```c
/*--------------------------------------------------------------------*/
/*                                                                    */
/*                                                                    */
/*  Simple USB-CDC to CAN Loopback Test Program                       */
/*                                                                    */
/*                                                                    */
/*--------------------------------------------------------------------*/


#include <stdio.h>
#include <conio.h>
#include <windows.h>









#define ASCII_ESC           27
#define ASCII_BEL           0x07
#define ASCII_CR            0x0d
#define ASCII_LF            0x0a
//
#define ASCII_CAN2_MARK     '\''




#define TIMEOUT_CONSTANT    10

#define STR_LENGTH          64

#define CUR_LINE_START      10




////////////////////////////////////////////////////////////////////////
//--------------------------
// Function prototypes
//--------------------------

void WaitMs(long DelayMs);
void CMD_WAIT_REPLY(void);
void CMD_WAIT_REPLY_MSG(void);

//-------------------------------------------------------------------------

void get_cursor(int* p_x_pos, int* p_y_pos);
void cursor(int x_pos, int y_pos);
void Erase_Screen(void);
void Fill_Lines(int StartLine,int Lines,unsigned char FillChar);

//-------------------------------------------------------------------------
```

```
void ClearComPort(HANDLE hComm);
BOOL WriteBuffer(HANDLE hComm,char * lpBuf, DWORD dwToWrite);
BOOL ReadBuffer(HANDLE hComm, char* lpBuf ,DWORD dwToRead);

//---------------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////////
//---------------------------
// Function implementations
//---------------------------

static CRITICAL_SECTION g_cs; // Critical section

HANDLE  WinStdOut = NULL;   // Console Handle
```

```
//---------------------------------------------------------------------------
void WaitMs(long DelayMs)
{//wait for a moment in ms

  if(DelayMs>0)
  {
    HANDLE hWaiter = CreateEvent(NULL, TRUE, FALSE, NULL);
    ResetEvent(hWaiter);
    WaitForSingleObject(hWaiter,DelayMs);
    CloseHandle(hWaiter);
    hWaiter=NULL;
  }//if(DelayMs>0)

}//WaitMs()
```

```
WORD gWait_reply_ms=10;         // Delay to wait the reply (ms)
/*
void CMD_WAIT_REPLY(void)
{
  WaitMs(gWait_reply_ms);
}//CMD_WAIT_REPLY()
*/
#define CMD_WAIT_REPLY() { WaitMs(gWait_reply_ms); }
```

```
WORD gWait_reply_msg_ms=20;       // Delay to wait the loopback message
reply (ms)
/*
void CMD_WAIT_REPLY_MSG(void)
{
  WaitMs(gWait_reply_msg_ms);
}//CMD_WAIT_REPLY_MSG()
*/
#define CMD_WAIT_REPLY_MSG() { WaitMs(gWait_reply_msg_ms); }
//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------

void get_cursor(int* p_x_pos, int* p_y_pos)
{
    // Get the cursor position

    CONSOLE_SCREEN_BUFFER_INFO consoleInfo;
    GetConsoleScreenBufferInfo(WinStdOut, &consoleInfo);
    *p_x_pos = consoleInfo.dwCursorPosition.X;
    *p_y_pos = consoleInfo.dwCursorPosition.Y;

}// get_cursor()

//---------------------------------------------------------------------------

void cursor(int x_pos, int y_pos)
{
    // Moves the cursor to the position specified by x_pos,y_pos

    COORD Newcurpos;
    Newcurpos.X = x_pos;
    Newcurpos.Y = y_pos;
    SetConsoleCursorPosition(WinStdOut, Newcurpos);

}// cursor()

//---------------------------------------------------------------------------

void Erase_Screen(void)
{
    // Fills the section of the screen that prints received data with
    spaces.

    int  c;
    char erasestr[81];

    for (c = 0; c < 81; c++)
        erasestr[c] = ' ';  // space

    erasestr[80] = '\0';

    cursor(0, CUR_LINE_START);
```

```
    for (c = CUR_LINE_START; c < 24; c++)
        printf(erasestr);

}// Erase_Screen()

//---------------------------------------------------------------------------

void Fill_Lines(int StartLine,int Lines,unsigned char FillChar)
{
    // Fills the section of the screen that prints received data with
    specific character.

  if(Lines)
  {//assume the screen is 80x23
    int  c;
    char erasestr[81];

    for (c = 0; c < 81; c++)
      erasestr[c] = FillChar;

    erasestr[80] = '\0';

    cursor(0, StartLine);

    for (c = StartLine; c < StartLine+Lines; c++)
      printf(erasestr);
  }
}// Fill_Lines()

//---------------------------------------------------------------------------









//---------------------------------------------------------------------------

void ClearComPort(HANDLE hComm)
{
  DWORD error;

  PurgeComm(hComm,PURGE_TXABORT|PURGE_RXABORT|PURGE_TXCLEAR|PURGE_RXCLEAR);
  ClearCommError(hComm, &error, NULL);
}//ClearComPort()


BOOL WriteBuffer(HANDLE hComm,char * lpBuf, DWORD dwToWrite)
{
  OVERLAPPED osWrite = {0};
  DWORD dwWritten;
  DWORD dwRes;
  BOOL fRes;
  // Create this write operation's OVERLAPPED structure hEvent.
  osWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
  if (osWrite.hEvent == NULL)
  // Error creating overlapped event handle.
  return FALSE;

  // Issue write
```

```
  if (!WriteFile(hComm, lpBuf, dwToWrite, &dwWritten, &osWrite))
  {
    if (GetLastError() != ERROR_IO_PENDING)
    {
      // WriteFile failed, but it isn't delayed. Report error.
      fRes = FALSE;
    }
    else
    {
      // Write is pending.
      dwRes = WaitForSingleObject(osWrite.hEvent, INFINITE);

      switch(dwRes)
      {
      // Overlapped event has been signaled.
      case WAIT_OBJECT_0:
      if (!GetOverlappedResult(hComm, &osWrite, &dwWritten, FALSE))
        fRes = FALSE;
      else
      {
        if (dwWritten != dwToWrite)
        {
          // The write operation timed out.
          // Decide if you want to abort or retry.
          fRes = FALSE;
        }
        else
        // Write operation completed successfully.
        fRes = TRUE;
      }
      break;
      deIsFault:
        // An error has occurred in WaitForSingleObject. This usually
        // indicates a problem with the overlapped event handle.
        fRes = FALSE;
      break;
      }
    }
  }
  else
  {
    // WriteFile completed immediately.
    if (dwWritten != dwToWrite)
    {
      // The write operation timed out.
      // Decide if you want to abort or retry.
      fRes = FALSE;
    }
    else
      fRes = TRUE;
  }

  CloseHandle(osWrite.hEvent);
  return fRes;
}//WriteBuffer()


BOOL ReadBuffer(HANDLE hComm, char* lpBuf ,DWORD dwToRead)
{
  BOOL fRes;
  COMSTAT comstat;
  BOOL  bRead = TRUE;
  BOOL  bResult = TRUE;
  DWORD dwError = 0;
  DWORD dwRead = 0;
  OVERLAPPED osRead = {0};
```

```
osRead.hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
if (osRead.hEvent == NULL)
// Error creating overlapped event handle.
return FALSE;


fRes = FALSE;
for(;;)
{
  EnterCriticalSection(&g_cs);
  bResult = ClearCommError(hComm, &dwError, &comstat);
  LeaveCriticalSection(&g_cs);

  if (comstat.cbInQue == 0)
  {
    // break out when all bytes have been read
    break;
  }

  if(bRead)
  {

    bResult=ReadFile(hComm,     // Handle
         lpBuf,                 // Incoming data
         dwToRead,              // Number of bytes to read
         &dwRead,               // Number of bytes read
         &osRead);

    if(!bResult)
    {

      DWORD dwError;
      switch (dwError = GetLastError())
      {
        case ERROR_IO_PENDING:
        {
          bRead = FALSE;
        }
        break;
        deIsFault:
          break;
      }//switch()
    }
    else
    {
      bRead=TRUE;
    }
  }
  else
  {

    bRead=TRUE;
    bResult = GetOverlappedResult(hComm, &osRead, &dwRead, TRUE);
    if (bResult)
    {
    //...
    }
  }
}//for()

if (dwRead != dwToRead)
{
  // The read operation timed out.
  // Decide if you want to abort or retry.
  fRes = FALSE;
}
```

```
  else
    fRes = TRUE;


  CloseHandle(osRead.hEvent);

  return fRes;
}//ReadBuffer()
//---------------------------------------------------------------------------

////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////

int main(int argv,char** argc)
{
  char version[6]={'1','.','0','.','3','\0'};// Software version


  int  cur_str      = 0;            // Current test string to send
  int  key_pressed  = 0;
  int x_pos = 0, y_pos = 8;
  CONSOLE_SCREEN_BUFFER_INFO consoleInfo; // Used to get cursor position.


  HANDLE       comport       = NULL;      // Handle for COM port
  DCB          comSettings;               // Contains various port settings
  COMMTIMEOUTS tempComTimeouts;           // Our temporary time-outs for
  COM1
  COMMTIMEOUTS savedComTimeouts;          // Stores the original time-outs
  DWORD        bytes_written = 0;         // Number of bytes written to port
  DWORD        bytes_read    = 0;         // Number of bytes read from port

  BOOL bStatus;                       // Return status of functions
  char devpath_str[STR_LENGTH];       // Device filepath to open
  char teststring[2][STR_LENGTH];     // Test patterns

  char* pstrArgc=0;
  DWORD test_times=1;                 // Test times
  int ComPortNo=0;                    // COM port No.
  int test_interval=0;                // Test interval (ms)
  int loopback_test_retry=3;          // Retry times of one loopback test
```

```
cycle


InitializeCriticalSection(&g_cs);

WinStdOut = GetStdHandle(STD_OUTPUT_HANDLE);


// Test Pattern
strcpy(teststring[0], "T01abcdef80102030405060708\r");
strcpy(teststring[1], "T02abcdef81122334455667788\r");



printf("############################################################\n");
printf(" USB Virtual Com Port to CAN Loopback Test Program.\n");
printf(" iBASE Technology Inc.\t\t\t Ver:%s\n",version);
printf("############################################################\n");
printf("\n");



if(argv>1)
{
  pstrArgc=argc[1];

  int iVal=0;
  iVal = strtoul (pstrArgc,NULL,10);
  if(iVal>=0)
  {
    ComPortNo=iVal;
    printf("ComPortNo:%d, ",ComPortNo);
  }
}
if(argv>2)
{
  pstrArgc=argc[2];

  unsigned long iVal=0;
  iVal = strtoul (pstrArgc,NULL,10);
  if(iVal>=0)
  {
    test_times=iVal;
    printf("test_times:%d, ",test_times);
  }
}
if(argv>3)
{
  pstrArgc=argc[3];

  int iVal=0;
  iVal = strtoul (pstrArgc,NULL,10);
  if(iVal>=0)
  {
    test_interval=iVal;
    printf("test_interval:%d, ",test_interval);
  }
}
if(argv>4)
{
  printf("\n");

  pstrArgc=argc[4];

  int iVal=0;
```

```
  iVal = strtoul (pstrArgc,NULL,10);
  if(iVal>=0)
  {
    gWait_reply_ms=iVal;
    printf("wait_reply_ms:%d, ",gWait_reply_ms);
  }
}
if(argv>5)
{
  pstrArgc=argc[5];

  int iVal=0;
  iVal = strtoul (pstrArgc,NULL,10);
  if(iVal>=0)
  {
    gWait_reply_msg_ms=iVal;
    printf("wait_reply_msg_ms:%d, ",gWait_reply_msg_ms);
  }
}
if(argv>6)
{
  pstrArgc=argc[6];

  int iVal=0;
  iVal = strtoul (pstrArgc,NULL,10);
  if(iVal>=0)
  {
    loopback_test_retry=iVal;
    printf("loopback_test_retry:%d, ",loopback_test_retry);
  }
}

//...
printf("\n");
```

```
char strPortNo[4];
if(ComPortNo==0)
{
  printf("input COM port number:\n");


  INT PortNoDigit=0;
  BOOL IsContinue=1;

  key_pressed  = 0;
  while(IsContinue)
  {
    while (kbhit())
    {
      key_pressed = getch();
      if(key_pressed==ASCII_ESC)
      {
        return 1;
      }

      if(key_pressed>='0' && key_pressed<='9')
      {
        strPortNo[PortNoDigit]=key_pressed;
        printf("%d",key_pressed-'0');
```

```
        PortNoDigit++;
      }
      else if(key_pressed == 0x0d)
      {
        strPortNo[PortNoDigit]='\0';

        int iVal=0;
        iVal = strtoul (strPortNo,NULL,10);
        if(iVal>=0)
        {
          ComPortNo=iVal;
        }

        IsContinue=0;
        break;
      }
      else  if(key_pressed!=0)
      {
        printf("invalid COM port number,please try again.\n");
        PortNoDigit=0;
      }
    }
  }
  //
  {
    int pos_x,pos_y;
    get_cursor(&pos_x,&pos_y);
    Fill_Lines((pos_y-1),2,' ');
    cursor(0,(pos_y-1));
  }
}


if(ComPortNo)
{
  printf("try to open COM%d...\t",ComPortNo);
  sprintf(devpath_str,"\\\\.\\COM%d",ComPortNo);
  sprintf(strPortNo,"%d",ComPortNo);
}



if(devpath_str)
{
  // Open COM port
  if ((comport =
          CreateFile(devpath_str,       //"\\\\.\\COM1"
          GENERIC_READ | GENERIC_WRITE, // for reading and writing
          0,                            // exclusive access
          NULL,                         // no security attributes
          OPEN_EXISTING,
          FILE_FLAG_OVERLAPPED,
          NULL)) == INVALID_HANDLE_VALUE)
  {
    printf("Unable to open COM%s.\n\n",strPortNo);
    printf("Press any key to exit.");
    getch();
    return(1);
  }


  printf("COM%s opened.\n\n",strPortNo);



  // Save time-out parameters for COM1
  bStatus = GetCommTimeouts(comport,&savedComTimeouts);
```

```
    if (bStatus == 0)
    {
      printf("GetCommTimeouts failure.\n\n");
      printf("Press any key to exit.");
      getch();
      return(1);
    }


    // Set time-outs.
    tempComTimeouts.ReadIntervalTimeout         = MAXDWORD;
    tempComTimeouts.ReadTotalTimeoutMultiplier  = MAXDWORD;
    tempComTimeouts.ReadTotalTimeoutConstant    = TIMEOUT_CONSTANT;
    tempComTimeouts.WriteTotalTimeoutMultiplier = TIMEOUT_CONSTANT;
    tempComTimeouts.WriteTotalTimeoutConstant   = TIMEOUT_CONSTANT;

    bStatus = SetCommTimeouts(comport,&tempComTimeouts);
    if (bStatus == 0)
    {
      printf("SetCommTimeouts failure.\n\n");
      printf("Press any key to exit.");
      getch();
      return(1);
    }


    // Set Port parameters.
    // We make a call to GetCommState() first in order to fill
    // the comSettings structure with all the necessary values.
    // Then we change the ones we want and call SetCommState().

    bStatus = GetCommState(comport, &comSettings);
    if (bStatus == 0)
    {
      printf("GetCommState failure.\n\n");
      printf("Press any key to exit.");
      getch();
      return(1);
    }
    comSettings.BaudRate = CBR_115200;
    comSettings.StopBits = ONESTOPBIT;
    comSettings.ByteSize = 8;
    comSettings.Parity   = NOPARITY;
    comSettings.fParity  = FALSE;
    bStatus = SetCommState(comport, &comSettings);
    if (bStatus == 0)
    {
      printf("GetCommTimeouts failure.\n\n");
      printf("Press any key to exit.");
      getch();
      return(1);
    }
  }//if(devpath_str)


  ClearComPort(comport);


  Erase_Screen();
  cursor(0,CUR_LINE_START);


//=============================================================================
// ASCII COMMAND OPERATION
//=============================================================================

  char strCmd[256];
```

```
  char strRcv[256];
  int TxSz=0;
  int RxSz=0;
  int Interval=0;
  BOOL IsCanPortOpen[2];
  BYTE ucCanPortStatus[2];



  IsCanPortOpen[0]=0;
  IsCanPortOpen[1]=0;
  ucCanPortStatus[0]=0;
  ucCanPortStatus[1]=0;


//------------------------
// Get Firmware Version
//------------------------

  sprintf(strCmd,"V\r");
  TxSz=strlen(strCmd);
  if(TxSz)
  {
    RxSz=6;
    int result=WriteBuffer(comport,strCmd,TxSz);
    if(result)
    {
      CMD_WAIT_REPLY();
      result=ReadBuffer(comport,strRcv,RxSz);
      if(result)
      {
        if(strRcv[0]=='V' && strRcv[5]==ASCII_CR)
        {
          printf("firmware
          version:V%c.%c.%c.%c\r\n",strRcv[1],strRcv[2],strRcv[3],strRcv[4])
          ;
        }
      }
    }

    if(result<=0)
    {
      printf("Get firmware version fail.\n\r");
      goto _Test_End;
    }

  }



//------------------------
// Get CAN port status
//------------------------

  sprintf(strCmd,"F\r");
  TxSz=strlen(strCmd);
  if(TxSz)
  {
    RxSz=4;
    int result=WriteBuffer(comport,strCmd,TxSz);
    if(result)
    {
      CMD_WAIT_REPLY();
      result=ReadBuffer(comport,strRcv,RxSz);
      if(result)
      {
```

```
        if(strRcv[0]=='F' && strRcv[3]==ASCII_CR)
        {
          char strStatus[8]={'\0'};
          sprintf(strStatus,"%c%c",strRcv[1],strRcv[2]);
          int iVal=0;
          iVal = strtoul (strStatus,NULL,16);
          if(iVal>=0)
          {
            ucCanPortStatus[0]=(iVal&0xFF);
          }

          printf("Get CAN1 status:0x%c%c\r\n",strRcv[1],strRcv[2]);
        }
      }
    }

    if(result<=0)
    {
      printf("Get CAN1 status fail.\n\r");
      goto _Test_End;
    }
  }



  sprintf(strCmd,"%cF\r",ASCII_CAN2_MARK);
  TxSz=strlen(strCmd);
  if(TxSz)
  {
    RxSz=5;
    int result=WriteBuffer(comport,strCmd,TxSz);
    if(result)
    {
      CMD_WAIT_REPLY();
      result=ReadBuffer(comport,strRcv,RxSz);
      if(result)
      {
        if(strRcv[0]==ASCII_CAN2_MARK && strRcv[1]=='F' &&
        strRcv[4]==ASCII_CR)
        {
          char strStatus[8]={'\0'};
          sprintf(strStatus,"%c%c",strRcv[2],strRcv[3]);
          int iVal=0;
          iVal = strtoul (strStatus,NULL,16);
          if(iVal>=0)
          {
            ucCanPortStatus[1]=(iVal&0xFF);
          }
          printf("Get CAN2 status:0x%c%c\r\n",strRcv[2],strRcv[3]);
        }
      }
    }

    if(result<=0)
    {
      printf("Get CAN2 status fail.\n\r");
      goto _Test_End;
    }
  }



//-------------------------------------------------
// Close CAN port if it has been opened
//-------------------------------------------------
```

```
if(ucCanPortStatus[0] & 0x80)
{
  IsCanPortOpen[0]=1;

  sprintf(strCmd,"C\r");
  TxSz=strlen(strCmd);
  if(TxSz)
  {
    RxSz=1;
    int result=WriteBuffer(comport,strCmd,TxSz);
    if(result)
    {
      CMD_WAIT_REPLY();
      result=ReadBuffer(comport,strRcv,RxSz);
      if(result)
      {
        if(strRcv[0]==ASCII_CR)
        {
          IsCanPortOpen[0]=0;
          printf("CAN1 closed.\r\n");
        }
      }
    }

    if(result<=0)
    {
      printf("CAN1 close fail.\n\r");
    }
  }
}
```

```
if(ucCanPortStatus[1] & 0x80)
{
  IsCanPortOpen[1]=1;

  sprintf(strCmd,"%cC\r",ASCII_CAN2_MARK);
  TxSz=strlen(strCmd);
  if(TxSz)
  {
    RxSz=2;
    int result=WriteBuffer(comport,strCmd,TxSz);
    if(result)
    {
      CMD_WAIT_REPLY();
      result=ReadBuffer(comport,strRcv,RxSz);
      if(result)
      {
        if(strRcv[0]==ASCII_CAN2_MARK && strRcv[1]==ASCII_CR)
        {
          IsCanPortOpen[1]=0;
          printf("CAN2 closed.\r\n");
        }
      }
    }

    if(result<=0)
    {
      printf("CAN2 close fail.\n\r");
    }
  }
}
```

```
//------------------------
// Set CAN port speed
//------------------------

  sprintf(strCmd,"S6\r");
  TxSz=strlen(strCmd);
  if(TxSz)
  {
    RxSz=3;
    int result=WriteBuffer(comport,strCmd,TxSz);
    if(result)
    {
      CMD_WAIT_REPLY();
      result=ReadBuffer(comport,strRcv,RxSz);
      if(result)
      {
        if(strRcv[0]=='S' && strRcv[1]=='6' && strRcv[2]==ASCII_CR)
        {
          printf("Set CAN1 in 500Kbps.\r\n");
        }
      }
    }

    if(result<=0)
    {
      printf("Set CAN1 speed fail.\n\r");
      goto _Test_End;
    }
  }



  sprintf(strCmd,"%cS6\r",ASCII_CAN2_MARK);
  TxSz=strlen(strCmd);
  if(TxSz)
  {
    RxSz=4;
    int result=WriteBuffer(comport,strCmd,TxSz);
    if(result)
    {
      CMD_WAIT_REPLY();
      result=ReadBuffer(comport,strRcv,RxSz);
      if(result)
      {
        if(strRcv[0]==ASCII_CAN2_MARK && strRcv[1]=='S' && strRcv[2]=='6' &&
        strRcv[3]==ASCII_CR)
        {
          printf("Set CAN2 in 500Kbps.\r\n");
        }
      }
    }

    if(result<=0)
    {
      printf("Set CAN2 speed fail.\n\r");
      goto _Test_End;
    }
  }



//---------------------------------------
// Setup masker and filter of CAN port
//---------------------------------------

  sprintf(strCmd,"m00000000\r");
```

```
TxSz=strlen(strCmd);
if(TxSz)
{
  RxSz=1;
  int result=WriteBuffer(comport,strCmd,TxSz);
  if(result)
  {
    CMD_WAIT_REPLY();
    result=ReadBuffer(comport,strRcv,RxSz);
    if(result)
    {
      if(strRcv[0]==ASCII_CR)
      {
        printf("Set CAN1 masker with 0x00000000\r\n");
      }
    }
  }

  if(result<=0)
  {
    printf("Set CAN1 masker fail.\n\r");
    goto _Test_End;
  }
}



sprintf(strCmd,"%cm00000000\r",ASCII_CAN2_MARK);
TxSz=strlen(strCmd);
if(TxSz)
{
  RxSz=2;
  int result=WriteBuffer(comport,strCmd,TxSz);
  if(result)
  {
    CMD_WAIT_REPLY();
    result=ReadBuffer(comport,strRcv,RxSz);
    if(result)
    {
      if(strRcv[0]==ASCII_CAN2_MARK && strRcv[1]==ASCII_CR)
      {
        printf("Set CAN2 masker with 0x00000000\r\n");
      }
    }
  }

  if(result<=0)
  {
    printf("Set CAN2 masker fail.\n\r");
    goto _Test_End;
  }
}




sprintf(strCmd,"M00000000\r");
TxSz=strlen(strCmd);
if(TxSz)
{
  RxSz=1;
  int result=WriteBuffer(comport,strCmd,TxSz);
  if(result)
  {
    CMD_WAIT_REPLY();
    result=ReadBuffer(comport,strRcv,RxSz);
```

```
      if(result)
      {
        if(strRcv[0]==ASCII_CR)
        {
          printf("Set CAN1 filter with 0x00000000\r\n");
        }
      }
    }

    if(result<=0)
    {
      printf("Set CAN1 filter fail.\n\r");

      goto _Test_End;
    }
  }



  sprintf(strCmd,"%cM00000000\r",ASCII_CAN2_MARK);
  TxSz=strlen(strCmd);
  if(TxSz)
  {
    RxSz=2;
    int result=WriteBuffer(comport,strCmd,TxSz);
    if(result)
    {
      CMD_WAIT_REPLY();
      result=ReadBuffer(comport,strRcv,RxSz);
      if(result)
      {
        if(strRcv[0]==ASCII_CAN2_MARK && strRcv[1]==ASCII_CR)
        {
          printf("Set CAN2 filter with 0x00000000\r\n");
        }
      }
    }

    if(result<=0)
    {
      printf("Set CAN2 filter fail.\n\r");
      goto _Test_End;
    }
  }




//-------------------------
// Open CAN port
//-------------------------

  sprintf(strCmd,"O\r");
  TxSz=strlen(strCmd);
  if(TxSz)
  {
    RxSz=1;
    int result=WriteBuffer(comport,strCmd,TxSz);
    if(result)
    {
      CMD_WAIT_REPLY();
      result=ReadBuffer(comport,strRcv,RxSz);
      if(result)
      {
        if(strRcv[0]==ASCII_CR)
```

```
      {
        IsCanPortOpen[0]=1;
        printf("CAN1 opened.\r\n");
      }
    }
  }

  if(result<=0)
  {
    printf("CAN1 open fail.\n\r");
    goto _Test_End;
  }
}



sprintf(strCmd,"%cO\r",ASCII_CAN2_MARK);
TxSz=strlen(strCmd);
if(TxSz)
{
  RxSz=2;
  int result=WriteBuffer(comport,strCmd,TxSz);
  if(result)
  {
    CMD_WAIT_REPLY();
    result=ReadBuffer(comport,strRcv,RxSz);
    if(result)
    {
      if(strRcv[0]==ASCII_CAN2_MARK && strRcv[1]==ASCII_CR)
      {
        IsCanPortOpen[1]=1;
        printf("CAN2 opened.\r\n");
      }
    }
  }

  if(result<=0)
  {
    printf("CAN2 open fail.\n\r");
    goto _Test_End;
  }
}



//clear the screen and draw a line,then set cursor from the specific line
  Erase_Screen();
Fill_Lines((CUR_LINE_START-1),1,'_');
cursor(0,CUR_LINE_START);



DWORD T1=0;               //record the ticks when test begin
DWORD T2=0;               //record the ticks when test end
FLOAT Tt=0;               //the elapsed ticks of test totally
DWORD TestCnt=0;          //counter of test times
DWORD ErrCnt=0;           //counter of test error
BOOL IsFault=false;       //the last test status



//
// begin test...
//
```

```
T1=GetTickCount();
for(TestCnt=0;TestCnt<test_times;TestCnt++)
{
  if(IsFault)
  {
    Fill_Lines(CUR_LINE_START,6,' ');//clear 6 lines
  }


  //
  // display result of the last test cycle on console screen
  //
  cursor(0,CUR_LINE_START);
  printf("remains %d times, Error Counter:%d,
  ",(test_times-(TestCnt+1)),ErrCnt);


  Tt=T2;
  if(T2>60000)
  {//min
    Tt/=60000;
    printf("elapsed time:%.1fmin ",Tt);
  }
  else if(T2>1000)
  {//sec
    Tt/=1000;
    printf("elapsed time:%.1fsec ",Tt);
  }
  else
  {//ms
    printf("elapsed time:%dms ",Tt);
  }
  printf("\r\n");



  //-------------------------------------------------
  // ping-pong message pattern between 2 CAN ports
  //-------------------------------------------------
  IsFault=false;

  if(IsCanPortOpen[0])
  {
    int retry=loopback_test_retry;
    while(retry--)
    {
      ClearComPort(comport);

      //-----------------------------------------------------------------------
      //send out 27bytes from CAN1: T01abcdef80102030405060708[CR]
      sprintf(strCmd,"%s",teststring[0]);

      TxSz=strlen(strCmd);
      if(TxSz)
      {

        strRcv[0]='\0';//clear
        //it should receive 30bytes: Z[CR]+'T01abcdef80102030405060708[CR]
        //that is the command status returned,
        //and followed by replied messsage text from CAN2 received.
        int result=WriteBuffer(comport,strCmd,TxSz);
        if(result)
        {
          CMD_WAIT_REPLY_MSG();

          RxSz=2+TxSz+1;
          result=ReadBuffer(comport,strRcv,RxSz);
```

```
        if(result)
        {
          if(strRcv[0]=='Z' && strRcv[1]==ASCII_CR)
          {
            cursor(0,CUR_LINE_START+1);
            printf("CAN1 transmit done.\r\n");


            BYTE* pReply=(BYTE*)&strRcv[2];
            if(pReply)
            {
              RxSz=TxSz+1;
              if(pReply[0]==ASCII_CAN2_MARK && pReply[1]=='T' &&
              pReply[RxSz-1]==ASCII_CR)
              {
                char strMsg[64];
                pReply[RxSz-1]='\0';
                sprintf(strMsg,"%s",(char*)&pReply[1]);
                cursor(0,CUR_LINE_START+2);
                printf("received feedback from CAN2:
                %s[CR]\r\n",strMsg);

                break;
              }
            }
          }
        }
      }

      if(result<=0)
      {
        Fill_Lines(CUR_LINE_START+1,2,' ');//clear 2 lines
        cursor(0,CUR_LINE_START+1);
        printf("CAN1 transmit fail\r\n");
      }
    }
    //-----------------------------------------------------------------
  }
  if(retry<0)
  {
    cursor(0,CUR_LINE_START+2);
    printf("can not receive loopback from CAN2.\r\n",strRcv);
    IsFault=true;
  }
}



if(IsCanPortOpen[1])
{
  int retry=loopback_test_retry;
  while(retry--)
  {
    ClearComPort(comport);

    //-----------------------------------------------------------------
    //send out 28bytes from CAN2: 'T02abcdef81122334455667788[CR]
    sprintf(strCmd,"%c%s",ASCII_CAN2_MARK,teststring[1]);

    TxSz=strlen(strCmd);
    if(TxSz)
    {
      strRcv[0]='\0';//clear
      //it should receive 30bytes: 'Z[CR]+T02abcdef81122334455667788[CR]
      //that is the command status returned,
      //and followed by replied messsage text from CAN1 received.
      int result=WriteBuffer(comport,strCmd,TxSz);
```

```
        if(result)
        {
          CMD_WAIT_REPLY_MSG();

          RxSz=3+TxSz-1;
          result=ReadBuffer(comport,strRcv,RxSz);
          if(result)
          {

            if(strRcv[0]==ASCII_CAN2_MARK && strRcv[1]=='Z' &&
            strRcv[2]==ASCII_CR)
            {
              cursor(0,CUR_LINE_START+3);
              printf("CAN2 transmit done.\r\n");

              BYTE* pReply=(BYTE*)&strRcv[3];
              if(pReply)
              {
                RxSz=TxSz-1;
                if(pReply[0]=='T' && pReply[RxSz-1]==ASCII_CR)
                {
                  char strMsg[64];
                  pReply[RxSz-1]='\0';
                  sprintf(strMsg,"%s",(char*)&pReply[0]);
                  cursor(0,CUR_LINE_START+4);
                  printf("received feedback from CAN1:
                  %s[CR]\r\n",strMsg);

                  break;
                }
              }
            }
          }
        }

        if(result<=0)
        {
          Fill_Lines(CUR_LINE_START+3,2,' ');//clear 2 lines
          cursor(0,CUR_LINE_START+3);
          printf("CAN2 transmit fail\r\n");
        }
      }
      //-----------------------------------------------------------------
    }
    if(retry<0)
    {
      cursor(0,CUR_LINE_START+4);
      printf("can not receive loopback from CAN1.\r\n",strRcv);
      IsFault=true;
    }
  }

  if(IsFault) ErrCnt++;



  Fill_Lines((CUR_LINE_START+5),1,'_');

  //delay for message display on console screen
  Sleep(test_interval);


//
// one cyclee test end
//
T2=(GetTickCount()-T1);
}//for()
```

```
  //show the final test result
  Erase_Screen();
  cursor(0,CUR_LINE_START);



  //
  // calculate elapsed test time, and display the test result on console
  screen
  //
  Tt=T2;
  if(T2>60000)
  {//min
    Tt/=60000;
    printf("\n\ntest finished.(Error:%d \/ %dtimes)
    (%.1fmin)\r\n",ErrCnt,TestCnt,Tt);
  }
  else if(T2>1000)
  {//sec
    Tt/=1000;
    printf("\n\ntest finished.(Error:%d \/ %dtimes)
    (%.1fsec)\r\n",ErrCnt,TestCnt,Tt);
  }
  else
  {//ms
    printf("\n\ntest finished.(Error:%d \/ %dtimes)
    (%dms)\r\n",ErrCnt,TestCnt,Tt);
  }









_Test_End:


  ClearComPort(comport);



  //------------------------
  // Close CAN port
  //------------------------

  if(IsCanPortOpen[0])
  {
    sprintf(strCmd,"C\r");
    Sleep(1);
    TxSz=strlen(strCmd);
    if(TxSz)
    {
      RxSz=1;
      int result=WriteBuffer(comport,strCmd,TxSz);
      if(result)
      {
        CMD_WAIT_REPLY();
        result=ReadBuffer(comport,strRcv,RxSz);
        if(result)
        {
```

```
        if(strRcv[0]==ASCII_CR)
        {
          IsCanPortOpen[0]=0;
          printf("CAN1 closed.\r\n");
        }
      }
    }

    if(result<=0)
    {
      printf("CAN1 close fail.\n\r");
    }
  }
}

if(IsCanPortOpen[1])
{
  sprintf(strCmd,"%cC\r",ASCII_CAN2_MARK);
  Sleep(1);
  TxSz=strlen(strCmd);
  if(TxSz)
  {
    RxSz=2;
    int result=WriteBuffer(comport,strCmd,TxSz);
    if(result)
    {
      CMD_WAIT_REPLY();
      result=ReadBuffer(comport,strRcv,RxSz);
      if(result)
      {
        if(strRcv[0]==ASCII_CAN2_MARK && strRcv[1]==ASCII_CR)
        {
          IsCanPortOpen[1]=0;
          printf("CAN2 closed.\r\n");
        }
      }
    }

    if(result<=0)
    {
      printf("CAN2 close fail.\n\r");
    }
  }
}




printf("\n\nTest terminated.\r\n");
printf("\n\nPress ESC to exit.\r\n");
while(key_pressed != ASCII_ESC)
{
    if (kbhit())
        key_pressed = getch();
}



// Restore time-out parameters
SetCommTimeouts(comport,&savedComTimeouts);
CloseHandle(comport);


DeleteCriticalSection(&g_cs);
```

```
  return(0);
}// main()
////////////////////////////////////////////////////////////////////////////
```