString(SOCKET s,char *szBuf,int nBufSize)	
Description:	Read a string from a socket.
Arguments:	**Return** - The size of data received, return -1 is error. **s** - Socket descriptor you want to bind. **szBuf** - Buffer for received data. **nBufSize** - The size of the input buffer.

Example:	`char szBuf[80];`
	`if(SocketGetString(s,szBuf,80)>0)`
	`  printf("The string read is '%s'\n",szBuf);`
	`else`
	`  printf(SocketGetString() error\n");`

BOOL SocketIsConnected(SOCKET s)	
Description:	Check connection of a socket.
Arguments:	**Return** - TRUE on connection and FALSE on disconnection. **s** - Socket descriptor you want to bind.

Example:	`while(SocketIsConnected(s)==TRUE)`
	`{`
	`  /* Do socket read/write ...*/`
	`}`

BOOL SocketIsTcpPortUsed(WORD wPort)	
Description:	Is this port number used by DSock TCP sockets ?
Arguments:	**Return** - TRUE: the port is used by DSock TCP sockets; false is not. **wPort** - Port number to detect.

Example:	`for(i=1;i<65535;i++)`
	`  if(SocketIsTcpPortUsed(i))`
	`  printf("port %u is used.\n");`

BOOL SocketIsUdpPortUsed(WORD wPort)	
Description:	Is this port number used by DSock UDP sockets ?
Arguments:	**Return** - TRUE: the port is used by DSock UDP sockets; false is not. **wPort** - Port number to detect.

Example:	`for(i=1;i<65535;i++)`
	`  if(SocketIsUdpPortUsed(i))`
	`  printf("port %u is used.\n");`

BOOL SocketFindFreeTcpPort()	
Description:	Find a free TCP port.
Arguments:	**Return** - Free TCP port number.

Example:	`printf("Find a free TCP port: %u\n",SocketFindFreeTcpPort());`

BOOL SocketFindFreeUdpPort()	
Description:	Find a free UDP port.
Arguments:	**Return** - Free UDP port number.
Example:	`printf("Find a free UDP port: %u\n",SocketFindFreeUdpPort());`

void SocketFlush(SOCKET s)	
Description:	Send pending data.
Arguments:	**s** - Socket descriptor you want to flash.
Example:	`SocketFlash(s);`

void SocketFlushNext(SOCKET s)	
Description:	Cause next transmission to have a flush.
Arguments:	**s** - Socket descriptor you want to have a flash at next transmission.
Example:	`SocketFlashNext(s);`

Index