RFID-232 사용자매뉴얼

2008. 9. 25

OL마이크로웨이브

http://olmicrowaves.com

목 차

1. RFID-232 소개

1) 제품 개요
2) 제품 사양
3) 설치 방법

2. 프로그래밍

1) 간단한 명령어 사용
2) **ISO15693** 명령어
3) 추가된 명령어
4) **RFID Monitor** 사용 방법

3. 예제 프로젝트

1) 강의실 출석관리시스템 **(EXCEL** 사용**)**
2) 재고관리시스템 **(Visual C++ 6.0** 사용**)**
가**.** 메인 응용프로그램 제작
나**. API**함수 라이브러리 제작
다**. API**함수 라이브러리의 사용

1. RFID-232 소개

1) 제품 개요

RFID-232는 ISO15693(ISO18000-3 Mode1) 13.35MHz RFID 표준 프로토콜 및 충돌방지(Anticollision) 기능을 지원하는 RFID 기반의 유비쿼터스 구축을 위한 RFID 리더 모듈로서 개발업체, 대학 및 연구기관, 학부 학생 등으로 하여금 RFID 제품, 실험·실습, 프로젝트 작품 등 다양한 용도로 활용될 수 있다.

RFID-232는 호스트와 시리얼통신시 RS232C레벨 혹은 TTL레벨을 점퍼 셋팅의 변경으로 선택할 수 있으며, 220V 상용전원 사용을 위한 9V DC정전압아답타용 잭이 장착되어 있고, 외부로부터 +5V DC정전압 전원을 공급받을 수 있는 포트도 제공되어 점퍼 셋팅으로 전원을 선택할 수 있어서 호스트 인터페이싱을 상황에 따라 용이하게 적용할 수 있다.

초보자들이 쉽게 RFID에 접근할 수 있도록 간단한 명령어 사용법이 매뉴얼에서 제공되며, 다양한 예제프로젝트를 통해 실습 과정과 소스를 제공한다.

RFID-232는 ISO15693 명령어를 모두 지원하며, 숙련자들이 필요한 명령어를 사용할 수 있도록 매뉴얼에 프로토콜이 자세히 설명되어 있으며, RFID Monitor 프로그램을 사용하여 각 명령어의 수행과 패킷을 확인할 수 있도록 하였다.

RFID-232는 본체와 안테나가 일체형으로 제작되어 별도의 안테나 없이 모듈 단독으로 사이트에 설치하여 사용될 수 있다.

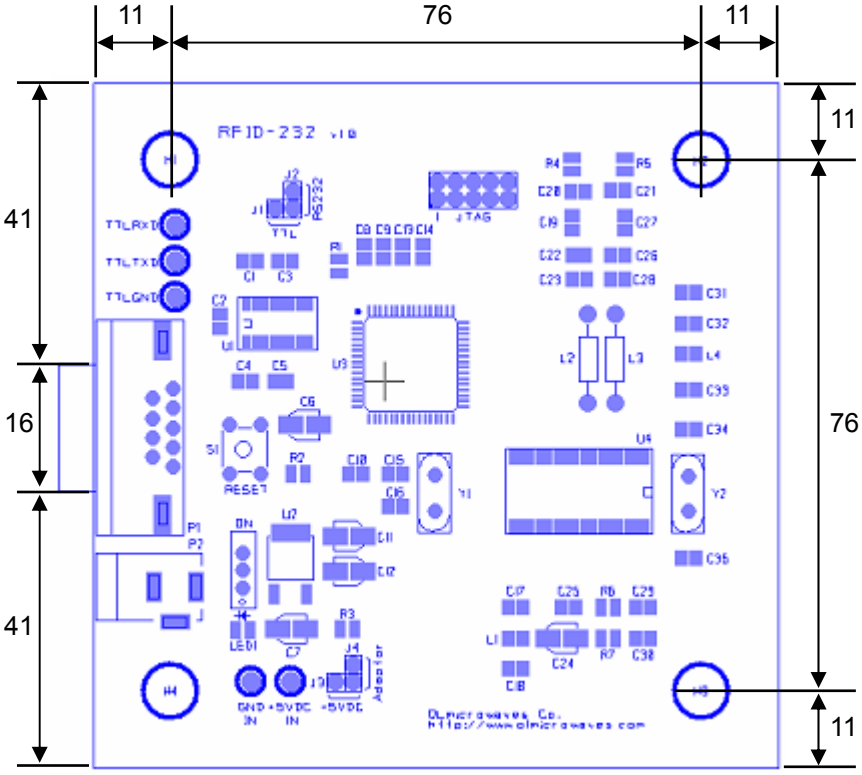
2) 제품 사양

가. 일반 사항

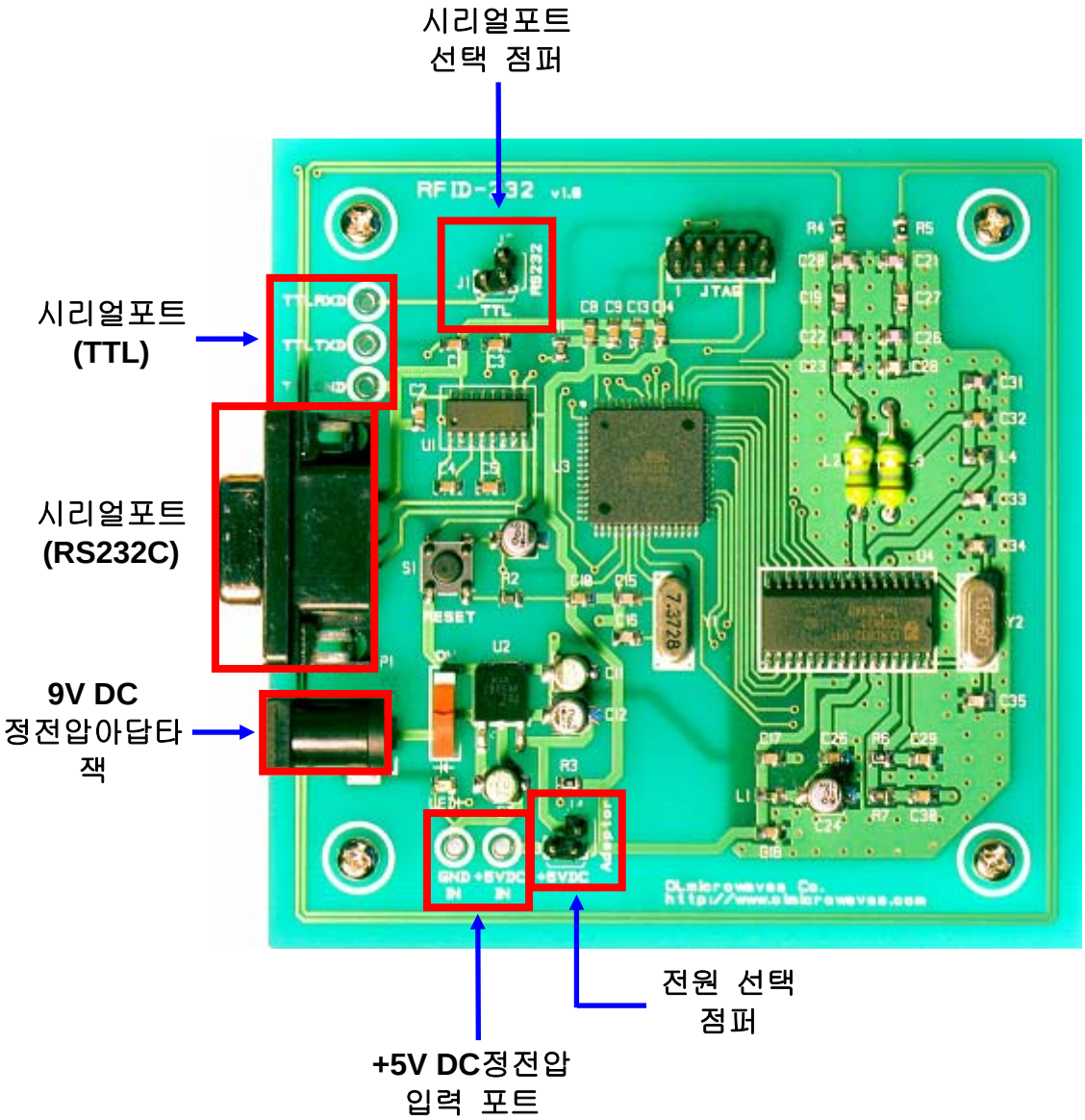
항 목	규 격
프로토콜	ISO 15693(ISO 18000-3 Mode1) (태그 제조업체: TI, Philips, Infenion…)
동작주파수	13.56 MHz
인식거리	10 Cm 이상 (태그 종류에 따라 변동)
전원	점퍼 선택: 67 mA @ 9V DC정전압아답타 혹은 60 mA @ 5V DC정전압
크기	98 x 98 x 15 mm
외부인터페이스 (시리얼통신)	파라메타 설정: 115.2 Kbps, 8 Data bit, 1 Stop bit, No parity, 흐름제어 없음 전기적 구분: 1) RS232C레벨 (9핀 DSUB, Female, 핀2:TXD, 핀3:RXD, 핀5:GND) 2) TTL레벨 (High:+5V, Low:GND)
동작온도	-25℃ ~ 80℃
동작습도	실내환경

나. 기구적 사양

단위: mm


11
76
11
11
41
16
41
76
11
RFID-232 v10
TTLRXD
TTLTXD
TTLGND
RS232
TTL
JTAG
RESET
LED1
GND +5VDC
IN IN
+5VDC
Adapter
OLmicrowaves Co.
http://www.olmicrowaves.com

다. 제품 외관

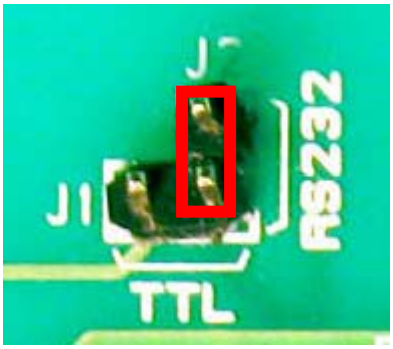

시리얼포트
선택 점퍼
시리얼포트
(TTL)
시리얼포트
(RS232C)
9V DC
정전압아답타
잭
전원 선택
점퍼
+5V DC정전압
입력 포트

3) 설치 방법

가. 점퍼 셋팅

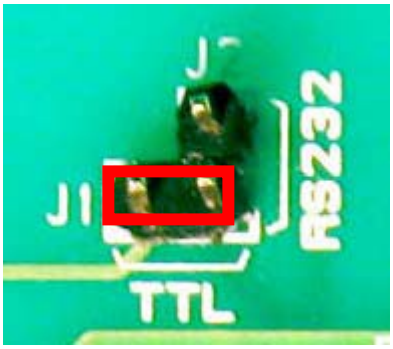
a). 시리얼포트 선택

9핀 컨넥터를 사용하여 RS232C 레벨로 시리얼통신을 하는 경우에는 점퍼를 J2에 장착한다.



J2에 점퍼 설정

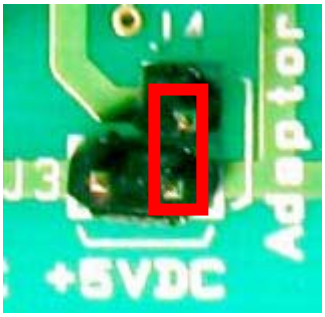
TTL 레벨로 시리얼통신을 하는 경우에는 점퍼를 J1에 장착한다.



J1에 점퍼 설정

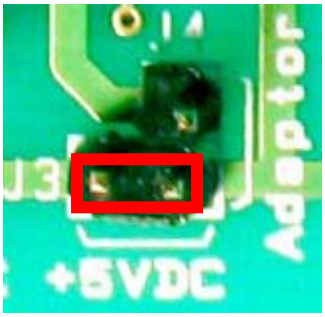
b). 전원 선택

9V DC정전압아답타를 전원으로 사용하는 경우에는 점퍼를 J4에 장착한다.



J4에 점퍼 설정

+5V DC정전압 입력을 전원으로 사용하는 경우에는 점퍼를 J3에 장착한다.



J3에 점퍼 설정

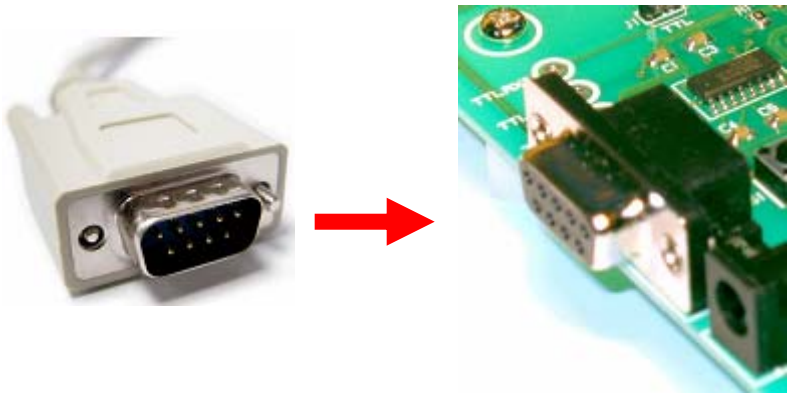
나. 케이블 접속

a). RS232C 시리얼포트 사용

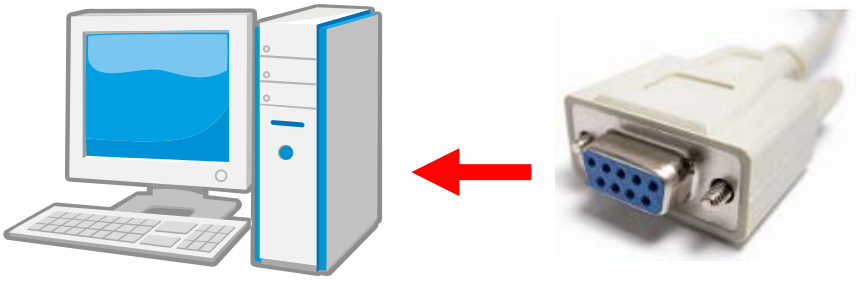
케이블 사양:

- Null Modem용 5Core
- 9핀 DSUB 컨넥터, Male to Female, 1:1 다이렉트
- 1.5 ~ 10M (사용자별 임의 선택)

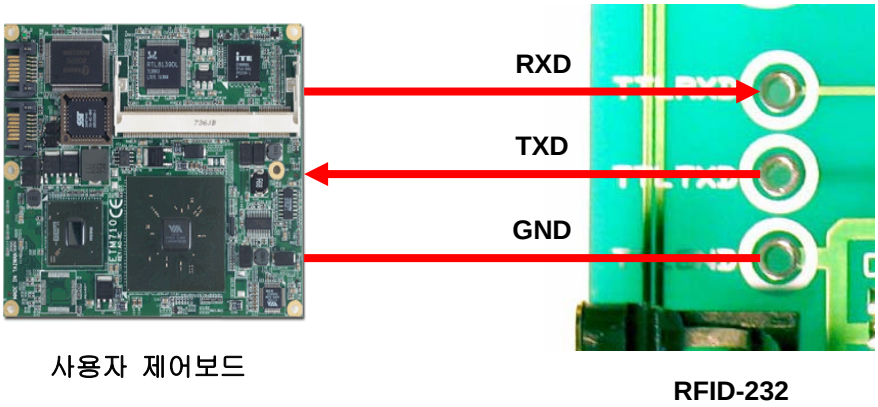
RFID-232측 접속:



PC측 접속:

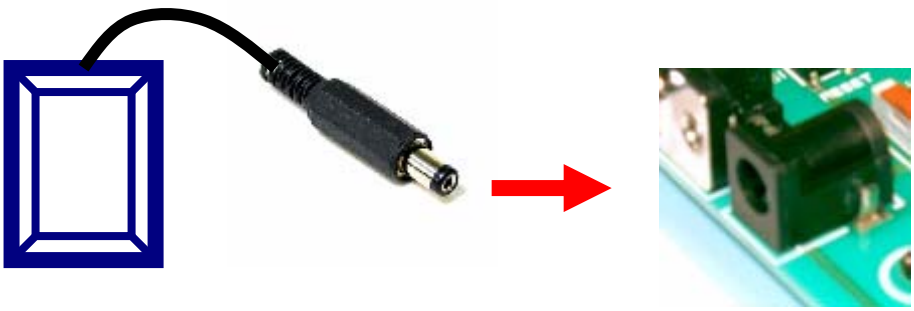


b). TTL 시리얼포트 사용



다. 전원 접속

a). 9V DC정전압아답타 사용

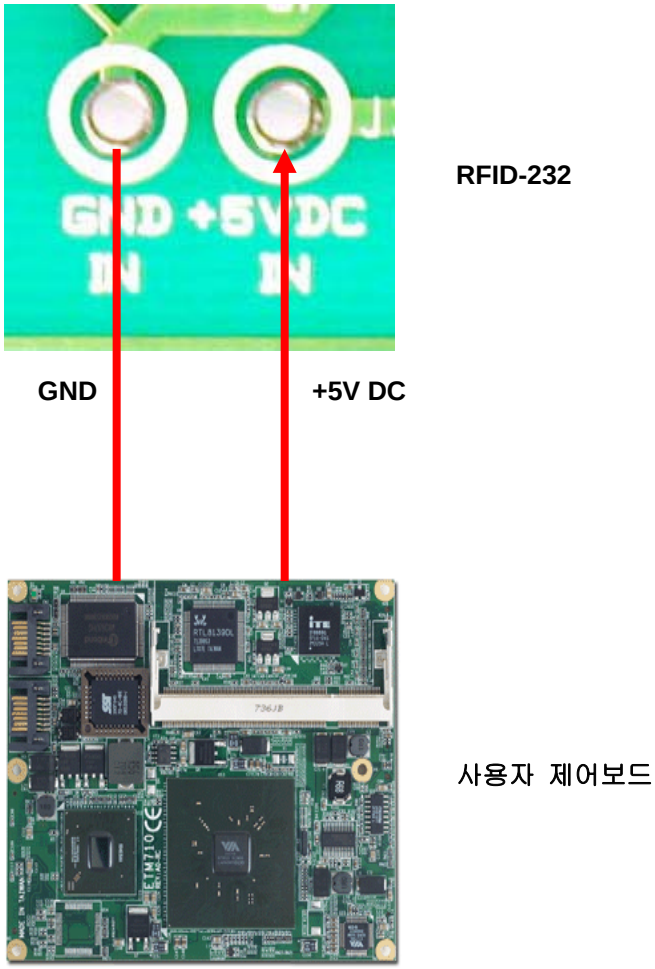


양극핀 지름 : 2mm

b). +5V DC정전압 입력 포트 사용

⚠ 주의

+5V DC정전압 입력 포트는 스위치를 경유하지 않는 관계로 전원이 접속되면 곧바로 전력이 공급되므로 입력 전압이 5V를 초과하지 않도록 미리 확인해야 한다.

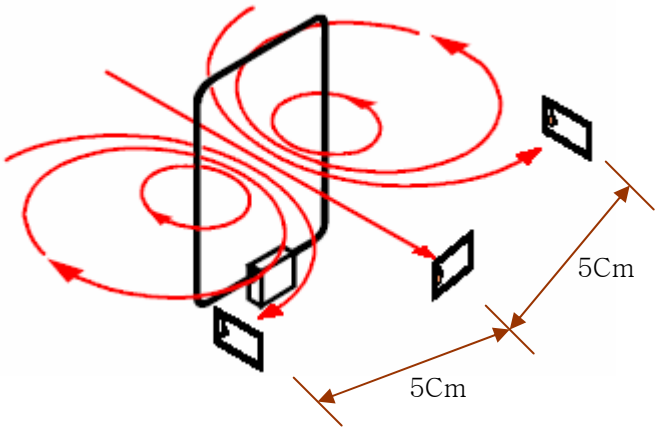


라. RFID Monitor 다운로드

RFID 명령어 실습 및 패킷 관찰을 위해 RFID Monitor를 다운로드한다. 프로그램은 당사 홈페이지(**http://www.olmicrowaves.com**)의 **Downloads** 메뉴 아래 RFID 카테고리에 있다.

마. 태그 사용 방법

Proximity Range(인식 범위 약10Cm인 RFID) 응용에서 일반적으로 IC카드 형태의 태그나 Inlay 라벨은 한 번에 한 개씩 RFID 리더에 스캔하는 방식으로 사용한다. 여러 개의 태그를 동시에 스캔하는 경우에는 인식 범위 내에서 태그 사이의 거리를 최소 5Cm 가량 유지하도록 해야 **Detune 현상**으로 인한 성능저하를 줄일 수 있다.



그림과 같이 리더 안테나에서 발생하는 자기력선의 방향은 원을 그리는 형태이므로 태그는 위치별로 이 자기력선과 직각이 되도록 하는 것이 성능을 최대로 발휘하는 방법이다.

2. 프로그래밍

1) 간단한 명령어 사용

ISO15693 프로토콜을 모르더라도 태그의 목록을 읽고, 태그의 구성을 파악하고, 태그에 정보를 써 넣거나 읽어 내는 동작을 구현할 수 있다.

※.1 모든 2자리 숫자는 1바이트의 16진수 표현임
※.2 숫자 사이의 공란은 설명을 위한 구분임

[Step1] InventoryAll : 인식 범위 내의 태그 목록을 충돌없이 모두 보여 준다.

Request 패킷(호스트 → RFID모듈) : 02 00 f0
Response 패킷(RFID모듈 → 호스트) 형식 : Length(자신 제외) + (Flag + DSFID + UID(8바이트)) 반복 예1) 태그 1개인 경우 : 0a 00 00 5c 08 89 09 00 01 04 e0 예2) 태그 2개인 경우 : 14 00 00 5c 08 89 09 00 01 04 e0 00 00 cc f4 66 24 00 00 07 e0 ※ UID 구조 : (LSB) 제조사별 일련번호 6바이트 + 제조사 코드 + e0 (MSB) ※ 제조사 코드 : Philips 04, Infineon 05, TI 07

[Step2] Get System Information : 지정된 태그의 구성 정보를 읽어 온다.

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + UID(8바이트) 원하는 태그의 UID를 사용한다. 예) 0a 22 2b 5c 08 89 09 00 01 04 e0

Response 패킷(RFID모듈 → 호스트)

형식 : Length(자신 제외) + Flag + Info flag + UID(8바이트) + (DSFID) + (AFI) + (Number of blocks) + (Block size) + (IC reference)

예) 0f 00 0f 5c 08 89 09 00 01 04 e0 00 00 1b 03 01

Info flag의 정의는 다음과 같다.

Bit Nb	Flag name	State	Description
Bit 1	DSFID	0	DSFID is not supported. DSFID field is not present
		1	DSFID is supported. DSFID field is present
Bit 2	AFI	0	AFI is not supported. AFI field is not present
		1	AFI is supported. AFI field is present
Bit 3	VICC memory size	0	Information on VICC memory size is not supported. Memory size field is not present.
		1	Information on VICC memory size is supported. Memory size field is present.
Bit 4	IC reference	0	Information on IC reference is not supported. IC reference field is not present.
		1	Information on IC reference is supported. IC reference field is present.
Bit 5	RFU		Shall be set to 0.
Bit 6	RFU		Shall be set to 0.
Bit 7	RFU		Shall be set to 0.
Bit 8	RFU		Shall be set to 0.

Number of blocks는 메모리의 블럭 개수를 나타내며 다음과 같다.

00 : 1개의 블럭으로 구성

~

ff : 256개 블럭으로 구성

따라서 상기 예제에서는 1b이므로 28개 블럭으로 메모리가 구성됨.

Block size는 각 블럭을 구성하는 바이트 수를 나타내며 다음과 같다.

00 : 1개의 바이트로 구성

~

1f : 32개 바이트로 구성

따라서 상기 예제에서는 03이므로 각 블럭은 4바이트임.

[Step3] Write Single Block : 지정된 태그에 정보를 기록한다.

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + UID(8바이트) + Block number + DB1 + DB2 + DB3 + DB4 원하는 태그의 UID와 임의의 블록 번호와 데이터를 적용한다. 예) 0f 22 21 5c 08 89 09 00 01 04 e0 1b 12 34 ab cd ※ Flag 값 : Philips태그→22, TI태그→62
Response 패킷(RFID모듈 → 호스트) 예1) 01 00 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조

[Step4] Read Single Block : 지정된 태그로부터 정보를 읽어 온다.

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + UID(8바이트) + Block number 원하는 태그의 UID와 블록 번호를 적용한다. 예) 0b 22 20 5c 08 89 09 00 01 04 e0 1b
Response 패킷(RFID모듈 → 호스트) 형식 : Length(자신 제외) + Flag + DB1 + DB2 + DB3 + DB4 예) 05 00 12 34 ab cd

2) ISO15693 명령어

가. Mandatory 명령어

Inventory (01) : 인식 범위 내에 있는 태그의 목록을 보여 준다.

Request 패킷(호스트 → RFID모듈)

형식 : Length(자신 제외) + Flag + 명령코드 + (AFI) + Mask Length + (Mask Value) + (Mask Value) + …

예) 03 06 01 00

※ Flag 값 : 기본(AFI 미사용, 16Slot모드)→**06**
AFI 사용→**기본 OR 10**
1Slot모드→**기본 OR 20**

※ Flag 비트에 관한 자세한 사항은 ISO15693-3 규격서를 참조바라며, 본 모듈에서는 모든 명령어에 대해서 Flag BIT8=0, BIT4=0, BIT2=1, BIT1=0으로 고정해서 사용해야 함.

※ Mask Length : Mask를 사용하지 않을 경우→**00**

※ 1Slot모드로 사용시 1개의 태그만 처리하며 인식범위 내에 2개 이상의 태그가 존재하면 충돌(Collision)을 발생한다. 1Slot모드는 여러 개의 태그가 동시에 접근하지 않는 응용에서 고속으로 처리할 경우에 유용하다. 필요한 경우에 호스트 프로그램에서 Anticollision 처리할 수 있도록 Collision 비트의 위치 정보를 보고한다.

※ 16Slot모드로 사용시 인식범위 내에 최하위 4비트가 서로 다른 16개의 태그를 충돌없이 동시에 처리한다. 최하위 4비트가 동일한 태그가 존재하면 충돌(Collision)을 발생한다. 이러한 경우에 호스트 프로그램에서 Anticollision 처리할 수 있도록 Collision 비트의 위치 정보를 보고한다.

Response 패킷(RFID모듈 → 호스트)

형식 : Length(자신 제외) + (Flag + DSFID + UID(8바이트)) 반복

예1) 1Slot모드인 경우 : 0a 00 00 5c 08 89 09 00 01 04 e0

예2) 1Slot모드 충돌 발생한 경우 : 0a 00 00 02 00 00 00 00 00 00 00

※ 02 → 맨 처음 충돌이 발생한 비트를 1로하고 그 이후의 비트는 0으로 처리

예3) 16Slot모드인 경우 : 1e 00 00 4a 38 9b 24 00 00 07 e0 00 00 5c 08 89 09 00 01 04 e0 00 00 4e 38 9b 24 00 00 07 e0

예4) 16Slot모드 충돌 발생한 경우 : 14 00 00 4a 38 9b 24 00 00 07 e0 00 00 1c 00 00 00 00 00 00 00

※ 1c → 맨 처음 충돌이 발생한 비트를 1로하고 그 이후의 비트는 0으로 처리

예5) 01 00 : 인식 범위 내에 태그가 없음

예6) 01 30 : 리더와 태그간 프레임 에러 발생

Stay quiet (02) : 지정된 태그를 침묵시킨다.

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + UID(8바이트) 원하는 태그의 UID를 적용한다. 예) 0a 22 02 5c 08 89 09 00 01 04 e0
Response 패킷(RFID모듈 → 호스트) 예1) 01 10 : 명령 수행 완료 예2) 01 11 : 명령 수행 실패

나. Optional 명령어

Read single block (20) : 지정된 태그의 지정된 블록의 내용을 읽어 온다.

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + (UID(8바이트)) + Block number 원하는 태그의 UID와 블록 번호를 적용한다. 예) 0b 22 20 5c 08 89 09 00 01 04 e0 1b ※ Select 모드인 경우(Select_flag 비트=1, Address_flag 비트=0) UID를 생략한다. ※ Custom_flag 비트를 set하면 Block security status 값을 읽어올 수가 있다. ※ Flag 비트에 관한 기타 자세한 사항은 ISO15693-3 규격서 참조.
Response 패킷(RFID모듈 → 호스트) 형식 : Length(자신 제외) + Flag + DB1 + DB2 + DB3 + DB4 예1) 05 00 12 34 ab cd : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

Write single block (21) : 지정된 태그의 지정된 블록에 내용을 써 넣는다.

Request 패킷(호스트 → RFID모듈)
형식 : Length(자신 제외) + Flag + 명령코드 + (UID(8바이트)) + Block number + DB1 + DB2 + DB3 + DB4 원하는 태그의 UID와 임의의 블록 번호와 데이터를 적용한다. 예) 0f 22 21 5c 08 89 09 00 01 04 e0 1b 12 34 ab cd ※ Flag 값 : Philips태그→22, TI태그→62 ※ Select 모드인 경우(Select_flag 비트=1, Address_flag 비트=0) UID를 생략한다. ※ Flag 비트에 관한 기타 자세한 사항은 ISO15693-3 규격서 참조.
Response 패킷(RFID모듈 → 호스트)
예1) 01 00 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

Lock block (22) : 지정된 태그의 지정된 블록을 Lock(Read Only) 시킨다.

⚠ 주의 **Lock이 설정된 이후에는 해당 블록의 데이터 수정이 불가능함.(복구 불가!)**

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + (UID(8바이트)) + Block number 원하는 태그의 UID와 임의의 블록 번호를 적용한다. 예) 0b 22 22 5c 08 89 09 00 01 04 e0 06 ※ Flag 값 : Philips태그→22, TI태그→62 ※ Select 모드인 경우(Select_flag 비트=1, Address_flag 비트=0) UID를 생략한다. ※ Flag 비트에 관한 기타 자세한 사항은 ISO15693-3 규격서 참조.
Response 패킷(RFID모듈 → 호스트) 예1) 01 00 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

Read multiple blocks (23) : 지정된 태그의 지정된 블록들의 내용을 읽어 온다.

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + (UID(8바이트)) + First block number + Number of blocks 원하는 태그의 UID와 첫번째 블록 번호와 읽어올 블록의 개수를 적용한다. 예) 0c 22 23 5c 08 89 09 00 01 04 e0 00 03 ※ Select 모드인 경우(Select_flag 비트=1, Address_flag 비트=0) UID를 생략한다. ※ Custom_flag 비트를 set하면 Block security status 값을 읽어올 수가 있다. ※ Flag 비트에 관한 기타 자세한 사항은 ISO15693-3 규격서 참조.
Response 패킷(RFID모듈 → 호스트) 형식 : Length(자신 제외) + Flag + DB11 + DB12 + DB13 + DB14 + DB21 + DB22 + DB23 + DB24 + … 예1) 11 00 11 12 13 14 21 22 23 24 31 32 33 34 41 42 43 44 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

Select (25) : 지정된 태그를 Select 상태로 설정한다.

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + UID(8바이트) 원하는 태그의 UID를 적용한다. 예) 0a 22 25 5c 08 89 09 00 01 04 e0
Response 패킷(RFID모듈 → 호스트) 예1) 01 00 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

Reset to ready (26) : 지정된 태그를 Ready 상태로 설정한다.

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + (UID(8바이트)) 원하는 태그의 UID를 적용한다. 예) 0a 22 26 5c 08 89 09 00 01 04 e0 ※ Select 모드인 경우(Select_flag 비트=1, Address_flag 비트=0) UID를 생략한다. ※ Select 모드가 아닌 경우 UID를 생략하면 Quiet 상태의 모든 태그에 동시에 적용된다. ※ Flag 비트에 관한 기타 자세한 사항은 ISO15693-3 규격서 참조.
Response 패킷(RFID모듈 → 호스트) 예1) 01 00 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

Write AFI (27): 지정된 태그의 AFI 값을 수정한다.

Request 패킷(호스트 → RFID모듈)
형식 : Length(자신 제외) + Flag + 명령코드 + (UID(8바이트)) + AFI 원하는 태그의 UID와 AFI 값을 적용한다. 예) 0b 22 27 5c 08 89 09 00 01 04 e0 01 ※ Flag 값 : Philips태그→22, TI태그→62 ※ Select 모드인 경우(Select_flag 비트=1, Address_flag 비트=0) UID를 생략한다. ※ Flag 비트에 관한 기타 자세한 사항은 ISO15693-3 규격서 참조.
Response 패킷(RFID모듈 → 호스트) 예1) 01 00 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

Lock AFI (28): 지정된 태그의 AFI를 Lock(Read Only) 시킨다.

⚠ 주의 **Lock이 설정된 이후에는 AFI의 수정이 불가능함.(복구 불가!)**

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + (UID(8바이트)) 원하는 태그의 UID를 적용한다. 예) 0a 22 28 5c 08 89 09 00 01 04 e0 ※ Flag 값 : Philips태그→22, TI태그→62 ※ Select 모드인 경우(Select_flag 비트=1, Address_flag 비트=0) UID를 생략한다. ※ Flag 비트에 관한 기타 자세한 사항은 ISO15693-3 규격서 참조.
Response 패킷(RFID모듈 → 호스트) 예1) 01 00 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

Write DSFID (29): 지정된 태그의 DSFID 값을 수정한다.

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + (UID(8바이트)) + DSFID 원하는 태그의 UID와 DSFID 값을 적용한다. 예) 0b 22 29 5c 08 89 09 00 01 04 e0 01 ※ Flag 값 : Philips태그→22, TI태그→62 ※ Select 모드인 경우(Select_flag 비트=1, Address_flag 비트=0) UID를 생략한다. ※ Flag 비트에 관한 기타 자세한 사항은 ISO15693-3 규격서 참조.
Response 패킷(RFID모듈 → 호스트) 예1) 01 00 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

Lock DSFID (2A): 지정된 태그의 DSFID를 Lock(Read Only) 시킨다.

⚠ 주의 **Lock이 설정된 이후에는 DSFID의 수정이 불가능함.(복구 불가!)**

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + (UID(8바이트)) 원하는 태그의 UID를 적용한다. 예) 0a 22 2a 5c 08 89 09 00 01 04 e0 ※ Flag 값 : Philips태그→22, TI태그→62 ※ Select 모드인 경우(Select_flag 비트=1, Address_flag 비트=0) UID를 생략한다. ※ Flag 비트에 관한 기타 자세한 사항은 ISO15693-3 규격서 참조.
Response 패킷(RFID모듈 → 호스트) 예1) 01 00 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

Get system information (2B): 앞 절 "간단한 명령어 사용" 참조.

Get block security status (2C): 지정된 태그의 지정된 블록들의 Block security status를 읽어 온다.

Request 패킷(호스트 → RFID모듈) 형식 : Length(자신 제외) + Flag + 명령코드 + (UID(8바이트)) + First block number + Number of blocks 원하는 태그의 UID와 첫번째 블록 번호와 읽어올 블록의 개수를 적용한다. 예) 0c 22 2c 5c 08 89 09 00 01 04 e0 00 09 ※ Select 모드인 경우(Select_flag 비트=1, Address_flag 비트=0) UID를 생략한다. ※ Flag 비트에 관한 기타 자세한 사항은 ISO15693-3 규격서 참조.
Response 패킷(RFID모듈 → 호스트) 형식 : Length(자신 제외) + Flag + Bss1 + Bss2 + Bss3 + Bss4 + … 예1) 0b 00 01 01 01 01 00 00 00 00 01 01 : 정상 응답 예2) 02 01 XX : 태그 오류 ※ XX : 태그로부터의 오류 코드 ※ 오류 코드에 관한 자세한 사항은 ISO15693-3 규격서 참조 예3) 01 20 : 태그로부터 응답 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

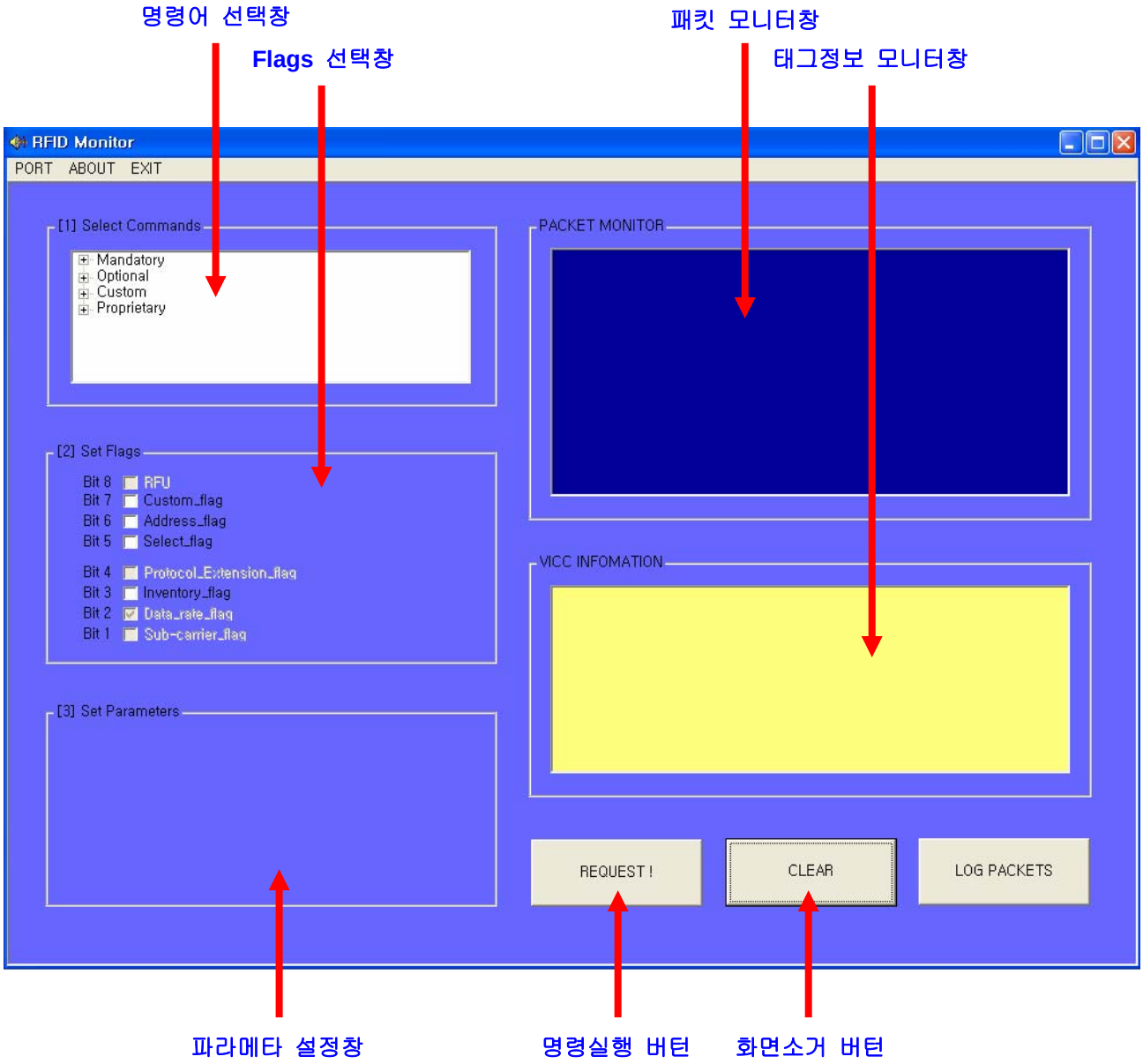
3) 추가된 명령어

InventoryAll : 인식 범위 내에 있는 태그의 목록을 충돌없이 모두 보여 준다.

Request 패킷(호스트 → RFID모듈) 형식1: AFI를 사용하지 않는 경우(default) : 02 00 f0 형식2: AFI를 사용하는 경우 : 03 10 f0 XX ※ XX : AFI 값
Response 패킷(RFID모듈 → 호스트) 형식: Length(자신 제외) + (Flag + DSFID + UID(8바이트)) 반복 예1) 태그 1개인 경우 : 0a 00 00 5c 08 89 09 00 01 04 e0 예2) 태그 2개인 경우 : 14 00 00 5c 08 89 09 00 01 04 e0 00 00 cc f4 66 24 00 00 07 e0 예3) 01 00 : 인식 범위 내에 태그가 없음 예4) 01 30 : 리더와 태그간 프레임 에러 발생

4) RFID Monitor 사용 방법

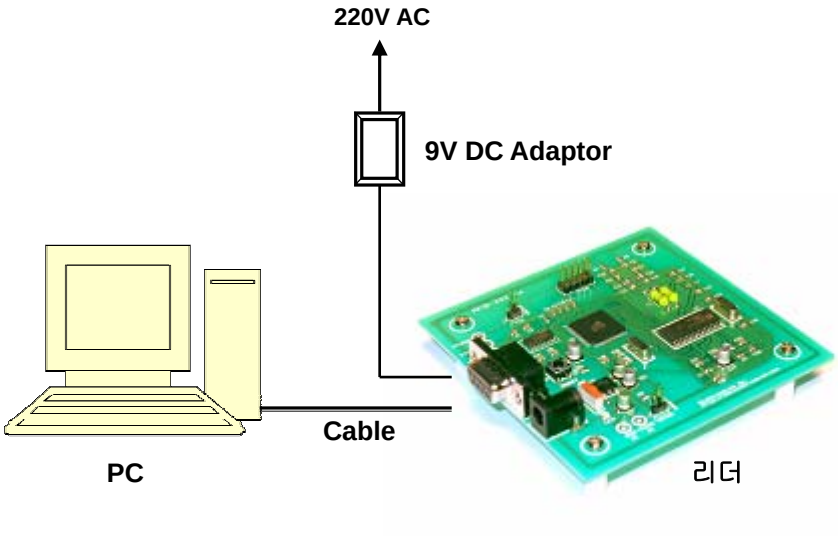
가. 화면 구성



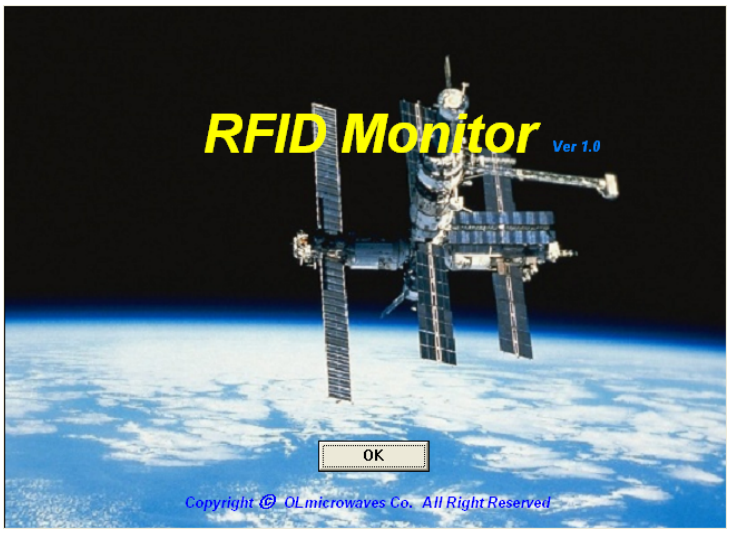
나. 사용 방법

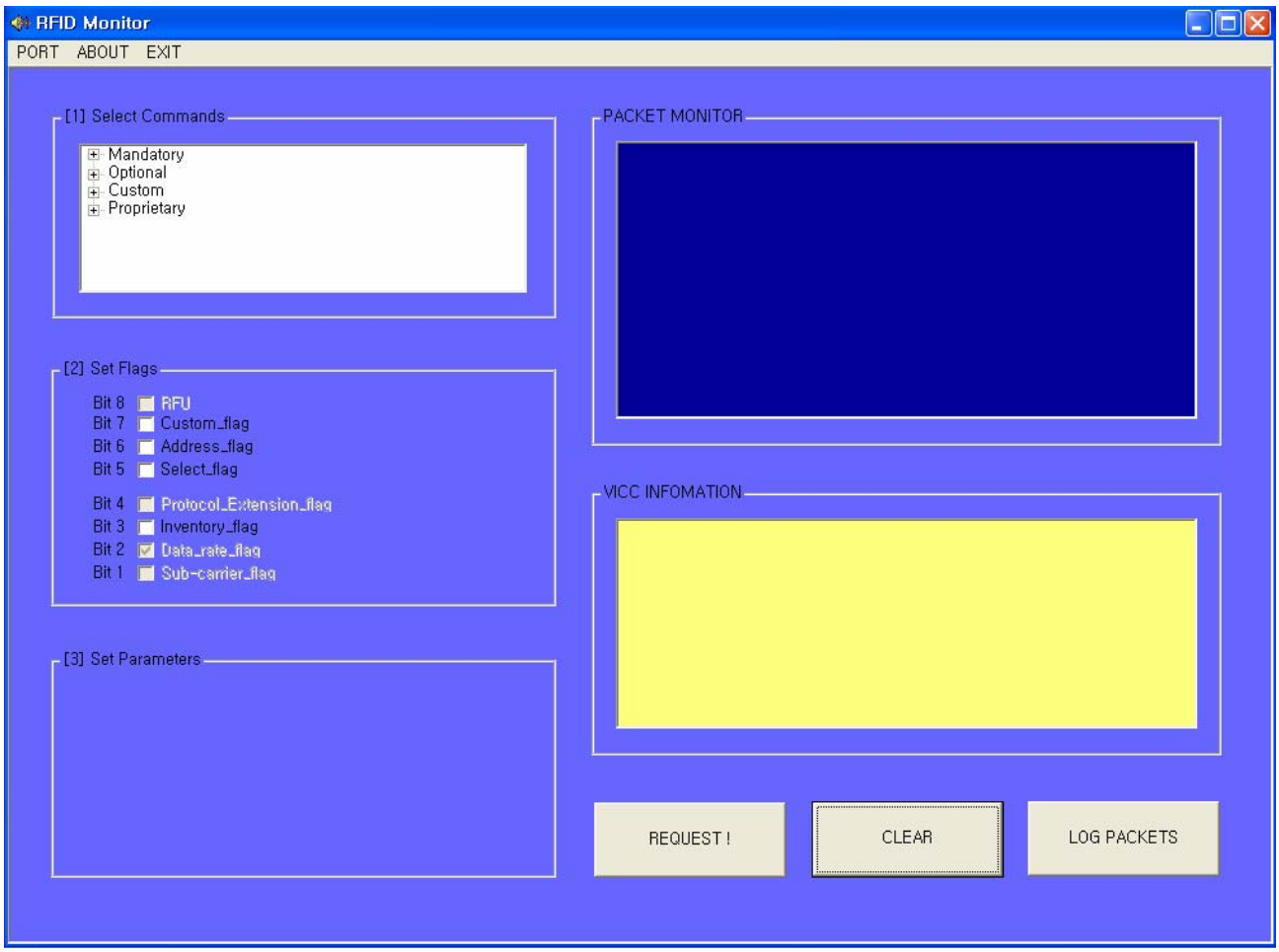
아래의 절차는 RFID Monitor 사용 방법을 설명하기 위한 예제로서, Inventory와 Write single block 및 Read single block 명령어를 RFID Monitor를 사용하여 학습해 본다.

[Step1] 리더 모듈과 RFID Monitor 프로그램(RFID Monitor.exe)을 설치한다.



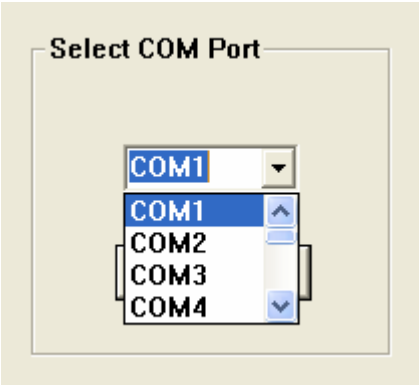
[Step2] RFID Monitor 프로그램을 실행한다.

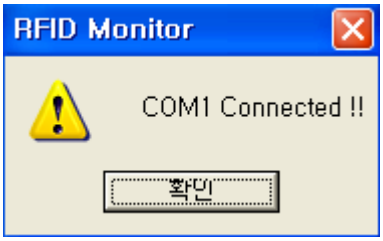




[Step3] 통신 포트를 선택한다.

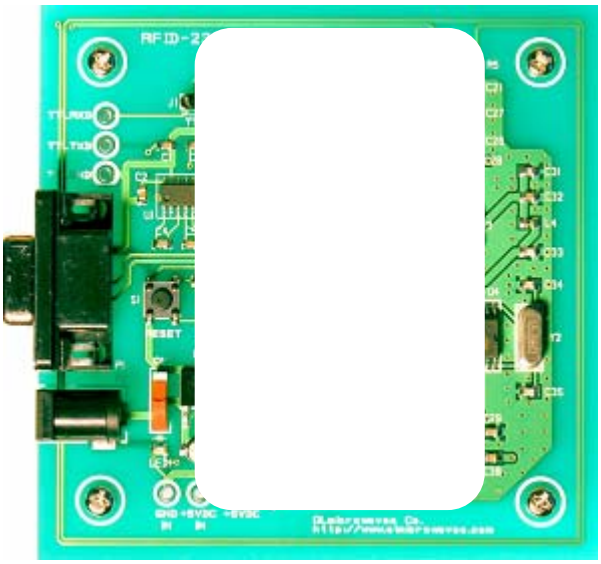
메뉴에서 **PORT**를 클릭하여 시리얼통신 포트를 선택한다.



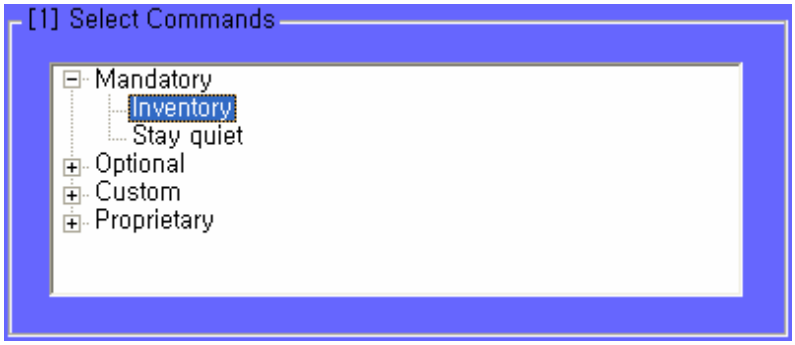


[Step.4] Inventory 명령어 수행

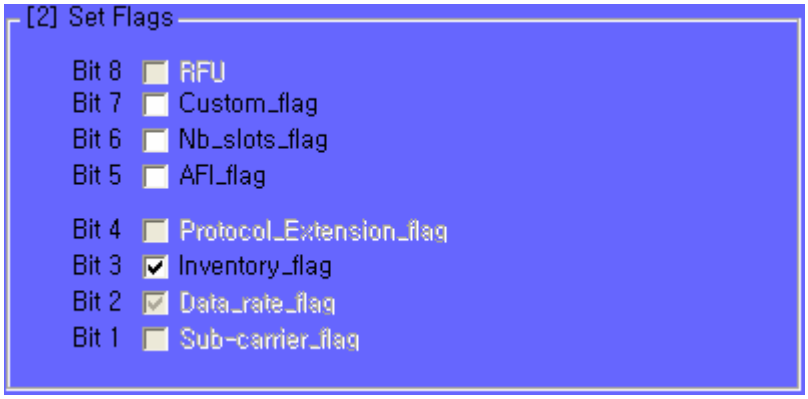
리더 모듈 위에 태그를 올려 놓는다.



[1] Select Commands 창에서 **Mandatory** 카테고리의 아래에 있는 **Inventory** 명령어를 클릭한다.



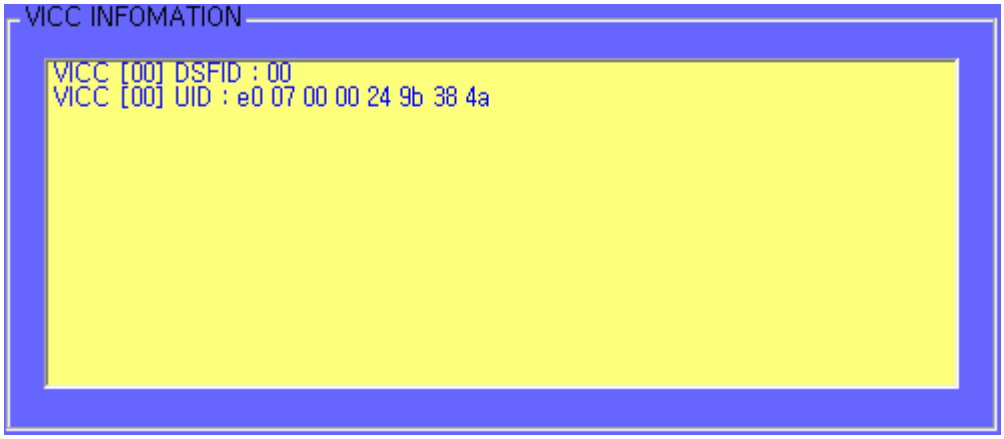
[2] Set Flags 창에서 0x06 의 값이 되도록 Flags 를 설정한다.



REQUEST ! 버턴을 클릭하여 명령어를 수행시킨다.

PACKET MONITOR 창에서 명령 패킷과 응답 패킷을 확인한다.

VICC INFOMATION 창에서 태그의 정보를 확인한다.

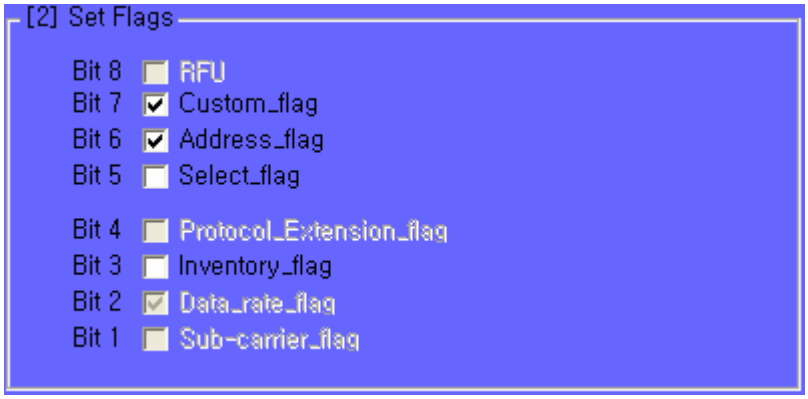


※ VICC[Vicinity Integrated Circuit Card] IC 카드 형태의 RFID 트랜스폰더

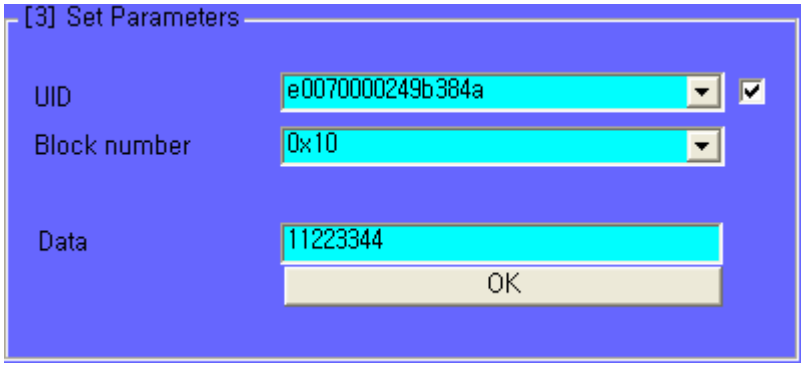
[Step.5] Write single block 명령어 수행

[1] Select Commands 창에서 **Optional** 카테고리의 아래에 있는 **Write single block** 명령어를 클릭한다.

[2] Set Flags 창에서 0x22(Philips 태그) 혹은 0x62(TI 태그)의 값이 되도록 Flags 를 설정한다.

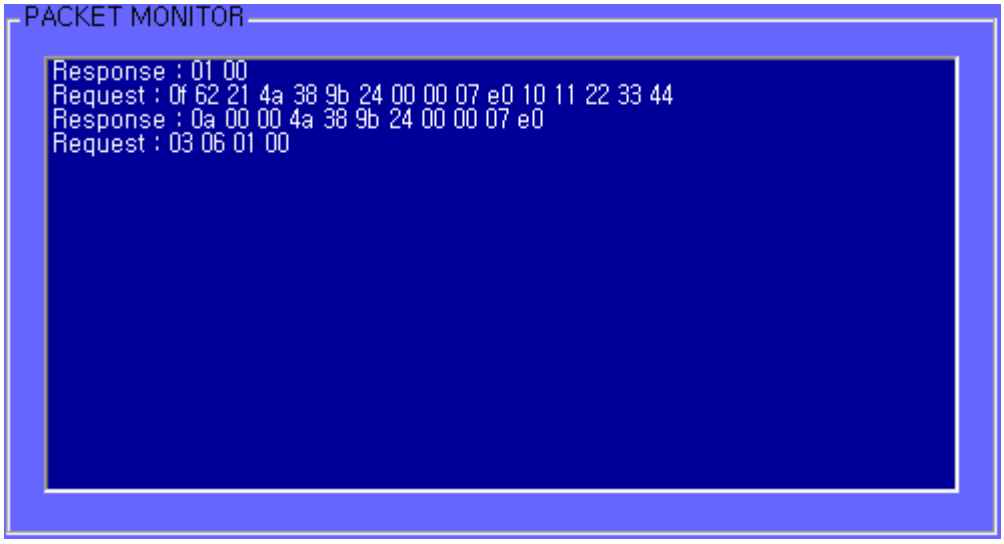


[3] Set Parameters 창에서 태그의 UID, 태그 메모리의 블록 번호를 지정하고, 저장하고자 하는 정보를 16 진수 형식으로 4 바이트를 연속으로 타이핑한 수 **OK** 버턴을 클릭한다.



REQUEST ! 버턴을 클릭하여 명령어를 수행시킨다.

PACKET MONITOR 창에서 명령 패킷과 응답 패킷을 확인한다.

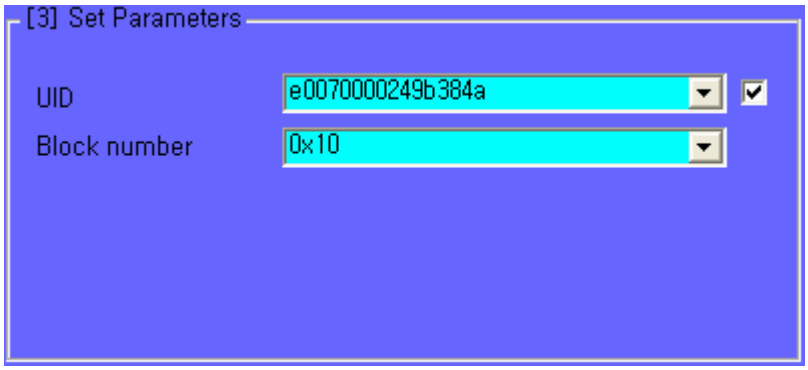


[Step.6] Read single block 명령어 수행

[1] Select Commands 창에서 **Optional** 카테고리의 아래에 있는 **Read single block** 명령어를 클릭한다.

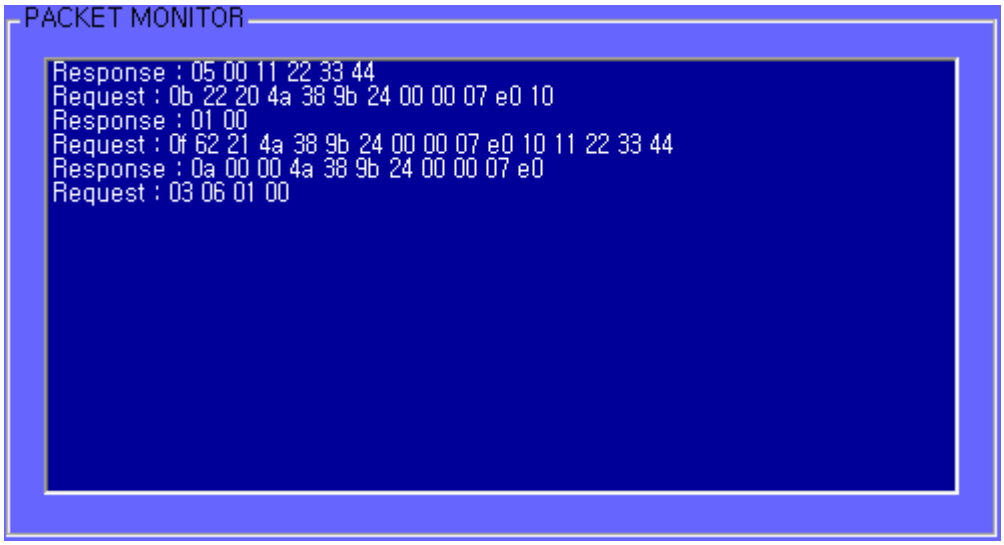
[2] Set Flags 창에서 0x22 의 값이 되도록 Flags 를 설정한다.

[3] Set Parameters 창에서 태그의 UID, 태그 메모리의 블록 번호를 지정한다.

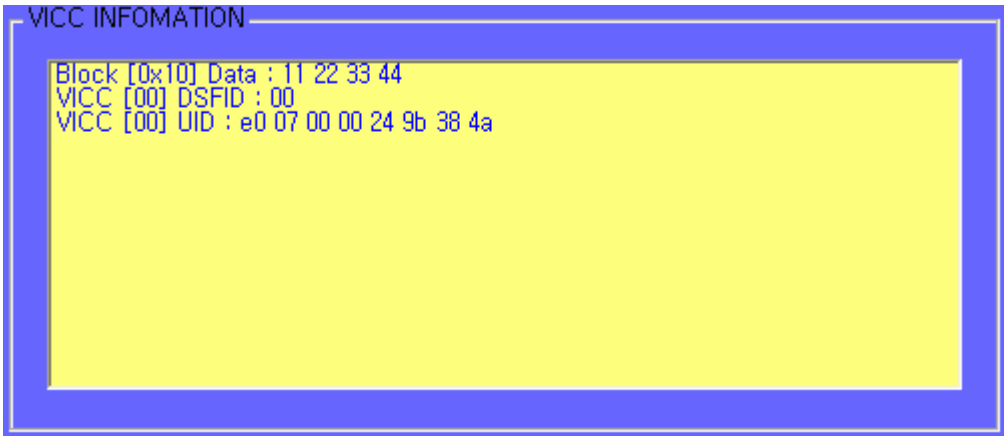


REQUEST ! 버턴을 클릭하여 명령어를 수행시킨다.

PACKET MONITOR 창에서 명령 패킷과 응답 패킷을 확인한다.



VICC INFOMATION 창에서 태그의 해당 블록의 정보를 확인한다.



위와 같은 방식으로 다른 명령어도 학습한다.

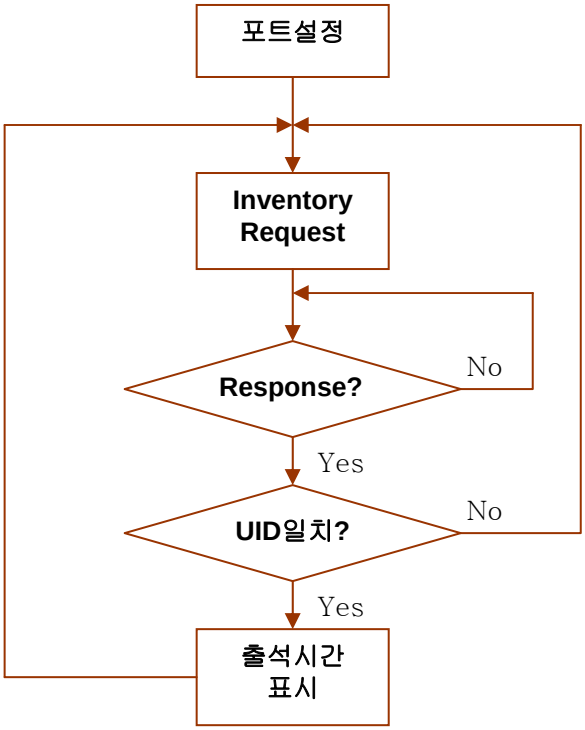
⚠ 주의 : **Lock 관련 명령어는 태그의 복구가 불가능하므로 신중하게 사용해야 함.**

3. 예제 프로젝트

1) 강의실 출석관리시스템 (EXCEL 사용)

일반적으로 PC 환경에서 시리얼통신을 구현하기 위해서는 **Visual C++**이나 **Visual Basic** 혹은 **Boland C++ Builder** 등을 사용해만 가능한 것으로 알고 있기 쉽다. 그러나 윈도우즈의 Win32 Call 기능을 제공하는 마이크로소프트 **엑셀(Excel)**로도 시리얼통신 기능을 구현할 수 있고, 이 것을 토대로 이화학 분야나 공학 등 여러 산업 분야에서 데이터 수집, 분석 및 제어를 위해 활용할 수 있으며, 특히 학부 학생들의 프로젝트 수행에 아주 유용하고 손쉬운 도구를 제공한다.

본 프로젝트에서는 '**stal**'님이 공개한 '**Win32Comm.bas**'를 활용하여 **Excel**에서 시리얼통신을 구현하고, RFID리더 모듈에 **Inventory** Request를 보내고 Response 수신하고 정보를 처리하는 방식으로 강의실 출석관리시스템을 구현해 본다.



<순서도>

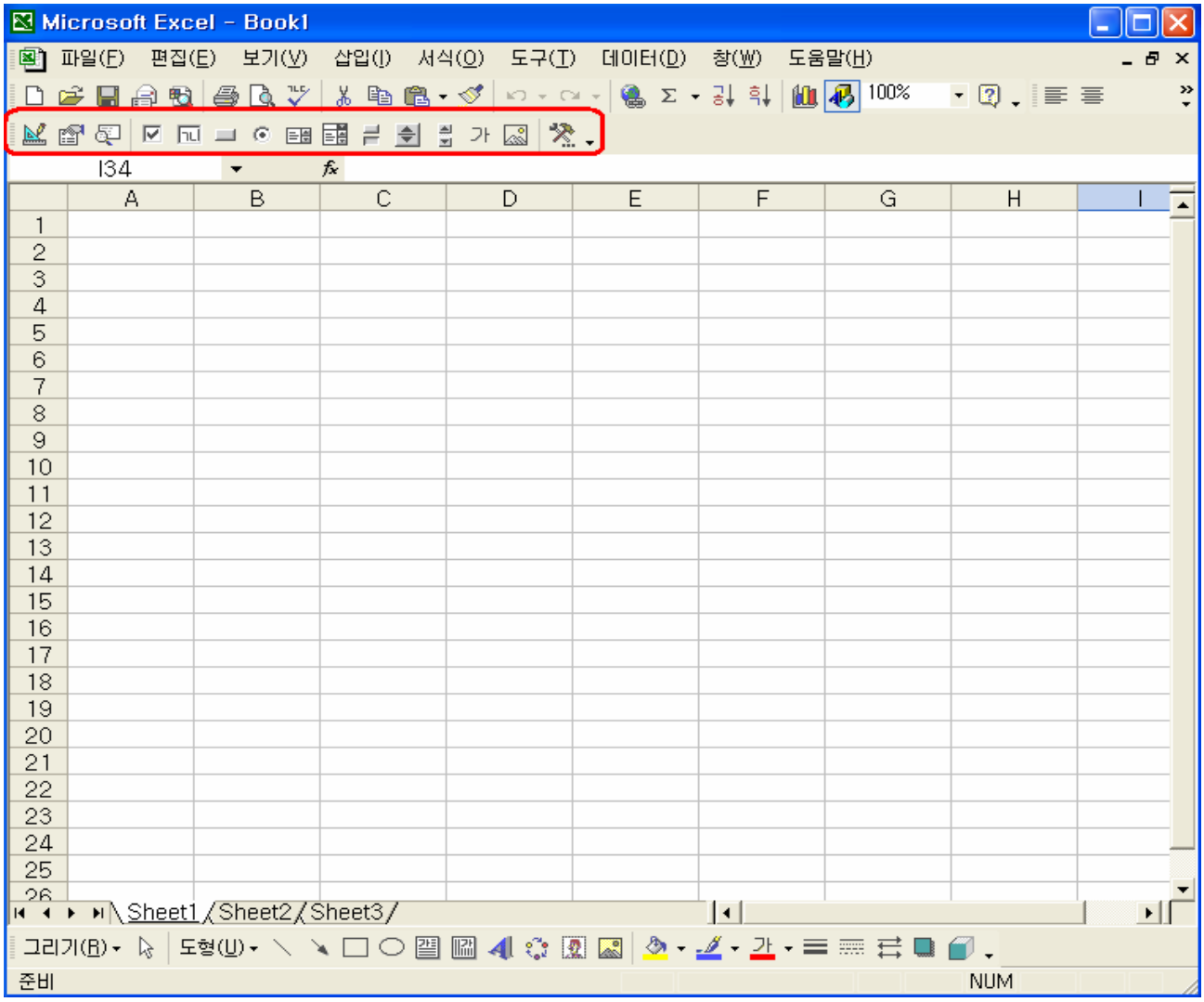
[Step.1] 관련 파일 다운로드

당사 홈페이지(**http://www.olmicrowaves.com**)의 **Downloads** 메뉴 아래 RFID 카테고리에서 다음의 파일을 다운로드하여 적당한 작업 디렉토리에 저장한다.

- **Win32Comm.bas**
- **Students.xls**

[Step.2] Excel VBA(Visual Basic for Applications) 활성화

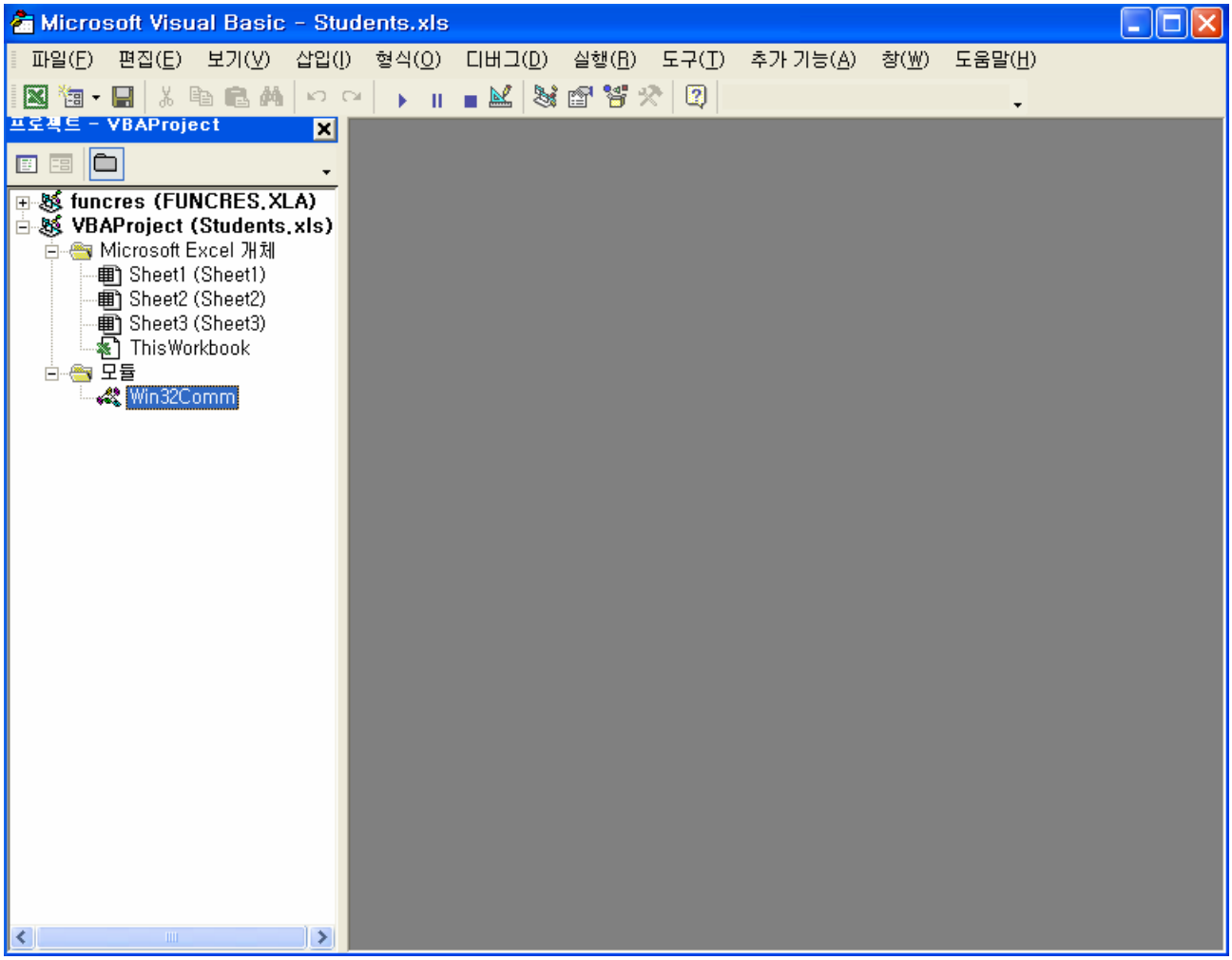
Excel을 실행한다.
메뉴에서 '보기'→'도구 모음'→'컨트롤 도구 상자'를 클릭하여 '컨트롤 도구 상자'를 적당한 곳에 위치시킨다.



[Step.3] Win32Comm.bas 모듈 삽입

Alt+F11 키를 눌러서 **Visual Basic Editor**를 활성화한다.

Visual Basic Editor의 좌측 프로젝트 창에서 마우스 우측 버턴을 클릭하여 팝업 메뉴을 열고 '파일 가져오기'를 클릭하여 **Win32Comm.bas** 모듈을 프로젝트에 삽입한다.



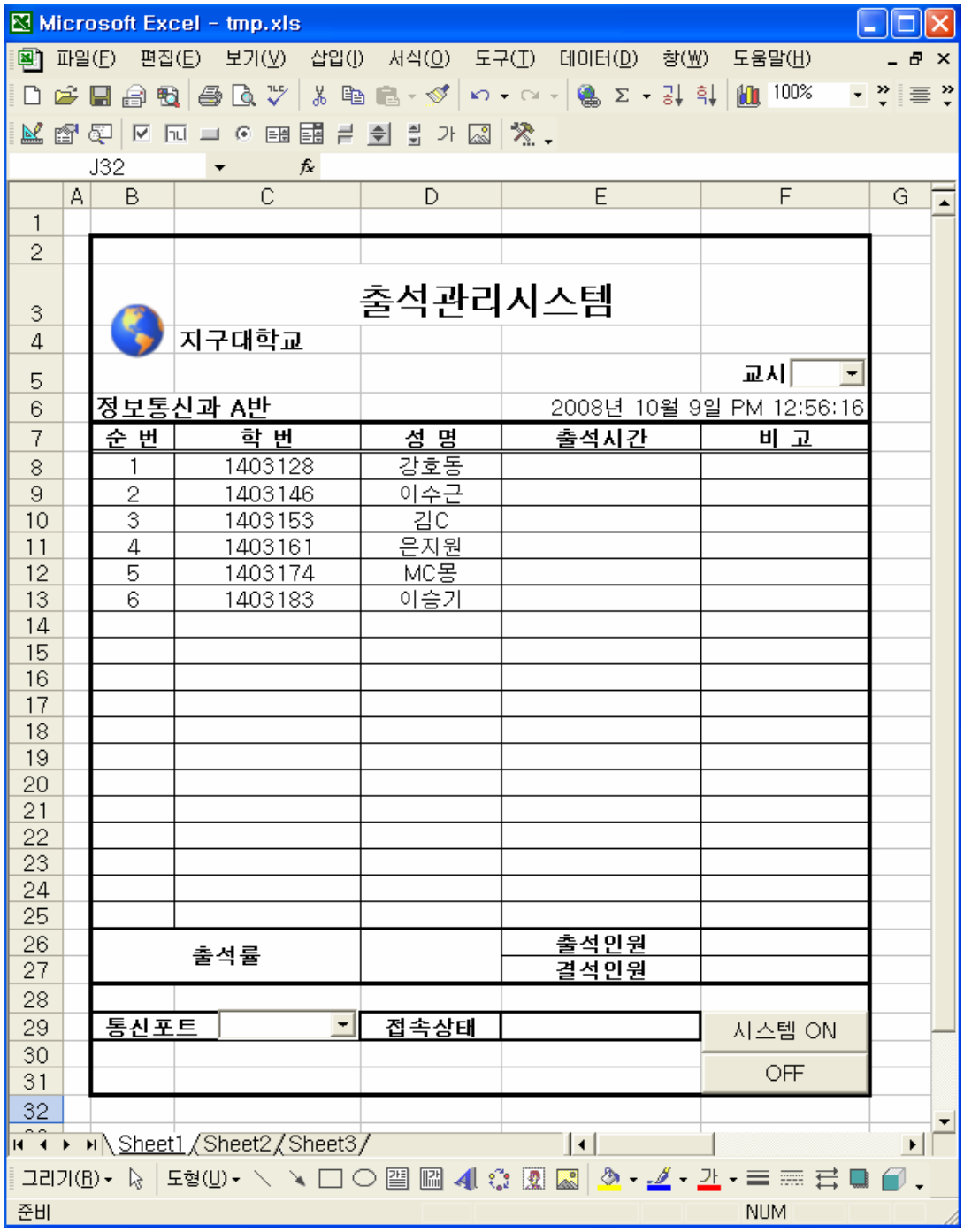
Alt+Q 키를 눌러서 **Visual Basic Editor**를 빠져 나온다.

적당한 파일명(=프로젝트 명)을 정하여 저장한다.

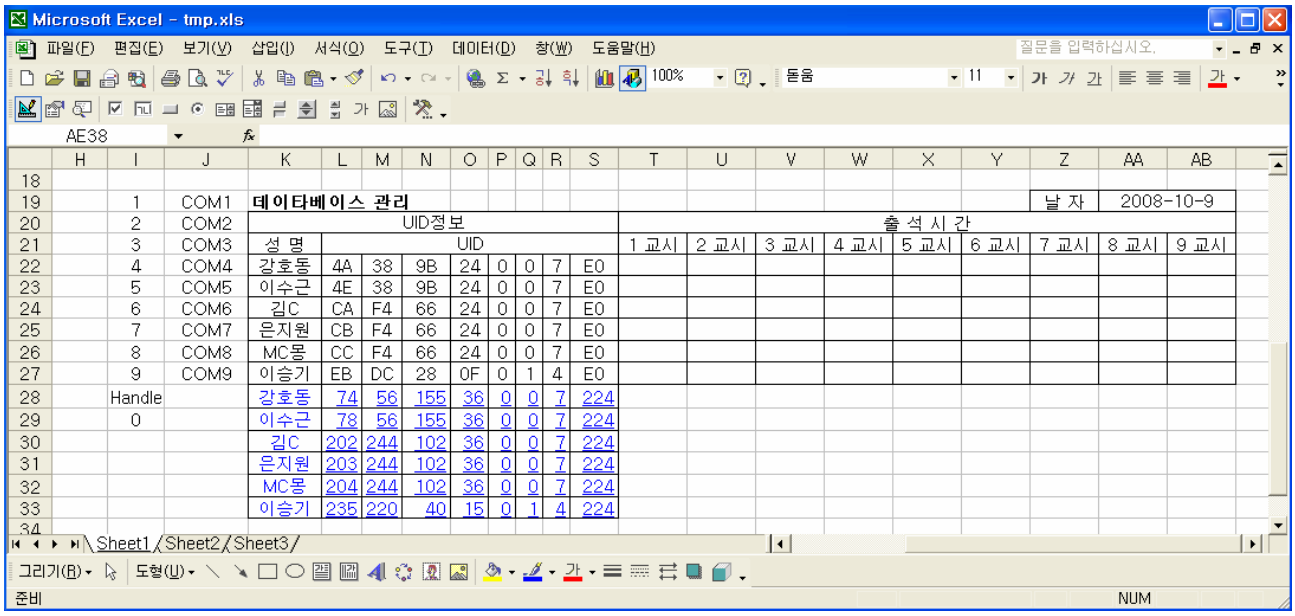
[Step.4] 화면 제작

'컨트롤 도구 상자'에서 하늘색 삼각자 모양의 버턴을 클릭하여 디자인 모드를 활성화한다. (동작을 실행할 경우에는 한 번 더 눌러서 디자인 모드를 끝낸다.)

예제 파일(**Students.xls**)을 참조하여 화면을 제작한다.



예제 파일(**Students.xls**)을 참조하여 화면의 동작에 필요한 데이터 값을 설정한다. 단, UID 값은 사용자별로 확인하여 기입한다.



[Step.5] 컨트롤 동작 프로그래밍

디자인모드에서 '시스템 ON'컨트롤을 더블클릭하여 **Visual Basic Editor**로 들어간다.

```
Private Sub CommandButton1_Click()

End Sub
```

통신포트를 설정하는 함수는 **CommOpen(COM1, "115200, n, 8, 1")**으로서 **Win32Comm** 모듈의 소스에 프로시저가 구현되어 있다. 포트 설정 결과로서 Handle 값이나(성공) 0(실패)을 돌려 준다.

다음은 통신포트를 설정하는 예이다.

```
Dim com As String

com = ComboBox1.Value   '콤보박스에서 포트 값을 읽어 온다.

If [i29] <> 0 Then        '시스템ON 버턴 중복 클릭으로 생성된 프로세서 제거
    Exit Sub
End If

[i29] = CommOpen(com, "115200, n, 8, 1")
```

```
If [i29] = 0 Then
    MsgBox "Cannot open COM Port.", , "Error"
    Exit Sub
End If

[e29] = "접속중"          '정상적으로 COM 포트 설정 완료
```

통신포트가 설정되었으면 **Inventory** Request를 리더 모듈에 송신하여 태그 UID를 요청한다.

```
Dim req_byte1 As Byte, req_byte2 As Byte, req_byte3 As Byte, req_byte4 As Byte

'Inventory 패킷 지정(Hexa 03 06 01 00)

req_byte1 = &H3       '
req_byte2 = &H6
req_byte3 = &H1
req_byte4 = &H0

CommWrite [i29], req_byte1
CommWrite [i29], req_byte2
CommWrite [i29], req_byte3
CommWrite [i29], req_byte4
```

CommWrite 설정된 통신포트로 한 바이트의 데이터를 송신하는 함수이다.

Inventory Request를 송신후 응답을 기다리다가 응답이 입력되면 지정된 버퍼에 저장한다. 이때, 응답을 기다리는 동안 컴퓨터가 먹동이 되지 않도록 **DoEvents** 함수를 호출해 둔다.

```
Do While CommNumRcv([i29]) = 0
    DoEvents     '입력 대기중 시스템 먹통 방지
Loop

Dim res_byte() As Byte, length As Byte

res_byte() = CommReadBytes([i29])    '시리얼 포트 입력을 변수에 저장
```

CommReadBytes는 시리얼통신 입력 버퍼로부터 데이터를 Byte형 배열로 돌려주는 함수이다.

리더 모듈로부터 태그의 UID를 획득한 후 데이터베이스에 등록되어 있는 UID와 일치 여부를 확인하여 처리한다.

```
Dim i As Long, j As Long, match As Long, class As Long

match = 0
class = ComboBox2.Value   '강의 교시 값

If res_byte(1) = 10 Then   '입력의 첫번째 값이 10(0x0a)이면 정상 응답으로 간주
    For i = 0 To 5          '6명의 학생
        For j = 0 To 7      '8바이트의 UID
            If Cells(i + 28, j + 12) = res_byte(4 + j) Then 'UID 셀과 입력 값과 비교
                match = match + 1
                If match = 8 Then   'UID 8바이트가 모두 일치하면..
                    Cells(i + 22, class + 19) = [e6]          '해당 학생의 해당 교시에 현재시간
                    표시
                End If
            End If
        Next
        match = 0
    Next
End If
```

디자인 모드에서 'OFF' 컨트롤을 더블클릭하여 **Visual Basic Editor**로 들어간다.

```
Private Sub CommandButton2_Click()

End Sub
```

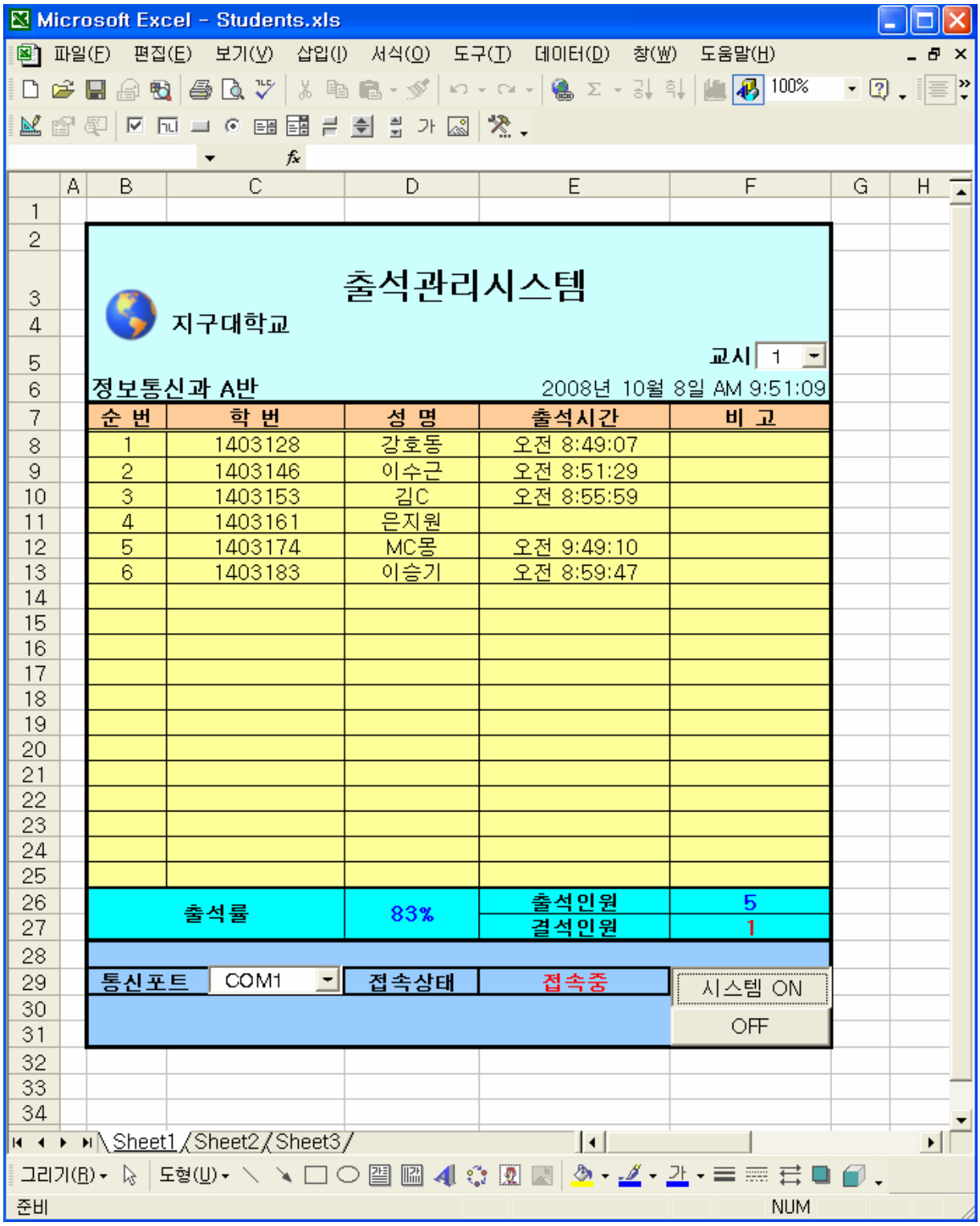
CommClose는 현재 설정되어 있는 통신포트를 종료하는 함수이다.
인자로서 **CommOpen**시 전달받은 Handle을 사용한다.

```
Private Sub CommandButton2_Click()

    If [i29] = 0 Then Exit Sub
    CommClose [i29]
    [i29] = 0
    [e29] = "끊어짐"

End Sub
```

[Step.6] 전체 동작 테스트

화면 구성과 컨트롤 프로그램이 완료되면 디자인 모드를 빠져나와 실행 모드에서 각 컨트롤 버턴을 클릭하면서 동작을 확인다.
리더 모듈에 태그를 하나씩 접근시켜서 출석 현황이 제대로 출력되는지 확인한다.
동작이 양호하면 색깔과 테두리를 적절히 설정하여 화면을 디자인한다.

예제 프로젝트에서는 구현되지 않았지만, 데이터베이스에 기록된 내용을 파일로 저장하는 로깅 기능을 추가하거나 여러 가지 섬세하고 편리한 기능을 추가하면 보다 더 실용적인 출석관리시스템이 될 것이다.



2) 재고관리시스템 (Visual C++ 6.0 사용)

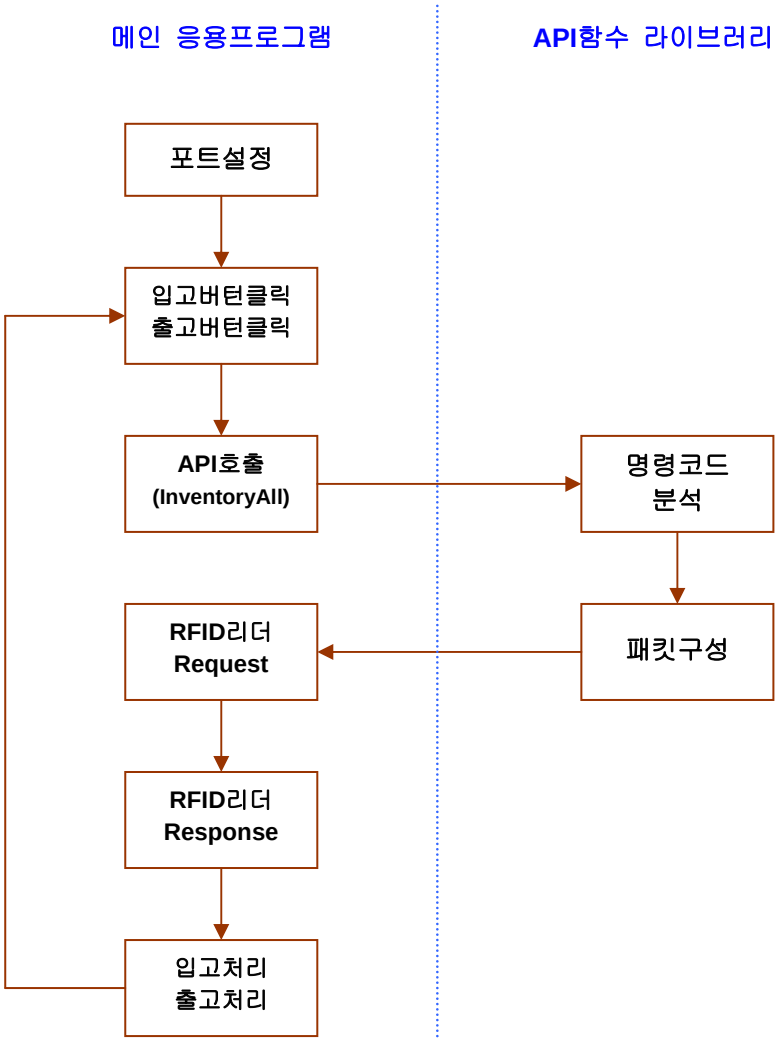
상품의 재고나 물류를 관리하기 위하여 과거로부터 일반적으로 바코트시스템이 사용되어 오고 있다. 그러나 바코드시스템은 태그의 단가가 낮고 취급이 간편하다는 장점이 있는 반면에 태그의 정보를 갱신하지 못하고 인식을 위해서는 반드시 바코드가 보이도록 해야 한다는 단점이 있다. 한편 이러한 바코드시스템의 단점은 RFID 기술을 사용하여 극복할 수 있으며 RFID시스템의 장점이라 할 수 있다.

본 프로젝트에서는 국내외적으로 점차 되입이 확산되어 가고 있는 RFID기술의 하나의 예로서 간단한 재고관리시스템의 호스트프로그램을 **Visual C++ 6.0**을 사용하여 제작해 본다.

호스트프로그램은 각기 응용마다 혹은 제작자마다 다양한 구조로 제작될 수 있다. 본 예제는 메인 응용프로그램과 API함수 라이브러리 구조로 되어 있으며, 독자들이 필요에 따라 함수를 라이브러리 형태의 API로 제작할 수 있는 기술을 습득할 수 있도록 하였다.

메인 응용프로그램은 GUI 화면을 생성하고, 사용자를 위한 입출력 기능을 구현하고, 시리얼통신을 구축하여 RFID리더와 데이터를 송수신하고, 재고관리시스템 전체의 흐름을 제어한다.

API함수 라이브러리는 메인 응용프로그램으로부터 특정 명령어(예로서, **Inventory-All**) 코드를 수신하고 그 명령어에 대한 패킷을 구성하여 리턴하는 기능을 수행한다. 라이브러리는 '**MFC AppWizard(dll)**'을 사용하여 '**Regular DLL using shared MFC DLL**', 즉 '**정규 MFC 공유 DLL**' 형태로 제작한다.

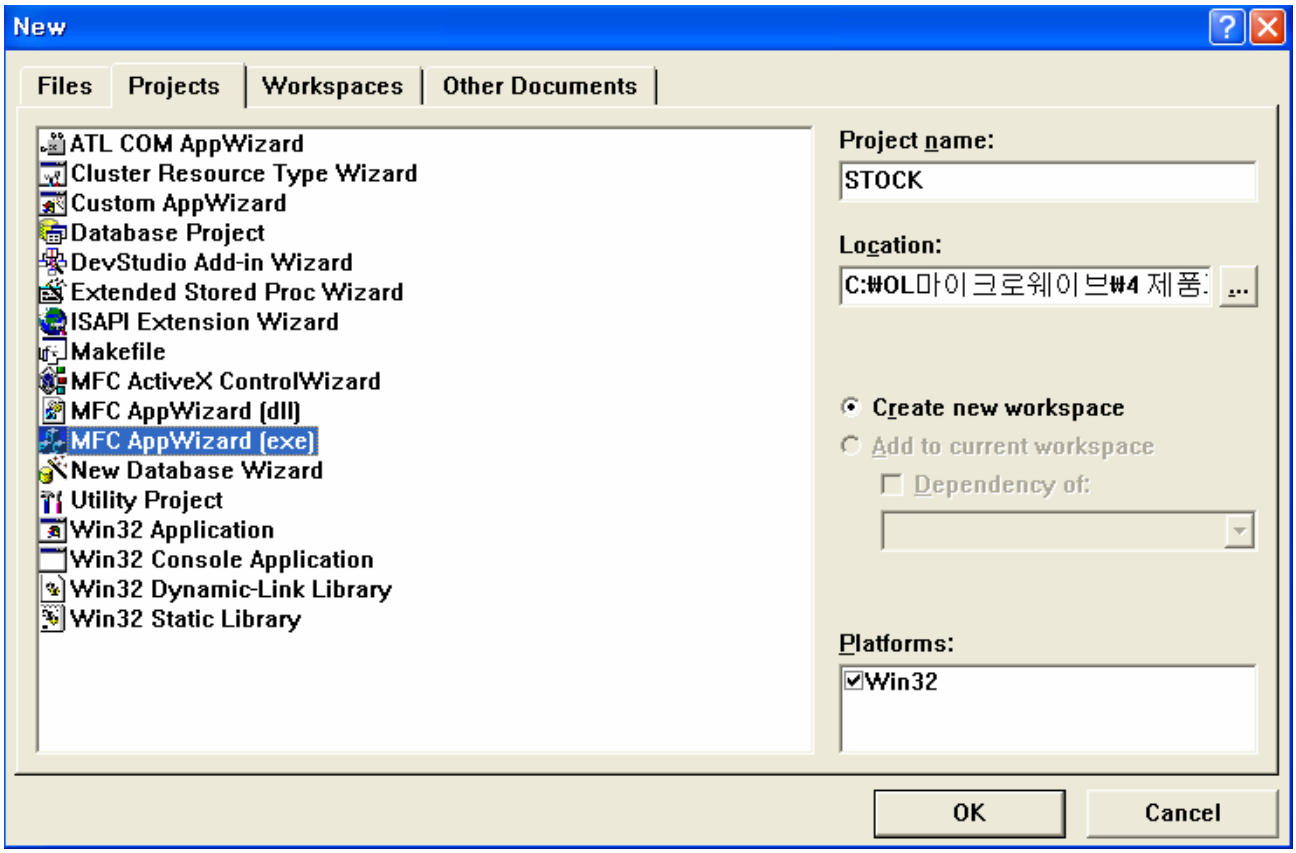

메인 응용프로그램
API함수 라이브러리
포트설정
입고버턴클릭
출고버턴클릭
API호출
(InventoryAll)
명령코드
분석
RFID리더
Request
패킷구성
RFID리더
Response
입고처리
출고처리

<순서도>

가. 메인 응용프로그램 제작

a). 기본 화면 생성

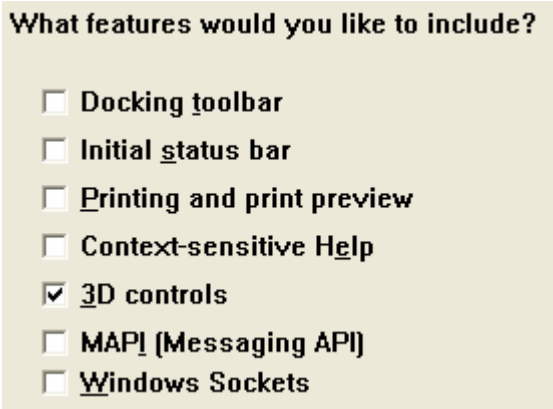
- **Visual C++ 6.0**을 실행한다.
- **File** 메뉴에서 **New**를 클릭하여 다음과 같이 **MFC AppWizard(exe)** 프로젝트를 생성한다.



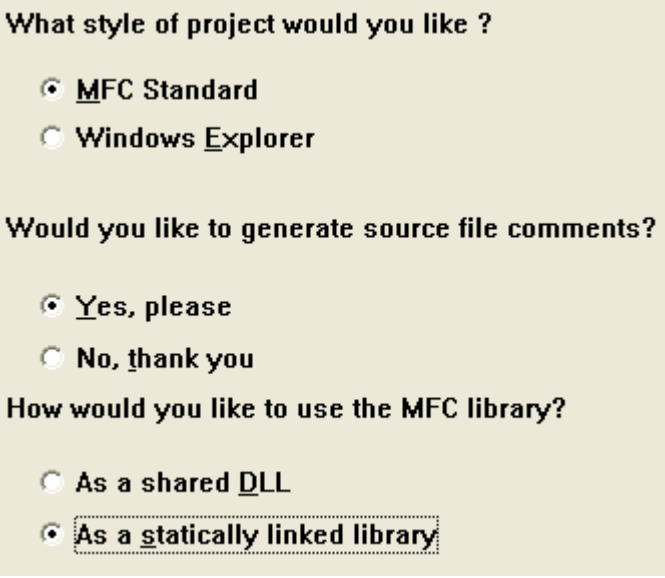
- MFC AppWizard-Step 1 화면에서 Single Document를 선택한다.

• Next를 2회 클릭하여 MFC AppWizard-Step 3 of 6 화면에서 ActiveX Control을 체크 해제한다.

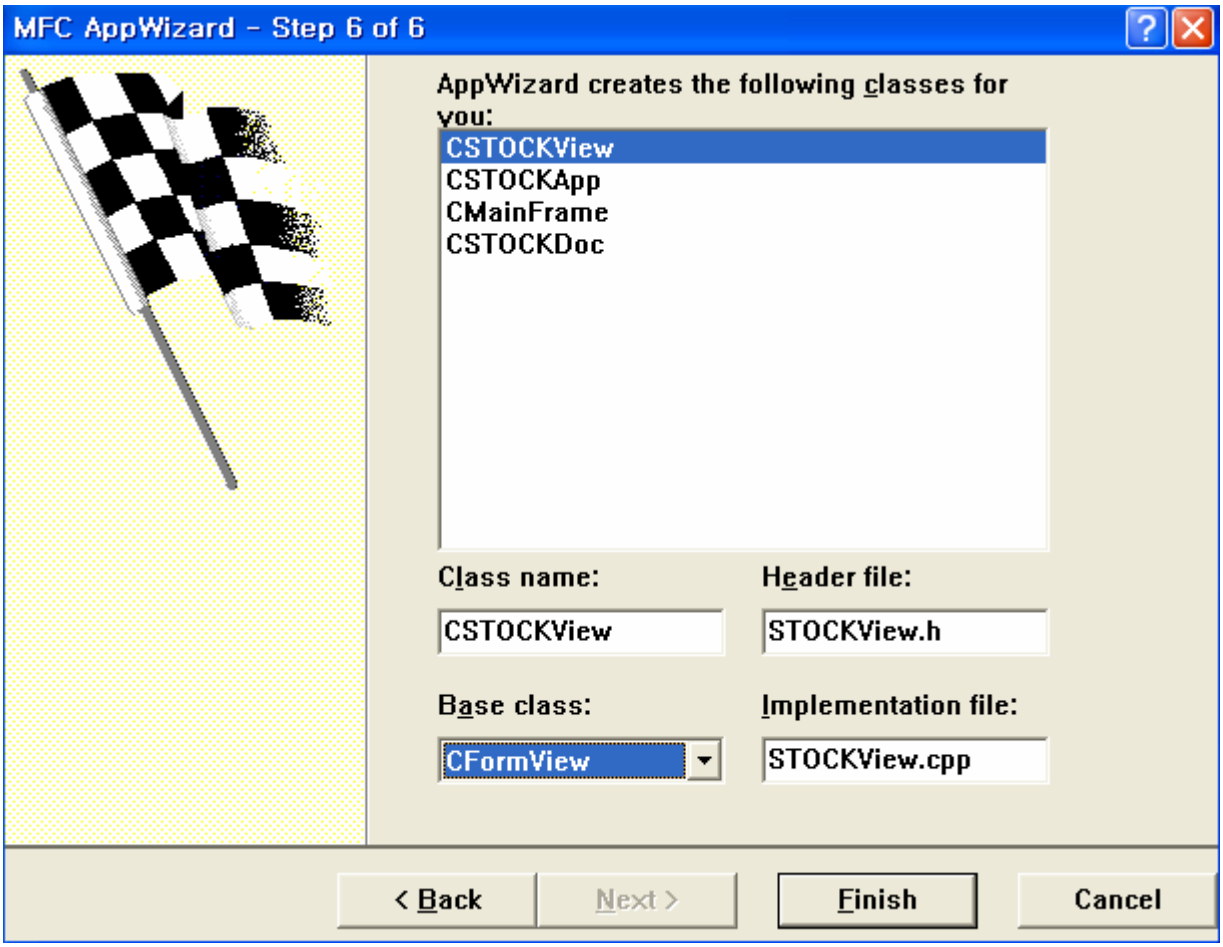
• Next를 1회 클릭하여 MFC AppWizard-Step 4 of 6 화면에서 3D controls를 제외하고 모두 체크 해제한다.



• Next를 1회 클릭하여 MFC AppWizard-Step 5 of 6 화면에서 다음과 같이 체크한다.



• Next를 1회 클릭하여 MFC AppWizard-Step 6 of 6 화면에서 CSTOCKView 클래스의 Base class를 CFormView로 선택하고 Finish, OK를 클릭한다.



b). 시리얼통신 구현

[Step.1] 관련 파일 다운로드

당사 홈페이지(**http://www.olmicrowaves.com**)의 **Downloads** 메뉴 아래 **RFID** 카테고리에서 다음의 파일을 다운로드하여 앞의 과정에서 생성된 **STOCK** 디렉토리에 저장한다.

- **Serial.h**
- **Serial.cpp**

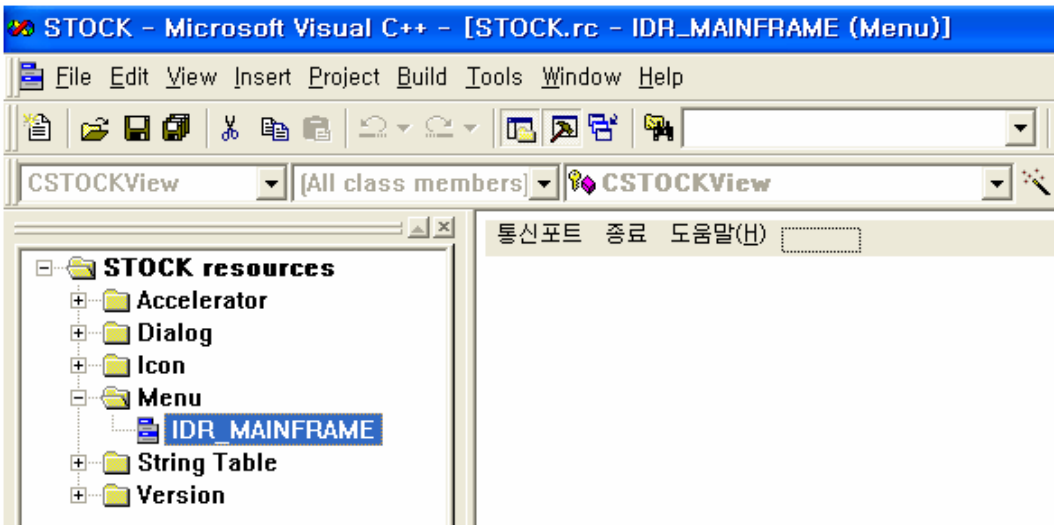
[Step.2] 시리얼통신 모듈 삽입

Resource View 창을 열고 **STOCK files** 위에서 마우스 오른쪽 버턴을 클릭하여 팝업 메뉴를 열고 **Add Files to Project**를 클릭하여 **Serial.h**와 **Serial.cpp**를 프로젝트에 삽입한다.

[Step.3] 메뉴 제작

• **Resource View** 창을 열고 **Menu** 아래에 있는 **IDR_MAINFRAME**을 더블클릭하여 메뉴 리소스 편집화면을 연다.

• **파일**과 **편집** 메뉴를 삭제하고 점선의 네모상자를 더블클릭하여 **통신포트**와 **종료** 메뉴를 추가한다.



• **통신포트**와 **종료** 메뉴 각각 속성(**Properties)**창에서 **Pop-up**을 체크해제하고 ID를 **ID_COMM**과 **ID_APP_EXIT**로 설정한다.

[Step.4] 통신포트 및 종료 메뉴 함수 생성

• **ClassWizard**(**View** 메뉴)를 연다.

• **Class name:**을 **CSTOCKView**, **Object Ids:**를 **ID_COMM**, **Messages:**를 **COMMAND**로 선택하고 **Add Function**과 **OK**를 차례로 클릭하여 통신포트 메뉴 구현 함수인 **OnComm()**을 생성한다.

• **Class name:**을 **CSTOCKView**, **Object Ids:**를 **ID_APP_EXIT**, **Messages:**를 **COMMAND**로 선택하고 **Add Function**과 **OK**를 차례로 클릭하여 종료 메뉴 구현 함수인 **OnAppExit()**을 생성한다.

• **OK**를 클릭하여 **ClassWizard**를 닫는다.

[Step.5] 통신포트 하위 대화상자 생성

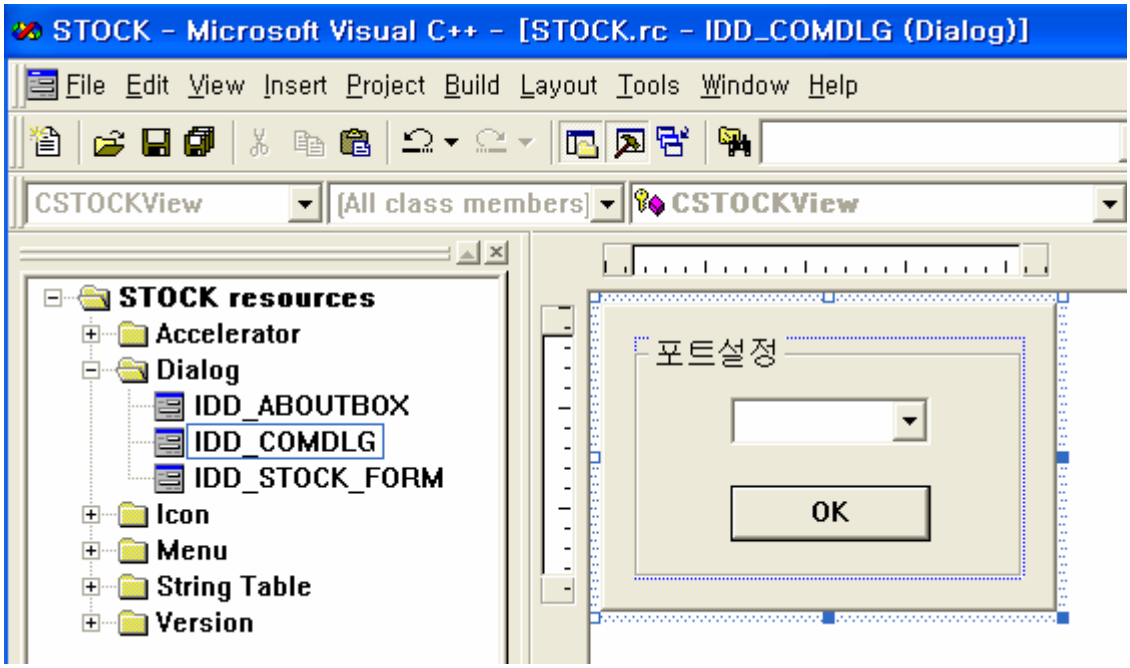
• **Resource View** 창을 열고 **Dialog** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Insert Dialog**를 클릭하여 새로운 대화상자를 만들고 속성 부메뉴를 열어서 **General**탭에서 ID를 **IDD_COMDLG**로 설정하고, **Styles**탭에서 **Title bar**를 체크 해제한다.

• **CANCEL** 버튼을 삭제한다.

• **Combo Box**를 1개 추가하고 속성 부메뉴를 열어서 **General**탭에서 ID를 **IDC_COMPORT**로 설정하고, **Data**탭에서 COM1~COM9까지 입력한다.
(※ 줄바꿈: **Ctrl+Enter)**

• **Combo Box**의 ▼를 클릭하고 점선상자를 아래로 드래그하여 가시영역을 설정한다.

• **Combo Box**와 **OK**버튼을 적절히 위치시키고 **Group Box**로 둘러싸고 **Group Box**의 **Caption**을 **포트설정**으로 입력한다.



[Step.6] IDD_COMDLG 를 위한 CComDlg 클래스 생성

• **ClassWizard(View** 메뉴)를 연다.

• **IDD_COMDLG**에 대한 새로운 클래스를 생성하겠는지 여부를 묻는 창이 뜨면 **OK**를 클릭하고, 그렇지 않으면 **Add Class**, **New**를 클릭한다.

• **Name**에 **CComDlg**로 입력하고, **Base class**를 **Cdialog**로 설정하고, **Dialog ID**를 **IDD_COMDLG**로 설정하고 **OK**를 클릭하여 **ClassWizard**를 닫는다.

[Step.7] CComDlg 클래스에 초기화 함수 추가

• **ClassWizard**를 열고, **Class name**에 **CComDlg**, **Object IDs**에 **CComDlg**, **Messages**에 **WM_INITDIALOG**를 선택하고 **Add Function**을 클릭하면 **OnInit-Dialog** 함수가 생성된다. **OK**를 클릭하여 **ClassWizard**를 닫는다.

[Step.8] CComDlg 클래스에 통신포트 설정을 위한 함수 추가

• **ClassWizard**를 열고, **Class name**에 **CComDlg**, **Object IDs**에 **IDC_COMPORT**, **Messages**에 **CBN_SELCHANGE**를 선택하고 **Add Function**을 클릭하면 **OnSel-changeComport** 함수가 생성된다. **OK**를 클릭하여 **ClassWizard**를 닫는다.

[Step.9] CComDlg 클래스에 필요한 변수 추가

• m_com : Combo Box로부터 통신포트 값을 전달하기 위한 변수.

ClassWizard를 열고, Member Variables탭을 클릭하고, **Class name**에 **CComDlg**, **Control IDs**에 **IDC_COMPORT**를 선택하고 **Add Variable**을 클릭한다. **Member variable name**에 **m_com**을 입력하고, **Category**에 **Control**을 선택하고, **Variable type**에 **CCombo Box**를 선택하고 **OK**를 클릭하여 **ClassWizard**를 닫는다.

• com : 통신포트 값을 **CComDlg** 클래스로부터 다른 클래스로 전달하기 위한 변수.

Class View 창을 열고 **CComDlg** 위에서 마우스 오른쪽 버턴을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **int**, **Varible Name**에 **com**을 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

[Step.10] OnSelchangeComport 함수를 작성한다.

CComDlg 클래스 아래 **OnSelchangeComport()**를 더블클릭하여 열고 다음의 소스를 코딩해 넣는다. 이 프로그램은 콤보박스에서 통신포트를 선택했을 때 선택된 포트값을 전역변수 **com** 에 저장하는 기능을 수행한다.

```
int tmp = m_com.GetCurSel();
if(tmp != CB_ERR)
{
        CString tmpr;
        m_com.GetLBText(tmp, tmpr);
        tmpr.Delete(0, 3);
        com = atoi(tmpr);
        if(com<=0) com = 0;
}
```

[Step.11] CSTOCKView 클래스에 필요한 변수 추가

• m_pSerial : 통신모듈인 CSERIAL클래스 타입의 변수로서 통신설정값을 전달.

Class View 창을 열고 **CSTOCKView** 위에서 마우스 오른쪽 버턴을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **CSERIAL**, **Varible Name**에 **m_pSerial**을 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

[Step.12] 통신포트 및 종료 메뉴 함수 작성

• **CSTOCKView** 클래스를 더블클릭하여 **STOCKView.h** 를 열고 상단에 **ComDlg.h** 를 추가한다.

```
// STOCKView.h : interface of the CSTOCKView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_STOCKVIEW_H__892656D8_52A0_40CD_9011_2AC456A0940E__INCLUDED_)
#define AFX_STOCKVIEW_H__892656D8_52A0_40CD_9011_2AC456A0940E__INCLUDED_

#include "Serial.h"     // Added by ClassView
#include "ComDlg.h"

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
```

• **CSTOCKView** 클래스 아래 **OnComm()**를 더블클릭하여 열고 다음의 소스를 코딩해 넣는다. 이 프로그램은 메뉴에서 **통신포트**를 클릭했을 때 **포트설정** 다이얼로그 박스가 팝업되고, 포트를 선택할 수 있도록 하는 기능을 수행한다.

```
CComDlg dlog;
if(dlog.DoModal() == IDOK)
{
        if((dlog.com > 0) && (dlog.com < 125))
        {
                m_pSerial.Stopcom();
                m_pSerial.Setting(dlog.com,115200,8,0,0); // 115.2Kbps,8 data bits
                m_pSerial.Init();
                m_pSerial.Startcom();
                m_pSerial.SetHwnd(this->m_hWnd);
        }
        else MessageBox("Out of COM range!");
}
```

• **CSTOCKView** 클래스 아래 **OnAppExit()**를 더블클릭하여 열고 다음의 소스를 코딩해 넣는다. 이 프로그램은 메뉴에서 **종료**를 클릭했을 때 설정된 통신포트를 해제하고 메인 응용프로그램을 종료하는 기능을 수행한다.

```
m_pSerial.Release();
ExitProcess(TRUE);
```

[Step.13] 통신포트 설정 기능 확인

지금까지의 작업을 저장하고 **Rebuild All**을 수행하여 **Error**와 **Warning**이 없는 것을 확인후 프로그램을 실행해본다.

통신포트 메뉴를 클릭하여 통신포트를 설정해본다.

종료 메뉴를 클릭하여 프로그램을 종료한다.

c). 재고관리 기능 구현

본 예제에서 구현할 재고관리 기능은 두 개의 버턴(입고관리, 출고관리)을 사용하여 제품(태그)을 입고처리하거나 출고처리하고 재고 리스트를 보여주는 간단한 동작을 수행한다. 입고처리와 출고처리 함수에서 사용되는 RFID 명령어 패킷을 API 함수 라이브러리를 호출하여 얻어 내는 방식으로 프로그램을 작성해본다.

[Step.1] 컨트롤 배치

• **Resource View** 창을 열고 **Dialog** 아래에 있는 **IDD_STOCK_FORM**을 더블클릭하여 메인 대화상자 리소스를 열고 대화상자의 크기를 적당히 크게한다.

• 대화상자 상단에 **Static Text**를 추가하고 **Caption**에 **재고관리시스템**으로 입력한다.

• **Push Button**을 3개를 가로 방향으로 차례로 추가하고 속성 부메뉴에서 **ID**와 **Caption**을 다음과 같이 지정한다.

IDC_BUTTON_STORE	**상품입고**
IDC_BUTTON_STOCK	**재고현황**
IDC_BUTTON_DELIVER	**상품출고**

• 상품입고 버턴 아래에 **List Box** 2개를 세로 방향으로 추가한다.

IDC_LIST1
IDC_LIST2

• 상품출고 버턴 아래에 **List Box** 2개를 세로 방향으로 추가한다.

IDC_LIST3
IDC_LIST4

• 재고현황 버턴 아래에 **List Box** 1개를 추가한다.

IDC_LIST5

• 각 **List Box**에 다음과 같이 **Static Text**를 추가한다.

UID (List Box1 앞)
입고시간 (List Box2 앞)
UID (List Box3 앞)
출고시간 (List Box4 앞)
UID 입고시간 (List Box5 위)

• **ClassWizard**를 열고, Member Variables탭을 클릭하고, **Class name**에 **CSTOCKView**, **Control IDs**에 **IDC_LIST1**를 선택하고 **Add Variable**을 클릭한다. **Member variable name**에 **m_Uidstore**를 입력하고, **Category**에 **Control**을 선택하고, **Variable type**에 **CListBox**를 선택하고 **OK**를 클릭하여 변수를 추가한다. Add Varialble을 계속하여 같은 방식으로 다음과 같이 변수를 추가한다.

IDC_LIST2 m_Timestore
IDC_LIST3 m_Uiddeliver
IDC_LIST4 m_Timedeliver
IDC_LIST5 m_Stock

[Step.2] 상품입고 버턴 구현

상품입고 버턴을 더블클릭하여 **OnButtonStore** 함수를 추가하고 다음의 소스를 코딩해 넣는다. 이 프로그램은 API 함수인 **GetPacket()**을 호출하여 RFID Request 패킷을 구성하고 시리얼통신 포트로 출력하는 기능을 수행한다.

```
int length;
unsigned char PACKET[100];

GetPacket(0x01, PACKET); // InvestoryAll packet called

length = PACKET[0];

m_pSerial.Output(&PACKET[0], length+1);

direction = 0;
```

[Step.3] 상품출고 버턴 구현

상품출고 버턴을 더블클릭하여 **OnButtonDeliver** 함수를 추가하고 다음의 소스를 코딩해 넣는다. 이 프로그램의 기능은 **상품입고** 버턴과 유사하며 입출고 플래그(direction)를 세트(1)하여 출고로 설정한다.

```
int length;
unsigned char PACKET[100];

GetPacket(0x01, PACKET); // InvestoryAll packet called

length = PACKET[0];

m_pSerial.Output(&PACKET[0], length+1);

direction = 1;
```

[Step.4] 시리얼통신 데이터 수신메시지 핸들러 생성

• **CSTOCKView** 클래스를 더블클릭하여 **STOCKView.h** 를 열고 시리얼통신 데이터 수신메시지 핸들을 위한 afx_msg 함수 선언을 추가한다.

```
// Generated message map functions
protected:
    //{{AFX_MSG(CSTOCKView)
    afx_msg void OnComm();
    afx_msg void OnAppExit();
    afx_msg void OnButtonStore();
```

```
afx_msg void OnButtonDeliver();
//}}AFX_MSG
afx_msg BOOL OnSerialreceive(UINT WParam , LONG LParam);
DECLARE_MESSAGE_MAP()
```

• **STOCKView.cpp** 를 열고 상단의 Message Map 부분에 시리얼통신 데이터 수신메시지 핸들러를 위한 Message Map 을 추가한다.

```
BEGIN_MESSAGE_MAP(CSTOCKView, CFormView)
    //{{AFX_MSG_MAP(CSTOCKView)
    ON_COMMAND(ID_COMM, OnComm)
    ON_COMMAND(ID_APP_EXIT, OnAppExit)
    ON_BN_CLICKED(IDC_BUTTON_STORE, OnButtonStore)
    ON_BN_CLICKED(IDC_BUTTON_DELIVER, OnButtonDeliver)
    //}}AFX_MSG_MAP
    ON_MESSAGE(WM_RECEIVEDATA, OnSerialreceive)
END_MESSAGE_MAP()
```

[Step.5] 시리얼통신 데이터 수신메시지 핸들러 함수 작성

• 재고관리시스템 기능 수행을 위한 몇 가지 변수 추가

Class View 창을 열고 **CSTOCKView** 위에서 마우스 오른쪽 버턴을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 다음의 변수를 추가한다.

unsigned char	**warehouse[10][20]**	**Public**
int	**emptyshelf[11]**	**Public**
int	**direction**	**Public**

• 변수 및 컨트롤 초기화

CSTOCKView 클래스 아래 **OnInitialUpdate()**를 더블클릭하여 열고 다음의 초기화 코드를 추가한다.

```
// 사용자 초기화
m_Uidstore.ResetContent();
m_Timestore.ResetContent();
m_Stock.ResetContent();
m_Uiddeliver.ResetContent();
m_Timedeliver.ResetContent();

for(int tmp=0; tmp<10; tmp++)
 {
        for(int tmp1=0; tmp1<20; tmp1++)
```

```
		{
			warehouse[tmp][tmp1] = 0;
		}
	}

	emptyshelf[0] = 10;
	for(tmp=1; tmp<=10; tmp++)
	{
		emptyshelf[tmp] = 0;
	}
```

• 시리얼통신 데이터 수신메시지 핸들러 함수 **OnSerialreceive** 작성

STOCKView.cpp 를 열고 맨 하단에 다음의 소스를 코딩하여 추가한다.
이 프로그램은 재고관리시스템의 주 프로세서로서, 시리얼통신 수신버퍼로부터 RFID 리더로부터의 응답 데이터(UID)를 읽고, 상품입고 혹은 상품출고 처리를 수행하고, 재고현황을 갱신하고 화면으로 출력하는 기능을 한다.

```
BOOL CSTOCKView::OnSerialreceive(UINT WParam, LONG LParam)
{
	int tmp, tmp1, length;
	unsigned char tmpc, rxd[100];
	CString uid, tmpr;

	length = (int)WParam;
	tmpr = "";

	for(tmp=0; tmp<length; tmp++)
	{
		m_pSerial.bufread.Rxchar(&tmpc); // 시리얼 수신버퍼에서 1바이트 읽는다.
		tmpr.Format("%02x",tmpc);
		uid += tmpr;
		rxd[tmp] = tmpc;
	}

	// 컨트롤 초기화
	m_Uidstore.ResetContent();
	m_Timestore.ResetContent();
	m_Stock.ResetContent();
	m_Uiddeliver.ResetContent();
	m_Timedeliver.ResetContent();

	int uidcompare = 0;
	for(tmp=0; tmp<10;tmp++) // 기존에 UID가 저장되어 있는지 확인
	{
		if(memcmp(&warehouse[tmp][0],&rxd[3],8) == 0) uidcompare = tmp +1;
	}

	if(direction == 0) // 상품입고 처리
	{
```

```
		if(length == 11 && uidcompare == 0) // 정상 UID 입력, 해당 UID 없음
		{
			if(emptyshelf[0] >0) // 창고에 UID가 1개 이상 저장되어 있음
			{
				int shelf = 0;
				while(emptyshelf[shelf+1] == 1) shelf++;

				for(tmp=3; tmp<=10; tmp++)
				{
					warehouse[shelf][tmp-3] = rxd[tmp];
				}
				emptyshelf[shelf+1] = 1; // 해당 쉘프 플래그를 Full로 설정
				emptyshelf[0]--; // Empty 쉘프 개수 감소

				SYSTEMTIME st;
				GetLocalTime(&st);

				warehouse[shelf][8] = (unsigned char)st.wHour;
				warehouse[shelf][9] = (unsigned char)st.wMinute;

				m_Uidstore.InsertString(0, uid.Mid(6,16));

				CString szTime;
				szTime.Format("%02d:%02d", st.wHour, st.wMinute);
				m_Timestore.InsertString(0, szTime);
			}
		}
	}

	if(direction == 1) // 상품출고 처리
	{
		if(length == 11 && uidcompare != 0) // 정상 UID 입력, 해당 UID 있음
		{
			for(tmp=0; tmp<=9;tmp++)
			{
				warehouse[uidcompare-1][tmp] = 0; // 창고에 해당 UID Clear
			}

			emptyshelf[uidcompare] = 0; // 해당 쉘프 플래그를 Empty로 설정
			emptyshelf[0]++; // Empty 쉘프 개수 증가

			SYSTEMTIME st;
			GetLocalTime(&st);

			m_Uiddeliver.InsertString(0, uid.Mid(6,16));

			CString szTime;
			szTime.Format("%02d:%02d", st.wHour, st.wMinute);
			m_Timedeliver.InsertString(0, szTime);
		}
	}

	// 재고현황 출력
	for(tmp=0; tmp<10; tmp++)
```

```
    {
        uid = "";
        for(tmp1=0; tmp1<=7; tmp1++)
        {
            tmpr.Format("%02x",warehouse[tmp][tmp1]);
            uid += tmpr;
        }

        tmpr.Format(" %02d:",warehouse[tmp][8]);
        uid += tmpr;
        tmpr.Format("%02d",warehouse[tmp][9]);
        uid += tmpr;

        m_Stock.InsertString(0, uid.Mid(0,22));
    }

    return TRUE;
}
```

지금까지의 작업을 저장하고 **Rebuild All**을 수행하여 **'GetPacket' : undeclared identifier**라는 오류 메시지 한 개만 발생하는 것을 확인한다. 이 것은 아직 API함수 라이브러리를 제작하지 않았기 때문에 발생하는 것이다.

나. API함수 라이브러리 제작

본 프로젝트에서는 API함수 라이브러리를 MFC DLL 형태로 제작해 본다.

DLL(Dynamic Link Library)이란 라이브러리가 프로그램 외부에 따로 독립적으로 실행 가능한 파일로 존재하고 필요시 로드할 수 있는 형태의 라이브러리를 말한다. DLL은 프로그램을 작성할 때 각 모듈별로 나누어서 프로그램을 작성하는 것을 가능하게 한다. 따라서 유지보수가 용이하고 여러 사람이 분담하여 프로그램을 제작할 수 있다. 반복적으로 이용되는 모듈은 DLL로 제작하여두면 프로그램을 효율적으로 사용할 수 있고, 프로그램 크기도 줄일 수 있다.

DLL의 종류

DLL에는 정규 DLL과 확장 DLL이 있다. 정규 DLL은 Win32 프로그램으로 설정되어 MFC를 사용하지 않은 다른 프로그램과도 원활히 연결할 수 있는 DLL이고 확장 DLL은 MFC 전용 프로그램을 위한 DLL이다.

정규(Regular) DLL

정규 DLL은 Win32 프로그래밍 환경에서 만든다. 즉 클래스 형태가 아닌 C 함수 형태로 DLL을 제작하기 때문에 정규 DLL은 MFC를 사용하지 않은 다른 프로그램에서도 사용할 수 있다. 그러나 내부적으로는 클래스 사용이 가능하다. 정규 DLL이 MFC 라이브러리를 사용할 경우 라이브러리를 공유하거나(Shared) 아니면 자체적으로 모두 가지고 있거나(Static) 둘 중 하나를 선택할 수 있다.

정규 DLL 작성법

void GetPacket()라는 함수를 DLL로 제작하는 경우 함수 선언시 다음과 같이 설정하여야 한다. 그러면 **GetPacket()** 함수는 DLL 외부에서 호출하여 사용할 수 있는 함수가 된다.

```
extern "C" __declspec (dllexport) void GetPacket ();
```

그런 다음, 함수명 앞에 **_declspec(dllexport)**를 붙이고 작성한다.

```
__declspec (dllexport) void GetPacket()
{
        함수 내용
}
```

함수 작성 완료후 **Rebuild All**하면 DLL 파일과 LIB 파일이 생성된다.

정규 DLL 사용법

응용프로그램에서 **void GetPacket()** 함수를 호출하려면 선언부에 다음과 같이 설정한다.

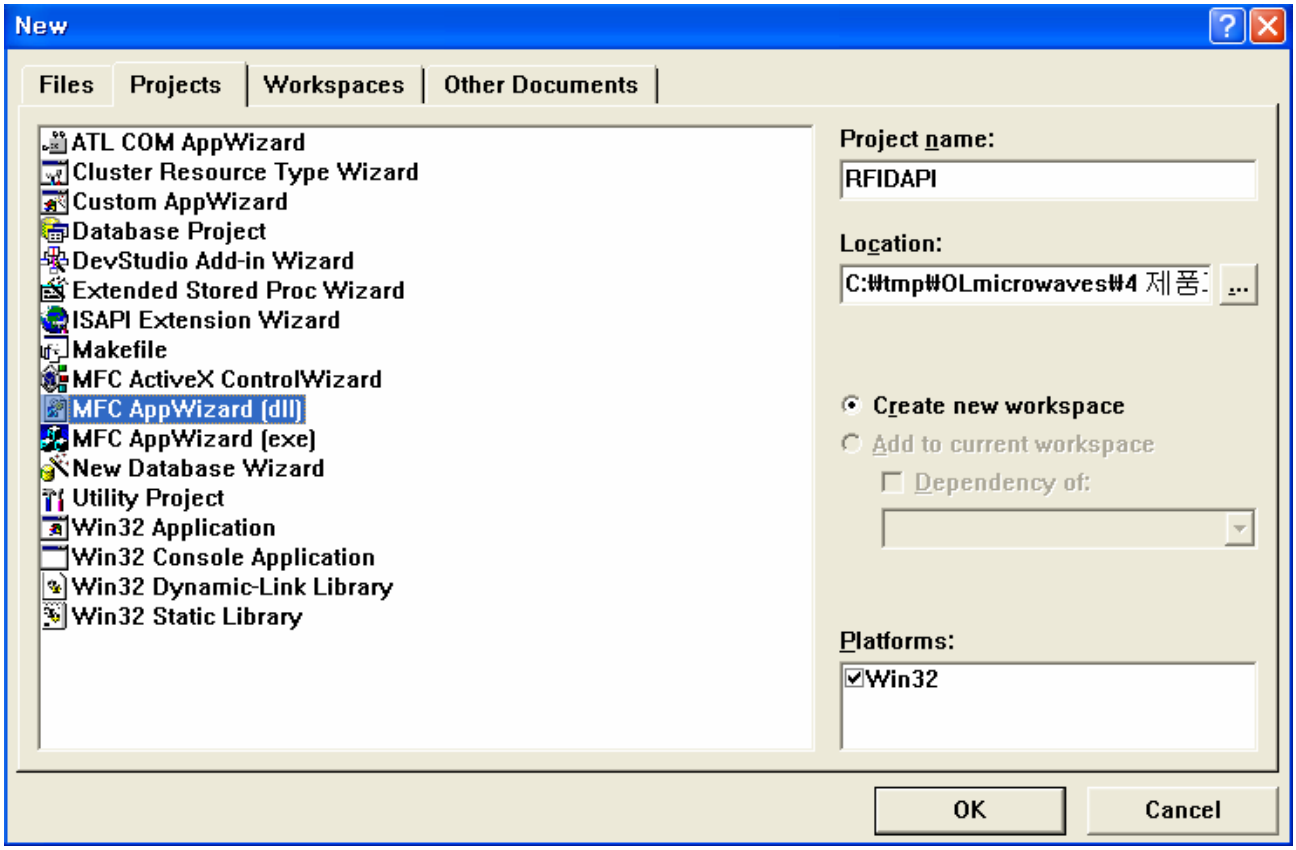
```
extern "C" __declspec (dllimport) void GetPacket ();
```

그리고 **Project** 메뉴 **Settings**에서 **Link**탭을 선택하여 앞서 생성된 LIB 파일명을 설정해 준다. (LIB 파일의 전체 경로를 지정하거나 프로젝트 디렉토리에 DLL 파일과 LIB 파일을 복사해 둔다.)

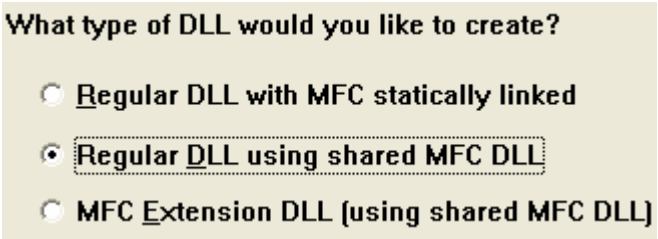
[Step.1] 기본 파일 생성

• **Visual C++ 6.0**을 실행한다.

• **File** 메뉴에서 **New**를 클릭하여 다음과 같이 **MFC AppWizard(dll)** 프로젝트를 생성한다.



• MFC AppWizard-Step 1 of 1 화면에서 Regular DLL using shared MFC DLL을 선택하고 Finish, OK를 클릭한다.



[Step.2] API 함수 작성

• **Class View** 창을 열고 **CRFIDAPIApp** 클래스 아래 **CRFIDAPIApp()**를 더블클릭하여 열고 맨 아래에 다음의 코드를 추가한다. 이 프로그램은 응용프로그램으로부터 호출되는 GetPacket() 함수를 선언하고 구현한다.

```
// API 함수 선언 및 구현
extern "C" __declspec (dllexport) void GetPacket(unsigned char CMD, unsigned char *PACKET);

__declspec (dllexport) void GetPacket(unsigned char CMD, unsigned char *PACKET)
{
	switch (CMD)
	{
		case 0x01 : break;		// 필요에 따라 추가

		case 0x20 : break;		// 필요에 따라 추가

		case 0x21 : break;		// 필요에 따라 추가

		// InventoryAll 명령어 패킷
		case 0xf0 : PACKET[0] = 0x02;
				PACKET[1] = 0x00;
				PACKET[2] = 0xf0;
				break;

		default : break;
	}
}
```

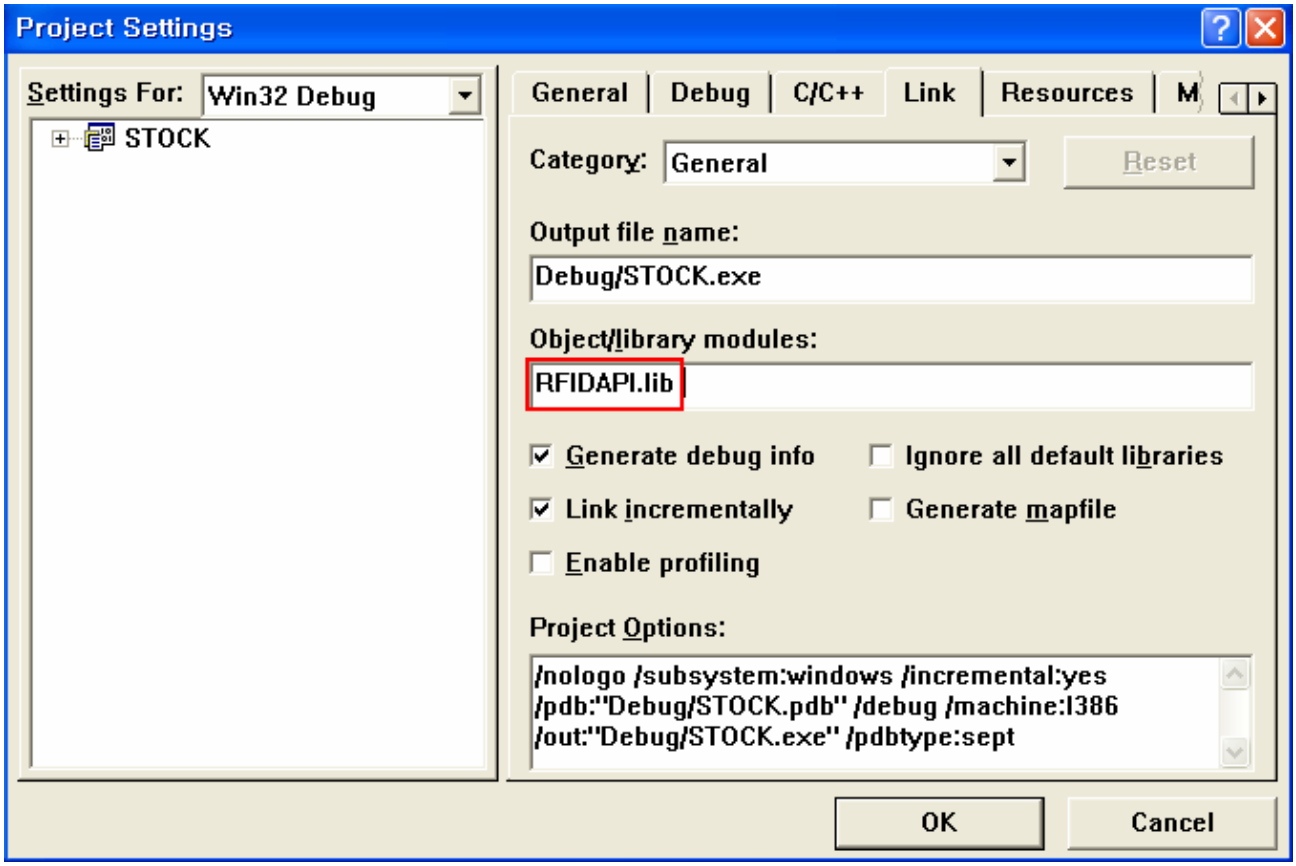
지금까지의 작업을 저장하고, **Rebuild** 메뉴의 **Set Active Configuration**에서 **Win32 Release**를 선택한 후 **Rebuild All**을 수행하여 **RFIDAPI.dll** 파일과 **RFIDAPI.lib** 파일을 생성한다.

다. API함수 라이브러리의 사용

• RFIDAPI 프로젝트의 Release 디렉토리에 있는 **RFIDAPI.dll** 파일과 **RFIDAPI.lib** 파일을 **STOCK** 프로젝트에 복사한다.

• **Visual C++ 6.0**을 실행하고 **STOCK** 프로젝트를 연다.

• **Project** 메뉴에서 **Settings**를 클릭하고 **Link**탭을 선택하여 다음과 같이 LIB 파일명을 설정해 준다.



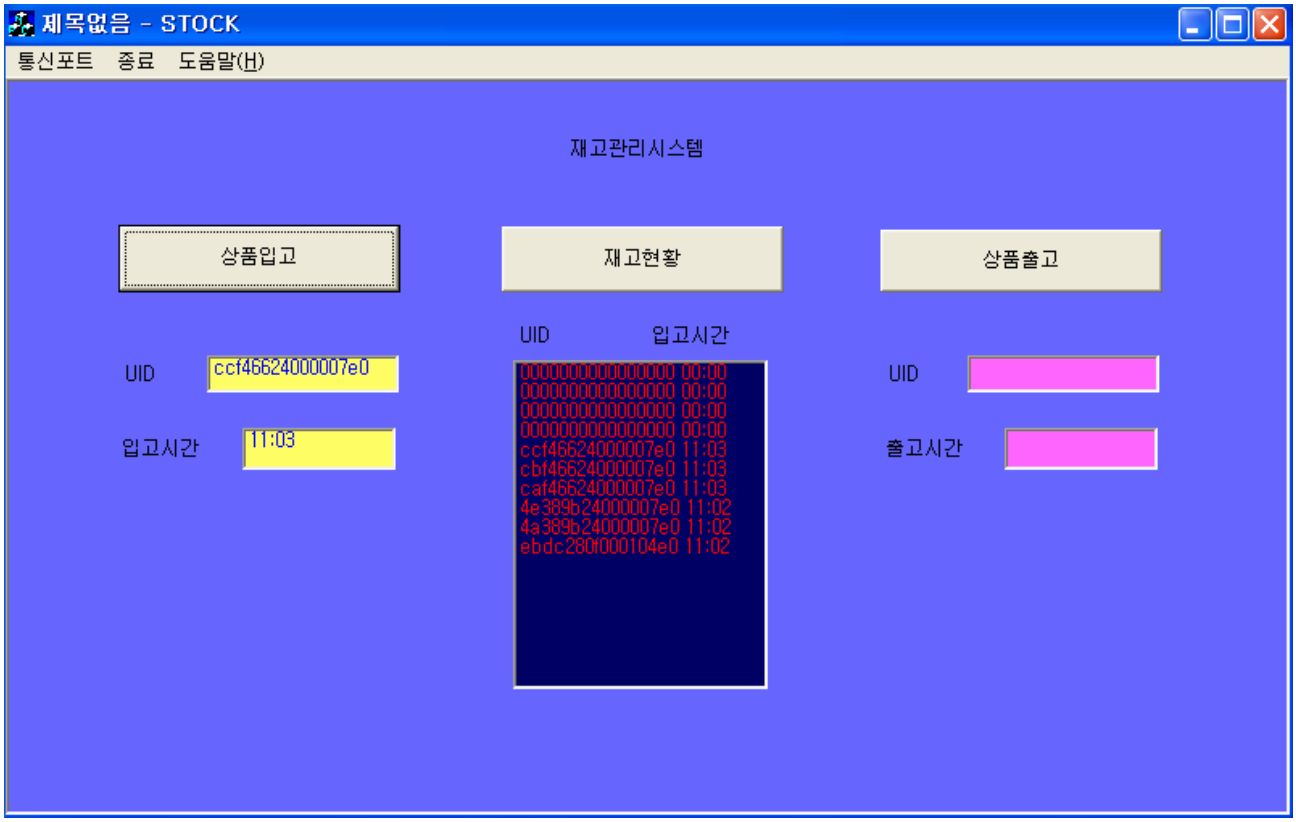
• **Class View** 창을 열고 **CSTOCKView** 클래스를 더블클릭하여 **STOCKView.h**를 열고 맨 아래에 다음의 코드를 추가한다.

```
extern "C" __declspec (dllimport) void GetPacket(unsigned char CMD, unsigned char *PACKET);
```

• 지금까지의 작업을 저장하고, **Rebuild** 메뉴의 **Set Active Configuration**에서 **Win32 Release**를 선택한 후 **Rebuild All**을 수행하여 **STOCK.exe** 파일을 생성한다.

• **RFIDAPI.dll** 파일과 **STOCK.exe** 파일을 동일한 디렉토리에 두고 **STOCK.exe**를 실행하여 메인 응용프로그램의 재고관리시스템 기능을 확인한다.

• **WM_CTRCOLOR**를 사용하여 적당한 색깔로 화면을 꾸민다.
※ 예제 프로젝트 소스의 **OnCtrColor()** 함수 참조



- 감사합니다 -

OL마이크로웨이브

홈페이지: http://www.olmicrowaves.com
E-mail: webmaster@olmicrowaves.com
기술지원 E-mail: imaman@hitel.net
연락처: 054-455-5453, 018-323-8134