10 장 윈도 95 용 통신 에뮬레이터

지금까지 내용을 잘 이해했다면 윈도 95 용 통신 에뮬레이터를 작성하는데 필요한 기본적인 지식은 어느 정도 갖추었다고 할 수 있겠다. 자, 이제 진짜 통신 에뮬레이터를 만들어 보자.

통신 에뮬레이터의 기본 구조

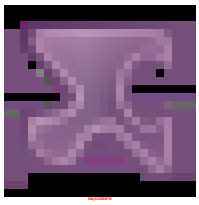
통신 프로그래밍에 대한 어떤 막연한 두려움같은 것들을 갖고 있다면, 이번 장에서 모두 깨버리고 넘어가도록 하자. 통신 프로그래밍이라는 것이 너무 어렵고 까다로운 작업이라는 선입견은 여기서 던져 버리고 가벼운 마음으로 시작할 수 있도록 먼저, 아주 간단한 통신 프로그램을 작성해 보도록 하자.

우선, 통신 프로그래밍, 그러니까 통신 에뮬레이터의 기본 구조에 대해 먼저 짚고 넘어가자. 통신에뮬레이터의 기본 기능은 사실 아주 단순하다. 전화선을 타고 모뎀으로 들어온 문자를 화면에 출력하는 것과 자판에서 입력한 문자를 다시 모뎀을 통해 전화선으로 보내 주는 작업이다. 여기서, 모뎀은 바로, 전화선을 통해 들어오는 아날로그 데이터를 PC 쪽에서 읽어낼 수 있도록 디지틀 데이터로 변환하는 역할과 그 반대 역할 즉, PC 쪽에서 입력한 디지틀 데이터를 전화선의 소리형태로 변환시켜주는 역할을 하게 된다. 통신하는 도중에 전화기를 들어 보면, '삐삐-지지직-'하는 소리를 들을 수 있는데, 이 소리가 바로 전화선을 통해 전달되는 아날로그 데이터이다. 이 소리 데이터를 다시 바이트 단위로 변환해서 모뎀의 수신 큐에 넣어 두게 되는 것이다. 아날로그에서 디지틀로, 디지틀에서 아날로그로 변환하는 것을 변조(modulation)-복조(demodulation)라고 하는데 이 변복조(MOd-DEMoulation)의 약자가 바로 MODEM 이다.

이처럼 통신 에뮬레이터의 기본 기능 중에서 대부분은 모뎀 하드웨어가 직접 해 주기 때문에 실제 프로그래머는 자판을 통해 입력된 문자와 모뎀을 통해 수신되는 문자에 대한 처리만 해주면 되는 것이다.

모뎀에 대해서는 모뎀의 명령어들에 대한 의미와 그 결과값에 대해서만 잘 알고 있으면 그만이다.

따라서 프로그래머가 신경써야 할 부분은 바로, 직렬통신을 담당하고 있는 범용 비동기 송수신기(UART : Universal Asynchronous Receiver Transmitter) 부분이다. 이 범용 비동기 송수신기를 흔히들 그냥 '통신 포트'라고도 부르는데, 개념적으로는 다음 그림과 같이 PC와 모뎀 사이에 있는 '입출력 큐'정도로 생각하면 우선은 이해하기 쉬울 것이다. 모뎀이 수신한 데이터는 직렬 통신 포트, 즉 UART 의 수신 버퍼에 들어가며, PC 쪽에서 직렬 통신 포트에 쓴 데이터는 UART 의 송신 버퍼에 들어가서 모뎀을 통해 연결된 원격지로 전송된다.



UART 는 모뎀을 통해 들어오는 비트 스트림을 바이트 형태로 바꿔주고, PC 쪽에서 보내는 바이트 형태의 데이터를 모뎀이 전송할 수 있도록 비트 스트림 형태로 바꿔주는 역할을 하는 장치이다. 이 UART 의 레지스터를 이용해서 모뎀을 제어하거나 할 수 있다. 따라서 MS-DOS 에서 직렬 통신 장치를 이용하는 통신 에뮬레이터같은 소프트웨어를 만들려면, UART 의 레지스터값을 읽고 쓰는 부분과 수신 버퍼를 감시하는 인터럽트 서비스 루틴을 작성해야 했다.

MS-DOS 에서의 통신 프로그램 구조

다음은 바이오스 인터럽트 14 번인 직렬통신 기능을 이용하는 예이다.

이 코드는

- Microsoft C for MS-DOS, versions 5.1, 6.0, 6.0a, and 6.0ax
- Microsoft C/C++ for MS-DOS, version 7.0
- Microsoft Visual C++ for Windows, versions 1.0 and 1.5

에서 수행된다.

```
#include <stdio.h>
#include <conio.h>
#include <bios.h>

void main(void)
{
  unsigned config;
  unsigned data, ch;
  config = (_COM_CHR8|_COM_STOP1|_COM_NOPARITY|_COM_2400);
  _bios_serialcom(_COM_INIT,0,config);

  while(1)
  {
    data = 0x0000;
    _bios_serialcom(_COM_RECEIVE,0,data);
    if (data != 0x0000)
      _putch((int)data);
    if (_kbhit())
    {
      ch = (unsigned)_getch();
      if(ch) _bios_serialcom(_COM_SEND,0,ch);
      else if((unsigned)_getch() == 45) break;
    }
  }
}
```

이 코드는 폴링(polling) 방식으로 직렬포트를 감시 하면서 수신된 데이터가 있으면 화면에 출력하고, 키보드 입력이 있으면 입력된 내용을 통신 포트로 보내는 일종의 간단한 통신 에뮬레이터이다. 모뎀은 COM1 에 연결된 것으로 가정했으며, 'ATL-X'를 누르면 프로그램은 종료된다. ALT-X 키를 누르면 첫 번째 getch()에서는 0 을 두 번째 getch()에서는 45 를 읽어 while 루프를 탈출하게 된다. 이 코드는 아주 짧고 간단해 보이지만, 실제로 썩 잘 동작한다. AT 명령을 입력해 볼 수도 있으며, 'ATD'명령을 이용해서 전화를 걸 수도 있다.

이 예제 코드에서 속도를 4800bps 이상으로 설정하면 제대로 동작하지 않는다.

```
config = (_COM_CHR8|_COM_STOP1|_COM_NOPARITY|_COM_9600);
```

이는 바이오스의 직렬통신 기능이 워낙 시원치 않기 때문으로, 4800bps 이상의 속도에서

도 제대로 동작하도록 하려면 직렬포트 감시를 폴링 방식에서 인터럽트 방식으로 바꾸고 , 바이오스 기능을 이용하지 않고 직접 포트에 데이터를 읽고 쓰는 inport, inportb, outport, outportb 등의 함수를 이용해서 UART 레지스터를 제어해야 한다.

다음 코드는 바이오스 기능(인터럽트 14)를 사용하지 않고, 직접 통신 포트를 제어하며 인터럽트 기반 방식으로 동작하는 예이다.

```
#include <stdio.h>
#include <conio.h>
#include <bios.h>

// serial port
#define COM1BASE        0x03F8

#define IRQ4    0x0C

#define THR     (0)     // Transmit Holding Register
#define RBR     (0)     // Receive Buffer Register
#define IER     (1)     // Interrupt Enable Register
#define IIR     (2)     // Interrupt Identification Register
#define LCR     (3)     // Line Control Register
#define MCR     (4)     // Modem Control Register
#define LSR     (5)     // Line Status Register
#define MSR     (6)     // Modem Status Register

// 8259 PIC
#define IMR     (0x21)
#define OCW2    (0x20)
#define MASKON          (0x0B)
#define MASKOFF         (0x0C)

#define EOI     (0x20)


#define MAXBUFSIZE      0x4000

char cCommQueue[MAXBUFSIZE];
int nHeadIndex = 0, nTailIndex = 0;

void interrupt (*OldVect)(unsigned bp, unsigned di,
                          unsigned si, unsigned ds,
                          unsigned es, unsigned dx,
                          unsigned cx, unsigned bx,
                          unsigned ax);

void interrupt CommISR(unsigned bp, unsigned di,
                          unsigned si, unsigned ds,
```

```
                                unsigned es, unsigned dx,
                                unsigned cx, unsigned bx,
                                unsigned ax)
{
  unsigned char cIIR, cData;
  enable();

  // read IIR
  cIIR = inportb(COM1BASE+IIR);
  if((cIIR & 0x04) == cIIR)
  {
    // read RBR
    cData = inportb(COM1BASE+RBR);
    nTailIndex %= MAXBUFSIZE;
    cCommQueue[nTailIndex++] = cData;
  }
        outportb(OCW2, EOI);

  _putch(cData);
}

void InitComm()
{
        // set baud rate 9600
        outportb(COM1BASE+LCR, 0x80);
        outport(COM1BASE, 12);

        // set 8 data bit, no parity, 1 stop bit
        outportb(COM1BASE+LCR, 0x03);

  outportb(MCR, 0x0B);
  outportb(IER, 0x01);

  OldVect = _dos_getvect(IRQ4);
  _dos_setvect(IRQ4, CommISR);

  outportb(IMR, inportb(IMR) & MASKON);
}

void CloseComm()
{
  outportb(MCR, 0x0B);
  outportb(IER, 0x01);

  outportb(IMR, inportb(IMR) & MASKOFF);
  _dos_setvect(IRQ4, OldVect);
}

int CommGetCh()
{
  if(nHeadIndex == nTailIndex)
```

```
    return -1;
  nHeadIndex %= MAXBUFSIZE;
  return cCommQueue[nHeadIndex++];
}

int CommPutCh(int c)
{
  while(!(inportb(LSR) & 0x20))
    ;
  outportb(THR, (unsigned char)c);
}

int CommPuts(char *str)
{
  while(*str)
    CommPutCh(*str++);
}

void main(void)
{
  InitComm();
  CommPuts("AT&FV1X4\r");

  while(1)
  {
    if (_kbhit())
    {
      ch = (unsigned)_getch();
      if(ch) CommPutCh(ch);
      else if((unsigned)_getch() == 45) break;
    }
  }
  CloseComm();
}
```

CommISR 이 인터럽트 서비스 루틴인데, 통신 포트에 어떤 변화가 일어나서 인터럽트가 발생하면 이 루틴이 수행된다. 이 인터럽트 서비스 루틴에서는 직렬 포트의 수신 버퍼가 읽을 수 있는 상태가 되면 수신 버퍼에서 데이터를 읽어 cCommQueue 라는 원형 큐 버퍼에 넣고, 그 데이터를 화면에 출력하게 된다. 그리고, main 함수에서는 그냥 키보드 입력이 있는지 검사하고 있다면 직렬 통신 포트에 써 넣는 작업을 하기만 하면 된다 . 어느 순간에만 통신 포트를 읽어보는 폴링 방식이 아니라, 통신 포트에서 일어나는 변화가 인터럽트 서비스 루틴을 호출하는 인터럽트 구동 방식이기 때문에 중간에 데이터를 잃게 된다거나 하는 일 없이 아주 안정적이다. 또한 바이오스 기능을 사용하지 않고 직접 직렬 포트와 기타 하드웨어에 접근하기 때문에 4800bps 이상의 고속 통신에서도 잘 동작한다.

상당히 복잡해 보이긴 하지만, 사실 한번 잘 정리해서 코딩해 두면, 아주 잘 동작한다. 직렬 통신 기능 자체에 대해서는 위의 예제코드로도 충분하다. 통신 포트를 지정할 수 있도록 코드를 추가하면 말이다. 따라서 MS-DOS 상의 통신 에뮬레이터를 작성하는 것이 어려운 것만은 아니라는 것을 알 수 있을 것이다. MS-DOS 상에서 통신 에뮬레이터를 작성할 때 시간이 많이 걸리고 어려운 작업은 아마도 그 이외의 작업들, 예를 들어, 풀다운 메뉴와 대화상자 처리 부분, 문서 편집기 부분, 기타 부가 기능들에 대한 것들일 것이다.

윈도 3.1 의 통신 프로그램 구조

그렇다면, 윈도 3.1 상의 통신 에뮬레이터는 어떤 식으로 구성될까? 윈도 3.1 에서 통신용 API 들이 일부 제공되기 때문에 MS-DOS 에서처럼 직접 UART 에 접근하거나, 그 내부의 레지스터들에 대해서는 알지 못해도 된다.

MS-DOS 에서는 직렬통신 포트에 어떤 변화가 발생되면 수행되는 인터럽트 서비스 루틴에서 통신 버퍼의 변화를 감시했었다. 하지만 윈도 3.1 에서는 직렬 통신 포트에 생기는 변화를 응용프로그램이 받을 수 있도록 메시지를 발생시켜준다. 윈도 3.0 까지만 해도 이런 메시지가 없었기 때문에 주 메시지 루프 안에서 통신 포트를 계속 검사하는 폴링 방식만이 지원되었다.

```
....
....

while(TRUE)
{
 if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
 {
  if(msg.message == WM_QUIT)
    break;
  else
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    }
 }
 else if(nComID >= 0)
 {
  if((nReadChars = ReadComm(nComID, cBuf, 128)) > 0)
  {
    if((nStart = lstrlen(cInBuf)) < 80)
```

```
    {
      for(i=0; i < nReadChars; i++)
        cInBuf[nStart+i] = cBuf[i];
      cInBuf[nReadChars + nStart] = 0;
      PostMessage(msg.hwnd, WM_USER+1, 0, 0L);
    }
    else
    {
      for(i=0; i < nReadChars; i++)
        cInBuf[nStart+i] = cBuf[i];
      cInBuf[nReadChars] = 0;
      PostMessage(msg.hwnd, WM_USER+1, 1, 0L);
    }
   }
   else
    GetCommError(nComID, &ComStat);
  }
}
....
....
```

이 방식은 MS-DOS 같은 단일 작업(single task) 환경에서는 잘 동작할 지 몰라도 윈도같은 다중 작업(multi task) 환경에서는 잘 동작하지 않는다. 왜냐하면, 어떤 다른 프로그램이 시스템 자원을 과도하게 사용하느라, 이 프로그램이 사용할 수 있는 시간이 아주 적어져 버린다면(메시지 루프를 수행할 시간이 적다면), 통신 포트의 수신 버퍼가 넘쳐버려 수신된 데이터를 잃어 버리게 되기 때문이다.

이런 문제점을 해결하기 위해 통신 포트에 어떤 변화가 일어나면 윈도 시스템이 메시지를 발생시켜 주는 방식이 윈도 3.1 부터 제공되기 시작하였다. WM_COMMNOTIFY 가 그 메시지이고 EnableCommNotification 은 이 메시지를 발생시킬 것인지 아닌지를 설정하는 함수이다.

```
BOOL EnableCommNotification(idComDev, hwnd, cbWriteNotify, cbOutQueue)

int idComDev;    /* communications-device identifier        */
HWND hwnd;     /* handle of window receiving messages */
int cbWriteNotify;          /* number of bytes written before notification      */
int cbOutQueue;             /* minimum number of bytes in output queue      */


ComIdDev : OpenComm()로 얻은 직렬 통신 장치에 대한 핸들
hwnd : WM_COMMNOTIFY 메시지를 받을 윈도 핸들,
      이 값이 NULL 이면 WM_COMMNOTIFY 메시지를 보내지 않는다.
```

cbWriteNotify : WM_COMMNOTIFY 메시지를 보내기 전에 통신 드라이버가
수신 버퍼에 저장한 바이트 수,
즉 수신 버퍼에 수신된 데이터의 바이트 수를 말한다.
cbOutQueue : 출력 버퍼의 데이터 수가 이 숫자보다 작아지면
WM_COMMNOTIFY 메시지를 발생시킨다.

WM_COMMNOTIFY

WM_COMMNOTIFY

wParam : 직렬 통신 장치에 대한 핸들
lParam : LOWORD(lParam)에 CN_EVENT, CN_RECEIVE, CN_TRANSMIT
CN_EVNET : 사건이 발생, 이 사건은 SetCommEventMask 로 설정하며,
일어난 사건의 종류를 알려면 GetCommEventMask 를 이용한다.
GetCommEventMask 를 부르면 이 사건은 소거된다.
CN_RECEIVE : 수신 큐에 데이터가 수신되었다.
이 때 수신된 데이터는 최소 cbWriteNotify 바이트이며
이 cbWriteNotify 는 EnableCommNotification 의 인수이다.
CN_TRANSMIT : 전송 큐에 있는 데이터가 cbOutQueue 보다 작다.
이 cbOutQueue 는 EnableCommNotification 의 인수이다.

윈도 3.1 에서는 EnableCommNotification 으로 적당한 메시지 발생 조건을 지정하고 나면, 통신 포트에 대해 다음과 같은 변화가 발생하면 WM_COMMNOTIFY 메시지를 발생시킨다.

수신 버퍼에 데이터가 수신되었을 때(CN_RECEIVE)

전송버퍼가 지정한 수 이하의 데이터가 되었을 때(CN_TRANSMIT)

통신 포트에 어떤 사건이 발생했을 때(CN_EVENT)

통신 포트에서 일어나는 사건이란, SetCommEventMask 란 함수로 설정한 사건을 말한다.

```
UINT FAR* SetCommEventMask(idComDev, fuEvtMask)

int idComDev;    /* device to enable         */
UINT fuEvtMask;/* events to enable         */


idComDev : OpenComm()로 얻은 직렬 통신 장치에 대한 핸들
fuEvtMask : 통신 포트에서 일어나는 사건으로 취급될 종류
   EV_DSR : DSR(Data Set Ready) 신호가 변경
   EV_CTS : CTS(Clear To Send) 신호가 변경
   EV_CTSS :
   EV_RLSD : RLSD(Receive Line Signal Detect) 신호가 변경
   EV_RLSDS :
   EV_BREAK : BREAK 신호가 검출
   EV_ERR : 오류 검출. CE_FRAME, CE_OVERRUN, CE_PARITY
   EV_PERR : 병렬 장치 오류 검출. CE_NDS, CE_IOE, CE_LOOP, CE_PTO
   EV_RING : Ring indicator 검출
   EV_RXCHAR : 수신 큐에 문자 수신
   EV_RXFLAG : 사건 문자(Event Character)가 수신 큐에 수신
      사건 문자는 DCB 의 EvtChar 에 지정한다.
   EV_TXEMPTY : 전송 큐에서 마지막 문자가 전송
```

윈도 3.1 에서 직렬 통신 프로그래밍은 직렬 포트 하드웨어의 레지스터를 설정하거나 송수신 큐에 대해서 관리할 필요가 없어졌다. 그래서 훨씬 손쉬워진 점도 있지만, 문자 전송시 이전에 전송한 문자가 아직 송신 큐에 그대로 남아 있는지 따위를 검사해 주어야 하는 번거로움도 있다.

통신 포트 열기

윈도 3.1 에서는 OpenComm 함수를 이용해서 통신 포트를 열고 그 핸들을 얻는데, 이 때, 송수신 큐의 크기를 지정하도록 되어 있다.

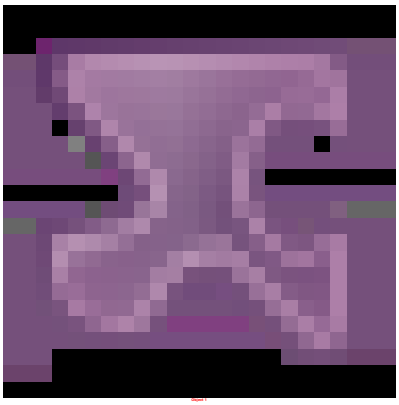
```
int OpenComm(lpszDevControl, cbInQueue, cbOutQueue)

LPCSTR lpszDevControl; /* address of device-control information  */
UINT cbInQueue;          /* size of receiving queue          */
UINT cbOutQueue;         /* size of transmission queue       */

lpszDevControl : "COMn", "LPTn"형식으로 지정한다.
   윈도는 직렬 통신 포트는 COM1 ~ COM9 까지,
   병렬 통신 포트는 LPT1 ~ LPT3 까지 지원한다.
```

```
    하지만, 윈도 표준 통신 드라이버는 COM1 ~ COM4 까지만 지원한다.
cbInQueue : 수신 큐의 바이트 단위 크기
cbOutQueue : 송신 큐의 바이트 단위 크기
```

윈도 3.1 에서 직렬 통신 포트를 통해 문자를 수신하는 것은, 직접 직렬 통신 포트를 읽는 것이 아니라, 통신 드라이버가 마련한 수신 큐를 읽는 것이다. 송신의 경우도 마찬가지로, 송신 큐에 데이터를 넣어두면 윈도 시스템은 이 송신 큐에 있는 데이터를 실제 직렬 통신 포트에 하나씩 쓰는 것이다.



이 그림의 전송 큐와 수신 큐는 직렬 포트 하드웨어에 있는 버퍼를 말하는 것이 아니라 윈도 시스템에서 관리되고 있는 버퍼를 의미하는 것이다.

통신 포트 초기화

통신 포트를 연 다음에는 열어 놓은 통신 포트를 적당하게 초기화해 주어야 한다. 이를 위해 먼저, DCB(Device Control Block) 구조체에 대해 살펴보아야 한다.

```
typedef struct tagDCB        /* dcb */
{
    BYTE Id;
    UINT BaudRate;
    BYTE ByteSize;
    BYTE Parity;
    BYTE StopBits;
    UINT RlsTimeout;
    UINT CtsTimeout;
    UINT DsrTimeout;
```

```
  UINT fBinary       :1;
  UINT fRtsDisable   :1;
  UINT fParity       :1;
  UINT fOutxCtsFlow  :1;
  UINT fOutxDsrFlow  :1;
  UINT fDummy        :2;
  UINT fDtrDisable   :1;

  UINT fOutX         :1;
  UINT fInX          :1;
  UINT fPeChar       :1;
  UINT fNull         :1;
  UINT fChEvt        :1;

  UINT fDtrflow      :1;
  UINT fRtsflow      :1;
  UINT fDummy2       :1;

  char XonChar;
  char XoffChar;
  UINT XonLim;
  UINT XoffLim;
  char PeChar;
  char EofChar;
  char EvtChar;
  UINT TxDelay;
} DCB;
```

Id : 통신 장치의 식별자. 최상위 비트가 1 이면 병렬 통신 장치를 가리킨다.

BaudRate : 상위바이트를 OxFF 로 하면, 하위바이트는 보 레이트 인덱스 값을

사용한다. 보 레이트 인덱스 값은 다음 중 하나이다.

CBR_110

CBR_4400

CBR_9200

CBR_8400

CBR_6000

CBR_28000

CBR_9600

CBR_14400

CBR_19200
CBR_38400
CBR_56000
CBR_128000
CBR_256000

상위 바이트가 0xFF 가 아니면, 이 변수값 자체가 실제 보 레이트가 된다.

ByteSize : 송수신 시 사용할 문자의 비트 수. 4 ~ 8 사이의 값을 갖는다.

Parity : 패리티 비트

EVENPARITY Even 패리티
MARKPARITY Mark 패리티
NOPARITY 패리티 없음
ODDPARITY Odd 패리티

StopBits : 정지 비트

ONESTOPBIT 1 정지 비트
ONE5STOPBITS 1.5 정지 비트
TWOSTOPBITS 2 정지 비트

RlsTimeout : 직렬 통신 장치가 RLSD (receive-line-signal-detect) 신호가

검출될 때까지 기다릴 최대 시간(밀리 초 단위)

RLSD 는 CD(carrier-detect) 신호를 말한다.

CtsTimeout : 직렬 통신 장치가 CTS (clear-to-send) 신호가

검출될 때까지 기다릴 최대 시간(밀리 초 단위)

DsrTimeout : 직렬 통신 장치가 DSR (data-set-ready) 신호가

검출될 때까지 기다릴 최대 시간(밀리 초 단위)

fBinary : 이진 상태를 지정한다.

이진 상태가 아니면 입력에 EofChar 문자가 있으면 데이터의 끝이라 판단한다.

fRtsDisable : RTS (request-to-send) 신호 끔 여부

fParity : 패리티 검사 여부

fOutxCtsFlow : CTS (clear-to-send) 신호를 출력 흐름 제어에서 검사하는지 여부

fOutxDsrFlow : DSR (data-set-ready) 신호를 출력 흐름 제어에서 검사하는지 여부

fDummy : 사용되지 않음

fDtrDisable : DTR (data-terminal-ready) 신호를 끔 여부

fOutX : 전송 시 XON/XOFF 흐름 제어를 사용할 지 여부

fInX : 수신 시 XON/XOFF 흐름 제어를 사용할 지 여부

fPeChar : 패리티 오류 때 받는 문자를 지정.

fNull : 수신 된 널 문자를 버릴 것인지 여부

fChEvt : EvtChar 문자의 수신이 사건(event)로 처리되는지 여부

fDtrflow : DTR (data-terminal-ready) 신호가 수신 흐름 제어에 사용되는지 여부

fRtsflow : RTS (ready-to-send) 신호가 수신 흐름 제어에 사용되는지 여부

fDummy2 : 사용안됨

XonChar : 송수신시 사용되는 XON 문자

XoffChar : 송수신시 사용되는 XOFF 문자

XonLim : XON 문자가 보내지기 전에 수신 큐에 허용된 최소 문자

XoffLim : XOFF 문자가 보내지기 전에 수신 큐에 허용된 최소 문자

PeChar : 패리티 오류시 대치될 문자 지정

EofChar : 데이터 끝을 지정할 문자

EvtChar : 사건으로 취급될 문자 지정

TxDelay : 사용되지 않음

이 장치 제어 블록(DCB) 구조체는 통신 포트 이용 시 설정해 주어야 하는 여러 항목을 정리해 놓은 것이다. 각 변수에 적절한 값들을 넣어주고 이 DCB 변수를 인수로 해서 SetCommState 함수를 호출해 줌으로써 초기화 작업을 마칠 수 있다. 그런데, 이 구조체 각각에 값들을 넣어주는 것은 아주 번거로운 일이 될 것이다. 그 개수도 개수지만 기본적으로 언제나 거의 같은 값들을 갖는 항목들도 많기 때문이다. 이런 작업을 단순하게 해 주는 함수가 바로 BuildCommDCB 라는 함수이다.

```
int BuildCommDCB(lpszDef, lpdcb)

LPCSTR lpszDef;/* address of device-control string          */
DCB FAR* lpdcb;/* address of device-control block          */


lpszDef : DOS 의 MODE 명령에서 사용되는 형식과 같은 형식의 문자열
lpdcb : 설정할 DCB 구조체에 대한 포인터
```

이 함수를 이용하면 간단히 DCB 구조체의 각 값들을 채울 수 있다. 말 그대로 DCB 구조체의 값들만 채우는 것이기 때문에 아직, 통신 포트에는 어떤 설정값도 반영되지 않고 있다. DCB 값에 따라 통신 포트를 설정하려면, SetCommState 함수를 호출해 주어야 한다.

```
int SetCommState(lpdcb)
```

```
const DCB FAR* lpdcb;    /* address of device control block            */
```

만약 lpszDef 에서 지정할 수 없는 항목들의 값을 지정해 주고 싶다면 BuildCommDCB 후에 직접 lpDcb 의 멤버 변수에 값을 써 넣고 SetCommState 를 불러 주면 된다.

마 지 막 으 로 , WM_COMMNOTIFY 메 시 지 를 위 해 SetCommEventMask 와 EnableCommNotification 을 함수를 불러 준다.

```
....
....

nComID = OpenComm("COM1", 4096, 4096);
BuildCommDCB("COM1:9600,N,8,1", &dcb);
dcb.Id = nComID;
dcb.BaudRate = CBR_38400;
SetCommState(&dcb);

SetCommEventMask(nComID, EV_RXCHAR);
EnableCommNotification(nComID, hwnd, 1,-1);
....
....
```

데이터 수신

MS-DOS 에서는 인터럽트 서비스 루틴에서 통신 포트로 들어온 데이터를 프로그램이 마련한 버퍼(원형 큐)에 넣어주는 역할을 하고, 포트에서 한문자 읽기는 단지 이 버퍼에서 한 문자를 읽어오는 것이었다. 직렬 포트에 데이터를 쓸 경우도 쓰기가 가능한 상태인지 검사한 다음, 쓰기가 가능한 상태가 되면 데이터를 쓰도록 한다. 윈도에서 역시 통신 포트에서 데이터를 읽어 오는 ReadComm 이란 함수 역시 통신 드라이버가 마련한 수신 큐에서 데이터를 읽어오게 된다. 이 때 수신 큐에 몇 개의 문자가 들어 있는지 알아 낼 수가 있는데 이는 GetCommError 함수를 이용하면 된다.

```
int ReadComm(idComDev, lpvBuf, cbRead)

int idComDev;    /* identifier of device to read from          */
void FAR* lpvBuf;          /* address of buffer for read bytes          */
int cbRead;        /* number of bytes to read          */
```

```
int GetCommError(idComDev, lpStat)

int idComDev;    /* communications device identifier        */
COMSTAT FAR* lpStat;    /* address of device-status buffer          */
```

이 함수를 호출하면, 현재 통신 포트의 상태가 lpStat 에 넘겨져 온다.

```
typedef struct tagCOMSTAT {    /* cmst                          */
    BYTE status;
    UINT cbInQue;
    UINT cbOutQue;
} COMSTAT;

status : 현재의 전송 상태
cbInQue : 수신 큐에 남아 있는 문자 수
cbOutQue : 전송 큐에 남아 있는 문자 수
```

따라서 다음과 같은 코드로 문자 수신을 하면 된다.

```
....
....
GetCommError(nComId, &ComStat);
cbInQue = ComStat.cbInQue;
if(cbInQue > 0) // 수신 큐에 데이터가 있다면
{
  cbReadChar = ReadComm(nComID, lpszBuf, cbInQue);
  ....
  ....
}
....
....
```

데이터 전송

데이터 송신은 WriteComm 함수를 이용해서 지정한 크기의 데이터를 전송 큐에 써 넣는다. 그렇지만, 언제나 송신 큐가 충분히 남아 있어서 데이터가 모두 전송 큐로 잘 보내지지는 않기 때문에 쓰려고 했던 데이터가 모두 써지는지를 감시해야 한다.

```
int WriteComm(idComDev, lpvBuf, cbWrite)

int idComDev;    /* identifier of comm. device       */
const void FAR* lpvBuf;  /* address of data buffer*/
int cbWrite;        /* number of bytes to write         */
```

전송 큐에 써 넣을 데이터가 lpBuf 이고 그 크기가 cbSize 라고 하자. lpIndex 는 lpBuf 상의 한 위치를 가리키는 포인터이다.

```
....
....
lpIndex = (LPSTR)lpBuf; // lpIndex 초기화
cbTotalSize = 0; // 실제로 전송 큐에 써진 데이터 크기

// 실제로 데이터가 모두 전송 큐에 써질 때까지 반복한다
while(cbTotalSize < cbSize)
{
  cbWriteChar = WriteComm(nComID, lpIndex, (cbSize-cbTotalSize));
  if(cbWriteChar < 0) // 오류 발생
    // 오류가 발생하면 쓴 데이터 바이트의 음수를 돌려 줌
    cbWriteChar = -cbWriteChar;
  lpIndex += cbWriteChar;
  cbTotalSize += cbWriteChar;
}
....
....
```

통신 포트 닫기

통신 포트를 닫고 송,수신 큐에 할당된 메모리를 해제한다.

```
int CloseComm(idComDev)

int idComDev;    /* device to close            */
```

통신 에뮬레이터를 작성하기 위한 기본적인 코드들은 거의 설명을 했기 때문에 위의 코드들을 뼈대로 해서 약간의 살만 붙이면 윈도 3.1 용 통신 에뮬레이터를 작성할 수 있을 것이다.

윈도 95 의 통신 프로그램 구조

Win32 에서는 통신 함수들이 사라져 버렸다(?). 윈도 95 에서 통신 프로그램을 작성하려고 도움말에서 OpenComm 이나 ReadComm, WriteComm 같은 통신 함수를 찾아 보았지만, 없다. 도움말이 잘못된 것일까? 물론 그렇지는 않다. 그렇다면 어떻게 된 일일까? 윈도 3.1 과 Win32 의 통신 함수를 비교해 보자.

다음은 윈도 3.1 의 통신 함수들이다.

```
BuildCommDCB
ClearCommBreak
CloseComm
EnableCommNotification
EscapeCommFunction
FlushComm
GetCommError
GetCommEventMask
GetCommState
OpenComm
ReadComm
SetCommBreak
SetCommEventMask
SetCommState
TransmitCommChar
UngetCommChar
```

WriteComm

음영으로 처리된 부분이 Win32 에서는 사라져 버린 함수들이다. OpenComm, ReadComm, WriteComm, CloseComm 등이 모두 사라져 버린 것을 알 수 있을 것이다. EnableCommNotification 함수도 없어져 버렸으니, WM_COMMNOTIFY 메시지도 당연히 없어져 버렸다. 이 메시지가 없어져 버렸으니, Win32 에서는 윈도 3.0 시절 처럼 메시지 루프나 타이머에서 통신 포트를 감시해야 할까? 아니다, Win32 는 완전한 선점형 멀티 태스킹 환경 아닌가? 멀티 쓰레드도 완전하게 지원된다. Win32 에서 통신 포트에 대한 감시는 바로, 별도의 쓰레드가 책임지게 되는 것이 일반적인 구조이다.

다음은 Win32 통신 함수들이다. 윈도 3.1 의 그것과 달라진 이름들도 많이 보이고, 또한 새로 추가된 함수들이 꽤 많아 보인다. 역상 처리된 함수들이 새로 추가된 함수들이다.

BuildCommDCB
BuildCommDCBAndTimeouts
ClearCommBreak
ClearCommError
CommConfigDialog
DeviceIoControl
EscapeCommFunction
GetCommConfig
GetCommMask
GetCommModemStatus
GetCommProperties
GetCommState
GetCommTimeouts
PurgeComm
SetCommBreak
SetCommConfig
SetCommMask
SetCommState
SetCommTimeouts
SetupComm
TransmitCommChar
WaitCommEvent

하지만, 어찌된 일인지 여기 Win32 통신 함수 부분에는 통신 포트를 열거나 통신 포트에 데이터를 쓰거나 통신 포트에서 데이터를 읽어오거나 통신 포트를 닫는 등의 함수가 없다. 그렇다면 Win32 에서는 어떤 식으로 통신 포트를 다룰까? Win32 에서는 통신 포트를 일종의

파일 자원으로 생각한다. 통신 포트에 대한 작업이, 열어서 읽거나 쓰고 나면 닫아주는 것이 파일 작업과 많이 닮았다는 생각이 들지 않는가? Win32 에서는 통신 포트의 여닫기를 파일 여닫기 함수를 이용하며 통신 포트에 데이터를 읽거나 쓰는 작업도 파일에 데이터를 읽거나 쓰는 함수를 이용한다. 즉, OpenComm 대신에 CreateFile 을 CloseComm 대신에 CloseHandle 을 사용하며 ReadComm 대신에 ReadFile 이나 ReadFileEx 를, WriteComm 대신에 WriteFile 이나 WriteFileEx 를 사용한다.

통신 포트를 파일 개념으로 사용하기 때문에 윈도 3.1 의 UngetCommChar 같은 함수는 지원될 수가 없다.

통신 포트 열기

OpenComm 에 대응되는 것이 CreateFile 이다.

```
HANDLE CreateFile(
  LPCTSTR lpFileName,  // pointer to name of the file
  DWORD dwDesiredAccess,      // access (read-write) mode
  DWORD dwShareMode,          // share mode
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,// pointer to security descriptor
  DWORD dwCreationDistribution,          // how to create
  DWORD dwFlagsAndAttributes,          // file attributes
  HANDLE hTemplateFile          // handle to file with attributes to copy
 );
```

DCB 구조체는 그대로 존재하지만 그 구조에는 약간의 변화가 생겼다.

```
typedef struct _DCB { // dcb
  DWORD DCBlength;          // sizeof(DCB)
  DWORD BaudRate;           // current baud rate
  DWORD fBinary: 1;         // binary mode, no EOF check
  DWORD fParity: 1;         // enable parity checking
  DWORD fOutxCtsFlow:1;     // CTS output flow control
  DWORD fOutxDsrFlow:1;     // DSR output flow control
  DWORD fDtrControl:2;      // DTR flow control type
  DWORD fDsrSensitivity:1;  // DSR sensitivity
  DWORD fTXContinueOnXoff:1; // XOFF continues Tx
  DWORD fOutX: 1;           // XON/XOFF out flow control
  DWORD fInX: 1;            // XON/XOFF in flow control
```

```
    DWORD fErrorChar: 1;       // enable error replacement
    DWORD fNull: 1;            // enable null stripping
    DWORD fRtsControl:2;       // RTS flow control
    DWORD fAbortOnError:1;     // abort reads/writes on error
    DWORD fDummy2:17;          // reserved
    WORD wReserved;            // not currently used
    WORD XonLim;               // transmit XON threshold
    WORD XoffLim;              // transmit XOFF threshold
    BYTE ByteSize;             // number of bits/byte, 4-8
    BYTE Parity;               // 0-4=no,odd,even,mark,space
    BYTE StopBits;             // 0,1,2 = 1, 1.5, 2
    char XonChar;              // Tx and Rx XON character
    char XoffChar;             // Tx and Rx XOFF character
    char ErrorChar;            // error replacement character
    char EofChar;              // end of input character
    char EvtChar;              // received event character
    WORD wReserved1;           // reserved; do not use
} DCB;
```

중첩 입출력

윈도 95 에서는 중첩 입출력이 지원되는데, 중첩 입출력이란, 하나의 스레드가 시간이 많이 걸리는 어떤 작업을 배경으로 수행하면서 다른 작업을 수행하는 것을 말한다. 윈도 95 에서는 직렬 포트에 대한 입출력 작업을 중첩 방식으로 이루어 지도록 할 수 있다. CreateFile 의 인수 dwFlagsAndAttributes 에 FILE_FLAG_OVERLAPPED 가 설정되어 있다면, 중첩 작업이 가능하게 된다. 중첩 작업이 가능한 상태가 되면 ReadFile 이나 WriteFile 에서 버퍼의 내용을 다 읽거나 쓰기 전에 함수 자체는 곧바로 복귀하게 된다. 실제 읽거나 쓰기 작업은 이제 배경(background)작업으로 이루어 지는 것이다.

데이터 수신

ReadComm 해당하는 것이 ReadFile 이다.

```
BOOL ReadFile(
    HANDLE hFile,              // handle of file to read
    LPVOID lpBuffer,           // address of buffer that receives data
    DWORD nNumberOfBytesToRead,        // number of bytes to read
```

```
  LPDWORD lpNumberOfBytesRead,       // address of number of bytes read
  LPOVERLAPPED lpOverlapped // address of structure for data
 );
```

```
HANDLE hComm;

....
....

DWORD Read(LPTSTR lpszBlock, int nMaxLength)
{
 // Read comm port by overlapped I/O
 BOOL fReadStat;
 COMSTAT ComStat;
 DWORD dwErrorFlags, dwError, dwLength;

 // 직렬 통신 포트에 수신된 데이터 수를 얻는다.
 ClearCommError(hComm, &dwErrorFlags, &ComStat);
 if(dwErrorFlags > 0) {
  TRACE("Comm port read error\n");
  return (DWORD)0;
 } // if
 dwLength = min((DWORD)nMaxLength, ComStat.cbInQue);
 if(dwLength > 0) { // 수신된 데이터가 있다면
  // 수신된 데이터를 읽는다.
          fReadStat = ReadFile(hComm, lpszBlock, dwLength, &dwLength,
&GetReadOS());
  if(!fReadStat) { // 오류가 발생하면
   if(GetLastError() == ERROR_IO_PENDING) {
    // 발생한 오류가 입출력 대기중 오류라면
    TRACE("IO Pending...\n");
    // 중첩 작업의 상황을 알아본다.
    while(!GetOverlappedResult(hComm, &GetReadOS(), &dwLength, TRUE))
    {
     // 중첩 작업이 아직 끝나지 않았다
     dwError = GetLastError();
     // 현재 상황이 입출력 작업이 끝나지 않은 상태라면
     // 다시 한번 중첩 작업 상황을 알아본다.
     if(dwError == ERROR_IO_INCOMPLETE)
     { // normal result if not finished
      continue;
     }
     else
     { // 중첩 작업 중 다른 오류 발생!!
      TRACE("<CE-%u>\n", dwError);
      ClearCommError(GetPortHandle(), &dwErrorFlags, &ComStat);
      if(dwErrorFlags > 0)
```

```
          {
            TRACE("<CE-%u>\n", dwErrorFlags);
          }
          break;
        } // else
      } // while
    } // if
    else
    { // 입력 대기중 외에 다른 오류 발생
      dwLength = 0;
      ClearCommError(hComm, &dwErrorFlags, &ComStat);
      if(dwErrorFlags > 0)
      {
        TRACE("<CE-%u>\n", dwErrorFlags);
      } // if
    } // else
  } // if
 } // if
 return dwLength;
}
```

위의 예는 중첩 작업으로 수신 큐를 읽으려 할 때, 입출력 대기 오류 발생하면, 중첩 작업이 완료될 때까지 수신 큐를 폴링하면서 데이터를 모두 읽어들이는 예이다. 개념적으로 윈도 3.1 의 데이터 수신 방법과 크게 다르지 않다. 하지만, 읽기 작업이 길어질 경우, 다 읽지 못했더라도 일단 ReadFile 함수에서는 복귀하고 다음 코드를 수행하게 된다. 이렇게 되면 중첩 작업 상황 검사하는 코드 부분이 수행되게 된다. 따라서 사실상 이 코드는 지정한 데이터를 다 읽을 때까지 폴링 상태에 있게 하기 때문에 중첩 작업 자체의 잇점을 얻을 수는 없다.

데이터 전송

WriteComm 에 해당하는 것이 WriteFile 이다.

```
BOOL WriteFile(
  HANDLE hFile,            // handle to file to write to
  LPCVOID lpBuffer,        // pointer to data to write to file
  DWORD nNumberOfBytesToWrite,        // number of bytes to write
  LPDWORD lpNumberOfBytesWritten,// pointer to number of bytes written
  LPOVERLAPPED lpOverlapped // addr. of structure needed for overlapped I/O
 );
```

```
HANDLE hComm;
....
....

DWORD Write(LPCTSTR lpszBlock, int nMaxLength)
{
  BOOL fWriteStat;
  COMSTAT ComStat;
  DWORD dwBytesWritten, dwError, dwErrorFlags;

  // 데이터를 송신 큐에 쓴다.
  fWriteStat = WriteFile(hComm, lpszBlock, nMaxLength,
                      &dwBytesWritten, &GetWriteOS());

  // 보통의 경우에는 WriteFile 함수가 TRUE 값을 돌려주기 때문에
  // 아래의 코드는 수행되지 않는 경우가 대부분이다.
  // 왜냐하면 직렬 통신 드라이버가 송신 큐에 쓴 데이터를 캐시(cache)하기
  // 때문이다.
  // 수천 바이트 정도의 작업은 중첩작업 없이 곧바로 수행된다.

  if(!fWriteStat) // 데이터를 미처 다 쓰지 못했다면
  {
    // 입출력 대기 상태라면
    if(GetLastError() == ERROR_IO_PENDING)
    {
      // 중첩 작업 상황을 알아본다.
           while(!GetOverlappedResult(hComm,  &GetWriteOS(),  &dwBytesWritten,
TRUE))
      { // 중첩 작업이 끝나지 않았다.
        dwError = GetLastError();
        // 상황이 입출력이 끝나지 않은 것이라면
        // 다시 한번 중첩 작업이 끝났는지 확인한다.
        if(dwError == ERROR_IO_INCOMPLETE)
        { // normal result if not finished
          continue;
        }
        else
        {
          // 중첩 작업 미완이 아닌 다른 오류가 발생
          TRACE("<CE-%u>\n", dwError);

          ClearCommError(hComm, &dwErrorFlags, &ComStat);
          if(dwErrorFlags > 0)
          {
            TRACE("<CE-%u>\n", dwErrorFlags);
          }
          break;
        }
      }
    }
  }
```

```
    else
    { // Write 에서 오류가 발생했으나 중첩 작업 때문은 아닌 다른 오류
      dwBytesWritten = 0;
      ClearCommError(hComm, &dwErrorFlags, &ComStat);
      if(dwErrorFlags > 0)
      {
        TRACE("<CE-%u>\n", dwErrorFlags);
      }
    }
  }
  return dwBytesWritten;
}
```

이런 방법은 사실 중첩 작업의 잇점을 거의 살리지 못하는 것이다. 쓰기 작업 자체를 다른 쓰레드로 동작하도록 한다면 Write 작업이 잘 수행되지 않아 중첩 작업이 반복 일어나게 되는 경우 생기는 수행 상의 문제를 해결할 수 있다. 하지만, Write 한 결과를 동기적으로 알아야 하는 경우가 많기 때문에 다른 쓰레드로 동작하도록 하면 그 구조가 좀 복잡하게 될 것이다.

데이터 송수신 시 중첩 작업을 시도하기는 했지만 사실상 윈도 3.1 과 그 개념에서는 큰 차이가 없다는 것을 알 수 있다. 이 정도 수준의 코드만으로도 충분히 안정되게 동작하는 통신 프로그램을 작성할 수 있다.

읽기나 쓰기 작업을 시도했는데 어느 정도의 시간동안 아무 반응이 없으면 문제가 발생한 것인데, 그 '어느 정도의 시간'을 설정하기 위해 SetCommTimeouts 이 있으며 COMMTIMEOUTS 구조체를 사용하게 된다.

```
BOOL SetCommTimeouts(
    HANDLE hFile,            // handle of communications device
      LPCOMMTIMEOUTS lpCommTimeouts// address of communications time-out
structure
  );
```

```
typedef struct _OVERLAPPED { // o
    DWORD  Internal;
    DWORD  InternalHigh;
    DWORD  Offset;
    DWORD  OffsetHigh;
    HANDLE hEvent;
```

```
} OVERLAPPED;
```

통신 포트 닫기

통신 포트를 닫을 때도 특별히 직렬 포트를 닫기 위한 함수가 존재하는 것이 아니라 단지 파일 핸들을 닫을 때 사용하는 일반적인 함수인 CloseHandle 을 사용한다.

```
BOOL CloseHandle(
    HANDLE hObject          // handle to object to close
);
```

통신 이벤트 감시

Win32 에서는 WM_COMMNOTIFY 메시지가 없어져 버렸기 때문에 통신 포트에서 일어나는 변화를 메시지 형태로 알려주지 않는다. 대신 Win32 는 진정한 의미의 멀티 태스킹 환경이기 때문에 멀티 스레드가 지원된다. 주 스레드(Main 부분)에서 통신 포트를 감시하는 별도의 스레드 하나를 만들어서 그 스레드로부터 통신 포트의 변화를 메시지로 받도록 하면 Win32 에서도 윈도 3.1 과 비슷한 구조로 통신 프로그램을 작성할 수 있다.

통신 포트를 감시하는 스레드에서는 통신 포트에 EV_RXCHAR 라는 사건이 발생하면 WM_COMMNOTIFY32 라는 사용자 정의 메시지를 주 스레드 쪽의 윈도에 넘겨준다. CN_EVENT32 를 lParam 에 함께 넘겨주도록 함으로써 메시지를 보낸 쪽을 확인하도록 한다. 다음은 정의 부분이다.

```
// 통신 사건 메시지를 정의
// WM_COMMNOTIFY32 를 사용자 정의 윈도 메시지로 정의하였다.
#define WM_COMMNOTIFY32     WM_USER+1
#define CN_EVENT32          0x04

#define COMM_MASK         EV_RXCHAR
```

다음 코드는 메시지 처리 부분이다. 수신 큐에 데이터가 들어왔다는 메시지를 받으면 이 데이터를 처리, 즉 화면 데이터를 갱신하고 화면도 갱신한다. 작업이 끝나면 사건 객체 하나를 신호상태(signaled state)로 만드는데, 이는 통신 포트 감시 스레드 쪽에 메시지 처리 작업이 끝났음을 알리기 위해서이다.

```
BEGIN_MESSAGE_MAP(CSerialCommView, CView)
        //{{AFX_MSG_MAP(CSerialCommView)
        ON_WM_CHAR()
        ON_WM_KILLFOCUS()
        ON_WM_SETFOCUS()
        ON_WM_SIZE()
        ON_WM_ERASEBKGND()
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
  // 사용자 정의 메시지
  ON_MESSAGE(WM_COMMNOTIFY32, OnCommNotify)
END_MESSAGE_MAP()

....
....

LRESULT CSerialCommView::OnCommNotify(WPARAM wParam, LPARAM lParam)
{
   int nLength;
   BYTE abIn[MAXBUF+1];
   CSerialCommDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   MSG msg;

   // 포트가 열려있지 않다면 사건객체를 신호상태(signaled state)로 하고 끝낸다
   if(!modem.GetCommPort()->IsOpen())
   {
      SetEvent(modem.GetCommPort()->GetPostEvent());
      return FALSE;
   }
   // 포트가 열려 있다면, 메시지가 쓰레드 함수에서 온 것인지 확인한다.
   if(LOWORD(lParam & CN_EVENT32) != CN_EVENT32)
   {
      SetEvent(modem.GetCommPort()->GetPostEvent());
      return FALSE;
   }

   do
   {
      // 메시지 큐에 있는 다른 윈도 메시지 처리
```

```
      while(PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
         AfxGetApp()->PumpMessage();

      // 메시지를 받았다면 수신 큐에 데이터가 있다는 뜻이므로 이를 읽어
      if(nLength = modem.GetCommPort()->Read((LPSTR)abIn, MAXBUF))
      {
         // 화면에 보여주고 화면을 갱신한다.
         pDoc->WriteBlock((LPSTR)abIn, nLength);
         UpdateWindow();
      }
   } while(nLength > 0);
   // 메시지가 처리되면 사건 객체를 신호상태(signaled state)로 한다.
   SetEvent(modem.GetCommPort()->GetPostEvent());
   return TRUE;
}
```

OnCommNotify 를 호출하게 되는 직렬 포트 감시 스레드는 어떤 조건 하에서 계속 루프를 돌고 있는 형식으로 작성하는 것이 일반적인다. 이 스레드 함수는 주 스레드와는 별도로 수행되기 때문에 루프를 돌면서 계속 대기 상태에 있더라도 주 스레드의 실행에는 거의 아무런 영향을 미치지 않는다.

```
UINT MonitorThreadProc(LPVOID pParam)
{
 CCommPort* pCommPortInfo = (CCommPort*)pParam;
 DWORD dwTransfer, dwEvtMask;
 OVERLAPPED os;

 // 중첩 작업을 위한 수동 리셋 사건 객체를 만든다.
 memset(&os, 0, sizeof(OVERLAPPED));
 os.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
 ASSERT(os.hEvent);

 // 사건 마스크를 설정한다.
 // EV_RXCHAR 는 수신 큐에 문자가 수신되는 사건을 말한다.
 if(!SetCommMask(pCommPortInfo->hComm, EV_RXCHAR))
  return FALSE;

 // 포트가 닫혔으면 쓰레드를 종료한다.
 while(pCommPortInfo->IsOpen())
 {
  dwEvtMask = 0;

  // 직렬 포트의 수신 큐에 문자가 수신될 때까지 기다린다.
  if(!WaitCommEvent(pCommPortInfo->hComm, &dwEvtMask, &os)) {
   if(GetLastError() == ERROR_IO_PENDING) {
```

```
      GetOverlappedResult(pCommPortInfo->hComm,
                          &os, &dwTransfer, TRUE);
      os.Offset += dwTransfer;
    }
   }
   if((dwEvtMask & EV_RXCHAR) == EV_RXCHAR) {
     // WM_COMMNOTIFY32 메시지가 처리되었는지 확인하기 위한 사건 객체
     // 메시지 보내기 전에 사건 객체를 비신호상태(non-signaled state)로.
     ResetEvent(pCommPortInfo->GetPostEvent());
     // 메시지를 보낸다.
                                          ((CWnd*)(pCommPortInfo->hNotifyWnd))-
>PostMessage(WM_COMMNOTIFY32,
                    (WPARAM)pCommPortInfo->hComm, (LPARAM)CN_EVENT32);
     // WM_COMMNOTIFY32 메시지가 처리될 때까지 기다린다.
     WaitForSingleObject(pCommPortInfo->GetPostEvent(), INFINITE);
   } // if
 } // while

 CloseHandle(os.hEvent);
 SetEvent(pCommPortInfo->GetMonitorThreadKilledEvent());

 return TRUE;
}
```

SetCommMask 함수는 윈도 3.1 의 SetCommEventMask 에 대응되는 함수이다. 윈도 3.1 의 경우에는 SetCommEventMask 에서 설정한 사건이 발생하면 WM_COMMNOTIFY 메시지를 발생시킬 수 있었다. 하지만 Win32 에서는 SetCommMask 로 WaitCommEvent 라는 대기 함수가 기다리는 사건을 지정해 주는 것이다. 메시지를 발생시키는 것이 아니라 별도의 스레드에서 사건을 대기할 수 있도록 지정하는 역할을 한다. 위의 예인 EV_RXCHAR 는 직렬 포트에 문자가 수신되었다는 사건을 말한다. 즉, 문자가 수신되면 WaitCommEvent 함수가 TRUE 를 반환하고 다음으로 진행한다는 의미이다.

```
BOOL SetCommMask(
   HANDLE hFile,           // handle of communications device
   DWORD dwEvtMask    // mask that identifies enabled events
  );
```

WaitCommEvent 함수는 중첩 작업으로 이루어 지기 때문에 이 작업이 즉시 끝나지 않으면 FALSE 를 반환하면서 함수를 마친다. 그리고 나서 GetLastError() 해서 ERROR_IO_PENDING 이면 작업이 배경(background)으로 이루어지고 있다는 의미가 된다. 배경작업의 상태는 GetOverlappedResult 함수로 알아 볼 수 있다.

```
BOOL WaitCommEvent(
    HANDLE hFile,            // handle of communications device
    LPDWORD lpEvtMask, // address of variable for event that occurred
    LPOVERLAPPED lpOverlapped,            // address of overlapped structure
   );
```

이로서 MS-DOS 와 윈도 3.1 과 Win32 의 통신 프로그램 구조에 대한 고찰을 끝냈다. 이런 저런 사소한 차이점들이 있긴 하지만 기본적으로는 같은 개념과 구조를 가진다는 것을 알 수 있다.

직렬 통신포트 클래스

자, 그렇다면 이제 실제로 윈도 95 상에서 통신 에뮬레이터를 만들어 보기로 하자. 먼저, 직렬 통신 포트에 대한 클래스를 만들어 보자. 다음 클래스를 한번 살펴 보자.

```
class CCommPort : public CObject
{
//protected: // create from serialization only
public:
        CCommPort();
        DECLARE_DYNCREATE(CCommPort)

// Attributes
protected:
        HANDLE                  m_hCommPort;
        COMM_PORT       m_portID;
        BOOL            m_fOpen;
        DCB             m_dcb;
        OVERLAPPED      m_osWrite, m_osRead;
        CWinThread*     m_pMonitorThread;
   HANDLE       m_hPostEvent;   // comm event notification event
        HANDLE                  m_hEventMonitorThreadKilled;
        HWND            m_hNotifyWnd;  // COMM_MSG notify window
   CPortSetting   m_portSetting;
   COMM_ERROR           m_error;

// Operations
public:
        char*           GetPortName(COMM_PORT portName)
```

```
                                    { return pstrCommPortName[portName]; }
  COMM_PORT   GetPortID() { return m_portID; }
        void            SetPortID(COMM_PORT portID) { m_portID = portID; }
        BOOL            GetOpenFlag() { return m_fOpen; }
        void            SetOpenFlag(BOOL fPortOpen) { m_fOpen = fPortOpen; }
        HANDLE                  GetPortHandle() { return m_hCommPort; }
        void            SetPortHandle(HANDLE hCommPort) { m_hCommPort =
hCommPort; }
        HANDLE                  GetReadOSEvent() { return m_osRead.hEvent; }
        void            SetReadOSEvent(HANDLE hReadOSEvent)
                                { m_osRead.hEvent = hReadOSEvent; }
        OVERLAPPED      GetReadOS() { return m_osRead; }
        HANDLE                  GetWriteOSEvent() { return m_osWrite.hEvent; }
        void            SetWriteOSEvent(HANDLE hWriteOSEvent)
                                { m_osWrite.hEvent = hWriteOSEvent; }
        OVERLAPPED      GetWriteOS() { return m_osWrite; }

        CWinThread*     GetMonitorThread() { return m_pMonitorThread; }
        void            SetMonitorThread(CWinThread* pMonitorThread)
                                { m_pMonitorThread = pMonitorThread; }
        HANDLE                  GetPostEvent() { return m_hPostEvent; }
        void            SetPostEvent(HANDLE hPostEvent)
                                { m_hPostEvent = hPostEvent; }
        HANDLE                  GetMonitorThreadKilledEvent()
                                { return m_hEventMonitorThreadKilled; }
        void            SetMonitorThreadKilledEvent(HANDLE
hEventMonitorThreadKilled)
             { m_hEventMonitorThreadKilled = hEventMonitorThreadKilled; }

        HWND            GetNotifyWindow() { return m_hNotifyWnd; }
        void            SetNotifyWindow(HWND hNotifyWnd)
                                { m_hNotifyWnd = hNotifyWnd; }
  CPortSetting   GetPortSetting() { return m_portSetting; }
  void       SetPortSetting(CPortSetting portSetting)
                          { m_portSetting = portSetting; }
   COMM_ERROR  GetError() { return m_error; }
   void       SetError(COMM_ERROR error) { m_error = error; }

   // Comm port setting
   DWORD      GetBaudRate() { return m_portSetting.GetBaudRate(); }
   void       SetBaudRate(DWORD dwBaudRate)
                          { m_portSetting.SetBaudRate(dwBaudRate); }
   BYTE       GetDataBits() { return m_portSetting.GetDataBits(); }
   void       SetDataBits(BYTE bDataBits)
                          { m_portSetting.SetDataBits(bDataBits); }
   BYTE       GetParity() { return m_portSetting.GetParity(); }
   void       SetParity(BYTE bParity) { m_portSetting.SetParity(bParity); }
          COMM_PORT    GetPort() { return m_portSetting.GetPort(); }
   void       SetPort(COMM_PORT port)
                          { m_portSetting.SetPort(port); SetPortID(port); }
   BOOL       GetRTSCTS() { return m_portSetting.GetRTSCTS(); }
```

```
    void        SetRTSCTS(BOOL fRTSCTS) { m_portSetting.SetRTSCTS(fRTSCTS); }
    BYTE        GetStopBits() { return m_portSetting.GetStopBits(); }
    void        SetStopBits(BYTE bStopBits)
                                { m_portSetting.SetStopBits(bStopBits); }
    BOOL        GetXONXOFF() { return m_portSetting.GetXONXOFF(); }
    void        SetXONXOFF(BOOL fXONXOFF)
                                { m_portSetting.SetXONXOFF(fXONXOFF); }
    BOOL        GetDTRDSR() { return m_portSetting.GetDTRDSR(); }
                    void                                    SetDTRDSR(BOOL      fDTRDSR)
{ m_portSetting.SetDTRDSR(fDTRDSR); }

    int         GetBaudRateIndex(DWORD dwBaudRate)
                                {  return  m_portSetting.GetBaudRateIndex(dwBaudRate);
}
    int         GetParityIndex(BYTE bParity)
                                { return m_portSetting.GetParityIndex(bParity); }
    int         GetStopBitsIndex(BYTE bStopBits)
                                { return m_portSetting.GetStopBitsIndex(bStopBits); }
    int         GetDataBitsIndex(BYTE bDataBits)
                                { return m_portSetting.GetDataBitsIndex(bDataBits); }

// Overrides
            // ClassWizard generated virtual function overrides
            //{{AFX_VIRTUAL(CCommPort)
            //}}AFX_VIRTUAL

// Implementation
public:
            void                StartMonitorThread();
//          void                KillMonitorThread();
            DWORD               Read(LPTSTR lpszBlock, int nMaxLength);
            DWORD               Write(BYTE bByte);
            DWORD               Write(LPCTSTR lpszBlock, int nMaxLength);
    DWORD           Write(LPCTSTR lpszBlock);
            void                Clear();
            BOOL                Open();
            void                Close();
            void                EnableMonitorThread();
            void                DisableMonitorThread();
            BOOL                SetDCB(DWORD BaudRate = CBR_14400,// current baud
rate
                                        BYTE ByteSize = 8,// number of bits/byte, 4-8
                                        BYTE        Parity      =       NOPARITY,//      0-
4=no,odd,even,mark,space
                                        BYTE StopBits = 0,// 0,1,2 = 1, 1.5, 2
                                        int nFlag = FC_RTSCTS);// Flow control flag
    BOOL            IsOpen();
    void            DispError();
    void            ReadProfile();
    void            WriteProfile();
    COMM_ERROR          DTR(BOOL fSetting);
```

```
        virtual ~CCommPort();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

};
```

직렬 통신 포트 클래스

CCommPort 클래스는 CObject 클래스를 상속받아 만들었는데, CObject 클래스는 대부분의 MFC 클래스에 대한 기본 클래스이다.

```
class CCommPort : public CObject
```

CObject 클래스를 상속받아 클래스를 만들면 다음과 같은 잇점이 있다. CObject 는 직렬화(serialization)라고 하는 기능과 실행 시 클래스 정보(run-time class information)를 얻는 기능, 진단(diagnostic) 출력 기능등이 있기 때문에 이 클래스를 상속받아 만든 클래스도 이들 기능을 이용할 수 있게 된다. 직렬화란 객체의 내용을 하드 디스크같은 저장 장치에 저장하거나 또는 반대로 저장 장치에서 읽어내어 다시 객체를 구성할 수 있도록 해 주는 기능을 말하는 것으로, 저장과 불러오기의 과정을 아주 단순하게 구성할 수 있도록 도와준다. 실행시 클래스 정보를 얻는 기능이란 실행 중의 객체의 클래스에 대한 정보를 얻을 수 있는 방법을 제공한다는 말이며, 진단 출력 기능이란 프로그램이 비정상적으로 종료되는 것과 같은 경우에 어떤 부분에서 문제가 발생했는지를 보여주는 기능을 말한다.

직렬 통신 포트에 대한 핸들

```
        HANDLE                  m_hCommPort;
```

Win32 환경에서는 통신 포트 장치를 일반적인 파일과 같은 개념으로 생각하며, 직렬 통신 포트를 열기 위해서 CreateFile 함수를 사용한다.

```
HANDLE CreateFile(
   LPCTSTR lpFileName,  // pointer to name of the file
   DWORD dwDesiredAccess,      // access (read-write) mode
   DWORD dwShareMode,          // share mode
   LPSECURITY_ATTRIBUTES lpSecurityAttributes,// pointer to security descriptor
   DWORD dwCreationDistribution,          // how to create
   DWORD dwFlagsAndAttributes,            // file attributes
   HANDLE hTemplateFile            // handle to file with attributes to copy
  );
```

이 함수를 이용해서 통신 포트를 열 때는 dwShareMode 는 반드시 0 으로 설정해야 한다. 왜냐하면 통신 포트와 같은 자원은 공유될 수 있는 것이 아니기 때문이다. 그리고, dwCreateDistribution 값은 OPEN_EXISTING 을 사용해야 한다. 새로 통신 포트 자원을 만드는 것이 아니고 이미 존재하는 것을 여는 것이기 때문이다. 그리고, hTemplateFile 역시 항상 NULL 이어야 한다.

dwFlagsAndAttributes 는 이 장치에 대한 읽기 쓰기 작업에서 중첩 입출력(overlapped I/O)을 사용할 것인지를 정해주는 인수이다. 이 값이 FILE_FLAG_OVERLAPPED 이면 ReadFile 이나, WriteFile 등의 입출력 함수들이 중첩 입출력 방식으로 동작하게 된다. 중첩 입출력에 대한 것은 이 클래스의 입출력 멤버함수인 Read, Write 에서 살펴보기로 하자.

통신 포트의 아이디

```
        COMM_PORT      m_portID;
```

통신 포트에 대한 핸들값은 내부적으로 사용되는 것이고, 사용자 입장에서는 어떤 통신 포트가 사용되는지에 대한 변수가 필요하다. COMM_PORT 는 다음과 같이 정의되어 있다.

```
typedef enum {
   COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8
} COMM_PORT ;
```

```

```

C++의 엄격한 형 검사 덕분에 COMM_PORT 를 하나의 새로운 형(type)으로 생각할 수가 있다. COMM_PORT 변수인 m_portID 변수는 COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8 이외의 값이 할당되면, 컴파일시 오류가 발생하게 된다.

통신 포트가 사용 중인가?

```
        BOOL            m_fOpen;
```

통신 포트를 열려 있는지 닫혀 있는지를 검사하는 변수이다. 통신 포트를 열면 TRUE 값을 가지면, 통신 포트를 닫으면 FALSE 값을 갖는다.

장치 제어 블록(DCB)

```
        DCB             m_dcb;
```

직렬 통신 포트에 대한 전반적인 설정값들을 저장하는 구조체가 바로 DCB 즉, 장치 제어 블록(Device Control Block)이다. 새로운 설정값을 실제로 통신 포트에 적용하려면 이 구조체의 적당한 멤버 변수에 값을 지정해 놓고, SetCommState 함수를 호출함으로써 직렬 통신 포트에 대한 설정값을 바꿀 수가 있다.

```
typedef struct _DCB { // dcb
    DWORD DCBlength;           // sizeof(DCB)
    DWORD BaudRate;            // current baud rate
    DWORD fBinary: 1;          // binary mode, no EOF check
    DWORD fParity: 1;          // enable parity checking
    DWORD fOutxCtsFlow:1;      // CTS output flow control
    DWORD fOutxDsrFlow:1;      // DSR output flow control
    DWORD fDtrControl:2;       // DTR flow control type
    DWORD fDsrSensitivity:1;   // DSR sensitivity
```

```
    DWORD fTXContinueOnXoff:1; // XOFF continues Tx
    DWORD fOutX: 1;            // XON/XOFF out flow control
    DWORD fInX: 1;             // XON/XOFF in flow control
    DWORD fErrorChar: 1;       // enable error replacement
    DWORD fNull: 1;            // enable null stripping
    DWORD fRtsControl:2;       // RTS flow control
    DWORD fAbortOnError:1;     // abort reads/writes on error
    DWORD fDummy2:17;          // reserved
    WORD wReserved;            // not currently used
    WORD XonLim;               // transmit XON threshold
    WORD XoffLim;              // transmit XOFF threshold
    BYTE ByteSize;             // number of bits/byte, 4-8
    BYTE Parity;               // 0-4=no,odd,even,mark,space
    BYTE StopBits;             // 0,1,2 = 1, 1.5, 2
    char XonChar;              // Tx and Rx XON character
    char XoffChar;             // Tx and Rx XOFF character
    char ErrorChar;            // error replacement character
    char EofChar;              // end of input character
    char EvtChar;              // received event character
    WORD wReserved1;           // reserved; do not use
} DCB;
```

이 DCB 구조체의 각 멤버 변수들에 적당한 값을 모두 채우는 것은 아주 번거로운 일이 될 것이다. 이런 일을 해 주는 함수가 바로 BuildCommDCB 인데, 이 함수는 도스의 MODE 명령에서 사용하는 방식과 같은 방식으로 DCB 구조체의 값을 바꿔준다. 이 함수는 DCB 구조체의 변수에 값만 넣어 줄 뿐, 실제 상태를 바꿔 주는 것은 아니기 때문에 이 후 꼭 SetCommState 를 호출해야 한다. 또는 이런 방법도 있다. 먼저, GetCommState 함수를 호출해서 현재 시스템의 상태를 DCB 구조체에 넣고, 자신이 바꾸고 싶은 멤버 변수에만 값을 지정한 다음, SetCommState 를 호출한다.

중첩 입출력(OVERLAPPED I/O)

```
        OVERLAPPED     m_osWrite, m_osRead;
```

중첩 입출력을 위한 구조체 OVERLAPPED 는 다음과 같다.

```
typedef struct _OVERLAPPED { // o
```

```
    DWORD  Internal;
    DWORD  InternalHigh;
    DWORD  Offset;
    DWORD  OffsetHigh;
    HANDLE hEvent;
} OVERLAPPED;
```

중첩 입출력 방식이란 어떤 방식인가? 일반적인 입출력은 다음과 같은 방식으로 일우어 진다. 예를 들어, WrtieFile 함수로 4096 바이트를 쓰기로 했다면, WriteFile 함수는 어떤 오류가 발생하기 전에는 4096 바이트를 모두 쓰기 전에는 복귀하지 않는다. 즉, 4096 바이트 모두를 쓰기 전에는 WriteFile 함수 내에서 머물게 된다. 읽기에 있어서도 마찬가지이다. 이런 방식에서는 만약 읽기,쓰기 같은 입출력 작업 시간이 길어지게 된다면, 프로그램에서는 다른 작업을 하지 못하고 기다리는 일이 생기게 된다.

중첩 입출력 방식에서는 입출력이 완료되기 전이라도 입출력 함수가 곧바로 복귀하고 실제 입출력 작업은 배경(background)으로 이루어 지는 방식을 말한다. 배경으로 이루어 지는 작업에 대한 결과는 GetOverlappedResult 함수를 이용해서 알아낼 수 있다. 그러니까 중첩 입출력 방식에서는 위의 예에서 4096 바이트를 WriteFile 하려고 할 때, 실제로 4096 바이트를 다 쓰기 전에도 WriteFile 함수는 복귀하고, 쓰기 작업이 배경으로 이루어진다는 것이다.

```
BOOL GetOverlappedResult(
    HANDLE hFile,              // handle of file, pipe, or communications device
    LPOVERLAPPED lpOverlapped,// address of overlapped structure
    LPDWORD lpNumberOfBytesTransferred,          // address of actual bytes count
    BOOL bWait    // wait flag
);
```

입출력을 위한 ReadFile 이나 WriteFile 도 OVERLAPPED 구조체를 인수로 가지며, 실제 입출력이 완료되면 OVERLAPPED 구조체의 hEvent 가 가리키는 사건 객체가 신호 상태가 된다.

m_osWrite, m_osRead 변수는 중첩 입출력 작업의 인수로서 사용된다.

```
BOOL ReadFile(
     HANDLE hFile,            // handle of file to read
    LPVOID lpBuffer,          // address of buffer that receives data
    DWORD nNumberOfBytesToRead,         // number of bytes to read
```

```
  LPDWORD lpNumberOfBytesRead,      // address of number of bytes read
  LPOVERLAPPED lpOverlapped // address of structure for data
);
```

```
BOOL WriteFile(
  HANDLE hFile,             // handle to file to write to
  LPCVOID lpBuffer,       // pointer to data to write to file
  DWORD nNumberOfBytesToWrite,       // number of bytes to write
  LPDWORD lpNumberOfBytesWritten,// pointer to number of bytes written
  LPOVERLAPPED lpOverlapped // addr. of structure needed for overlapped I/O
);
```

읽기나 쓰기 작업 후에 배경으로 이루어 지는 작업에 대한 결과는 GetOverlappedResult 함수를 이용해서 얻을 수 있다. ReadFile 이나 WriteFile 의 인수로 사용되는 lpOverlapped 변수의 hEvent 멤버 변수에는 수동 리셋 사건 객체(manual reset event object)가 할당된다.

사건 객체(event object)란 무엇이고 왜 필요한 것인가? 윈도 95 는 다중 스레드(multi-thread) 프로그램이 가능하다고들 한다. 사건 객체는 다중 스레드 환경에서 스레드간의 통신을 위해 필요한 동기화 객체이다.

프로세스(process)란 자신만의 주소공간과 코드, 데이터 그리고 운영체제의 여러 자원으로 이루어진 실행 중인 프로그램을 말하는 것이다. 운영체제의 자원이란 파일같은 것을 말한다. 즉, 쉽게 생각해서 우리가 보통 프로그램이라고 말하는 것이다. 한글을 실행시켰다면, 한글 프로세스가 수행되고 있다고 생각하면 된다. 이러한 프로세스 내에는 하나 이상의 스레드(thread)가 포함되어 있는데, 윈도 3.1 이하에서는 하나의 프로세스 내에 하나의 스레드만이 존재했다. 그러나 윈 95 환경에서는 하나의 프로세스 내에 여러개의 스레드가 존재할 수가 있는데, 생성된 스레드는 그 스레드를 생성한 프로세스의 메모리를 공유하며, 다른 자원들도 공유한다. 즉, 프로세스는 하나 이상의 스레드로 구성된다고 생각하면 된다. 여러개의 스레드 중에서 처음 프로세스가 시작된 부분, C 로 생각하면 main 함수가 있는 스레드를 주 스레드(primary thread)라고 한다.

만약 다중 스레드로 프로그램을 작성한다면 어떤 문제가 발생할까? 하나의 작업이 끝나고 다음 작업을 하는 식으로 순차적으로 작업이 이루어지는 것이 아니기 때문에 이들 사이에서 어떻게 동기(synchronization)를 맞추느냐가 아주 중요한 문제가 된다. 프로세스나 스레드 간의 통신이 필요하다는 말인데, 이런 동기화 문제를 해결하기 위한 여러 가지 방법이 있다.

윈도 95 에서 제공하는 동기화 객체에는 임계부분(critical section)객체와 뮤텍스(mutex) 객체, 세마포어(semaphore)객체, 사건(event) 객체가 있다. 이 중에서 임계 부분 객체를 제외하고는 모두 커널 객체라고 하는데, 커널 객체는 두가지 상태값을 갖는다. 두가지 상태란 신호상태(signal state)와 비신호상태(nonsignal state)인데, 커널 객체는 언제나 이 둘 중의 하나의 상태에 있게 된다. 신호상태와 비신호 상태의 의미는 여러 가지로 설명이 될 수 있겠지만, 이렇게 생각해 보자. 신호상태는 객체 자신의 입장에서 보면 객체가 사용할 수 있는 상태로 되었다는 것을 말하며, 스레드의 입장에서 보면 어떤 작업이 완료되었다는 의미이다. 소화제 광고 중에 아무개 개그맨이 "나 소화 다 됐어요"하면서 손을 드는 장면이 있는데, 바로 이렇게 어떤 일이 다 되었다고 손을 드는 상태가 바로 신호상태인 것이다. 소화가 다 되었으니 이제 다른 것을 또 먹을 수가 있다는 의미도 된다. 어떤 객체가 신호상태가 될 때까지 계속 기다리는 함수를 대기함수라고 한다. 하나의 객체가 신호상태가 되기를 기다리는 함수가 WaitForSingleObject 이고 여러개의 객체에 대해서 신호상태가 되기를 기다리는 함수가 WaitForMultipleObjects 이다.

WaitForSingleObject 란 hHandle 이 가리키는 객체가 신호상태가 될 때까지 기다린다는 의미이다.

```
DWORD WaitForSingleObject(
  HANDLE hHandle,       // handle of object to wait for
  DWORD dwMilliseconds          // time-out interval in milliseconds
 );
```

```
DWORD WaitForMultipleObjects(
  DWORD nCount,         // number of handles in handle array
  CONST HANDLE *lpHandles,    // address of object-handle array
  BOOL bWaitAll,          // wait flag
  DWORD dwMilliseconds          // time-out interval in milliseconds
 );
```

MFC 에도 동기화를 위한 객체들의 클래스들이 제공되며 대기함수를 위한 클래스로는 CSingleLock 와 CMultiLock 이 있는데, 이들 클래스의 멤버 함수인 Lock 이 바로 객체가 사용가능할 때까지 기다리는 함수인데, 이것이 WaitForXXXObjectX 대기 함수에 해당하겠다.

읽기, 쓰기함수에서 사용되는 lpOverlapped 구조체의 hEvent 변수는 시스템에서 수동 리셋 사건 객체가 사용된다고 했는데, 이는 시스템에서 사용하게 되는 사건객체로서

ReadFile 이나 WriteFile 의 인수로 사용될 OVERLAPPED 구조체의 hEvent 변수에 미리 사건 객체를 만들어 할당해 주어야 한다. 읽기 쓰기 함수에서는 이 사건 객체를 따로 조작해 주어야 하는 일은 없다. 앞서 말했듯이 단지 CreateEvent 함수를 이용해서 사건 객체를 수동 리셋, 초기 상태는 비신호상태로 만들어 m_osRead.hEvent 와 m_osWrite.hEvent 에 넣어주는 일만 해 주면 된다. 수동 리셋은 객체가 신호 상태가 된 다음 ResetEvent 함수를 불러서 수동으로 사건 객체를 리셋 해주어야 하는 객체를 말하고, 자동 리셋은 대기 함수가 끝나고 나면 윈도 시스템이 자동으로 리셋 해 주는 것을 말한다. 따라서 대기 함수 이후에 곧바로 사건 객체를 리셋 시키고 싶다면 자동 리셋 사건 객체를 만들어 사용하면 되고, 원하는 순간에 리셋을 임의로 하고 싶다면 수동 리셋 사건 객체를 만들어 사용하면 된다.

사건객체의 신호, 비신호 상태를 지정해 주는 일은 앞으로 살펴볼 통신 포트 감시 스레드와 주 스레드 사이에서 이루어 지게 되므로, 그 때 자세히 다루기로 하자.

통신 포트 감시 스레드

```
        CWinThread*     m_pMonitorThread;
HANDLE      m_hPostEvent;   // comm event notification event
        HWND            m_hNotifyWnd;  // COMM_MSG notify window
```

윈도 3.1 에서는 통신 포트에 어떤 변화가 생기면 WM_COMMNOTIFY 라는 메시지를 발생시켰다. 하지만 윈도 95 에서는 이 메시지가 없어져 버렸다. 윈도 95 에서 WM_COMMNOTIFY 가 하던 일을 해 주기 위해서는 통신 포트를 감시하는 부분을 스레드로 만들어서, 이 스레드가 주 윈도에 메시지를 보내주도록 해 주면 된다.

윈도 95 에서 WM_COMMNOTIFY 메시지를 부활 시켜보자.

```
// define communication event message
#define WM_COMMNOTIFY32     WM_USER+1
#define CN_EVENT32          0x04
```

WM_COMMNOTIFY32 라는 사용자 정의 메시지를 정의한다. 이제 이 메시지를 발생시켜줄 스레드를 만들어야 한다. MFC 에선 스레드를 사용자 인터페이스 스레드(user interface thread)와 작업 스레드(worker thread)로 구분한다.

사용자 인터페이스 스레드는 독자적으로 사용자 인터페이스 메시지에 대한 처리를 해 주는 스레드를 말한다. 사용자 인터페이스 스레드는 CWinThread 에서 상속받아 클래스를 만들며, DECLARE_DYNCREATE, IMPLEMENT_DYNCREATE 매크로를 사용해서 구현해야 한다. 또한, 주 스레드를 만들 때처럼, InitInstance 함수를 재정의(override)해서 사용해야 한다. 응용프로그램 클래스인 CWinApp 도 CWinThread 를 상속받아 만드는데, 마치 새로운 응용프로그램 클래스를 만들 듯이 스레드를 만들게 된다. 사용자 인터페이스 스레드는 CRuntimeClass 를 인자로 해서 AfxBeginThread 함수를 호출하면 된다.

작업 스레드는 사용자의 입력을 다루거나, 메시지 처리 부분을 갖지 않으며, 전형적인 배경(background) 작업을 하는데 사용된다. 프린트 작업이 배경으로 이루어 지게 하고 싶거나 또는 특정한 어떤 하드웨어를 계속 감시하게 하고 싶다면, 이 작업 스레드를 사용하면 된다. 작업 스레드 역시 AfxBeginThread 함수를 사용하지만, 인자로서 스레드 함수의 시작 주소를 주면 된다. 이 스레드 함수는 반환값이 UINT 이고 인수로는 LPVOID lpParam 을 가지며 다음과 같이 선언되어 사용된다.

```
UINT MonitorThreadProc(LPVOID lpParam)
```

직렬 통신 포트 클래스의 m_pMonitorThread 는 바로 통신 포트를 감시하고 있는 스레드 함수의 시작 주소를 갖게 된다.

그렇다면 직렬 통신 포트 감시 스레드를 먼저 살펴보자.

```
/*
*       File:                   MonitorThread.cpp
*
*       Contains:               implementation of worker thread
*
*       Project:
*
*       Copyright:              (c) 1997 by Kwon, Jae-Rock. All right Reserved.
*
*       Written by:             Kwon, Jae-Rock
*
*       Change History(most recent first) :
*
*       <1> 97.03
*
*/
```

```
#include "stdafx.h"
#include "CommPort.h"
#include "MonitorThread.h"

UINT MonitorThreadProc(LPVOID pParam)
{
	CCommPort* pCommPortInfo = (CCommPort*)pParam;
    DWORD dwTransfer, dwEvtMask;
	OVERLAPPED os;

	memset(&os, 0, sizeof(OVERLAPPED));
	os.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
	ASSERT(os.hEvent);

	if(!SetCommMask(pCommPortInfo->GetPortHandle(), COMM_MASK))
		return FALSE;

	// if port is close, this thread is terminated by exiting the proc
	while(pCommPortInfo->GetOpenFlag())
	{
		dwEvtMask = 0;
		if(!WaitCommEvent(pCommPortInfo->GetPortHandle(),
&dwEvtMask, &os)) {
			if(GetLastError() == ERROR_IO_PENDING) {
				GetOverlappedResult(pCommPortInfo-
>GetPortHandle(),
                                                                &os,
&dwTransfer, TRUE);
				os.Offset += dwTransfer;
			}
		}
		if((dwEvtMask & COMM_MASK) == COMM_MASK) {
        ResetEvent(pCommPortInfo->GetPostEvent());
        ((CWnd*)(pCommPortInfo->GetNotifyWindow()))->PostMessage(
                                WM_COMMNOTIFY32,(WPARAM)pCommPortInfo-
>GetPortHandle(),
         (LPARAM)CN_EVENT32);
        // wait for comm event notificaton complete
        WaitForSingleObject(pCommPortInfo->GetPostEvent(), INFINITE);
		} // if
	} // while

	CloseHandle(os.hEvent);
	SetEvent(pCommPortInfo->GetMonitorThreadKilledEvent());

	return TRUE;
}
```

먼저, 첫줄을 보자.

```
        CCommPort* pCommPortInfo = (CCommPort*)pParam;
```

pParam 에는 직렬 통신 포트 클래스에 대한 포인터가 담겨 온다. 이 포인터를 이용해서 주 스레드의 정보를 넘겨 주게 된다.

그리고는 사건 객체 하나를 만드는데, 이 사건 객체는 WaitCommEvent 함수를 위한 것이다.

```
BOOL WaitCommEvent(
  HANDLE hFile,            // handle of communications device
  LPDWORD lpEvtMask, // address of variable for event that occurred
  LPOVERLAPPED lpOverlapped,            // address of overlapped structure
 );
```

WaitCommEvent 함수는 윈도 3.1 에는 없던 함수로서 통신 포트에서 지정된 사건이 일어날 때까지 기다리는 함수이다. 사건 지정은 SetCommMask 함수로 하며 이 함수는 윈도 3.1 의 SetCommEventMask 에 대응되는 함수이다. SetCommMask 로 설정한 사건이 통신 포트에서 일어날 때까지 WaitCommEvent 함수는 기다리게 된다.

이 작업 스레드는 통신 포트가 열려 있다면(pCommPortInfo->GetOpenFlag()) 계속해서 무한 루프를 돌면서 통신 포트에서 일어나는 사건을 감시하도록 되어 있다. 만약 주 스레드에서 통신 포트를 닫아 버리면, 이 작업 스레드도 자동적으로 종료된다. 왜냐하면 while 루프를 빠져나와 return TRUE 함으로써 함수가 종료되기 때문이다. 이처럼 작업 스레드는 이렇게 어떤 조건 하에서 루프를 돌면서 작업을 하도록 설계되는 것이 보통이다.

발생한 사건이 지정한 사건과 같다면, 통신 포트 클래스의 m_hPostEvent (pCommPortInfo->GetPostEvent())를 리셋시킨다.

```
ResetEvent(pCommPortInfo->GetPostEvent());
```

이 m_hPostEvent 멤버 변수는 바로 이 직렬 포트 감시 스레드에서 주 스레드로 WM_COMMNOTIFY32 메시지를 보내는데, 이 메시지를 받은 주 스레드 쪽에서 메시지 처리

가 제대로 끝났는지를 검사하기 위한 것이다. 주 스레드로 메시지를 보내기는 하는데, 통신 포트 클래스 입장에서는 어떤 곳으로 메시지를 보내야 할 지를 알아야 한다. 통신 포트 클래스를 사용하는 주 스레드가 자신의 윈도 핸들을 m_hNotifyWnd 멤버 변수에 넣어 주어야만 직렬 포트에서 일어나는 사건에 대해 WM_COMMNOTIFY32 메시지를 받을 수 있게 된다.

주 스레드로 메시지를 보내기 전에 m_hPostEvent 객체를 리셋(비신호 상태:non signaled state)해 두고 나서, WM_COMMNOTIFY32 메시지를 보낸다.

```
((CWnd*)(pCommPortInfo->GetNotifyWindow()))->PostMessage(
                                        WM_COMMNOTIFY32,(WPARAM)pCommPortInfo-
>GetPortHandle(),
          (LPARAM)CN_EVENT32);
```

메시지를 보내고 나서는 WaitForSingleObject 함수를 이용해서 m_hPostEvent 객체가 신호상태(signaled state)가 될 때까지 기다린다.

```
// wait for comm event notificaton complete
WaitForSingleObject(pCommPortInfo->GetPostEvent(), INFINITE);
```

WM_COMMNOTIFY32 메시지를 처리하는 주 스레드 쪽에서는 모든 처리를 마친 다음 꼭 m_hPostEvent 객체를 신호상태로 만들어 주어야만 MonitorThreadProc 이 제대로 동작하게 된다.

WM_COMMNOTIFY32 메시지를 처리하는 부분을 살짝 엿보면 다음과 같다.

```
LRESULT CSerialCommView::OnCommNotify(WPARAM wParam, LPARAM lParam)
{
   int nLength;
   BYTE abIn[MAXBUF+1];
   CSerialCommDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   MSG msg;


   if(!modem.GetCommPort()->IsOpen())
   {
      SetEvent(modem.GetCommPort()->GetPostEvent());
```

```
        return FALSE;
    }
    if(LOWORD(lParam & CN_EVENT32) != CN_EVENT32)
    {
        SetEvent(modem.GetCommPort()->GetPostEvent());
        return FALSE;
    }
    do
    {
        while(PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
            AfxGetApp()->PumpMessage();

        if(nLength = modem.GetCommPort()->Read((LPSTR)abIn, MAXBUF))
        {
            pDoc->WriteBlock((LPSTR)abIn, nLength);
            UpdateWindow();
        }
    } while(nLength > 0);

    SetEvent(modem.GetCommPort()->GetPostEvent());
    return TRUE;
}
```

직렬 통신 포트 설정 클래스

```
CPortSetting    m_portSetting;
```

CPortSetting 클래스는 직렬 통신 포트 설정 상태 저장을 위한 클래스이다. 이 클래스에는 보레이트, 데이터비트, 패리티 등에 대한 정보가 들어 있는데, 이 클래스는 실제 레지스트리에 저장될 정보이기도 하다.

```
class CPortSetting
{
// Attributes
protected:
        DWORD       m_dwBaudRate;
        BYTE        m_bDataBits;
        BYTE        m_bParity;
        COMM_PORT   m_port;
```

```
        BOOL        m_fRTSCTS;
        BYTE        m_bStopBits;
        BOOL        m_fXONXOFF;
        BOOL        m_fDTRDSR;

public:
   DWORD       GetBaudRate() { return m_dwBaudRate; }
   void        SetBaudRate(DWORD dwBaudRate) { m_dwBaudRate = dwBaudRate;
}
   BYTE        GetDataBits() { return m_bDataBits; }
   void        SetDataBits(BYTE bDataBits) { m_bDataBits = bDataBits; }
   BYTE        GetParity() { return m_bParity; }
   void        SetParity(BYTE bParity) { m_bParity = bParity; }
        COMM_PORT   GetPort() { return m_port; }
   void        SetPort(COMM_PORT port) { m_port = port; }
   BOOL        GetRTSCTS() { return m_fRTSCTS; }
   void        SetRTSCTS(BOOL fRTSCTS) { m_fRTSCTS = fRTSCTS; }
   BYTE        GetStopBits() { return m_bStopBits; }
   void        SetStopBits(BYTE bStopBits) { m_bStopBits = bStopBits; }
   BOOL        GetXONXOFF() { return m_fXONXOFF; }
   void        SetXONXOFF(BOOL fXONXOFF) { m_fXONXOFF = fXONXOFF; }
   BOOL        GetDTRDSR() { return m_fDTRDSR; }
   void        SetDTRDSR(BOOL fDTRDSR) { m_fDTRDSR = fDTRDSR; }

// Implementation
public:
   int         GetBaudRateIndex(DWORD dwBaudRate);
   int         GetParityIndex(BYTE bParity);
   int         GetStopBitsIndex(BYTE bStopBits);
   int         GetDataBitsIndex(BYTE bDataBits);
};
```

직렬 통신 포트 오류 종류

```
COMM_ERROR  m_error;
```

직렬 통신 포트에 대한 작업 중에 일어나는 오류에 대한 값을 저장하는 멤버 변수이다. 일단은 세 개의 오류값만을 정의해 두었지만, 차차 확장해 나가도록 하자.

```
typedef enum {
```

```
    PORT_NO_ERROR, PORT_OPEN_ERROR, PORT_HANSHAKE_LINE_IN_USE
} COMM_ERROR;
```

멤버 변수 값 읽기, 쓰기

직렬 통신 포트 클래스 내의 멤버 변수들의 값을 읽거나 값을 바꾸기 위해 인라인 함수들이 제공된다.

```
...
...

// Operations
public:
        char*           GetPortName(COMM_PORT portName)
                                { return pstrCommPortName[portName]; }

        COMM_PORT       GetPortID() { return m_portID; }
        void            SetPortID(COMM_PORT portID) { m_portID = portID; }
        BOOL            GetOpenFlag() { return m_fOpen; }
        void            SetOpenFlag(BOOL fPortOpen) { m_fOpen = fPortOpen; }
        HANDLE                  GetPortHandle() { return m_hCommPort; }
        void            SetPortHandle(HANDLE hCommPort) { m_hCommPort =
hCommPort; }
        HANDLE                  GetReadOSEvent() { return m_osRead.hEvent; }
        void            SetReadOSEvent(HANDLE hReadOSEvent)
                                { m_osRead.hEvent = hReadOSEvent; }
        OVERLAPPED      GetReadOS() { return m_osRead; }
        HANDLE                  GetWriteOSEvent() { return m_osWrite.hEvent; }
        void            SetWriteOSEvent(HANDLE hWriteOSEvent)
                                { m_osWrite.hEvent = hWriteOSEvent; }
        OVERLAPPED      GetWriteOS() { return m_osWrite; }

        CWinThread* GetMonitorThread() { return m_pMonitorThread; }
        void            SetMonitorThread(CWinThread* pMonitorThread)
                                { m_pMonitorThread = pMonitorThread; }
        HANDLE                  GetPostEvent() { return m_hPostEvent; }
        void            SetPostEvent(HANDLE hPostEvent)
                                { m_hPostEvent = hPostEvent; }
        HANDLE                  GetMonitorThreadKilledEvent()
                                { return m_hEventMonitorThreadKilled; }
        void            SetMonitorThreadKilledEvent(HANDLE
hEventMonitorThreadKilled)
                { m_hEventMonitorThreadKilled = hEventMonitorThreadKilled; }
}
```

이론적으로 생각해 보면 멤버 변수에 접근할 수 있는 함수를 제공하는, 이런 방식으로 코딩하는 것이 좋다. 왜냐하면, 외부 함수에서 클래스 내의 멤버 변수를 직접 건드리지 못하도록 하면, 나중에 멤버 변수들의 이름이 바뀌거나 하더라도, 다른 소스 코드 부분은 전혀 손대지 않고, 이 클래스의 멤버 변수들만 바꿔주면 되기 때문이다. 이외에도 여러 가지 잇점이 있긴 하지만, 멤버변수들에 대해 모두 이렇게 GetXXX, SetXXX 함수를 만들어 주는 것도 보통 번거로운 일이 아니다. 실제로 MFC 의 클래스들도 멤버 변수들을 위한 접근자가 주어지지 않고, 그냥 변수를 직접 사용하도록 해 놓은 것들이 상당히 많다. 우리도 이 직렬 포트 클래스에서만 이런 식으로 Get, Set 함수를 제공하고 예제 코드 작성에는 이런 방식을 사용하지 않겠다.

직렬 포트 클래스의 멤버 함수

```
....
....

// Implementation
public:
        void            StartMonitorThread();
//      void            KillMonitorThread();
        DWORD           Read(LPTSTR lpszBlock, int nMaxLength);
        DWORD           Write(BYTE bByte);
        DWORD           Write(LPCTSTR lpszBlock, int nMaxLength);
        DWORD       Write(LPCTSTR lpszBlock);
        void            Clear();
        BOOL            Open();
        void            Close();
        void            EnableMonitorThread();
        void            DisableMonitorThread();
        BOOL            SetDCB(DWORD BaudRate = CBR_14400, // current baud
rate
                                BYTE ByteSize = 8,      // number of bits/byte,
4-8
                                BYTE Parity = NOPARITY,// 0-
4=no,odd,even,mark,space
                                BYTE StopBits = 0,// 0,1,2 = 1, 1.5, 2
                                int nFlag = FC_RTSCTS);// Flow control flag
    BOOL      IsOpen();
    void      DispError();
    void      ReadProfile();
    void      WriteProfile();
```

```
    COMM_ERROR DTR(BOOL fSetting);
}
```

직렬 포트 감시 스레드 시작하기

```
        void            StartMonitorThread();
```

이 멤버 함수는 직렬 포트를 감시하는 작업 스레드(worker thread)를 시작하기 위한 함수이다. 배경 작업 위한 스레드로는 작업 스레드가 적당하다는 것은 이미 살펴보았다. 작업 스레드는 AfxBeginThread 함수를 이용해서 만들며 인수로는 작업 스레드의 시작 주소와 pParam 으로 전달될 매개 변수를 지정해 주면 된다.

```
void CCommPort::StartMonitorThread()
{
        if(GetMonitorThread() == NULL)
        { // if comm port don't have monitor thread
                CWinThread* pMonitorThread =
                        AfxBeginThread(MonitorThreadProc, this,
                                THREAD_PRIORITY_BELOW_NORMAL);
                SetMonitorThread(pMonitorThread);
        }

        TRACE("Monitor Thread started.\n");
}
```

직렬 포트에서 데이터 읽기

```
        DWORD           Read(LPTSTR lpszBlock, int nMaxLength);
```

직렬 포트에서 최대 nMaxLength 만큼 데이터를 읽어서 lpszBlock 가 가리키도록 한다.

```
////////////////////////////////////////////////////////////////////////////
// if return 0, some error occur.
DWORD CCommPort::Read(LPTSTR lpszBlock, int nMaxLength)
{
	// Read comm port by overlapped I/O
	BOOL fReadStat;
	COMSTAT ComStat;
	DWORD dwErrorFlags, dwError, dwLength;

	ClearCommError(GetPortHandle(), &dwErrorFlags, &ComStat);
	if(dwErrorFlags > 0) {
		TRACE("Comm port read error\n");
		return (DWORD)0;
	}
	dwLength = min((DWORD)nMaxLength, ComStat.cbInQue);
	if(dwLength > 0) {
		fReadStat = ReadFile(GetPortHandle(), lpszBlock, dwLength,
                                                        &dwLength,
&GetReadOS());
		if(!fReadStat) {
			if(GetLastError() == ERROR_IO_PENDING) {

				TRACE("IO Pending...\n");
				while(!GetOverlappedResult(GetPortHandle(),

	&GetReadOS(), &dwLength, TRUE))
				{
					dwError = GetLastError();
					if(dwError == ERROR_IO_INCOMPLETE)
					{ // normal result if not finished
						continue;
					}
					else
					{
						// an error occurred, try to
recover
						TRACE("<CE-%u>\n", dwError);


	ClearCommError(GetPortHandle(), &dwErrorFlags, &ComStat);
						if(dwErrorFlags > 0)
						{
							TRACE("<CE-%u>\n",
dwErrorFlags);
						}
						break;
					}
				}
			}
			else
```

```
                        { // some other error occured
                                dwLength = 0;
                                ClearCommError(GetPortHandle(),
&dwErrorFlags, &ComStat);
                                if(dwErrorFlags > 0)
                                {
                                        TRACE("<CE-%u>\n", dwErrorFlags);
                                }
                        }
                }
        }
        return dwLength;
}
```

먼저, 입력 큐에 얼마만큼의 데이터가 수신되었는지를 ClearCommError 함수를 통해 알아낸다. ComStat 구조체의 cbInQue 와 cbOutQue 를 보면, 얼마만큼의 데이터가 수신되고, 얼마만큼의 데이터가 출력되기 위해 큐에 남아 있는지를 알 수 있다. 수신 큐에 들어 있는 만큼만 ReadFile 함수를 통해 읽어 들인다. 읽기는 중첩 작업으로 이루어 지는데, 실상 이 함수는 중첩 작업으로 읽기를 하기는 하지만, 곧바로 while 루프를 돌면서 GetOverlappedResult 함수로 중첩 작업의 결과를 폴링하고 있기 때문에 중첩 작업의 잇점을 충분히 살리고 있다고 할 수는 없다. 하지만, 원한다면 이 입출력이 지연되는 동안 다른 작업을 수행하도록 할 수는 있으니 완전히 의미없다고 할 수는 없겠다.

직렬 포트에 데이터 쓰기

```
        DWORD           Write(BYTE bByte);
        DWORD           Write(LPCTSTR lpszBlock, int nMaxLength);
        DWORD       Write(LPCTSTR lpszBlock);
```

세 함수는 기본적으로 같은 함수들이다. 단지, Write(BYTE bByte)함수는 하나의 바이트를 직렬 포트에 쓰는 함수이고, Write(LPCTSTR lpszBlock, int nMaxLength) 함수는 nMaxLength 만큼을 쓰는 함수이다. Write(LPCTSTR lpszBlock)는 lpszBlock 이 문자열인 경우, 이 문자열 길이만큼만 직렬 포트에 쓰게 된다.

```
DWORD CCommPort::Write(BYTE bByte)
```

```
{
	BOOL fWriteStat;
	COMSTAT ComStat;
	DWORD dwBytesWritten, dwError, dwErrorFlags;

	fWriteStat = WriteFile(GetPortHandle(), (LPTSTR)&bByte, 1,
                                                &dwBytesWritten,
&GetWriteOS());

	if(!fWriteStat)
	{
		if(GetLastError() == ERROR_IO_PENDING)
		{
			while(!GetOverlappedResult(GetPortHandle(),
&GetWriteOS(),

&dwBytesWritten, TRUE))
			{
				dwError = GetLastError();
				if(dwError == ERROR_IO_INCOMPLETE)
				{ // normal result if not finished
					continue;
				}
				else
				{
					// an error occurred, try to recover
					TRACE("<CE-%u>\n", dwError);

					ClearCommError(GetPortHandle(),
&dwErrorFlags, &ComStat);
					if(dwErrorFlags > 0)
					{
						TRACE("<CE-%u>\n",
dwErrorFlags);
					}
					break;
				}
			}
		}
		else
		{ // some other error occured
			dwBytesWritten = 0;
			ClearCommError(GetPortHandle(),		&dwErrorFlags,
&ComStat);
			if(dwErrorFlags > 0)
			{
				TRACE("<CE-%u>\n", dwErrorFlags);
			}
		}
	}
	return dwBytesWritten;
```

```
}
```

```
DWORD CCommPort::Write(LPCTSTR lpszBlock, int nMaxLength)
{
	BOOL fWriteStat;
	COMSTAT ComStat;
	DWORD dwBytesWritten, dwError, dwErrorFlags;

	fWriteStat = WriteFile(GetPortHandle(), lpszBlock,
                                                nMaxLength,   &dwBytesWritten,
&GetWriteOS());

	if(!fWriteStat)
	{
		if(GetLastError() == ERROR_IO_PENDING)
		{
			while(!GetOverlappedResult(GetPortHandle(),
&GetWriteOS(),

&dwBytesWritten, TRUE))
			{
				dwError = GetLastError();
				if(dwError == ERROR_IO_INCOMPLETE)
				{ // normal result if not finished
					continue;
				}
				else
				{
					// an error occurred, try to recover
					TRACE("<CE-%u>\n", dwError);

					ClearCommError(GetPortHandle(),
&dwErrorFlags, &ComStat);
					if(dwErrorFlags > 0)
					{
						TRACE("<CE-%u>\n",
dwErrorFlags);
					}
					break;
				}
			}
		}
		else
		{ // some other error occured
			dwBytesWritten = 0;
			ClearCommError(GetPortHandle(),		&dwErrorFlags,
&ComStat);
```

```
                        if(dwErrorFlags > 0)
                        {
                                TRACE("<CE-%u>\n", dwErrorFlags);
                        }
                }
        }
        return dwBytesWritten;
}
```

```
DWORD CCommPort::Write(LPCTSTR lpszBlock)
{
        BOOL fWriteStat;
        COMSTAT ComStat;
        DWORD dwBytesWritten, dwError, dwErrorFlags;
  int nMaxLength = strlen(lpszBlock);

    fWriteStat = WriteFile(GetPortHandle(), lpszBlock, nMaxLength,
                                        &dwBytesWritten, &GetWriteOS());

        if(!fWriteStat)
        {
                if(GetLastError() == ERROR_IO_PENDING)
                {
                        while(!GetOverlappedResult(GetPortHandle(),
&GetWriteOS(),

&dwBytesWritten, TRUE))
                        {
                                dwError = GetLastError();
                                if(dwError == ERROR_IO_INCOMPLETE)
                                { // normal result if not finished
                                        continue;
                                }
                                else
                                {
                                        // an error occurred, try to recover
                                        TRACE("<CE-%u>\n", dwError);
                                        ClearCommError(GetPortHandle(),
&dwErrorFlags, &ComStat);
                                        if(dwErrorFlags > 0)
                                        {
                                                TRACE("<CE-%u>\n",
dwErrorFlags);
                                        }
                                        break;
                                }
                        }
```

```
                }
                else
                { // some other error occured
                        dwBytesWritten = 0;
                        ClearCommError(GetPortHandle(),          &dwErrorFlags,
&ComStat);
                        if(dwErrorFlags > 0)
                        {
                                TRACE("<CE-%u>\n", dwErrorFlags);
                        }
                }
        }
        return dwBytesWritten;
}
```

이 쓰기 함수들 역시 중첩 작업으로 쓰기 작업을 수행하며 WriteFile 함수를 이용해서 쓰기 작업을 시작하고 나서, WriteFile 함수는 곧바로 복귀하고 실행은 계속해서 다음으로 진행하게 된다. 다음부분에서는 GetOverlappedResult 함수를 이용해서 중첩 작업이 끝날 때까지 계속해서 기다린다.

통신 포트 입출력 버퍼 지우기

```
        void            Clear();
```

통신 포트의 입출력 버퍼를 깨끗이 지우는 함수이다.

```
void CCommPort::Clear()
{
        PurgeComm(GetPortHandle(), PURGE_TXABORT | PURGE_RXABORT
                                                |       PURGE_TXCLEAR       |
PURGE_RXCLEAR);
}
```

PurgeComm 함수는 윈도 3.1 의 FlushComm 에 대응되는 함수이다.

통신 포트 열기

```
BOOL		Open();
```

통신 포트 열기 함수에 아무런 인수가 없는데, 이것은 포트 설정 클래스 멤버 변수(CPortSetting m_portSetting)에 지정해 준대로 통신 포트를 열도록 설계되었기 때문이다. 통신 포트에 대한 설정값이 레지스트리에 저장되고 나면 이 통신 포트 열기 함수는 레지스트리에 저장된 정보를 이용하게 되기 때문에 아무런 인수 없이 동작하게 되는 것이다.

```
BOOL CCommPort::Open()
{
	if(GetOpenFlag())
		return FALSE;

  ////////////////////////////////////
  //  Read port setting value
  COMM_PORT portID    = GetPortSetting().GetPort();
	DWORD dwBaudRate    = GetPortSetting().GetBaudRate();
	// number of bits/byte, 4-8
	BYTE bByteSize      = GetPortSetting().GetDataBits();
	// 0-4=no,odd,even,mark,space
	BYTE bParity        = GetPortSetting().GetParity();
	// 0,1,2 = 1, 1.5, 2
	BYTE bStopBits      = GetPortSetting().GetStopBits();
  int nFlag;
  nFlag = GetPortSetting().GetRTSCTS() ? FC_RTSCTS : 0;
  nFlag |= GetPortSetting().GetDTRDSR() ? FC_DTRDSR : 0;
  nFlag |= GetPortSetting().GetXONXOFF() ? FC_XONXOFF : 0;

  ////////////////////////////////////
	//          Open Comm Port
	HANDLE hCommPort = CreateFile(GetPortName(portID),
                                      GENERIC_READ                    |
GENERIC_WRITE,
                                      0,          // Exclusive access
                                      NULL, // no security attrs
                                      OPEN_EXISTING,
                                      FILE_ATTRIBUTE_NORMAL           |
FILE_FLAG_OVERLAPPED,
                                      NULL);
```

```
        if(hCommPort == (HANDLE)-1)
   {
      SetError(PORT_OPEN_ERROR);
                return FALSE;
   }

        SetPortHandle(hCommPort);
        SetOpenFlag(TRUE);

        ////////////////////////////////////////
        SetupComm(GetPortHandle(), RXQUEUESIZE, TXQUEUESIZE);

        ////////////////////////////////////////
        // set up for overlapped non-blocking I/O
        COMMTIMEOUTS CommTimeOuts;
  CommTimeOuts.ReadIntervalTimeout = MAXDWORD;
  CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
  CommTimeOuts.ReadTotalTimeoutConstant = 0;
  CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
  CommTimeOuts.WriteTotalTimeoutConstant = 5000;
        SetCommTimeouts(GetPortHandle(), &CommTimeOuts);

        ////////////////////////////////////////////
        // set up connection
        if(SetDCB(dwBaudRate, bByteSize, bParity, bStopBits, nFlag)) {
                SetMonitorThread(NULL);                 //      Monitor      thread
initialize
                StartMonitorThread();           // Monitor Thread Start
                // assert DTR
                DTR(TRUE);
        }
        else {
                SetOpenFlag(FALSE);
                CloseHandle(GetPortHandle());
                return FALSE;
        }

        ////////////////////////////////////////
        // Create Events
        HANDLE hPostEvent = CreateEvent(NULL,           // no security

        TRUE,   // manual reset

        TRUE,   // intial event is set

        NULL);  // no name
        HANDLE hEventMonitorThreadKilled = CreateEvent(NULL,

        TRUE,   // manual reset

        FALSE,  // intial event is reset
```

```
        NULL);   // no name
        SetPostEvent(hPostEvent);
        SetMonitorThreadKilledEvent(hEventMonitorThreadKilled);

        HANDLE hReadOSEvent = CreateEvent(NULL,      // no security
                                                                TRUE,
        // manual reset
                                                                FALSE,
        // initial event is reset
                                                                NULL);
        // no name
        HANDLE hWriteOSEvent = CreateEvent(NULL,// no security
                                                                TRUE,
        // manual reset
                                                                FALSE,
        // initial event is reset
                                                                NULL);
        // no name
        SetReadOSEvent(hReadOSEvent);
        SetWriteOSEvent(hWriteOSEvent);
        GetReadOS().Offset = GetWriteOS().Offset = 0;

        // Clear comm buffer
        PurgeComm(GetPortHandle(), PURGE_TXABORT | PURGE_RXABORT
                                                        |           PURGE_TXCLEAR           |
PURGE_RXCLEAR);

        TRACE("Open Comm Port\n");
        return TRUE;
}
```

먼저, m_portSetting 클래스로부터 설정값들을 읽어온다. 이 값들은 사용자가 대화상자를 통해 값을 변경하면 이 클래스에 반영이 되도록 되어 있다.

그 다음에는 CreateFile 함수를 통해 통신 포트를 연다. 이 때 중첩 입출력을 사용한다는 플래그를 설정하는 것을 잊지 말것.

SetupComm 함수를 통해 입출력 큐의 크기를 설정해 준다.

그리고는 시간 초과(time outs)에 대한 설정을 해 주어야 한다. 시간 초과는 중첩작업으로 입출력 작업이 일어날 때, 정해진 일정시간 이내에 입출력이 이루어지지 않으면, 발생하게 된다.

```
typedef struct _COMMTIMEOUTS {   // ctmo
    DWORD ReadIntervalTimeout;
    DWORD ReadTotalTimeoutMultiplier;
    DWORD ReadTotalTimeoutConstant;
    DWORD WriteTotalTimeoutMultiplier;
    DWORD WriteTotalTimeoutConstant;
} COMMTIMEOUTS,*LPCOMMTIMEOUTS;
```

시간 초과(timeout)를 계산하는 방법은 다음과 같다. 예를 들어 문자를 수신하는 경우에, 하나의 문자가 수신되고 두 번째 문자가 수신될 때까지의 시간간격이 "정해진" 시간보다 길어지면 시간 초과가 발생하게 되는 것이다. 이 "정해진" 시간을 계산하는 방법은 이렇다.

총시간 초과 = (Multiplier*바이트 수) + Constant

1024 바이트의 데이터를 중첩작업으로 쓰기작업을 시도했다고 하자. 이 경우 WriteTotalTimeoutMultiplier 이 2 이고, WriteTotalTimeoutConstant 가 5000 이라면, 총시간 초과는 (2 * 1024) + 5000 = 7048 즉, 7.048 초이다. 이 경우에는 7.048 초가 지나도 쓰기 작업이 완료되지 않는다면, 시간초과 오류가 발생하게 된다.

통신포트 닫기

```
        void                Close();
```

통신 포트를 닫고, 사용하던 사건 객체의 핸들도 모두 반납한다.

```
void CCommPort::Close()
{
        if(GetOpenFlag() == FALSE)
                return;

        SetOpenFlag(FALSE); // monitor thread killed by exiting the proc
        SetCommMask(GetPortHandle(), 0);

        DWORD dwExitCode;

        if(GetExitCodeThread(GetMonitorThread()->m_hThread,     &dwExitCode)
```

```
&&
                dwExitCode == STILL_ACTIVE)
        {       // Wait for monitor thread killed
                WaitForSingleObject(GetMonitorThreadKilledEvent(), INFINITE);
        }
        SetMonitorThread(NULL);

        // drop DTR
        DTR(FALSE);
        // flush buffer
        PurgeComm(GetPortHandle(), PURGE_TXABORT | PURGE_RXABORT
                                                |       PURGE_TXCLEAR       |
PURGE_RXCLEAR);

        CloseHandle(GetPortHandle());

        CloseHandle(GetPostEvent());
        CloseHandle(GetMonitorThreadKilledEvent());

        CloseHandle(GetReadOSEvent());
        CloseHandle(GetWriteOSEvent());
        TRACE("Close Comm Port\n");
}
```

통신포트 감시 스레드 활성화

```
        void            EnableMonitorThread();
```

통신 포트에서 일어나는 사건을 감시하는 스레드의 기능을 활성화 시킨다. 이 함수는 스레드 자체를 수행시킨다거나 하지는 않고, 단지 통신 포트 감시 이벤트 마스크만을 설정해 줄 뿐이다.

이런 방식 대신, ResumeThread, SuspendThread 함수 쌍을 이용해도 좋다. SuspendThread 가 스레드 수행을 일시 중지시키는 함수이고, ResumeThread 가 일시 중지된 스레드의 수행을 재개하는 함수이다. SuspendThread 함수를 호출하면 suspend count 라고 하는 숫자가 하나 증가하는데, 이 숫자는 SuspendThread 함수가 호출될 때마다 하나씩 증가한다. 반대로 ResumeThread 가 호출되면 이 숫자가 하나씩 감소한다. 이 suspend count 라는 값이 0 이 되어야 즉, SuspendThread 가 연속 호출된 숫자만큼

ResumeThread 가 호출되어야 스레드 수행이 재개되도록 되어 있기 때문에 두 함수의 짝을 잘 맞춰 주어야 한다.

통신 포트 사건 마스크를 설정해 주는 방식으로 스레드 활성화, 비활성화를 한다면 함수를 여러번 불러도 아무 문제가 없다.

```
void CCommPort::EnableMonitorThread()
{
	SetCommMask(GetPortHandle(), COMM_MASK);
}
```

통신포트 감시 스레드 비활성화

```
	void		DisableMonitorThread();
```

통신 포트에서 일어나는 사건을 감시하는 스레드의 기능을 비활성화 시킨다. 이 함수는 스레드 자체를 종료시킨다거나 하지는 않고, 단지 통신 포트 감시 이벤트 마스크만을 0, 즉 어떤 사건에 대해서도 감시하지 말 것을 설정해 줄 뿐이다. 이렇게 하면 통신 포트 감시 스레드에서는 통신 포트에서 일어나는 사건을 감지하지 못함으로써 스레드 자체가 비활성화된 것과 같은 효과를 얻을 수 있다.

```
void CCommPort::DisableMonitorThread()
{
	SetCommMask(GetPortHandle(), 0);
}
```

장치 제어 블록(DCB) 설정하기

```
	BOOL		SetDCB(DWORD BaudRate = CBR_14400, // current baud
```

```
rate
                                    BYTE ByteSize = 8,        // number of bits/byte,
4-8
                                    BYTE        Parity        =        NOPARITY,//        0-
4=no,odd,even,mark,space
                                    BYTE StopBits = 0,// 0,1,2 = 1, 1.5, 2
                                    int nFlag = FC_RTSCTS);// Flow control flag
```

직렬 통신 자원에 대한 설정값 정의를 위해 DCB 구조체가 사용된다.

```
typedef struct _DCB { // dcb
   DWORD DCBlength;          // sizeof(DCB)
   DWORD BaudRate;           // current baud rate
   DWORD fBinary: 1;         // binary mode, no EOF check
   DWORD fParity: 1;         // enable parity checking
   DWORD fOutxCtsFlow:1;     // CTS output flow control
   DWORD fOutxDsrFlow:1;     // DSR output flow control
   DWORD fDtrControl:2;      // DTR flow control type
   DWORD fDsrSensitivity:1;  // DSR sensitivity
   DWORD fTXContinueOnXoff:1; // XOFF continues Tx
   DWORD fOutX: 1;           // XON/XOFF out flow control
   DWORD fInX: 1;            // XON/XOFF in flow control
   DWORD fErrorChar: 1;      // enable error replacement
   DWORD fNull: 1;           // enable null stripping
   DWORD fRtsControl:2;      // RTS flow control
   DWORD fAbortOnError:1;    // abort reads/writes on error
   DWORD fDummy2:17;         // reserved
   WORD wReserved;           // not currently used
   WORD XonLim;              // transmit XON threshold
   WORD XoffLim;             // transmit XOFF threshold
   BYTE ByteSize;            // number of bits/byte, 4-8
   BYTE Parity;              // 0-4=no,odd,even,mark,space
   BYTE StopBits;            // 0,1,2 = 1, 1.5, 2
   char XonChar;             // Tx and Rx XON character
   char XoffChar;            // Tx and Rx XOFF character
   char ErrorChar;           // error replacement character
   char EofChar;             // end of input character
   char EvtChar;             // received event character
   WORD wReserved1;          // reserved; do not use
} DCB;
```

이 구조체의 각 변수에 적당한 값을 넣어주고, SetCommState 함수를 호출해 주면 설정값이 직렬 통신 장치에 반영이 된다. 이 구조체의 항목이 너무 많기 때문에 일일이 값을 지정

해 주기가 힘들다. 따라서, 보통은 BuildCommDCB 라는 함수를 이용하는데, 이 함수는 MS-DOS 의 mode 명령의 형식으로 직렬통신 장치를 설정할 수 있도록 해 준다.

```
BOOL BuildCommDCB(
  LPCTSTR lpDef,          // pointer to device-control string
  LPDCB lpDCB // pointer to device-control block
 );
```

lpDef 에 "COM1: baud=9600 parity=N data=8 stop=1" 식의 mode 명령의 명령행 인수를 넣어주면 lpDCB 에 적당한 값들을 채워 주게 되어 있다. BuildCommDCB 는 DCB 구조체의 값을 채워주기만 할 뿐, 그 값이 실제 직렬 통신 장치에는 전혀 반영되지 않는다. DCB 구조체의 내용을 직렬 통신 장치에 반영시키려면 SetCommState 함수를 이용해야 한다.

```
BOOL SetCommState(
  HANDLE hFile,           // handle of communications device
  LPDCB lpDCB // address of device-control block structure
 );
```

```
BOOL GetCommState(
  HANDLE hFile,           // handle of communications device
  LPDCB lpDCB // address of device-control block structure
 );
```

DCB 구조체에 값을 채워주는 방법에는 위 방법 외에도 다음과 같은 방법이 있다. GetCommState 라는 함수는 현재 직렬 통신 장치의 상태를 lpDCB 로 읽어오는 함수이다. 이 함수를 이용해서 현재 상태를 lpDCB 에 모두 채워 넣은 후, 변경하고 싶은 변수만 새로 설정한 다음, SetCommState 함수를 호출해 주는 방법이다.

```
BOOL CCommPort::SetDCB(DWORD dwBaudRate,                                //
current baud rate
                                                BYTE bByteSize,
// number of bits/byte, 4-8
                                                BYTE bParity,
// 0-4=no,odd,even,mark,space
                                                BYTE bStopBits,
```

```
// 0,1,2 = 1, 1.5, 2
                                                    int nFlag)                          //
Flow control flag
{
        m_dcb.DCBlength = sizeof(DCB);
        GetCommState(GetPortHandle(), &m_dcb);

        m_dcb.BaudRate = dwBaudRate;
        m_dcb.ByteSize = bByteSize;
        m_dcb.Parity = bParity;
        m_dcb.StopBits = bStopBits;

        // set hardware control
        BYTE bSet = (BYTE)((nFlag & FC_DTRDSR) != 0);
        m_dcb.fOutxDsrFlow = bSet;
        m_dcb.fDtrControl    =    (bSet)    ?    DTR_CONTROL_HANDSHAKE    :
DTR_CONTROL_ENABLE;

        bSet = (BYTE)((nFlag & FC_RTSCTS) != 0);
        m_dcb.fOutxCtsFlow = bSet;
        m_dcb.fRtsControl    =    (bSet)    ?    RTS_CONTROL_HANDSHAKE    :
RTS_CONTROL_ENABLE;

        m_dcb.fInX = m_dcb.fOutX = bSet;
        m_dcb.XonChar = ASCII_XON;
        m_dcb.XoffChar = ASCII_XOFF;
        m_dcb.XonLim = 100;
        m_dcb.XoffLim = 100;

        m_dcb.fBinary = TRUE;
        m_dcb.fParity = TRUE;

        TRACE("Setup DCB\n");
        return SetCommState(GetPortHandle(), &m_dcb);
}
```

이 함수는 다음과 같이 기본 인자(default parameter)를 지정해 주었기 때문에 단순 목적으로 사용할 때는 인자 없이 그냥 SetDCB()만 호출해 주면 된다.

```
BOOL SetDCB(DWORD BaudRate = CBR_14400,// current baud rate
                        BYTE ByteSize = 8,                  //    number    of
bits/byte, 4-8
                        BYTE Parity = NOPARITY,             //                0-
4=no,odd,even,mark,space
                        BYTE StopBits = 0,                  // 0,1,2 = 1, 1.5,
2
```

```
                        int nFlag = FC_RTSCTS);             // Flow control flag
```

설정값을 레지스트리에 저장하기, 레지스트리에서 읽기

```
void      ReadProfile();
```

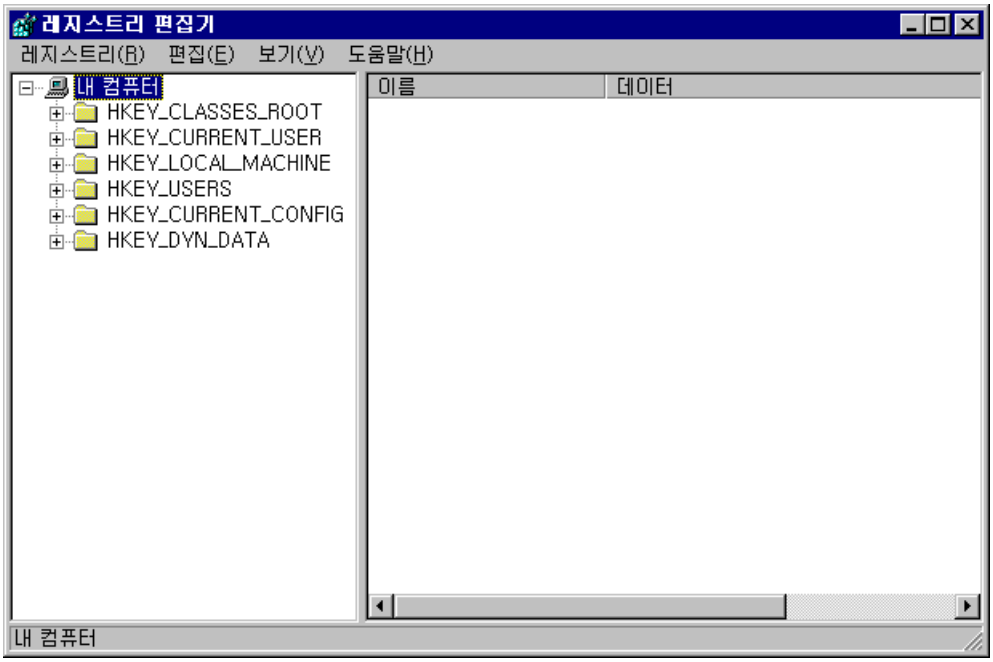
```
void      WriteProfile();
```

윈도 3.1 에서는 윈도의 시스템이나 응용프로그램들의 정보를 확장자가 '.INI'인 파일에 저장해 두었다. 대표적으로 'WIN.INI'와 'SYSTEM.INI'가 있다. 이 밖에도 각 응용프로그램들도 윈도 디렉토리에 자신들의 INI 파일들을 작성해 놓았다. 때문에 응용프로그램을 설치하고 사용하다, 응용프로그램을 지우려고 할 때, 응용프로그램이 설치된 디렉토리를 지우더라도 윈도 디렉토리에 설치된 INI 파일은 삭제되지 않고 남아 있게 되는 경우가 대부분이다. 때문에 시간이 좀 흐르고 나면 윈도 디렉토리에는 쓰지 않는 수많은 INI 파일들이 존재하게 되며, 어느 것이 어떤 응용프로그램에 필요한 것인지도 알 지 못하게 된다. 최근의 프로그램들은 대부분 Uninstall 프로그램을 같이 설치해 주기 때문에 이러한 문제는 많이 줄어들었다. 그리고, 시스템 INI 파일이 윈도 디렉토리 밑에 일반 파일 형태로 존재하기 때문에 실수로 지워 질 수도 있고, 만약 지워진 것이 시스템에 관련된 것이라면 윈도 시스템 자체를 사용할 수 없게 되기도 한다.

윈도 95 에서는 윈도 3.1 의 INI 를 통한 시스템 관리 방법을 벗어나 레지스트리라고 하는 일종의 시스템 정보 데이터베이스를 사용한다. 윈도 디렉토리 밑에 있는 SYSTEM.DAT 와 USER.DAT 가 바로 그 데이터 파일이며, SYSTEM.DA0 와 USER.DA0 가 백업 파일이다. 만약 레지스트리 데이터 파일이 깨지거나 또는 레지스트리 편집 도중에 실수로 어떤 정보를 지워버렸다면 바로 이 SYSTEM.DA0 와 USER.DA0 백업 파일을 이용해서 이전 상태로 복구할 수 있다.

```
C:\WINDOWS\>COPY SYSTEM.DA0 SYSTEM.DAT
C:\WINDOWS\>COPY USER.DA0 USER.DAT
```

윈도 95 에는 레지스트리를 정보를 보거나 편집하기 위한 도구로서 레지스트리 편집기가 제공된다. 윈도 디렉토리 밑에 'REGEDIT.EXE'란 파일이 바로 레지스트리 편집기이다.

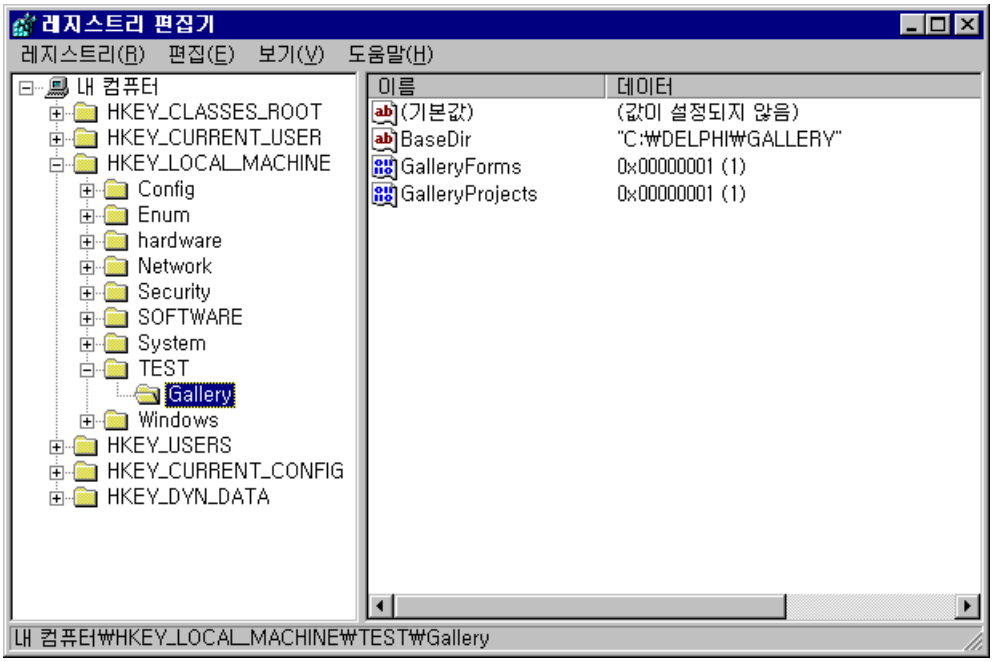


이 편집기에서 폴더 모양으로 되어 있는 것이 '키'이고 키에는 값들을 적어 넣을 수 있는데, 값은 문자열이나, 이진 값, 또는 DWORD 값 세 개 중의 하나이다. 값은 값의 이름과 값의 데이터로 구성되는데, 정확한 비교는 아니지만, INI 의 SECTION 과 ITEM 에 대응시켜 본다면 키가 INI 파일 자체 또는 SECTION 에 해당하고, 값의 이름이 ITEM 에 해당한다고 생각하면 된다.

만약 TEST.INI 라는 파일의 내용이 다음과 같다고 하자.

```
[Gallery]
BaseDir=C:\DELPHI\GALLERY
GalleryProjects=1
GalleryForms=1
```

[Gallery]가 SECTION 이고, BaseDir, GalleryProjects, GalleryForms 는 ITEM 이라고 한다. 이것을 레지스트리 상에 나타내 보면 다음과 같은 모양이 될 것이다.



윈도 95 에는 레지스트리에 키나 값을 추가하거나 삭제하기 위한 API 함수가 존재한다. 그렇다면 이제 INI 파일은 사용할 수 없는가? 아니다. 여전히 INI 관련 함수들은 잘 동작되며, API 들도 굳건히 살아 남아 있다. 또한 윈도 3.1 에서 쓰였던 INI 관리 함수들을 이용해서 레지스트리를 관리하는 방법도 있다. SetRegistryKey 라는 함수를 사용하면 되는데 보통 InitInstance 함수에서 호출하게 된다. SetRegistryKey 라는 함수는 CWinApp 의 멤버 함수이다.

```
void SetRegistryKey( LPCTSTR lpszRegistryKey );
```

인수로는 키의 이름이 들어가는데, 키 이름은 보통 회사 이름을 사용한다. 이 키는 레지스트리에서 HKEY_CURRENT_USER\Software\<키 이름>으로 등록되는데 이 아래로 다시 \<응용프로그램의 이름>\<section 이름>\<값의 이름>이 들어가기 때문에 회사이름으로 하는 것이 좋다. 이 함수를 호출하면 윈도 디렉토리 밑에 INI 파일을 만드는 대신 레지스트리에 키를 만들고, 이렇게 만들어진 레지스트리 키는 역시 CWinApp 의 멤버 함수인 GetProfileInt, GetProfileString, WriteProfileInt, WriteProfileString 함수로 관리된다.


```
void CCommPort::ReadProfile()
{
  int  nBaudRate, nDataBits, nParity, nPort, nStopBits;
  BOOL fRTSCTS, fXONXOFF, fDTRDSR;

  CWinApp* pApp = AfxGetApp();

        nBaudRate = pApp->GetProfileInt(COMM_PORT_SECTION,
                                        strPortItem[ITEM_BAUDRATE],

        GetPortSetting().GetBaudRateIndex(DEFAULT_BAUDRATE));

        nDataBits = pApp->GetProfileInt(COMM_PORT_SECTION,
                                        strPortItem[ITEM_DATABITS],

        GetPortSetting().GetDataBitsIndex(DEFAULT_DATABITS));
        nParity          =     pApp->GetProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_PARITY],

        GetPortSetting().GetParityIndex(DEFAULT_PARITY));
        nPort            =     pApp->GetProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_PORT],
                                        DEFAULT_PORT);
        fRTSCTS          =     pApp->GetProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_RTSCTS],
                                        DEFAULT_RTSCTS);
        nStopBits = pApp->GetProfileInt(COMM_PORT_SECTION,
```

```
                                        strPortItem[ITEM_STOPBITS],

        GetPortSetting().GetStopBitsIndex(DEFAULT_STOPBITS));
        fXONXOFF        =       pApp->GetProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_XONXOFF],
                                        DEFAULT_XONXOFF);
        fDTRDSR         =       pApp->GetProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_DTRDSR],
                                        DEFAULT_DTRDSR);

  SetBaudRate(dwBaudRateTable[nBaudRate]);
        SetDataBits(bDataBitsTable[nDataBits]);
        SetParity(bParityTable[nParity]);
        SetPort(portTable[nPort]);
        SetRTSCTS(fRTSCTS);
        SetStopBits(bStopBitsTable[nStopBits]);
        SetXONXOFF(fXONXOFF);
        SetDTRDSR(fDTRDSR);
}
```

```
void CCommPort::WriteProfile()
{
   int  nBaudRate, nDataBits, nParity, nPort, nStopBits;
   BOOL fRTSCTS, fXONXOFF, fDTRDSR;
   CWinApp* pApp = AfxGetApp();

   nBaudRate = GetBaudRateIndex(GetBaudRate());
   nDataBits = GetDataBitsIndex(GetDataBits());
   nParity   = GetParityIndex(GetParity());
   nPort     = GetPort();
   nStopBits = GetStopBitsIndex(GetStopBits());
   fRTSCTS   = GetRTSCTS();
   fXONXOFF  = GetXONXOFF();
   fDTRDSR   = GetDTRDSR();

        pApp->WriteProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_BAUDRATE],
                                                nBaudRate);
        pApp->WriteProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_DATABITS],
                                                nDataBits);
        pApp->WriteProfileInt(COMM_PORT_SECTION,    strPortItem[ITEM_PARITY],
nParity);
        pApp->WriteProfileInt(COMM_PORT_SECTION,      strPortItem[ITEM_PORT],
nPort);
        pApp->WriteProfileInt(COMM_PORT_SECTION,   strPortItem[ITEM_RTSCTS],
fRTSCTS);
```

```
        pApp->WriteProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_STOPBITS],
                                                nStopBits);
        pApp->WriteProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_XONXOFF],
                                                fXONXOFF);
        pApp->WriteProfileInt(COMM_PORT_SECTION,  strPortItem[ITEM_DTRDSR],
fDTRDSR);
}
```

직렬 포트 클래스 소스코드

다음은 직렬 포트 클래스의 구현 부분인 CommPort.h 와 CommPort.cpp 파일이다.

```
/*
*       File:                   CommPort.h
*
*       Contains:
*
*       Project:        Communication module for Intellegent User Interface.
*
*       Copyright:              (c) 1997 by Kwon, Jae-Rock. All right Reserved.
*
*       Written by:             Kwon, Jae-Rock
*
*       Change History(most recent first) :
*
*       <1> 97.03
*
*/

#ifndef _COMMPORT_H_
#define _COMMPORT_H_

//
#define MAXBUF          80

// define REGISTRY KEY
#define COMM_REGISTRY_KEY   "Chungnyun Soft"
#define COMM_PORT_SECTION   "CommPort Settings"

// define communication event message
#define WM_COMMNOTIFY32     WM_USER+1
```

```
#define CN_EVENT32          0x04

// define ASCII
#define ASCII_DLE                   0x10
#define ASCII_ETX                   0x03
#define ASCII_CAN                   0x18
#define ASCII_EOT                   0x04

#define ASCII_BEL       0x07
#define ASCII_BS        0x08
#define ASCII_TAB       0x09
#define ASCII_LF                    0x0A
#define ASCII_CR                    0x0D
#define ASCII_XON                   0x11
#define ASCII_XOFF                  0x13
#define ASCII_ESC       0x1B

////////////////////////////////////////////////////////////////////////////
// RX QUEUE SIZE
#define RXQUEUESIZE       4096 // Receive Queue Size
#define TXQUEUESIZE       4096 // Transmit Queue Size

#define COMM_MASK         EV_RXCHAR

////////////////////////////////////////////////////////////////////////////
// Flow control flags
#define FC_DTRDSR       0x01
#define FC_RTSCTS       0x02
#define FC_XONXOFF      0x04

////////////////////////////////////////////////////////////////////////////
#define MAXCOMMPORTS          8

typedef enum {
   COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8
} COMM_PORT ;

typedef enum {
   PORT_NO_ERROR, PORT_OPEN_ERROR, PORT_HANSHAKE_LINE_IN_USE
} COMM_ERROR;

extern char *pstrCommPortName[];

////////////////////////////////////////////////////////////////////////////
// Communication Port class ver 0.1

extern DWORD dwBaudRateTable[];
extern BYTE bParityTable[];
extern BYTE bStopBitsTable[];
extern BYTE bDataBitsTable[];
extern COMM_PORT portTable[];
```

```
class CPortSetting
{
// Attributes
protected:
   DWORD      m_dwBaudRate;
          BYTE        m_bDataBits;
          BYTE        m_bParity;
          COMM_PORT   m_port;
          BOOL        m_fRTSCTS;
          BYTE              m_bStopBits;
          BOOL    m_fXONXOFF;
          BOOL        m_fDTRDSR;

// Operations
public:
   DWORD      GetBaudRate() { return m_dwBaudRate; }
   void       SetBaudRate(DWORD dwBaudRate) { m_dwBaudRate = dwBaudRate;
}
   BYTE       GetDataBits() { return m_bDataBits; }
   void       SetDataBits(BYTE bDataBits) { m_bDataBits = bDataBits; }
   BYTE       GetParity() { return m_bParity; }
   void       SetParity(BYTE bParity) { m_bParity = bParity; }
          COMM_PORT   GetPort() { return m_port; }
   void       SetPort(COMM_PORT port) { m_port = port; }
   BOOL       GetRTSCTS() { return m_fRTSCTS; }
   void       SetRTSCTS(BOOL fRTSCTS) { m_fRTSCTS = fRTSCTS; }
   BYTE       GetStopBits() { return m_bStopBits; }
   void       SetStopBits(BYTE bStopBits) { m_bStopBits = bStopBits; }
   BOOL       GetXONXOFF() { return m_fXONXOFF; }
   void       SetXONXOFF(BOOL fXONXOFF) { m_fXONXOFF = fXONXOFF; }
   BOOL       GetDTRDSR() { return m_fDTRDSR; }
   void       SetDTRDSR(BOOL fDTRDSR) { m_fDTRDSR = fDTRDSR; }

// Implementation
public:
   int        GetBaudRateIndex(DWORD dwBaudRate);
   int        GetParityIndex(BYTE bParity);
   int        GetStopBitsIndex(BYTE bStopBits);
   int        GetDataBitsIndex(BYTE bDataBits);
};

class CCommPort : public CObject
{
//protected: // create from serialization only
public:
          CCommPort();
          DECLARE_DYNCREATE(CCommPort)

// Attributes
protected:
```

```
        HANDLE          m_hCommPort;
        COMM_PORT       m_portID;
        BOOL            m_fOpen;
        DCB             m_dcb;
        OVERLAPPED      m_osWrite, m_osRead;
        CWinThread*     m_pMonitorThread;
  HANDLE      m_hPostEvent;   // comm event notification event
        HANDLE          m_hEventMonitorThreadKilled;
        HWND            m_hNotifyWnd;                   //     COMM_MSG
notify window
  CPortSetting m_portSetting;
  COMM_ERROR  m_error;

// Operations
public:
        char*           GetPortName(COMM_PORT portName)
                        { return pstrCommPortName[portName]; }

  COMM_PORT   GetPortID() { return m_portID; }
        void            SetPortID(COMM_PORT portID) { m_portID = portID; }
        BOOL            GetOpenFlag() { return m_fOpen; }
        void            SetOpenFlag(BOOL fPortOpen) { m_fOpen = fPortOpen; }
        HANDLE          GetPortHandle() { return m_hCommPort; }
        void            SetPortHandle(HANDLE  hCommPort)  {  m_hCommPort  =
hCommPort; }
        HANDLE          GetReadOSEvent() { return m_osRead.hEvent; }
        void            SetReadOSEvent(HANDLE hReadOSEvent)
                        { m_osRead.hEvent = hReadOSEvent; }
        OVERLAPPED      GetReadOS() { return m_osRead; }
        HANDLE          GetWriteOSEvent() { return m_osWrite.hEvent; }
        void            SetWriteOSEvent(HANDLE hWriteOSEvent)
                        { m_osWrite.hEvent = hWriteOSEvent; }
        OVERLAPPED      GetWriteOS() { return m_osWrite; }

        CWinThread* GetMonitorThread() { return m_pMonitorThread; }
        void            SetMonitorThread(CWinThread* pMonitorThread)
                        { m_pMonitorThread = pMonitorThread; }
        HANDLE          GetPostEvent() { return m_hPostEvent; }
        void            SetPostEvent(HANDLE hPostEvent)
                        { m_hPostEvent = hPostEvent; }
        HANDLE          GetMonitorThreadKilledEvent()
                        { return m_hEventMonitorThreadKilled; }
        void            SetMonitorThreadKilledEvent(HANDLE
hEventMonitorThreadKilled)
          { m_hEventMonitorThreadKilled = hEventMonitorThreadKilled; }

        HWND            GetNotifyWindow() { return m_hNotifyWnd; }
        void            SetNotifyWindow(HWND hNotifyWnd) { m_hNotifyWnd =
hNotifyWnd; }
  CPortSetting GetPortSetting() { return m_portSetting; }
  void       SetPortSetting(CPortSetting portSetting)
```

```
                                        { m_portSetting = portSetting; }
  COMM_ERROR  GetError() { return m_error; }
  void        SetError(COMM_ERROR error) { m_error = error; }

   // Comm port setting
   DWORD      GetBaudRate() { return m_portSetting.GetBaudRate(); }
   void       SetBaudRate(DWORD dwBaudRate)
                                        { m_portSetting.SetBaudRate(dwBaudRate); }
   BYTE       GetDataBits() { return m_portSetting.GetDataBits(); }
   void       SetDataBits(BYTE bDataBits)
                                        { m_portSetting.SetDataBits(bDataBits); }
   BYTE       GetParity() { return m_portSetting.GetParity(); }
   void       SetParity(BYTE bParity) { m_portSetting.SetParity(bParity); }
          COMM_PORT   GetPort() { return m_portSetting.GetPort(); }
   void       SetPort(COMM_PORT port)
                                        { m_portSetting.SetPort(port); SetPortID(port); }
   BOOL       GetRTSCTS() { return m_portSetting.GetRTSCTS(); }
   void       SetRTSCTS(BOOL fRTSCTS) { m_portSetting.SetRTSCTS(fRTSCTS); }
   BYTE       GetStopBits() { return m_portSetting.GetStopBits(); }
   void       SetStopBits(BYTE bStopBits)
                                        { m_portSetting.SetStopBits(bStopBits); }
   BOOL       GetXONXOFF() { return m_portSetting.GetXONXOFF(); }
                  void                                  SetXONXOFF(BOOL      fXONXOFF)
{ m_portSetting.SetXONXOFF(fXONXOFF); }
   BOOL       GetDTRDSR() { return m_portSetting.GetDTRDSR(); }
   void       SetDTRDSR(BOOL fDTRDSR) { m_portSetting.SetDTRDSR(fDTRDSR); }

   int        GetBaudRateIndex(DWORD dwBaudRate)
                                        {                                                 return
m_portSetting.GetBaudRateIndex(dwBaudRate); }
   int        GetParityIndex(BYTE bParity)
                                        { return m_portSetting.GetParityIndex(bParity); }
   int        GetStopBitsIndex(BYTE bStopBits)
                                        {                                                 return
m_portSetting.GetStopBitsIndex(bStopBits); }
   int        GetDataBitsIndex(BYTE bDataBits)
                                        {                                                 return
m_portSetting.GetDataBitsIndex(bDataBits); }

// Overrides
          // ClassWizard generated virtual function overrides
          //{{AFX_VIRTUAL(CCommPort)
          //}}AFX_VIRTUAL

// Implementation
public:
          void              StartMonitorThread();
          DWORD             Read(LPTSTR lpszBlock, int nMaxLength);
          DWORD             Write(BYTE bByte);
          DWORD             Write(LPCTSTR lpszBlock, int nMaxLength);
  DWORD       Write(LPCTSTR lpszBlock);
```

```
        void            Clear();
        BOOL            Open();
        void            Close();
        void            EnableMonitorThread();
        void            DisableMonitorThread();
        BOOL            SetDCB(DWORD BaudRate = CBR_14400,// current baud
rate
                                BYTE ByteSize = 8,
        // number of bits/byte, 4-8
                                BYTE Parity = NOPARITY,
        // 0-4=no,odd,even,mark,space
                                BYTE StopBits = 0,
        // 0,1,2 = 1, 1.5, 2
                                int nFlag = FC_RTSCTS);         //   Flow
control flag
  BOOL      IsOpen();
  void      DispError();
  void      ReadProfile();
  void      WriteProfile();
  COMM_ERROR  DTR(BOOL fSetting);

        virtual ~CCommPort();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

};

#endif
//////////////////////////////////////////////////////////////////////////////
```

```
/*
*       File:                   CommPort.cpp
*
*       Contains:
*
*       Project:        Communication module for Intellegent User Interface.
*
*       Copyright:              (c) 1997 by Kwon, Jae-Rock. All right Reserved.
*
*       Written by:             Kwon, Jae-Rock
*
*       Change History(most recent first) :
*
```

```
*        <1> 97.03
*
*/

#include "stdafx.h"
#include "CommPort.h"
#include "MonitorThread.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNCREATE(CCommPort, CObject)

#define MAX_BAUDRATE_TBL    15
#define MAX_PARITY_TBL      5
#define MAX_STOPBITS_TBL    3
#define MAX_DATABITS_TBL    5

////////////////////////////////////////////////////////////////////////////
//
char *pstrCommPortName[] = {
   "COM1", "COM2", "COM3", "COM4", "COM5", "COM6", "COM7", "COM8"
};

DWORD dwBaudRateTable[] = {
  CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400, CBR_4800,
          CBR_9600, CBR_14400, CBR_19200, CBR_38400, CBR_56000,
          CBR_57600, CBR_115200, CBR_128000, CBR_256000
};

BYTE bParityTable[] = {
   NOPARITY, ODDPARITY, EVENPARITY, MARKPARITY, SPACEPARITY
};

BYTE bStopBitsTable[] = {
   ONESTOPBIT, ONE5STOPBITS, TWOSTOPBITS
};

BYTE bDataBitsTable[] = {
   4, 5, 6, 7, 8
};

COMM_PORT portTable[] = {
   COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8
};

CString strError[] = {
   "", "Can not open the serial communication port"
```

```
};

enum {
   ITEM_BAUDRATE,
   ITEM_DATABITS,
   ITEM_PARITY,
   ITEM_PORT,
   ITEM_RTSCTS,
   ITEM_STOPBITS,
   ITEM_XONXOFF,
   ITEM_DTRDSR
};

CString strPortItem[] = {
   "BaudRate",
          "DataBits",
          "Parity",
          "Port",
          "RTSCTS",
          "StopBits",
          "XONXOFF",
          "DTRDSR"
};

////////////////////////////////////////////////////////////////////////////
// default setting for comm port
#define DEFAULT_BAUDRATE    CBR_38400
#define DEFAULT_DATABITS    8
#define DEFAULT_PARITY      NOPARITY
#define DEFAULT_PORT        COM1
#define DEFAULT_RTSCTS      TRUE
#define DEFAULT_STOPBITS    ONESTOPBIT
#define DEFAULT_XONXOFF     TRUE
#define DEFAULT_DTRDSR      TRUE

////////////////////////////////////////////////////////////////////////////
// Table convertor
int CPortSetting::GetBaudRateIndex(DWORD dwBaudRate)
{
   for(int i=0; i < MAX_BAUDRATE_TBL; i++)
      if(dwBaudRate == dwBaudRateTable[i])
         return i;
   return -1;
}

int CPortSetting::GetParityIndex(BYTE bParity)
{
   for(int i=0; i < MAX_PARITY_TBL; i++)
      if(bParity == bParityTable[i])
         return i;
   return -1;
```

```
}

int CPortSetting::GetStopBitsIndex(BYTE bStopBits)
{
   for(int i=0; i < MAX_STOPBITS_TBL; i++)
      if(bStopBits == bStopBitsTable[i])
         return i;
   return -1;
}

int CPortSetting::GetDataBitsIndex(BYTE bDataBits)
{
   for(int i=0; i < MAX_DATABITS_TBL; i++)
      if(bDataBits == bDataBitsTable[i])
         return i;
   return -1;
}

////////////////////////////////////////////////////////////////////////////
// CCommPort Constructor/Destructor

CCommPort::CCommPort()
{
         // prevent for getting comm port handle before port open
         SetOpenFlag(FALSE);

         // default setting for comm port
  m_portSetting.SetBaudRate(DEFAULT_BAUDRATE);
         m_portSetting.SetDataBits(DEFAULT_DATABITS);
         m_portSetting.SetParity(DEFAULT_PARITY);
         m_portSetting.SetPort(DEFAULT_PORT);
         m_portSetting.SetRTSCTS(DEFAULT_RTSCTS);
         m_portSetting.SetStopBits(DEFAULT_STOPBITS);
         m_portSetting.SetXONXOFF(DEFAULT_XONXOFF);
         m_portSetting.SetDTRDSR(DEFAULT_DTRDSR);
}

CCommPort::~CCommPort()
{
}

////////////////////////////////////////////////////////////////////////////
// Operation

/*CString         CCommPort::GetName(COMM_PORT commPortName)
{
         return strCommPortName[commPortName];
}*/


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

```
// CCommPort implementation

void CCommPort::StartMonitorThread()
{
	if(GetMonitorThread() == NULL)
	{ // if comm port don't have monitor thread
		CWinThread* pMonitorThread =
			AfxBeginThread(MonitorThreadProc, this,
				THREAD_PRIORITY_BELOW_NORMAL);
		SetMonitorThread(pMonitorThread);
	}

	TRACE("Monitor Thread started.\n");
}

/*
void CCommPort::KillMonitorThread()
{
	// MonitorThread is terminated by close comm port
	// we only set the MonitorThread variable to NULL;
	SetMonitorThread(NULL);
}
*/

//////////////////////////////////////////////////////////////////////////
// if return 0, some error occur.
DWORD CCommPort::Read(LPTSTR lpszBlock, int nMaxLength)
{
	// Read comm port by overlapped I/O
	BOOL fReadStat;
	COMSTAT ComStat;
	DWORD dwErrorFlags, dwError, dwLength;

	ClearCommError(GetPortHandle(), &dwErrorFlags, &ComStat);
	if(dwErrorFlags > 0) {
		TRACE("Comm port read error\n");
		return (DWORD)0;
	}
	dwLength = min((DWORD)nMaxLength, ComStat.cbInQue);
	if(dwLength > 0) {
		fReadStat = ReadFile(GetPortHandle(), lpszBlock, dwLength,
					&dwLength, &GetReadOS());
		if(!fReadStat) {
			if(GetLastError() == ERROR_IO_PENDING) {

				TRACE("IO Pending...\n");
				while(!GetOverlappedResult(GetPortHandle(),
					&GetReadOS(),	&dwLength,
TRUE))
				{
					dwError = GetLastError();
```

```
                                        if(dwError == ERROR_IO_INCOMPLETE)
                                        { // normal result if not finished
                                                continue;
                                        }
                                        else
                                        {
                                                // an error occurred, try to
recover
                                                TRACE("<CE-%u>\n", dwError);


        ClearCommError(GetPortHandle(), &dwErrorFlags, &ComStat);
                                                if(dwErrorFlags > 0)
                                                {
                                                        TRACE("<CE-%u>\n",
dwErrorFlags);
                                                }
                                                break;
                                        }
                                }
                        }
                        else
                        { // some other error occured
                                dwLength = 0;
                                ClearCommError(GetPortHandle(),
&dwErrorFlags, &ComStat);
                                if(dwErrorFlags > 0)
                                {
                                        TRACE("<CE-%u>\n", dwErrorFlags);
                                }
                        }
                }
        }
        return dwLength;
}

DWORD CCommPort::Write(BYTE bByte)
{
        BOOL fWriteStat;
        COMSTAT ComStat;
        DWORD dwBytesWritten, dwError, dwErrorFlags;

        fWriteStat = WriteFile(GetPortHandle(), (LPTSTR)&bByte, 1,
                                        &dwBytesWritten, &GetWriteOS());

        if(!fWriteStat)
        {
                if(GetLastError() == ERROR_IO_PENDING)
                {
                        while(!GetOverlappedResult(GetPortHandle(),
&GetWriteOS(),
```

```
                                        &dwBytesWritten, TRUE))
                        {
                                dwError = GetLastError();
                                if(dwError == ERROR_IO_INCOMPLETE)
                                { // normal result if not finished
                                        continue;
                                }
                                else
                                {
                                        // an error occurred, try to recover
                                        TRACE("<CE-%u>\n", dwError);

                                        ClearCommError(GetPortHandle(),
&dwErrorFlags, &ComStat);
                                        if(dwErrorFlags > 0)
                                        {
                                                TRACE("<CE-%u>\n",
dwErrorFlags);
                                        }
                                        break;
                                }
                        }
                }
                else
                { // some other error occured
                        dwBytesWritten = 0;
                        ClearCommError(GetPortHandle(),         &dwErrorFlags,
&ComStat);
                        if(dwErrorFlags > 0)
                        {
                                TRACE("<CE-%u>\n", dwErrorFlags);
                        }
                }
        }
        return dwBytesWritten;
}

DWORD CCommPort::Write(LPCTSTR lpszBlock, int nMaxLength)
{
        BOOL fWriteStat;
        COMSTAT ComStat;
        DWORD dwBytesWritten, dwError, dwErrorFlags;

        fWriteStat = WriteFile(GetPortHandle(), lpszBlock, nMaxLength,
                                        &dwBytesWritten, &GetWriteOS());

        if(!fWriteStat)
        {
                if(GetLastError() == ERROR_IO_PENDING)
                {
                        while(!GetOverlappedResult(GetPortHandle(),
```

```
&GetWriteOS(),
                                        &dwBytesWritten, TRUE))
                        {
                                dwError = GetLastError();
                                if(dwError == ERROR_IO_INCOMPLETE)
                                { // normal result if not finished
                                        continue;
                                }
                                else
                                {
                                        // an error occurred, try to recover
                                        TRACE("<CE-%u>\n", dwError);

                                        ClearCommError(GetPortHandle(),
&dwErrorFlags, &ComStat);
                                        if(dwErrorFlags > 0)
                                        {
                                                TRACE("<CE-%u>\n",
dwErrorFlags);
                                        }
                                        break;
                                }
                        }
                }
                else
                { // some other error occured
                        dwBytesWritten = 0;
                        ClearCommError(GetPortHandle(),         &dwErrorFlags,
&ComStat);
                        if(dwErrorFlags > 0)
                        {
                                TRACE("<CE-%u>\n", dwErrorFlags);
                        }
                }
        }
        return dwBytesWritten;
}

DWORD CCommPort::Write(LPCTSTR lpszBlock)
{
        BOOL fWriteStat;
        COMSTAT ComStat;
        DWORD dwBytesWritten, dwError, dwErrorFlags;
   int nMaxLength = strlen(lpszBlock);

   fWriteStat = WriteFile(GetPortHandle(), lpszBlock, nMaxLength,
                                        &dwBytesWritten, &GetWriteOS());

        if(!fWriteStat)
        {
                if(GetLastError() == ERROR_IO_PENDING)
```

```
		{
			while(!GetOverlappedResult(GetPortHandle(),
&GetWriteOS(),
					&dwBytesWritten, TRUE))
			{
				dwError = GetLastError();
				if(dwError == ERROR_IO_INCOMPLETE)
				{ // normal result if not finished
					continue;
				}
				else
				{
					// an error occurred, try to recover
					TRACE("<CE-%u>\n", dwError);

					ClearCommError(GetPortHandle(),
&dwErrorFlags, &ComStat);
					if(dwErrorFlags > 0)
					{
						TRACE("<CE-%u>\n",
dwErrorFlags);
					}
					break;
				}
			}
		}
		else
		{ // some other error occured
			dwBytesWritten = 0;
			ClearCommError(GetPortHandle(),		&dwErrorFlags,
&ComStat);
			if(dwErrorFlags > 0)
			{
				TRACE("<CE-%u>\n", dwErrorFlags);
			}
		}
	}
	return dwBytesWritten;
}

void CCommPort::Clear()
{
	PurgeComm(GetPortHandle(), PURGE_TXABORT | PURGE_RXABORT
					|	PURGE_TXCLEAR		|
PURGE_RXCLEAR);
}

BOOL CCommPort::SetDCB(DWORD dwBaudRate,				//
current baud rate
					BYTE bByteSize,
// number of bits/byte, 4-8
```

```
                                                    BYTE bParity,
// 0-4=no,odd,even,mark,space
                                                    BYTE bStopBits,
// 0,1,2 = 1, 1.5, 2
                                                    int nFlag)                  //
Flow control flag
{
        m_dcb.DCBlength = sizeof(DCB);
        GetCommState(GetPortHandle(), &m_dcb);

        m_dcb.BaudRate = dwBaudRate;
        m_dcb.ByteSize = bByteSize;
        m_dcb.Parity = bParity;
        m_dcb.StopBits = bStopBits;

        // set hardware control
        BYTE bSet = (BYTE)((nFlag & FC_DTRDSR) != 0);
        m_dcb.fOutxDsrFlow = bSet;
        m_dcb.fDtrControl = (bSet) ? DTR_CONTROL_HANDSHAKE :
DTR_CONTROL_ENABLE;

        bSet = (BYTE)((nFlag & FC_RTSCTS) != 0);
        m_dcb.fOutxCtsFlow = bSet;
        m_dcb.fRtsControl = (bSet) ? RTS_CONTROL_HANDSHAKE :
RTS_CONTROL_ENABLE;

        m_dcb.fInX = m_dcb.fOutX = bSet;
        m_dcb.XonChar = ASCII_XON;
        m_dcb.XoffChar = ASCII_XOFF;
        m_dcb.XonLim = 100;
        m_dcb.XoffLim = 100;

        m_dcb.fBinary = TRUE;
        m_dcb.fParity = TRUE;

        TRACE("Setup DCB\n");
        return SetCommState(GetPortHandle(), &m_dcb);
}

BOOL CCommPort::Open()
{
        if(GetOpenFlag())
                return FALSE;

  ////////////////////////////////////
  //  Read port setting value
  COMM_PORT portID = GetPortSetting().GetPort();
        DWORD dwBaudRate = GetPortSetting().GetBaudRate();
        BYTE bByteSize = GetPortSetting().GetDataBits(); // number of bits/byte,
4-8
        BYTE bParity = GetPortSetting().GetParity();// 0-
```

```
4=no,odd,even,mark,space
        BYTE bStopBits = GetPortSetting().GetStopBits(); // 0,1,2 = 1, 1.5, 2

  int nFlag;
  nFlag = GetPortSetting().GetRTSCTS() ? FC_RTSCTS : 0;
  nFlag |= GetPortSetting().GetDTRDSR() ? FC_DTRDSR : 0;
  nFlag |= GetPortSetting().GetXONXOFF() ? FC_XONXOFF : 0;

  ////////////////////////////////////
        //      Open Comm Port
        HANDLE hCommPort = CreateFile(GetPortName(portID),
                                                GENERIC_READ                    |
GENERIC_WRITE,
                                                0,      // Exclusive access
                                                NULL, // no security attrs
                                                OPEN_EXISTING,
                                                FILE_ATTRIBUTE_NORMAL           |
FILE_FLAG_OVERLAPPED,
                                                NULL);
        if(hCommPort == (HANDLE)-1)
   {
      SetError(PORT_OPEN_ERROR);
                return FALSE;
   }

        SetPortHandle(hCommPort);
        SetOpenFlag(TRUE);

        ////////////////////////////////////////////
        SetupComm(GetPortHandle(), RXQUEUESIZE, TXQUEUESIZE);

        ////////////////////////////////////////////
        // set up for overlapped non-blocking I/O
        COMMTIMEOUTS CommTimeOuts;
  CommTimeOuts.ReadIntervalTimeout = MAXDWORD;
  CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
  CommTimeOuts.ReadTotalTimeoutConstant = 0;
  CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
  CommTimeOuts.WriteTotalTimeoutConstant = 5000;
        SetCommTimeouts(GetPortHandle(), &CommTimeOuts);

        /////////////////////////////////////////////
        // set up connection
        if(SetDCB(dwBaudRate, bByteSize, bParity, bStopBits, nFlag)) {
                SetMonitorThread(NULL);                 //      Monitor      thread
initialize
                StartMonitorThread();           // Monitor Thread Start
                // assert DTR
                DTR(TRUE);
        }
        else {
```

```
                SetOpenFlag(FALSE);
                CloseHandle(GetPortHandle());
                return FALSE;
        }

        ////////////////////////////////////////
        // Create Events
        HANDLE hPostEvent = CreateEvent(NULL,           // no security

        TRUE,   // manual reset

        TRUE,   // intial event is set

        NULL);  // no name
        HANDLE hEventMonitorThreadKilled = CreateEvent(NULL,

        TRUE,   // manual reset

        FALSE,  // intial event is reset

        NULL);  // no name
        SetPostEvent(hPostEvent);
        SetMonitorThreadKilledEvent(hEventMonitorThreadKilled);

        HANDLE hReadOSEvent = CreateEvent(NULL,     // no security
                                                                TRUE,
        // manual reset
                                                                FALSE,
        // initial event is reset
                                                                NULL);
        // no name
        HANDLE hWriteOSEvent = CreateEvent(NULL,// no security
                                                                TRUE,
        // manual reset
                                                                FALSE,
        // initial event is reset
                                                                NULL);
        // no name
        SetReadOSEvent(hReadOSEvent);
        SetWriteOSEvent(hWriteOSEvent);
        GetReadOS().Offset = GetWriteOS().Offset = 0;

        // Clear comm buffer
        PurgeComm(GetPortHandle(), PURGE_TXABORT | PURGE_RXABORT
                                                        |       PURGE_TXCLEAR          |
PURGE_RXCLEAR);

        TRACE("Open Comm Port\n");
        return TRUE;
}
```

```
void CCommPort::Close()
{
	if(GetOpenFlag() == FALSE)
		return;

	SetOpenFlag(FALSE); // monitor thread killed by exiting the proc
	SetCommMask(GetPortHandle(), 0);

	DWORD dwExitCode;

	if(GetExitCodeThread(GetMonitorThread()->m_hThread,	&dwExitCode)
&&
		dwExitCode == STILL_ACTIVE)
	{	// Wait for monitor thread killed
		WaitForSingleObject(GetMonitorThreadKilledEvent(), INFINITE);
	}
	SetMonitorThread(NULL);

	// drop DTR
	DTR(FALSE);
	// flush buffer
	PurgeComm(GetPortHandle(), PURGE_TXABORT | PURGE_RXABORT
						|	PURGE_TXCLEAR		|
PURGE_RXCLEAR);

	CloseHandle(GetPortHandle());

	CloseHandle(GetPostEvent());
	CloseHandle(GetMonitorThreadKilledEvent());

	CloseHandle(GetReadOSEvent());
	CloseHandle(GetWriteOSEvent());
	TRACE("Close Comm Port\n");
}

BOOL CCommPort::IsOpen()
{
   return GetOpenFlag();
}

void CCommPort::EnableMonitorThread()
{
	SetCommMask(GetPortHandle(), COMM_MASK);
}

void CCommPort::DisableMonitorThread()
{
	SetCommMask(GetPortHandle(), 0);
}

COMM_ERROR CCommPort::DTR(BOOL fSetting)
```

```
{
        if(m_dcb.fDtrControl == DTR_CONTROL_HANDSHAKE)
      return PORT_HANSHAKE_LINE_IN_USE;
   EscapeCommFunction(GetPortHandle(), (fSetting) ? SETDTR : CLRDTR);
   return PORT_NO_ERROR;
}

void CCommPort::DispError()
{
   COMM_ERROR error;

   error = GetError();
   if(error == PORT_NO_ERROR)
      return;
   AfxMessageBox(strError[error]);
}

void CCommPort::ReadProfile()
{
   int  nBaudRate, nDataBits, nParity, nPort, nStopBits;
   BOOL fRTSCTS, fXONXOFF, fDTRDSR;

   CWinApp* pApp = AfxGetApp();

        nBaudRate = pApp->GetProfileInt(COMM_PORT_SECTION,
                                        strPortItem[ITEM_BAUDRATE],

        GetPortSetting().GetBaudRateIndex(DEFAULT_BAUDRATE));

        nDataBits = pApp->GetProfileInt(COMM_PORT_SECTION,
                                        strPortItem[ITEM_DATABITS],

        GetPortSetting().GetDataBitsIndex(DEFAULT_DATABITS));
        nParity         =       pApp->GetProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_PARITY],

        GetPortSetting().GetParityIndex(DEFAULT_PARITY));
        nPort           =       pApp->GetProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_PORT],
                                        DEFAULT_PORT);
        fRTSCTS         =       pApp->GetProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_RTSCTS],
                                        DEFAULT_RTSCTS);
        nStopBits = pApp->GetProfileInt(COMM_PORT_SECTION,
                                        strPortItem[ITEM_STOPBITS],

        GetPortSetting().GetStopBitsIndex(DEFAULT_STOPBITS));
        fXONXOFF        =       pApp->GetProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_XONXOFF],
                                        DEFAULT_XONXOFF);
        fDTRDSR         =       pApp->GetProfileInt(COMM_PORT_SECTION,
```

```
strPortItem[ITEM_DTRDSR],
                                                DEFAULT_DTRDSR);

   SetBaudRate(dwBaudRateTable[nBaudRate]);
            SetDataBits(bDataBitsTable[nDataBits]);
            SetParity(bParityTable[nParity]);
            SetPort(portTable[nPort]);
            SetRTSCTS(fRTSCTS);
            SetStopBits(bStopBitsTable[nStopBits]);
            SetXONXOFF(fXONXOFF);
            SetDTRDSR(fDTRDSR);
}

void CCommPort::WriteProfile()
{
    int  nBaudRate, nDataBits, nParity, nPort, nStopBits;
    BOOL fRTSCTS, fXONXOFF, fDTRDSR;
    CWinApp* pApp = AfxGetApp();

    nBaudRate = GetBaudRateIndex(GetBaudRate());
    nDataBits = GetDataBitsIndex(GetDataBits());
    nParity   = GetParityIndex(GetParity());
    nPort     = GetPort();
    nStopBits = GetStopBitsIndex(GetStopBits());
    fRTSCTS   = GetRTSCTS();
    fXONXOFF  = GetXONXOFF();
    fDTRDSR   = GetDTRDSR();

            pApp->WriteProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_BAUDRATE],
                                                        nBaudRate);
            pApp->WriteProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_DATABITS],
                                                        nDataBits);
            pApp->WriteProfileInt(COMM_PORT_SECTION,    strPortItem[ITEM_PARITY],
nParity);
            pApp->WriteProfileInt(COMM_PORT_SECTION,      strPortItem[ITEM_PORT],
nPort);
            pApp->WriteProfileInt(COMM_PORT_SECTION,   strPortItem[ITEM_RTSCTS],
fRTSCTS);
            pApp->WriteProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_STOPBITS],
                                                        nStopBits);
            pApp->WriteProfileInt(COMM_PORT_SECTION,
strPortItem[ITEM_XONXOFF],
                                                        fXONXOFF);
            pApp->WriteProfileInt(COMM_PORT_SECTION,  strPortItem[ITEM_DTRDSR],
fDTRDSR);
}

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

```
// CCommPort diagnostics

#ifdef _DEBUG
void CCommPort::AssertValid() const
{
	CObject::AssertValid();
}

void CCommPort::Dump(CDumpContext& dc) const
{
	CObject::Dump(dc);
}
#endif //_DEBUG

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

모뎀 클래스

모뎀이란 우리말로 옮기면 변복조기(MOulator-DEModulator) 정도가 되겠다. 현재의 전화망은 음성 데이터를 송수신할 때, 전화국에서 가정까지는 아날로그 방식이 사용되고 있다. 따라서 전화선을 통해 데이터 통신을 하려면, 일단 가정내 PC 에서 전화국까지 데이터를 보내기 위해 컴퓨터 상의 정보를 디지틀 형태에서 아날로그 형태로 변환해 주어야 한다. 이 과정을 변조라고 한다. 반대로 가정 내 PC 에서 데이터를 받으려면 전화선을 통해 전달된 아날로그 신호를 디지틀 신호로 변환해 주어야 하는데, 이 과정을 복조라고 한다. 이런 변복조 과정을 수행해 주는 장치를 모뎀이라고 한다.

모뎀과 인터페이스는 모뎀 하드웨어에서 제공하는 명령어들을 이용하면 되는데, 데이터 통신에 사용되는 모뎀 명령은 헤이즈 사의 AT 명령어 집합이 표준처럼 사용되고 있다. 세부적인 일부 명령들은 모뎀 칩 제작사 마다 약간씩 다를 수 있으나, 일반적으로 많이 사용되는 모뎀 명령어들은 모두 이 헤이즈 사의 AT 명령어 집합을 따른다. 헤이즈 사의 AT 명령어 집합은 이전 명령을 반복하는 명령인 'A/'을 제외하고는 모두 'AT'로 시작되며, 끝은 '\r', 즉 캐리지 리턴으로 끝났다.

앞서 작성한 직렬 통신 포트 클래스를 기반으로 모뎀 명령어를 처리하기 위한 모뎀 클래스를 작성해 보자. 모뎀 클래스 역시 CObject 를 상속받아 만든다. 클래스 사용자의 입장에서 보면, 직렬 통신 포트 클래스는 전혀 고려하지 않고, 단지 이 모뎀 클래스만을 다루면 제대로

동작하도록 해 보자.

```
class CModem : public CObject
{
//protected: // create from serialization only
public:
	CModem();
	DECLARE_DYNCREATE(CModem)

// Attributes
protected:
	CCommPort	m_commPort;
	DIAL_MODE	m_dialMode;
  DWORD	m_dwConnectTimeout;
  DWORD	m_dwResponseTimeout;
  CString		m_strInitCommand;

// Operations
public:
	CCommPort*	GetCommPort() { return &m_commPort; }
	DIAL_MODE	GetDialMode() { return m_dialMode; }
	void		SetDialMode(DIAL_MODE dialMode) { m_dialMode = dialMode; }
  DWORD	GetConnectTimeout() { return m_dwConnectTimeout; }
  void	SetConnectTimeout(DWORD dwConnectTimeout)
				{ m_dwConnectTimeout = dwConnectTimeout; }
  DWORD	GetResponseTimeout() { return m_dwResponseTimeout; }
  void	SetResponseTimeout(DWORD dwResponseTimeout)
				{ m_dwResponseTimeout = dwResponseTimeout; }
  CString	GetInitCommand() { return m_strInitCommand; }
  void	SetInitCommand(CString strInitCommand)
				{ m_strInitCommand = strInitCommand; }

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CModem)
	//}}AFX_VIRTUAL

// Implementation
public:
	BOOL	Open(DIAL_MODE dialMode=TONE_DIAL);
	void		Close();
	MODEM_ERROR Initialize();
	MODEM_ERROR Dial(CString strNo);
	MODEM_ERROR HangUp();
	MODEM_ERROR Answer();
```

```
  BOOL        SendCommand(CString strCommand);
  MODEM_ERROR WaitForConnection();
  MODEM_ERROR WaitForResponse();
  void        ReadLine(char* lpszBuf, int nBufSize);
  MODEM_CONNECT_ERROR ReadConnectError(char* pszBuf);

          virtual ~CModem();
#ifdef _DEBUG
          virtual void AssertValid() const;
          virtual void Dump(CDumpContext& dc) const;
#endif

};
```

몇가지 멤버 함수들의 반환값이 MODM_ERROR 는 다음과 같이 정의되어 있다.

```
typedef enum {
  MODEM_SUCCESS, MODEM_NO_RESPONSE,
  MODEM_NO_CONNECTION, MODEM_DISCONNECT_FAILED
} MODEM_ERROR;
```

모뎀 클래스에는 모뎀 클래스를 통한 통신 포트 초기화하기, 전화걸기, 전화받기, 전화끊기 등의 기능이 필요하다. 모뎀 클래스의 멤버 변수로는 다음과 같은 것들이 있다.

```
          CCommPort       m_commPort;
          DIAL_MODE       m_dialMode;
  DWORD       m_dwConnectTimeout;
  DWORD       m_dwResponseTimeout;
  CString                 m_strInitCommand;
```

m_commPort 는 직렬 통신 포트 클래스인 CCommPort 형의 변수이다. 직렬 통신 포트에 대한 조작은 이 변수를 통해 이루어진다.

m_dialMode 는 전화거는 방식을 말하는 것으로 흔히들 전자식, 기계식으로 구분한다. 요즘 쓰이는 버튼을 누르면 DTMF 톤이라고 부르는 '삐-'소리가 나는 방식이 바로 전자식이고, 옛날에 다이얼을 돌리면 '뚜르르륵' 소리가 나던 방식이 기계식이다. 일부 전화기에는 전자식과 기계식을 전환할 수 있는 장치가 있는데, 이것을 기계식으로 돌려 놓고 전화를 걸어보면,

요즘은 옛날 영화에서나 들을 수 있는 다이얼 신호음, '뚜르르륵'하는 소리가 들리면서 전화가 걸리는 것을 볼 수 있다. 모뎀의 전화걸기 명령은 'ATD'인데, 여기에 전자식인 경우에는 'T'(tone), 기계식인 경우에는 'P'(pulse)를 붙이고, 전화번호를 넣어주면 된다. 즉, 555-2555 국에 전화를 걸 때, 전자식인 경우에는 'ATDT 555-2555\r'를, 기계식인 경우에는 'ATDP 555-2555'를 입력하면 된다.

```
typedef enum {
   TONE_DIAL, PULSE_DIAL
} DIAL_MODE;
```

m_dwConnectTimeout 은 전화를 걸고 나서, 연결될 때까지의 최대 시간을 지정해 주기 위한 변수이다. 전화 걸기 명령으로 전화를 걸어서, 연결이 되면, 모뎀은 'CONNECT xxxx'라는 응답을 주게 되는데, (xxxx 는 접속 성공한 속도) 연결 성공 여부는 'CONNECT'로 시작하는 응답을 받았는지 여부로 알 수 있다. 예를 전화걸기 대화상자를 만들었다고 하자, 전화 걸기를 시도한 후, 몇초이내에 전화가 걸리지 않으면 재시도하도록 하거나 또는 걸기 취소를 하고 싶다면, 다음과 같이 하면 된다.

```
CModem modem;

DWORD dwTimeout;
MODEM_CONNECT_ERROR connectError;
char szBuf[MAXBUF];

modem.Open();
modem.SetConnectTimeout(60000); // wait until 60 seconds
modem.Dial("555-2555");                    // dial to 555-2555

dwTimeout = GetTickCount() + GetConnectTimeout();
while(GetTickCount() < dwTimeout)
{
   ReadLine(szBuf, MAXBUF);  // read input queue until line feed
   connectError = ReadConnectError(szBuf); // szBuf match error string ??
   if(connectError != CE_SUCCESS)
      return MODEM_NO_CONNECTION;
   if(strstr(szBuf, "CONNECT"))            // "CONNECT" is in szBuf ?
      return MODEM_SUCCESS;
}
return MODEM_NO_CONNECTION;
```

m_dwResponseTimeout 은 AT 명령어를 입력한 다음, "OK"응답이 올 때까지 기다리는 시간을 정하기 위한 변수이다.

m_dwConnectTimeout 과 m_dwResponseTimeout 값은 모뎀 명령 응답 처리에 꼭 필요한 변수이기 때문에 모뎀 클래스의 생성자 CModem::CModem() 부분에서 SetConnectTimeout()과 SetResponseTimeout()함수를 이용해 그 값을 정해 주어야 한다.

```
CModem::CModem()
{
   SetConnectTimeout(60000L);
   SetResponseTimeout(2000L);
}
```

m_strInitCommand 는 모뎀 초기화 명령을 담아 둘 변수이다. Initialize()멤버 함수에서 사용된다.

모뎀 열기

```
BOOL       Open(DIAL_MODE dialMode=TONE_DIAL);
```

모뎀을 연다는 개념은 사실, 적당한 것이 아니다. 하지만 하부에 직렬 포트를 고려하지 않고 개념적으로 생각한다면 모뎀은 사실, 입출력 버퍼를 가지는 통신 장치 정도로 생각해도 큰 무리는 없을 듯하다. 마치 파일 입출력처럼 모뎀 사용을 위해 모뎀을 열어, 명령어를 주고 응답을 받고 하는 작업을 수행하다가 사용이 끝나면 모뎀을 닫아 주면 된다. 물론 모뎀 열기 함수 내부에서는 직렬 통신 포트에 대한 열기 처리를 해 준다.

모뎀 클래스의 여러 멤버 변수들을 사용하려면 항상 가장 먼저 Open 함수를 호출해 주어야 한다.

```
BOOL CModem::Open(DIAL_MODE dialMode)
```

```
{
  GetCommPort()->ReadProfile(); // read setting value from registry
        if(GetCommPort()->Open())         // open serial comm. port
   {
     SetDialMode(dialMode);    // set dialing mode
           Initialize();                        // send initialize AT command
     return TRUE;
   }
   else
   {
      GetCommPort()->DispError();
      return FALSE;
   }
}
```

기본 인수로는 전자식 전화걸기 상태가 지정된다.

모뎀 닫기

```
        void            Close();
```

모뎀 클래스의 사용이 끝나면 모뎀을 닫는다. 이 함수는 직렬 포트에 대한 사용을 실제로 끝내기 때문에 전화가 걸려있다면, 전화가 끊긴다. 이 함수 호출후라면 다른 응용프로그램에서도 모뎀 자원을 이제 사용할 수 있게 된다.

```
void CModem::Close()
{
        GetCommPort()->Close();
}
```

OK 응답 기다리기

```
MODEM_ERROR WaitForResponse();
```

모뎀에 명령을 내린 후, 모뎀으로부터 "OK"응답이 나올 때까지 기다린다. 기다리는 시간은 SetResponseTimeout()함수를 통해 지정한다.

```
MODEM_ERROR CModem::WaitForResponse()
{
   DWORD dwTimeout;
   char szBuf[MAXBUF];

   dwTimeout = GetTickCount() + GetResponseTimeout();
   while(GetTickCount() < dwTimeout)
   {
      ReadLine(szBuf, MAXBUF);
      if(!strncmp(szBuf, "OK", 2))
         return MODEM_SUCCESS;
   }
   return MODEM_NO_RESPONSE;
}
```

CONNECT 응답 기다리기

```
MODEM_ERROR WaitForConnection();
```

전화걸기 후 "CONNECT"로 시작되는 응답이 들어올 때까지 기다린다. 기다리는 시간은 SetConnectTimeout 함수를 통해 지정한다.

```
MODEM_ERROR CModem::WaitForConnection()
{
   DWORD dwTimeout;
   MODEM_CONNECT_ERROR connectError;
   char szBuf[MAXBUF];

   dwTimeout = GetTickCount() + GetConnectTimeout();
```

```
    while(GetTickCount() < dwTimeout)
    {
        ReadLine(szBuf, MAXBUF);
        connectError = ReadConnectError(szBuf);
        if(connectError != CE_SUCCESS)
            return MODEM_NO_CONNECTION;
        if(strstr(szBuf, "CONNECT"))
            return MODEM_SUCCESS;
    }
    return MODEM_NO_CONNECTION;
}
```

직렬 포트 입력 버퍼에서 한 줄 읽어오기

```
void        ReadLine(char* lpszBuf, int nBufSize);
```

직렬 통신 포트에 들어 온 데이터를 읽는다. 데이터를 읽다가 LF(Line Feed: ASCII 0x0A)를 만나면 읽기를 중단한다. 모뎀 명령어에 대한 응답을 읽는 함수로 쓰인다.

```
void CModem::ReadLine(char* lpszBuf, int nBufSize)
{
    char cChar;

    while(TRUE)
    {
        if(GetCommPort()->Read(&cChar, 1) != 1)
            break;
        *lpszBuf++ = (cChar);
        if(--nBufSize <= 1)
            break;
        if(cChar == ASCII_LF)
            break;
    }
    *lpszBuf = NULL;
}
```

모뎀에 명령어 보내기

```
BOOL        SendCommand(CString strCommand);
```

모뎀 명령어를 보낸다. strCommand 로 들어오는 명령어 문자열에 CR(Carriage Return: ASCII 0x0D)을 붙여서 직렬 통신 포트에 쓰면 명령이 입력된 것이다. 명령을 입력한 다음, "OK"응답이 올 때까지 기다린다.

```
BOOL CModem::SendCommand(CString strCommand)
{
   GetCommPort()->Write(strCommand);
   GetCommPort()->Write("\r");

   return WaitForResponse();
}
```

초기화 작업

```
        MODEM_ERROR Initialize();
```

이 함수를 통해 모뎀 초기화 명령을 수행한다. 초기화 명령을 위한 문자열은 m_strInitCommand 변수에 지정하면 Initialize() 함수에서 명령이 수행된다.

```
MODEM_ERROR CModem::Initialize()
{
            SendCommand(GetInitCommand());

   return WaitForResponse();
```

```
}
```

전화 걸기

```
        MODEM_ERROR Dial(CString strNo);
```

strNo 인 전화번호로 전화를 건다. 전화번호 문자열에 들어가는 공백이나 '-', '(', '('등의 문자들은 무시되기 때문에 strNo 는 "(02) 555-2555"같은 식으로 입력을 하면 된다. 전자식과 기계식 지정은 SetDialMode() 함수를 통해 한다. 전화걸기 명령어를 보낸 다음에는 "CONNECT"로 시작하는 응답이 들어올 때까지 기다린다.

```
MODEM_ERROR CModem::Dial(CString strNo)
{
        GetCommPort()->Write("ATD");
   GetCommPort()->Write((GetDialMode() == TONE_DIAL) ? "T" : "P");
        GetCommPort()->Write(strNo);
        GetCommPort()->Write("\r");

        return WaitForConnection();
}
```

전화 끊기

```
        MODEM_ERROR HangUp();
```

모뎀에는 두가지 상태가 있다. 하나는 보통 명령이 입력되는 명령 상태이고, 다른 하나는

연결이 성립되어서, 입력되는 데이터가 상대방으로 그대로 전달되는 온라인 상태이다. 즉, 통신 에뮬레이터를 실행시킨 다음, 화면에 "AT"라고 입력하면 "OK"라는 응답이 나오는 그런 상태를 명령 상태라고 한다. PC 통신 서비스 업체에 전화를 거는 경우를 생각해 보자. ATD 명령으로 전화를 걸어서 "삐--삐리리릭--"소리가 나고 "CONNECT xxx"응답이 뜨면서 연결이 되는데, 연결이 되고 나면 이제 AT 로 시작하는 명령에 대해 더 이상 응답하지 않는다. 입력되는 데이터는 모두 전화선 너머의 연결된 상대방으로 전달되는 것이다. 즉, 연결이 되어 사용자 번호와 비밀번호를 입력하는 상태가 되면, 입력된 문자는 더 이상 명령으로 해석되지 않고, 곧바로 상대방으로 전달된다. 이런 상태를 온라인 상태라고 한다. 그렇다면, 한가지 의문이 들만하다. 명령이 입력되지 않는데, 연결된 이후에 어떻게 전화를 끊을 수 있을까? 물론 전화가 끊기면 다시 명령 상태가 된다. 온라인 상태에서 전화를 끊지 않은 상태에서 명령어 상태로 나올 방법이 제공되어야 하겠는데, 이 명령어가 바로 "+++"이다. 일부 PC 통신 서비스 업체에 접속해 보면, 전화를 끊을 때, "+++ATH"를 입력하라는 말이 나오는데, 직접 키보드로 입력해서는 전화가 끊기지 않는다. 왜냐하면 입력한 '+'가 각각 그냥 데이터로 인식되기 때문이다. 일정간격으로 '+'가 세 번 입력되면, 온라인 명령 상태로 빠져 나오게 되는데, 이 상태에서 전화끊기 명령인 "ATH"를 주면 전화가 끊긴다. 이 방법 외에 하드웨어 적으로 전화를 끊을 수도 있는데, 방법은 DTR 을 일정시간 동안 떨어 뜨리는 것이다. DTR 을 1 초 정도 떨어 뜨리면, 대부분 전화가 끊긴다고 한다. 1 초 후에 다시 원래대로 DTR 을 올려 주면 된다. 아래 구현 부분에서는 일단 하드웨어적으로 끊기를 시도해 보고 나서, 결과에 상관없이 다시 한번 소프트웨어 적으로 끊기를 시도하도록 해 놓았다.

```
MODEM_ERROR CModem::HangUp()
{
        DWORD dwTimeout;

  GetCommPort()->DTR(FALSE);
  dwTimeout = GetTickCount()+1000L;
  while(GetTickCount() < dwTimeout)
      ;
        GetCommPort()->DTR(TRUE);

  GetCommPort()->Write("+++");
  WaitForResponse();
        GetCommPort()->Write("ATH0\r");
  if(WaitForResponse() == MODEM_SUCCESS)
     return MODEM_SUCCESS;

        return MODEM_DISCONNECT_FAILED;
}
```

전화 받기

```
	MODEM_ERROR Answer();
```

전화 벨이 몇번 울린 후 모뎀이 자동으로 전화를 받도록 하려면 모뎀의 S0 레지스터를 조작해 주면 된다. S0 레지스터에 전화 벨 수를 지정해 주면, 지정된 횟수만큼 전화벨이 울리고 난 뒤, 자동으로 모뎀이 전화를 받는다. 예를 들어, 누군가와 1:1 통신을 하고 싶다면, 에뮬레이터에서 "ATS0=2"이라고 입력한 다음, 기다리면, 전화벨이 두 번 울리고 나면, 모뎀은 자동으로 전화를 받는다.

S0 레지스터에 벨 수를 지정하지 않으면 계속 화면에 'RING'이란 문자가 출력되면서 벨이 울리는 것을 볼 수 있는데, 이 상태에서 "ATA"라는 전화받기 명령을 입력하면 전화가 연결된다.

```
MODEM_ERROR CModem::Answer()
{
	GetCommPort()->Write("ATA");

	return WaitForConnection();
}
```

응답오류 검사

```
MODEM_CONNECT_ERROR ReadConnectError(char* pszBuf);
```

직렬 통신 포트에서 읽은 데이터가 오류를 뜻하는 문자열 중 하나인지 검사한다. 일치하는

오류문자열이 있으면 해당 오류의 인덱스 값을 준다.

```
MODEM_CONNECT_ERROR CModem::ReadConnectError(char* lpszBuf)
{
   for(int i=0; szConnectError[i]; i++)
   {
      if(strstr(lpszBuf, szConnectError[i]))
         return (MODEM_CONNECT_ERROR)i; // connection error
   }
   return CE_SUCCESS;
}
```

모뎀 클래스 구현

```
/*
*        File:                  Modem.h
*
*        Contains:
*
*        Project:          Terminal Emulator for Win95
*
*        Copyright:             (c) 1997 Kwon, Jae-Rock. All right Reserved.
*
*        Written by:            Kwon, Jae-Rock
*
*        Change History(most recent first) :
*
*        <1> 97.03
*
*/
#ifndef _MODEM_H_
#define _MODEM_H_

#include "CommPort.h"

typedef enum {
  MODEM_SUCCESS, MODEM_NO_RESPONSE,
         MODEM_NO_CONNECTION, MODEM_DISCONNECT_FAILED
} MODEM_ERROR;

typedef enum {
  CE_SUCCESS, CE_NO_CARRIER, CE_ERROR,
```

```
        CE_NO_DIALTONE, CE_BUSY, CE_NO_ANSWER
} MODEM_CONNECT_ERROR;

typedef enum {
   MODEM_INIT, MODEM_ANSWER, MODEM_DIAL, MODEM_HANGUP
} MODEM_COMMAND;

typedef enum {
   TONE_DIAL, PULSE_DIAL
} DIAL_MODE;

/////////////////////////////////////////////////////////////////////////////
// Modem command class ver 0.1

class CModem : public CObject
{
//protected: // create from serialization only
public:
        CModem();
        DECLARE_DYNCREATE(CModem)

// Attributes
protected:
        CCommPort       m_commPort;
        DIAL_MODE       m_dialMode;
  DWORD      m_dwConnectTimeout;
  DWORD      m_dwResponseTimeout;
        CString         m_strInitCommand;

// Operations
public:
        CCommPort*      GetCommPort() { return &m_commPort; }
        DIAL_MODE       GetDialMode() { return m_dialMode; }
        void            SetDialMode(DIAL_MODE   dialMode)   {   m_dialMode   =
dialMode; }
  DWORD      GetConnectTimeout() { return m_dwConnectTimeout; }
  void       SetConnectTimeout(DWORD dwConnectTimeout)
                                  { m_dwConnectTimeout = dwConnectTimeout; }
  DWORD      GetResponseTimeout() { return m_dwResponseTimeout; }
  void       SetResponseTimeout(DWORD dwResponseTimeout)
                                  {          m_dwResponseTimeout          =
dwResponseTimeout; }
  CString    GetInitCommand() { return m_strInitCommand; }
  void       SetInitCommand(CString strInitCommand)
                                  { m_strInitCommand = strInitCommand; }

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CModem)
        //}}AFX_VIRTUAL
```

```
// Implementation
public:
  BOOL       Open(DIAL_MODE dialMode=TONE_DIAL);
          void                Close();
          MODEM_ERROR Initialize();
          MODEM_ERROR Dial(CString strNo);
          MODEM_ERROR HangUp();
          MODEM_ERROR Answer();
  BOOL       SendCommand(CString strCommand);
  MODEM_ERROR WaitForConnection();
  MODEM_ERROR WaitForResponse();
  void       ReadLine(char* lpszBuf, int nBufSize);
  MODEM_CONNECT_ERROR ReadConnectError(char* pszBuf);

          virtual ~CModem();
#ifdef _DEBUG
          virtual void AssertValid() const;
          virtual void Dump(CDumpContext& dc) const;
#endif

};

/////////////////////////////////////////////////////////////////////////////

#endif
```

```
/*
*         File:                         Modem.cpp
*
*         Contains:
*
*         Project:            Terminal Emulator for Win95
*
*         Copyright:                    (c) 1997 Kwon, Jae-Rock. All right Reserved.
*
*         Written by:                   Kwon, Jae-Rock
*
*         Change History(most recent first) :
*
*         <1> 97.03
*
*/

#include "stdafx.h"
#include "Modem.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNCREATE(CModem, CObject)

char *szConnectError[] = {
    "SUCCESS", "NO CARRIER", "ERROR", "NO DIALTONE", "BUSY", "NO ANSWER",
NULL
}; // SUCCESS is dummy value

//////////////////////////////////////////////////////////////////////////
// Modem Constructor/Destructor

CModem::CModem()
{
   SetConnectTimeout(60000L);
   SetResponseTimeout(2000L);
   SetInitCommand("ATZ");
   SetDialMode(TONE_DIAL);
}

CModem::~CModem()
{
   if(GetCommPort()->IsOpen())
      Close();
}

//////////////////////////////////////////////////////////////////////////
// CModem Operation

BOOL CModem::Open(DIAL_MODE dialMode)
{
  GetCommPort()->ReadProfile();
          if(GetCommPort()->Open())
  {
    SetDialMode(dialMode);
             Initialize();
    return TRUE;
  }
  else
  {
    GetCommPort()->DispError();
    return FALSE;
  }
}

void CModem::Close()
```

```
{
        GetCommPort()->Close();
}

//////////////////////////////////////////////////////////////////////
// CModem implementation
MODEM_CONNECT_ERROR CModem::ReadConnectError(char* lpszBuf)
{
   for(int i=0; szConnectError[i]; i++)
   {
      if(strstr(lpszBuf, szConnectError[i]))
         return (MODEM_CONNECT_ERROR)i; // connection error
   }
   return CE_SUCCESS;
}

void CModem::ReadLine(char* lpszBuf, int nBufSize)
{
   char cChar;

   while(TRUE)
   {
      if(GetCommPort()->Read(&cChar, 1) != 1)
         break;
      *lpszBuf++ = (cChar);
      if(--nBufSize <= 1)
         break;
      if(cChar == ASCII_LF)
         break;
   }
   *lpszBuf = NULL;
}

BOOL CModem::SendCommand(CString strCommand)
{
   GetCommPort()->Write(strCommand);
   GetCommPort()->Write("\r");

   return WaitForResponse();
}

MODEM_ERROR CModem::Initialize()
{
        SendCommand(GetInitCommand());
   return WaitForResponse();
}

MODEM_ERROR CModem::Dial(CString strNo)
{
        GetCommPort()->Write("ATD");
   GetCommPort()->Write((GetDialMode() == TONE_DIAL) ? "T" : "P");
```

```
   GetCommPort()->Write(strNo);
           GetCommPort()->Write("\r");

           return WaitForConnection();
}

MODEM_ERROR CModem::HangUp()
{
          DWORD dwTimeout;

  GetCommPort()->DTR(FALSE);
  dwTimeout = GetTickCount()+1000L;
  while(GetTickCount() < dwTimeout)
      ;
          GetCommPort()->DTR(TRUE);

  GetCommPort()->Write("+++");
  WaitForResponse();
          GetCommPort()->Write("ATH0\r");
  if(WaitForResponse() == MODEM_SUCCESS)
     return MODEM_SUCCESS;

          return MODEM_DISCONNECT_FAILED;
}

MODEM_ERROR CModem::Answer()
{
          GetCommPort()->Write("ATA");

          return WaitForConnection();
}

MODEM_ERROR CModem::WaitForConnection()
{
   DWORD dwTimeout;
   MODEM_CONNECT_ERROR connectError;
   char szBuf[MAXBUF];

   dwTimeout = GetTickCount() + GetConnectTimeout();
   while(GetTickCount() < dwTimeout)
   {
      ReadLine(szBuf, MAXBUF);
      connectError = ReadConnectError(szBuf);
      if(connectError != CE_SUCCESS)
         return MODEM_NO_CONNECTION;
      if(strstr(szBuf, "CONNECT"))
         return MODEM_SUCCESS;
   }
   return MODEM_NO_CONNECTION;
}
```

```
MODEM_ERROR CModem::WaitForResponse()
{
   DWORD dwTimeout;
   char szBuf[MAXBUF];

   dwTimeout = GetTickCount() + GetResponseTimeout();
   while(GetTickCount() < dwTimeout)
   {
      ReadLine(szBuf, MAXBUF);
      if(!strncmp(szBuf, "OK", 2))
         return MODEM_SUCCESS;
   }
   return MODEM_NO_RESPONSE;
}

/////////////////////////////////////////////////////////////////////////////
// CModem diagnostics

#ifdef _DEBUG
void CModem::AssertValid() const
{
          CObject::AssertValid();
}

void CModem::Dump(CDumpContext& dc) const
{
          CObject::Dump(dc);
}
#endif //_DEBUG

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