AN001:

CUWIN에서 MODBUS사용법

컨트롤 가이드

COMFILE
TECHNOLOGY
컴파일 테크놀로지㈜

CUWIN 및 MODBUS 소개

CUWIN은 고성능의 산업용 제어기로 사용자 인터페이스 및 외부제어를 편리하게 구성할 수 있도록 설계된 제품입니다.

CUWIN은 기본적으로 Embedded OS인 Windows CE5.0을 탑재 하고 있어 안정된 시스템을 구성할 수 있습니다. 또한 .Net Compact Framework를 지원하고 있어 강력하고 방대한 Library를 사용 하여 프로그램 개발 시 코딩 량을 줄여 개발 및 테스트 기간을 단축시켜줍니다. 그밖에 PLC모듈(TinyPLC, CUBLOC)등과 연동하여 MODBUS 통신을 통해 PLC의 I/O를 원격에서 제어 하실 수 있습니다.

<CUWIN3500>

<CUWIN3100>

TinyPLC: TinyPLC는 “반도체형 PLC” 로 마치 원칩마이컴 처럼 PCB 에 장착하여 사용할 수 있는 제품입니다. PLC 에서 사용하는 프로그래밍 방식인 “레더 로직”을 그대로 사용할 수 있어 기존 PLC 사용자들이 편하게 접근할 수 있고, 무엇보다 비용부담과 배선의 불편함을 없앤 제품입니다.

<TPC93A>

<TPC91A>

CUBLOC: CUBLOC 은 “반도체형 임베디드 컴퓨터”입니다. 마이컴을 사용하는 곳에 CUBLOC 을 대신 사용할 수 있습니다. CUBLOC을 사용하게 되면, 별도의 개발장비나 컴파일러가 필요 없이, 무료로 제공되는 통합개발환경 소프트웨어로 개발할 수 있으며, 어셈블리어나 C 언어보다는 배우기 쉽고, 사용하기 쉬운 베이직언어나 레더 로직을 사용할 수 있습니다.

<CUBLOC SERIES>

MODBUS: MODBUS는 MODICON사에서 자사의 PLC를 위하여 개발된 PLC접속 프로토콜로써, PLC와 외부기기와의 인터페이스를 위하여 고안된 통신방식입니다.
마스터-슬레이브의 개념을 가지고 운용되며 마스터는 데이터를 제공하는 능동적 디바이스이고 슬레이브는 데이터를 받아들이고, 응답하는 수동적 디바이스를 의미 합니다, 즉 슬레이브에서는 오로지 마스터가 보낸 데이터에 대한 응답만 가능할 뿐, 자체적으로 데이터를 송신할 수는 없습니다.

슬레이브는 자기 자신의 고유 어드레스를 가지고 있으며, 마스터에서는 이 Slave Address를 사용해서, 여러 개의 슬레이브 중 특정 슬레이브를 하나만을 지정해서 통신할 수 있습니다.

<MASTER – SLAVE 구조>

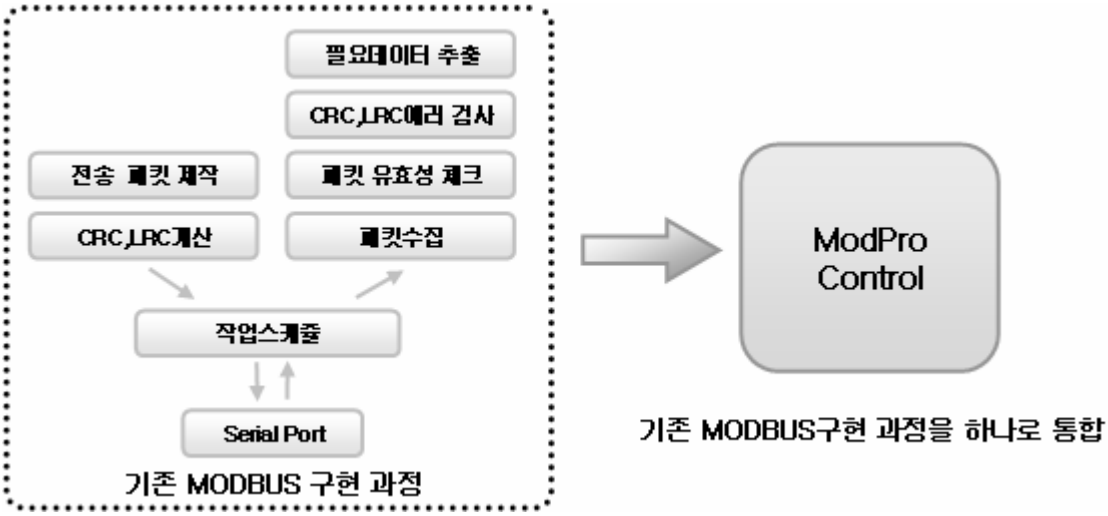
1:1연결 방식은 RS232, RS422 방식을 사용하고, 1:N연결에는 RS485방식을 사용합니다. Slave Address는 1:1방식에서도 사용합니다.

마스터가 보내는 하나의 메시지 집합을 “프레임”이라 부르며 프레임은 슬레이브 어드레스, 펑션 코드, 데이터, 에러체크 코드 등으로 구성되어 있습니다.

ModProControl 개요

CUWIN은 WINDOWS CE 기반의 터치 컨트롤러로 PLC와 MODBUS통신으로 연결하여 PLC의 I/O를 원격제어 하실 수 있습니다. 하지만 MODBUS를 소프트웨어로 구현하기 위해선 시리얼에서 데이터를 수집, 분석하는 작업이나 CRC, LRC 에러체크, 시리얼 포트를 컨트롤하기 위한 여러 작업등 해야 할 작업량이 많습니다.

그래서 MODBUS 프로토콜의 사용을 좀 더 쉽고 편리하게 사용하기 위해 Serial Port를 내장한 ModProControl을 제작하였습니다.



<MODBUS 구현과정의 통합>

ModProControl은 .NET Component를 기반으로 하여 제작한 MODBUS 프로토콜용 Serial Port 컨트롤입니다.

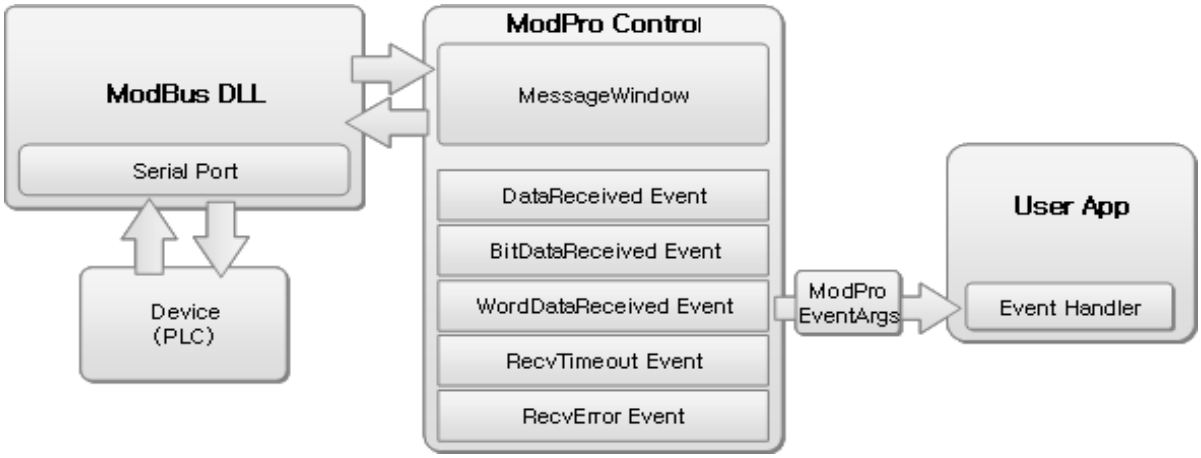
제공되는 Component의 DLL 파일을 추가만 하면 기존에. net에서 제공되는 Button이나 Label, Serial Port등과 같은 컨트롤처럼 쉽게 사용할 수 있습니다.

기본적으로 MODBUS에서 사용하는 Function을 기능을 제공하고 있고, User가 비록 MODBUS란 프로토콜을 모른다 하더라도 슬레이브 어드레스와 액세스할 주소만 알고 있다면 간단하게

값을 읽어올 수 있습니다. 뿐만 아니라, 반복 수행의 Auto 쿼리와 즉시 수행의 Manual 쿼리의 두 가지 형태로 통신을 할 수 있어 User가 코딩 해야 할 분량이 더욱 적어졌습니다.

또한, 자사의 PLC 제품 이외 MODBUS를 사용하는 타사 제품에서도 동작이 가능하고 MODBUS를 사용하지 않거나 이를 변경하여 사용하는 제품에서도 동작이 가능하도록 범용으로 사용이 가능한 데이터 전송 기능도 제공하고 있습니다.

ModProControl의 구조



<ModProControl의 구조>

ModBus DLL : ModBus 프로토콜의 핵심기능을 담당하는 DLL 라이브러리로 Serial Port를 집적 액세스 하기 때문에 유저가 따로 시리얼포트를 설정할 필요가 없습니다.

ModProControl : ModBus DLL의 기능을 .net Component형식으로 지원하기 위한 컨트롤로 User에게 이벤트 형식으로 수신 데이터를 제공합니다.

ModProEventArgs : 이벤트가 발생하였을 때 이벤트 핸들러의 인자로 넘어가는 데이터로 여기에 수행한 작업의 UID와 작업에 대한 응답 데이터를 포함하고 있습니다.

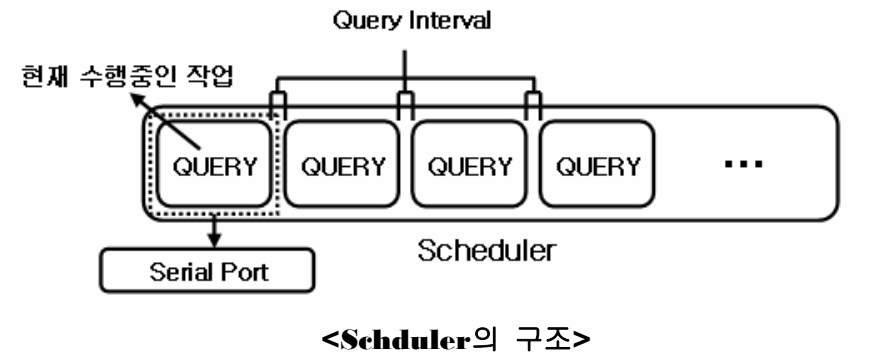
Device: User가 컨트롤 할 Device(PLC)로 ModBus DLL과 직접 통신합니다.

User App : User가 ModPro Control을 이용하여 제작한 Application

Scheduler와 Auto, Manual에 대해서

Modbus DLL은 작업들을 보다 효율적으로 관리하기 위한 Scheduler를 가지고 있습니다. 이 Scheduler는 등록되어있는 작업들이 유효한지를 판단하고 시리얼로 전송하는 역할을 담당하고 있습니다.

만약 Scheduler가 정상적으로 시리얼 전송을 마쳤을 경우 Device에서는 요청(Request)에 해당하는 응답(Response)를 보내주게 되고, 이 응답이 지정해둔 Timeout 이내에 도착하면 이를 User에게 Event형식으로 제공하게 됩니다. 하지만 지정해둔 Timeout 이내에 도착하지 못하면 Timeout Event를 발생시키게 됩니다. 이런 과정으로 하나의 작업이 완료하게 됩니다. 하나의 작업이 완료되면 queryInterval의 간격이 지난 후 다음 작업을 수행하게 됩니다. 만약 작업을 모두 마친 경우 등록 되어있는 작업을 다시 반복 수행합니다.



<Schduler의 구조>

이렇게 Scheduler에서 동작하는 작업을 등록하기 위해선 AutoWordRead, AutoBitWrite처럼 Auto로 시작하는 메소드를 통해 작업을 등록하게 되면 Scheduler에 등록되어 반복적으로 수행하게 되어있습니다. 이는 프로그램에서 주기적으로 특정 메모리영역이나 릴레이의 상태를 반복적으로 쓰거나 읽는 작업이 필요할 시에 유용하게 사용할 수 있습니다.

이와 반대로 Manual로 시작하는 메소드를 통해 작업을 등록할 수도 있습니다. Manual로 작업을 요청하게 되면 Scheduler에 등록되지 않고 호출 즉시 Scheduler에 등록 되어있는 작업을 무시하고 가장 최우선적으로 한번 수행하는 작업입니다. 이는 프로그램에서 Button이나 기타 컨트롤을 이용하여 사용하는 사용자의 입력을 통해 디바이스의 상태를 변경할 때 유용하게 사용할 수 있습니다.

지금까지 Scheduler와 Auto, Manual의 의미에 대해 알아보았습니다. 이제 실제 Control에서 이 Auto와 Manual을 어떤 방법으로 제공하고 이를 어떤 방법으로 사용 해야 하는지에 대해서 알아보겠습니다.

Auto 메소드

```
void AutoBitRead(int nUid, int nSlaveAddress, int nStartAddress, int nLength)
void AutoWordRead(int nUid, int nSlaveAddress, int nStartAddress, int nLength)
void AutoBitWrite(int nUid, int nSlaveAddress, int nStartAddress, bool bWritingBit)
void AutoWordWrite(int nUid, int nSlaveAddress, int nStartAddress, int nWritingWord)
void AutoMultiBitWrite(int nUid, int nSlaveAddress, int nStartAddress, int nLength, Byte[] pbyRawData)
void AutoMultiWordWrite(int nUid, int nSlaveAddress, int nStartAddress, int nLength, Byte[] pbyRawData)

void AutoUserProtocol(int nUid, int nResponseLength, byte[] pby)
```

Manual 메소드

```
void ManualBitRead(int nUid, int nSlaveAddress, int nStartAddress, int nLength)
void ManualWordRead(int nUid, int nSlaveAddress, int nStartAddress, int nLength)
void ManualBitWrite(int nUid, int nSlaveAddress, int nStartAddress, bool bWritingBit)
void ManualWordWrite(int nUid, int nSlaveAddress, int nStartAddress, int nWritingWord)
void ManualMultiBitWrite(int nUid, int nSlaveAddress, int nStartAddress, int nLength, Byte[] pbyRawData)
void ManualMultiWordWrite(int nUid, int nSlaveAddress, int nStartAddress, int nLength, Byte[] pbyRawData)

void ManualUserProtocol(int nUid, int nResponseLength, byte[] pby)
```

<AUTO와 MANUAL의 공통 기능>

다음과 같이 원형만 보시더라도 메소드 이름을 제외하곤 양쪽 모두 같은 형태인 것을 확인할 수 있습니다. 둘의 차이점은 단지 반복 수행인가? , 최우선수행인가? 의 차이 입니다.

우선 각 메소드들의 기능을 확인하면

BitRead : 비트(릴레이)의 상태를 읽어옵니다.
WordRead : 워드(데이터영역)의 값을 읽어옵니다.
BitWrite : 비트(릴레이)의 상태를 변경합니다.
WordWrite : 워드(데이터영역)의 값을 변경합니다.
MultiBitWrite : 여러 개의 비트(릴레이)의 상태를 변경합니다.
MultiWordWrite : 여러 개 워드(데이터영역)의 값을 변경합니다.
UserProtocol : 범용으로 사용할 수 있는 기능으로 바이트 배열에 데이터를 전송합니다.

각 메소드들의 인자를 보면 모든 메소드에 공통적으로 들어가는 nUid인자가 있습니다. 이 것은 현재 등록할 작업의 ID입니다. 이 ID를 통해 작업을 관리할 수 있고, 이를 통해 이벤트에서 수신한 데이터가 어떤 작업의 응답인지를 확인할 수 있습니다.

UserProtocol을 제외한 나머지의 인자를 보면 공통적으로 nSlaveAddress와 nStartAddress가 있음을 확인 할 수 있습니다. 이 인자들은 각각 PLC의 Slave Address와 읽거나 쓸 PLC의 Physical Address입니다.

그 이외의 인자들은 각 명령들에 따라 분류하여 알아보겠습니다.
BitRead, WordRead의 Length : 읽어올 크기
BitWrite, WordWrite의 bWritingBit나 nWritingWord : Write할 값
MultiBitWrite, MultiWordWrite의 Length : 데이터를 쓸 크기
MultiBitWrite, MultiWordWrite의 pbyRawData : 쓸 데이터를 의미합니다.
UserProtocol은 아래 “UserProtocol에 대하여” 장에서 좀더 자세히 다루도록 하겠습니다.

다음 4개 함수는 작업을 관리하는 기능으로 반복 수행중인 작업을 제거하거나 멈춤, 재동작 시키는 일을 합니다.

public bool pauseItem(int nUid) 해당 UID의 작업을 멈추고

public bool resumeItem(int nUid) 헤당 UID의 작업을 재동작 합니다.

public void removeAutoItem(int nUid) 반복되는 해당 작업을 제거하고

public void removeAllAutoItems() 모든 작업을 제거합니다.

아래의 2개 함수는 작업중 문제가 발생하였을시 대처 방법을 정의 해 두는 역할을 하는 함수입니다.

public void EnableRetryForManual(bool bEnable, int nRetryCount, int nIncreaseTime)

Manual로 작업이 반복 수행 되지 않기 때문에 등록된 작업이 실패 시 nRetryCount 만큼 재전송하여 문제가 존재하는가를 확인하고, 재전송 시 마다 nIncreaseTime 만큼 Timeout시간이 증가하여 수신하도록 하여 데이터를 보다 확실히 수신 받을 수 있는 기능을 제공합니다. 만약 이렇게 재전송을 했는데도 실패한다면 TimeoutEvent가 발생됩니다.

public void SetAutoCustomTimeOut(int nUid, int nTimeOut)

Auto로 등록되는 작업은 반복 수행작업을 하게 되는데 중요도가 높거나 혹은 수행 속도가 높아야 해서 일반적으로 적용되는 Timeout이 아닌 별도의 Timeou을 적용하고 싶을 때 사용하도록 합니다.

User Protocol에 대해서

ModProControl은 Modbus Protocol이외의 User가 직접 작성한 프로토콜이나 그 외 Modbus가 아닌 프로토콜을 지원하기 위해 User Protocol 형식의 방법을 지원하고 있습니다.

```
public void SetUserProtocolMode(bool bUserProtocolMode)
public void AutoUserProtocol(int nUid, int nResponseLength, byte[] pby)
public void ManualUserProtocol(int nUid, int nResponseLength, byte[] pby)
```

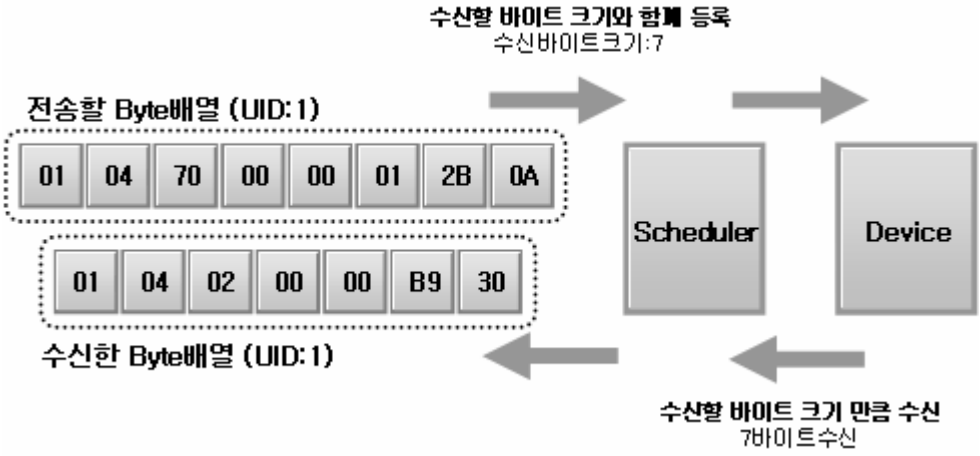
위 3개의 메소드를 통해 User Protocol을 이용할 수 있는데

public void SetUserProtocolMode(bool bUserProtocolMode)은 프로그램 시작시 호출해야 하는 함수로 UserProtocol의 사용여부를 지정하는 메소드입니다.

public void AutoUserProtocol(int nUid, int nResponseLength, byte[] pby)

public void ManualUserProtocol(int nUid, int nResponseLength, byte[] pby)

은 기존의 Auto와 Manual 방식을 그대로 따릅니다.



<USER PROTOCOL의 실행 구조>

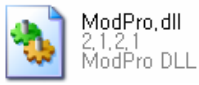
위 메소드에서 사용하는 인자를 보면 nUid는 해당 작업의 id를 의미하며

nResponseLength 수신받을 데이터의 크기를 의미합니다. 수신받을 데이터의 크기를 인자로 주어야 하는 이유는 각기 서로 다른 프로토콜의 경우 응답에 따른 요청 역시 모두 다를것입니다. 이 때, 하나의 완성된 프레임을 수신받기 위해선 이 요청 작업에 해당하는 응답 프레임의 크기를 알아야만 합니다.

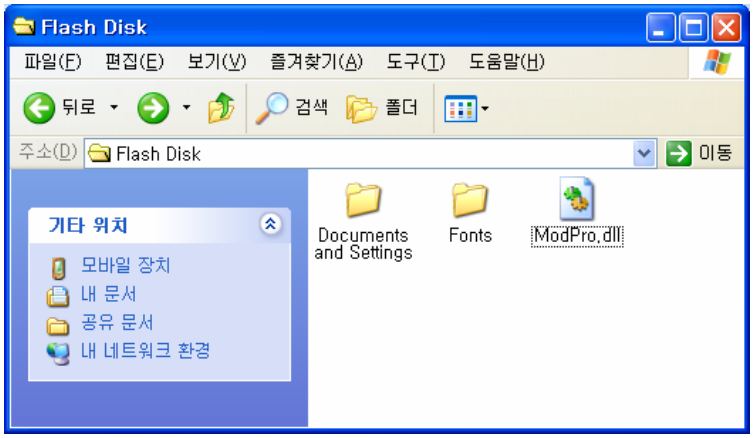
마지막 pby는 전송할 작업의 데이터 입니다.
이 pby는 단순히 byte배열 형태로 되어있어 유저가 어떠한 형식의 프로토콜을 작성하더라도 단순히 보낼 데이터의 byte배열만 취득할 수 있으면 제한 없이 사용할 수 있습니다.

ModProControl의 설정 방법

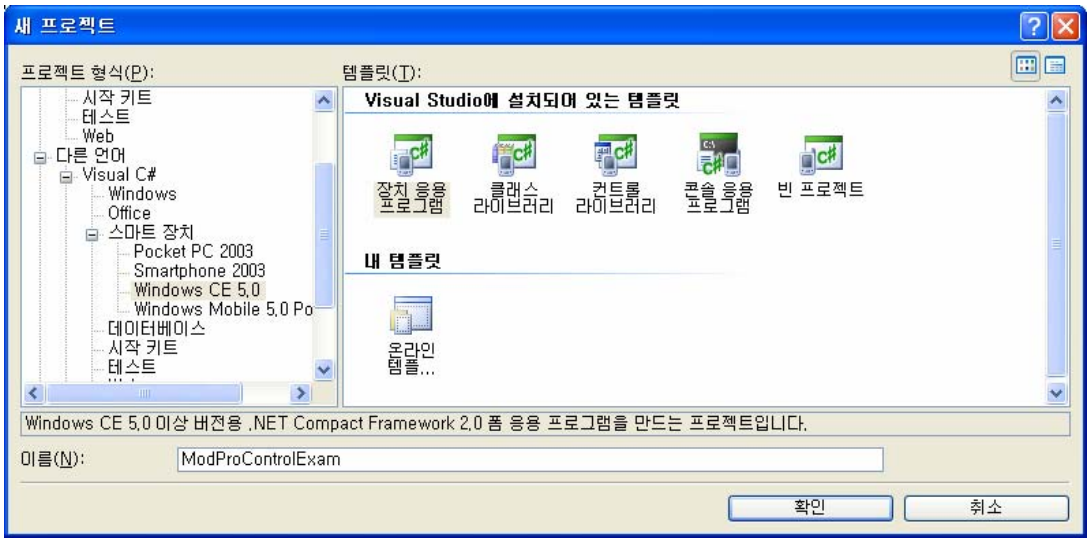
1. ModPro.DLL 파일과 ModProControl.DLL 파일을 다운로드 받습니다.



2. ModPro.DLL 파일은 CUWIN의 Flash Disk 폴더로 복사합니다.

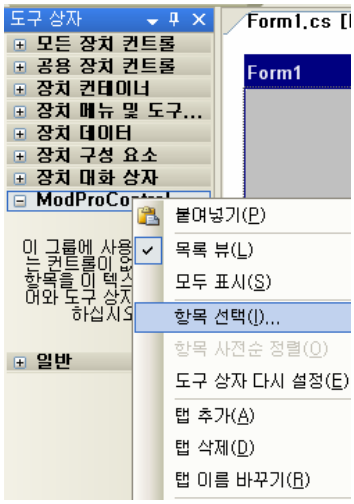


3. 다음과 같이

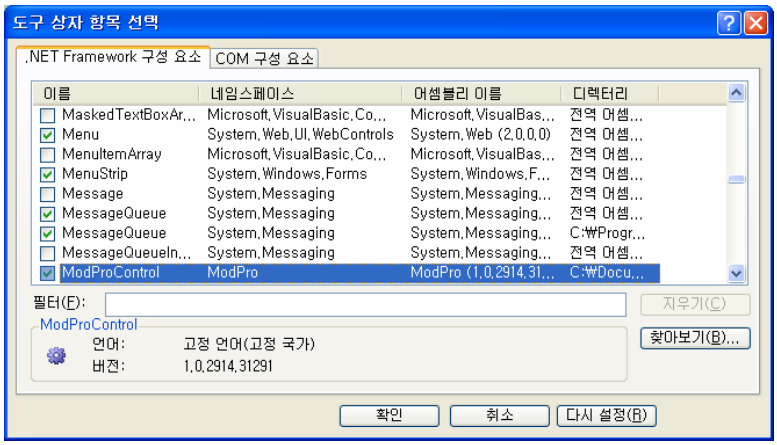


Visual Studio 2005를 실행시켜 파일 → 새 프로젝트 → Visual Basic 혹은 Visual C#의 스마트 장치의 Windows CE 5.0 → 장치 응용 프로그램을 선택하여 프로젝트를 새로 만듭니다.

4.Visual Studio 2005의 좌측의 도구상자의 원하는 탭에서 항목선택을 선택 하면



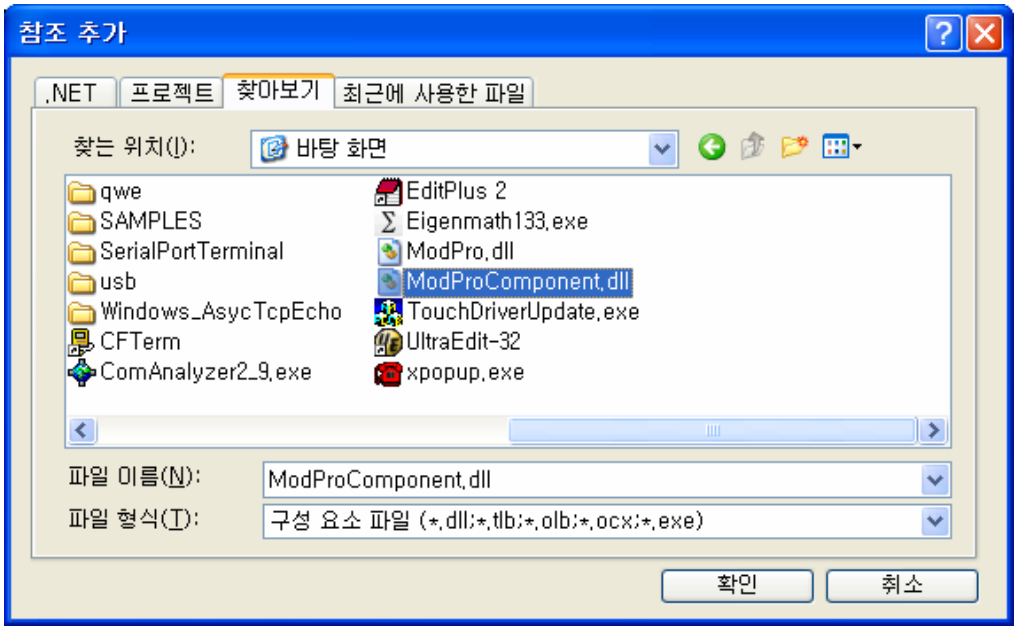
도구상자 항목 선택창에서 찾아보기 버튼을 눌러 ModProControl.Dll 파일을 선택하면 다음과 같이 ModProControl이 나타납니다.



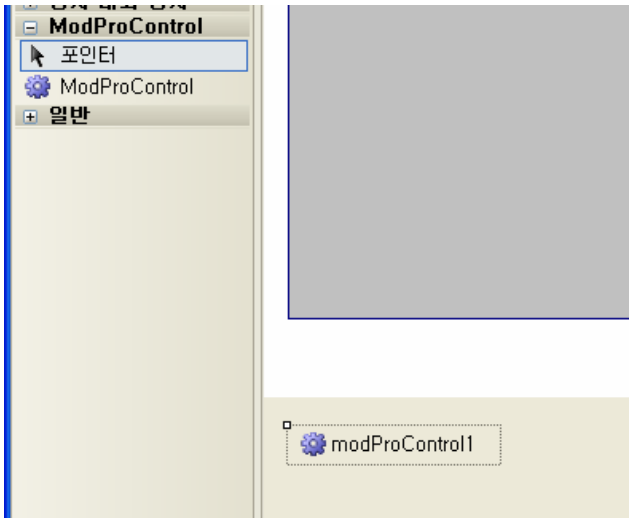
이제 확인을 누르시면

ModProControl이 등록된 것을 확인할 수 있습니다.

5. 이제 이 컨트롤을 사용하기 위한 참조 추가를 하여야 합니다.
참조추가는 우측 솔루션 탐색기의 참조에서 오른쪽 버튼 → 참조추가 혹은
상단 메뉴의 프로젝트→참조추가를 선택하여 추가 할 수 있습니다.



찾아보기 탭에서 ModProControl.dll파일을 선택 후 확인을 누릅니다.



마지막으로 ModProControl을 드래그하여 폼 위에 놓으면 다음과 같은 컨트롤을 사용할 수 있게 됩니다.

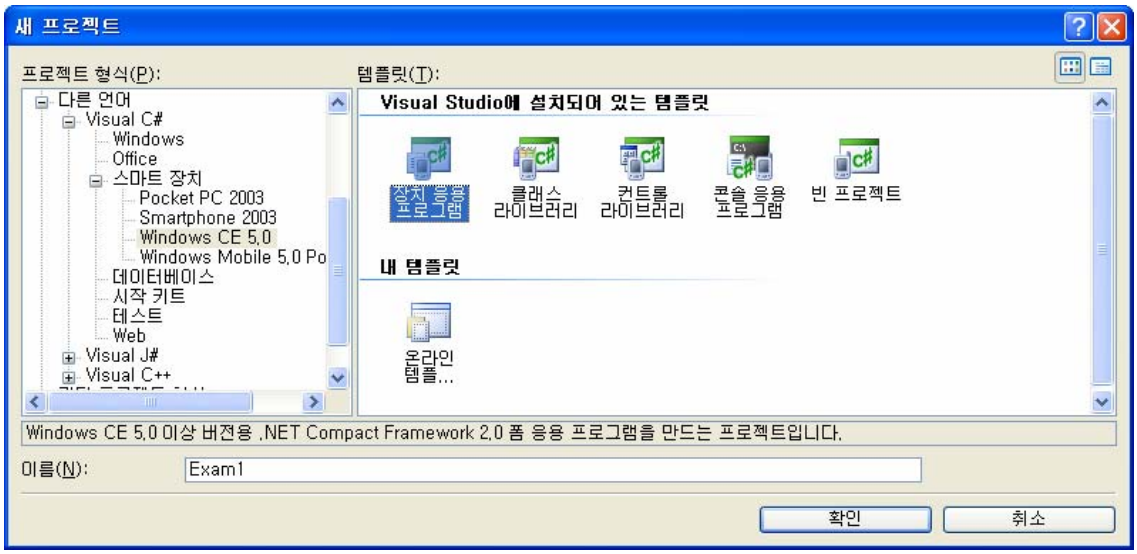
ModProControl을 이용한 예제

다음은 ModProControl을 이용한 예제를 통해 ModPro Control의 사용법을 익혀 보겠습니다.

Exam1. 다음은 PLC의 특정 번지의 메모리를 액세스 할 수 있는 간단한 예제 프로그램입니다.

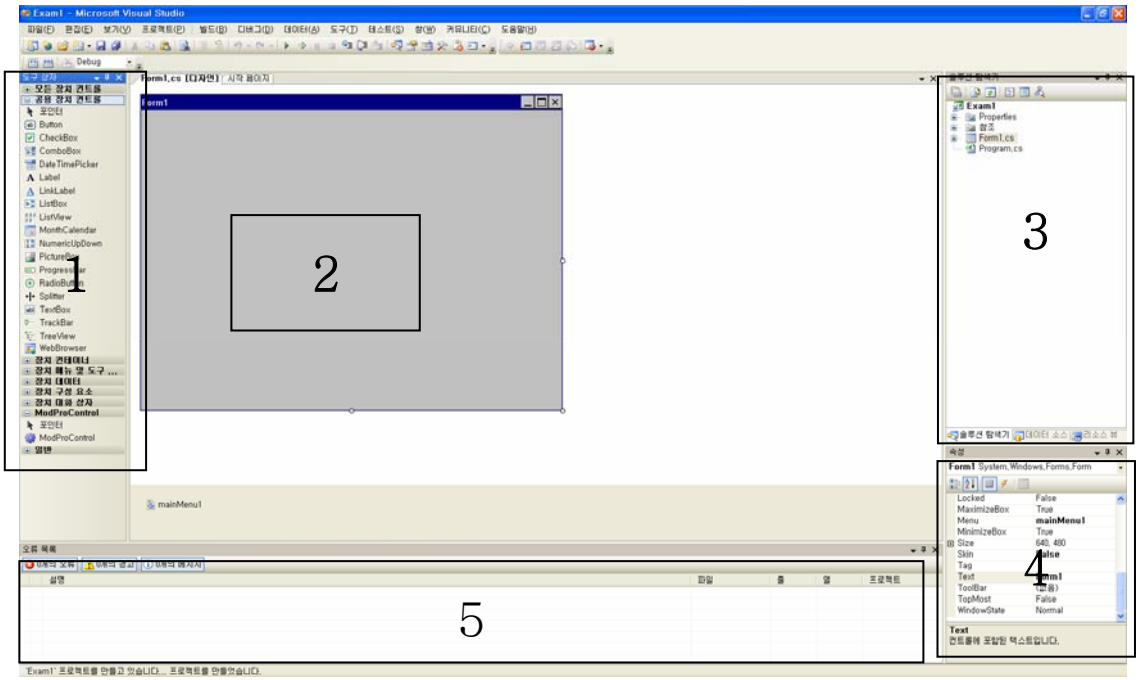
0. CUWIN 매뉴얼은 읽어 보셨나요 안 읽어 보셨다면, 매뉴얼을 읽고 설정을 마친 후에 시작하여 주세요.
0. 위의 컨트롤 사용법을 보지 않았다면 사용법을 보고 설정을 마친 후에 시작해 주세요.

1. Visual Studio 2005를 실행하여 다른 언어 → Visual C# → 스마트 장치 → Windows CE 5.0을 선택하고 우측에 장치 응용 프로그램을 선택하고 아래 이름에 “Exam1”이라 넣은 후 확인을 선택하여 주십시오

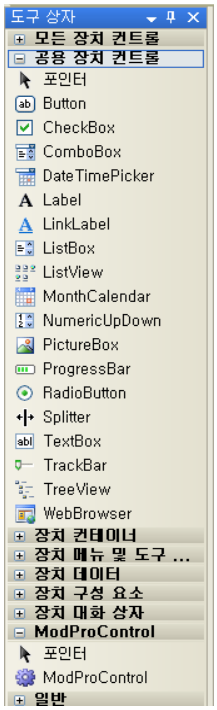


CUWIN은 Windows CE 5.00이 포팅되어 있고 .net Compact Framework를 사용할 수 있습니다. 위와 같이 선택을 하게 되면 .net Compact Framework를 사용하는 프로젝트를 생성하게 됩니다.

2. 다음은 Visual Studio의 화면 구성에 대해 한번 확인해 보겠습니다.



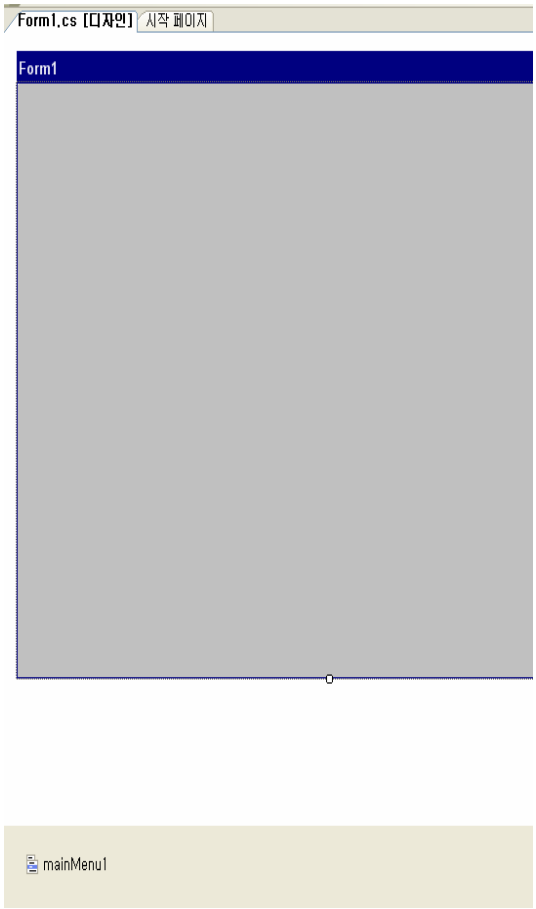
1.



다음은 도구상자입니다.
기본적으로 제공하는
Button, Label, ListBox, TextBox등을 사용할 수 있고
이 이외 추가 적으로 사용할 수 있는 컨트롤들이 있습니다.

위 설정 과정에서 정상적으로 ModProControl 등록을 마쳤다면 다음과 같이 ModProControl이 등록된 것을 확인할 수 있습니다.

2.

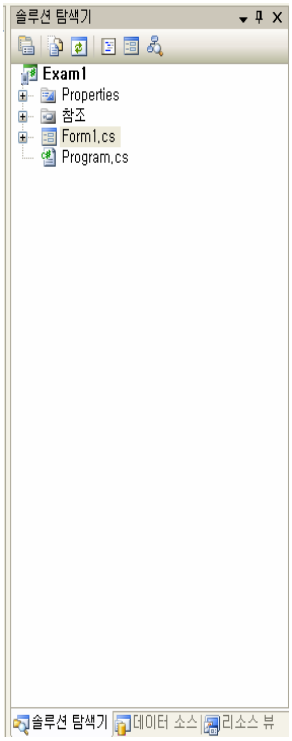


다음은 작업 영역입니다.

폼을 디자인하고, 코드를 작성하며 클래스다이어그램을 확인하고, 각종 설정 등의 작업을 할 수 있는 공간으로 폼과 아래엔 MainMenu컨트롤이 있는데 아래 등록되어 있는 MainMenu 컨트롤은 프로젝트 생성시 자동으로 등록되어있는 것입니다.

이번 예제에서는 메뉴 사용이 없으니 선택 후 Del 키를 눌러 삭제를 해 주세요.

3.

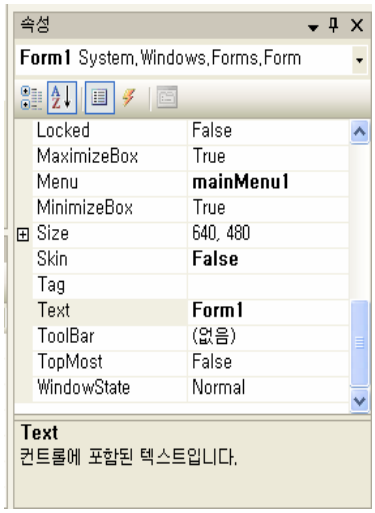


솔루션 탐색기 부분으로 우측 상단에 위치해있고

솔루션 탐색기 뿐 아니라, 데이터 소스, 리소스 뷰로 사용되며 하단에 선택 창이 있습니다.

프로젝트의 전체적인 소스파일과 리소스 등을 등록, 삭제 등 관리할 수 있는 창입니다.

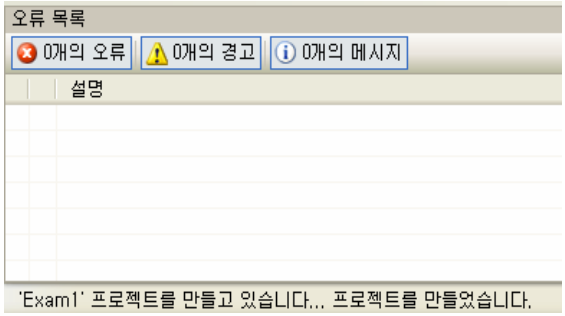
4.



속성 영역으로 프로젝트나 클래스, 컨트롤, 리소스 등의 속성을 지정할 수 있는 창입니다.

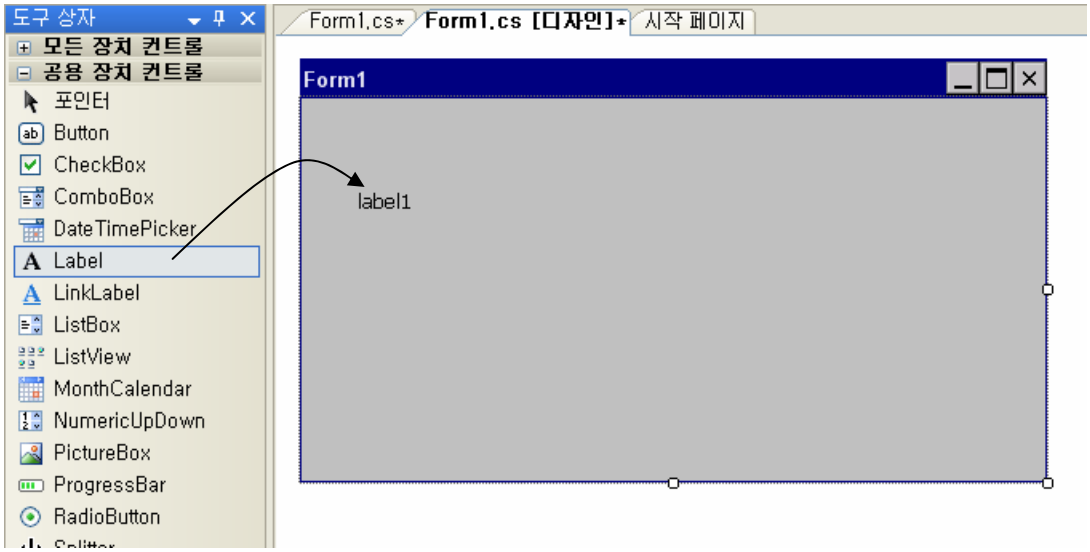
상단쪽엔 클래스 이름이 나타나며
그 아래엔 정렬 형태와 속성, 이벤트 선택하는 것이 있는데 번개 모양 선택 시 이벤트를 선택할 수 있는 창이 뜨며 번개모양 좌측에 리스트 모양을 누르면 속성이 뜨게 됩니다.

현재는 속성이 선택되어 있습니다.

5. 

다음은 오류 목록을 나타내는 창으로 코드상 문법오류나 경고 등을 나타내는 창입니다.

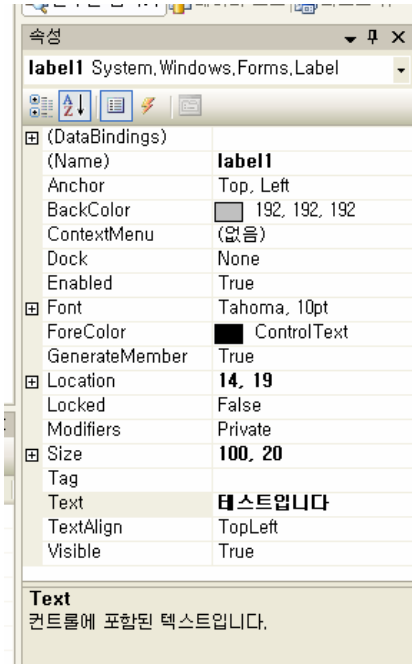
3. 작업영역 아래 쪽에 MainMenu 컨트롤이 등록되어 있다면 지우시고, 이제 폼을 구성을 할 것 입니다. 폼 구성하는 법은 우측 도구상자에서 원하는 컨트롤을 선택하여 폼 위에 끌어다 놓으시면 됩니다.



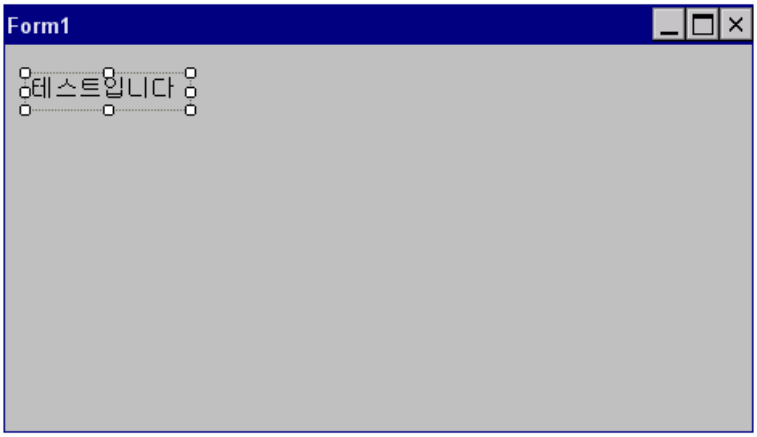
4. 이젠 끌어다 놓은 label의 속성을 변경해 보겠습니다.

우선 label은 선택하고 속성 창을 봐주세요.

속성 창에 Text에 “테스트입니다” 라고 입력하였습니다.

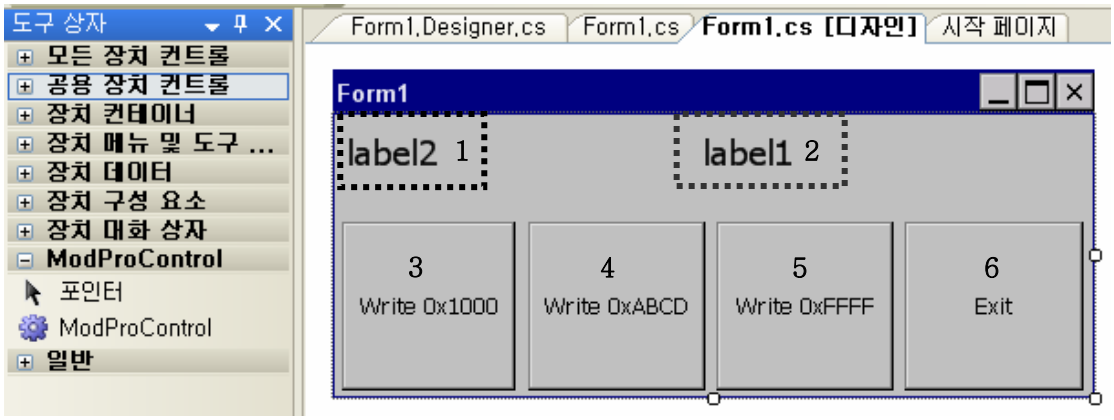


실제 Label을 보면



위와 같이 label1에서 “테스트입니다” 로 변경한 것을 확인할 수 있습니다.

5. 이번엔 위와 같은 방법으로 다음과 같이 폼을 구성 해 보십시오

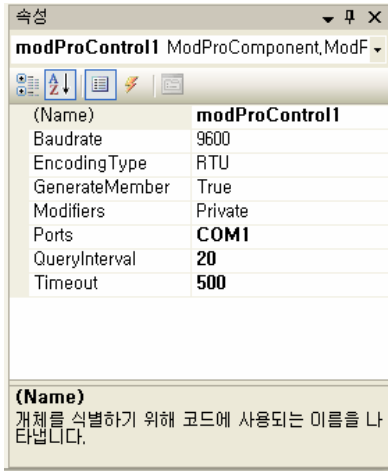


컨트롤의 속성은 다음과 같습니다..

No	Name	Text	Font
1	Label2	Label2	Tahoma, 14pt
2	Label1	Label1	Tahoma, 14pt
3	Button2	Write 0x1000	Tahoma, 10pt
4	Button3	Write 0xABCD	Tahoma, 10pt
5	Button4	Write 0xFFFF	Tahoma, 10pt
6	Button1	Exit	Tahoma, 10pt

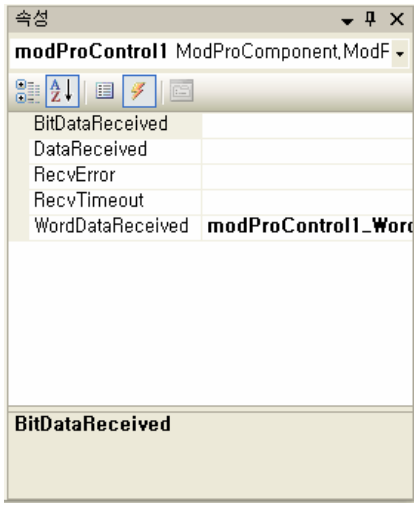
6. 좌측에 ModProControl을 드래그 하여 폼 위에 올려두면 다음과 같이 하단에 컨트롤이 등록됩니다.

7. 이제 ModProControl의 속성을 변경할 것입니다.



Ports를 COM1으로 두었고 QueryInterval을 20, Timeout을 500으로 두었습니다.
Ports는 사용할 포트의 이름입니다.
QueryInterval을 하나의 작업을 완료한 이후 다음 작업을 시작하기 전까지의 Interval 간격입니다. 이 속도를 줄이면 쿼리를 전송하는 시간이 빨라지고 늘이면 쿼리 전송시간이 느려집니다.
Timeout은 쿼리를 전송하고 응답이 오기까지의 제한 시간입니다. 만약 Timeout을 1000으로 두었다면 1초 이내로 응답이 오지 않을 경우 Timeout이 되어 RecvTimeout이벤트가 발생하게 됩니다.

8. 이젠 이벤트를 등록할 것입니다. 이벤트는 아주 간단하게 등록할 수 있습니다.



WordDataReceived 이벤트를 더블 클릭하면 그에 해당하는 이벤트 핸들러 함수가 자동으로 생성됩니다.
핸들러 함수는 이벤트가 발생하였을 때 자동으로 호출되는 함수라 생각하면 쉽습니다.

9. 이제 아래는 소스를 코딩 해야 할 차례입니다.

폼위에서 우측 버튼을 누르고 코드보기를 선택하여 주십시오
그럼 아래와 같은 화면이 뜰 것입니다.

Exam1.Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Exam1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }


        private void modProControl1_WordDataReceived(ModProComponent.ModProEventArgs e)
        {

        }
    }
}
```

public Form1()은 생성자로 Form이 생성될 때 실행됩니다.

InitializeComponent();
아래에 다음과 같이 작성하여 주십시오.

```
public Form1()
{
    InitializeComponent();
    modProControl1.InitializeModProS();
    modProControl1.AutoWordRead(1, 1, 0x8000, 1);
}
```

modProControl1.InitialIzeModProS ();

속성값으로 컨트롤을 초기화하는 메소드입니다.

modProControl1.AutoWordRead (1, 0x1, 0x8000, 1);

이 작업의 UID는 1(1번째 인자)입니다.

이 작업은 슬레이브 어드레스가 0x1(2번째인자)인 PLC의 0x8000(3번째인자)번지의 값을 하나(4번째 인자) 반복적으로 읽어오는 작업을 수행합니다.

이제 Control부터 WordData를 수신받았을 때 실행되는 WordDataReceived를 작성해야합니다.

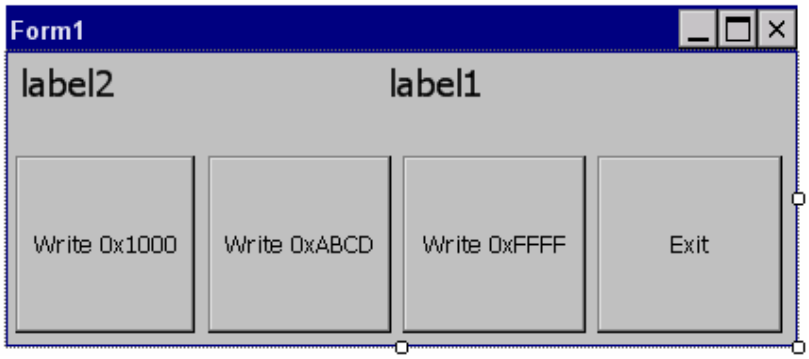
우선 먼저 using으로 System.Collections을 사용한다 정의 합니다. (1)

그리고 클래스의 멤버변수로 m_nCount 변수를 int형으로 선언해주십시오. (2)

그리고 아래와 같이 코딩해 주십시오 (3)

```
using System;
using System.Collections.Generic;
using System.Collections; //(1)
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Exam1
{
    public partial class Form1 : Form
    {
        int m_nCount = 0; //(2)
        public Form1()
        {
            InitializeComponent();
            modProControl1.InitialIzeModProS();
            modProControl1.AutoWordRead(1, 1, 0x8000, 1);
        }

        private void modProControl1_WordDataReceived(ModProComponent.ModProEventArgs e)
        {
            //(3)
            ArrayList datas = e.GetDatas();
            m_nCount++;

            label1.Text = "Value:" + (Convert.ToInt32(datas[0])).ToString("X") + " ";
            label2.Text = "Count:" + m_nCount.ToString();
        }
    }
}
```

각 버튼을 클릭했을 때 수행하는 코드를 작성해야 합니다.



우선 Write 0x1000을 더블 클릭하십시오. 코드에 다음과 같이 입력 합니다.

```
private void button2_Click(object sender, EventArgs e)
{
        //UID:2 작업, 슬레이브 어드레스가 1인 PLC의 0x8000영역에 0x1000이라 쓴다.
        modProControl1.ManualWordWrite(2, 1, 0x8000, 0x1000);
}
```

마찬가지로 Write 0xABCD를 더블 클릭 하여 아래와 같이 입력 합니다.

```
private void button3_Click(object sender, EventArgs e)
{
        //UID:2 작업, 슬레이브 어드레스가 1인 PLC의 0x8000영역에 0xABCD라 쓴다.
        modProControl1.ManualWordWrite(2, 1, 0x8000, 0xABCD);
}
```

다음은 Write 0xABCD를 더블 클릭 하여 아래와 같이 입력 합니다.

```
private void button4_Click(object sender, EventArgs e)
{
        //UID:2 작업, 슬레이브 어드레스가 1인 PLC의 0x8000영역에 0xFFFF이라 쓴다.
        modProControl1.ManualWordWrite(2, 1, 0x8000, 0xFFFF);
}
```

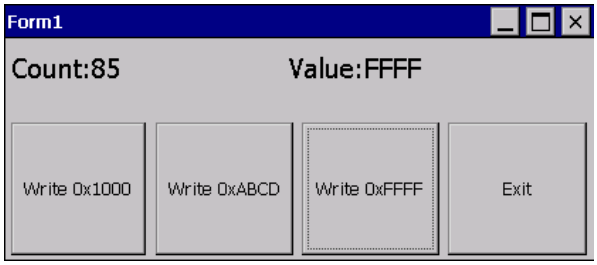
다음은 Exit를 더블 클릭 하여 아래와 같이 입력 합니다.

```
private void button1_Click(object sender, EventArgs e)
{
        //ModProControl를 닫고 프로그램을 종료합니다
        modProControl1.Close();
        Application.Exit();
}
```

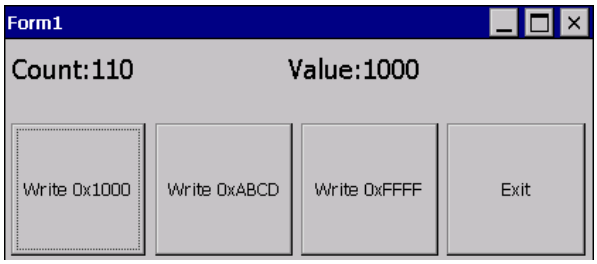
10. 이제 실행을 해야 합니다. 컴퓨터와 CUWIN이 정확히 연결되었는지 확인하십시오
CUWIN과 PLC혹은 CUBLOC이 정확히 연결되었는지 확인하십시오. CUWIN뒤에 스위치가 232에 놓였는지 485에 놓였는지 확인하여 정확히 배선하여 주십시오

11. 정상적으로 연결하였다면 실행하도록 합니다.
실행화면입니다.
Value의 값은 0x8000 번지의 값을 읽어와 디스플레이 하는 중이고
각 버튼은 0x8000번지의 서로 다른 값을 Write하도록 되어있습니다.

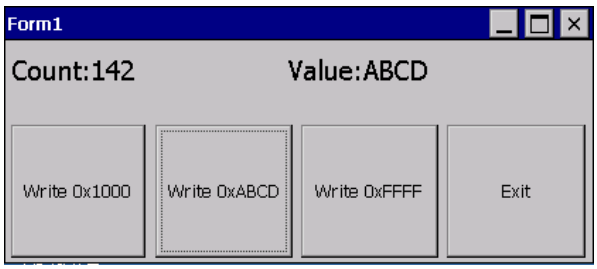
Write 0xFFFF(button4) 클릭 시



Write 0x1000(button2) 클릭 시

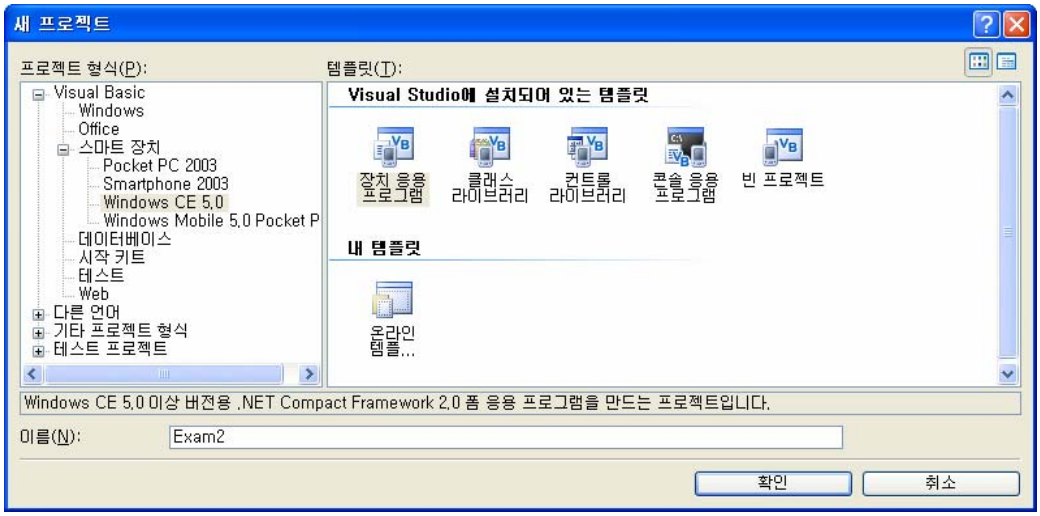


Write 0xABCD(button3) 클릭 시



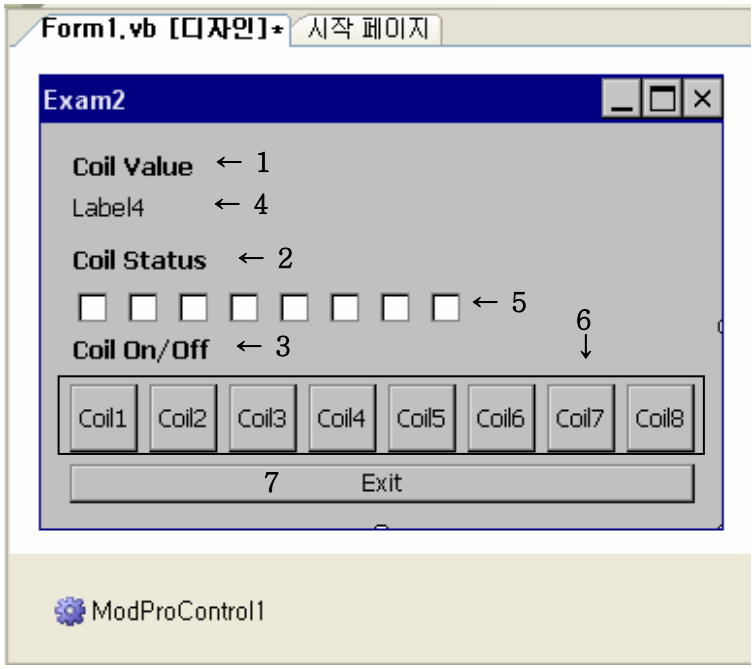
Exam2. 다음은 PLC의 Coil(릴레이)을 컨트롤하고 그 상태를 읽어 오는 간단한 예제 프로그램입니다.

0. 프로젝트는 Visual Basic을 이용하여 생성하십시오. 위의 C# 생성법과 동일합니다.



이름은 “Exam2” 입니다.

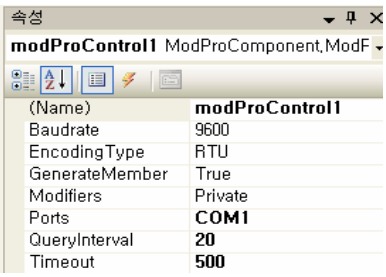
1. Exam1을 참조하여 다음과 같이 폼을 구성 해 보십시오



2. 폼과 각 컨트롤의 속성은 다음과 같습니다.

No	Name	Text	Font
1	Label1	“Coil Value”	Tahoma, 10pt, style=Bold
2	Label2	“Coil Status”	Tahoma, 10pt, style=Bold
3	Label3	“Coil On/Off”	Tahoma, 10pt, style=Bold
4	Label4	변경없음	변경없음
5	CheckBox1	“” (공백)	변경없음
	CheckBox2	“”	변경없음
	CheckBox3	“”	변경없음
	CheckBox4	“”	변경없음
	CheckBox5	“”	변경없음
	CheckBox6	“”	변경없음
	CheckBox7	“”	변경없음
	CheckBox8	“”	변경없음
6	Button1	Coil1	변경없음
	Button2	Coil2	변경없음
	Button3	Coil3	변경없음
	Button4	Coil4	변경없음
	Button5	Coil5	변경없음
	Button6	Coil6	변경없음
	Button7	Coil7	변경없음
	Button8	Coil8	변경없음
7	Button9	Exit	변경없음

ModProControl의 속성은 다음과 같습니다.

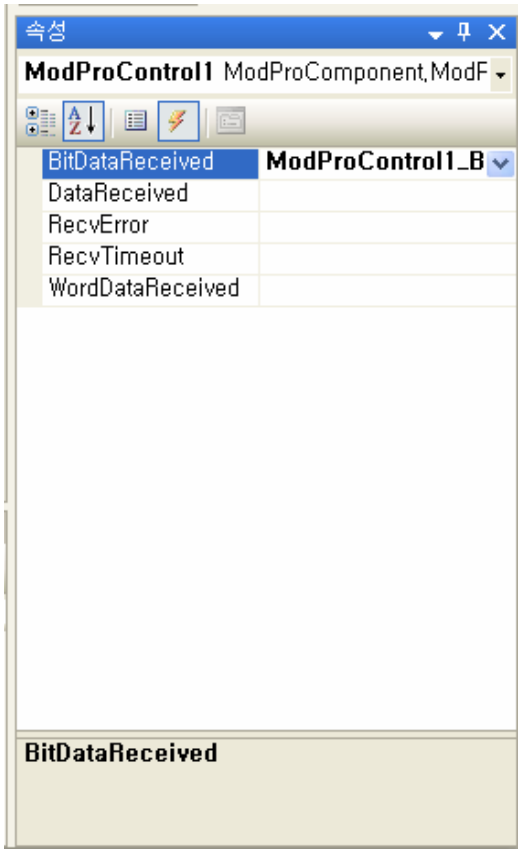


3. 이제 소스코드를 작성할 차례 입니다.
우선 폼을 더블 클릭하여 주십시오

```
Public Class Form1
    Dim m_bArray(8) As Boolean                          '(1)
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
                                                System.EventArgs) Handles MyBase.Load
        ModProControl1.InitiallzeModProS()              '(2)
        ModProControl1.AutoBitRead(1, 1, &H18, 8)
    End Sub
End Class
```

폼을 더블 클릭하면 위의 굵은 글씨 부분을 제외한 코드가 생성 될 것입니다.
그 안에 굵은 글씨의 코드를 넣어주십시오.

위 코드의
(1)번 코드는 Boolean형식의 개수가 8개인 배열입니다. 이 배열은 릴레이의 상태를 저장하고 있습니다.
(2)번 코드는 ModProControl을 컨트롤을 지정한 속성값으로 초기화하고 &H18(0x18)번지부터 시작하여 8개의 코일의 상태를 주기적으로 반복하며 읽어오는 작업을 등록하는 것입니다.



ModProControl의 BitDataReceived를 더블클릭 하여 주십시오.

그럼 아래와 같이 ModProControl1_BitDataReceived란 메소드가 생기게 됩니다.

이 메소드는 ModProControl1.BitDataReceived의 헨들러 함수입니다.

아래처럼 굵은 글씨의 코드를 넣어주십시오

```
Private Sub ModProControl1_BitDataReceived(ByVal e As ModProComponent.ModProEventArgs)
            Handles ModProControl1.BitDataReceived
        Dim datas As ArrayList
        Dim i As Integer
        datas = e.GetDatas()
        Label4.Text = datas(0).ToString()

        Dim n As Integer = Convert.ToInt32(datas(0))

        For i = 0 To 7
            If ((n And 1) = 1) Then
                m_bArray(i) = True
            Else
```

```
            m_bArray(i) = False
        End If
        n = n >> 1
    Next
    CheckBox1.Checked = m_bArray(7)
    CheckBox2.Checked = m_bArray(6)
    CheckBox3.Checked = m_bArray(5)
    CheckBox4.Checked = m_bArray(4)
    CheckBox5.Checked = m_bArray(3)
    CheckBox6.Checked = m_bArray(2)
    CheckBox7.Checked = m_bArray(1)
    CheckBox8.Checked = m_bArray(0)
End Sub
```

위 내용은 이벤트 발생시 인자로 넘어온 ModProEventArgs에서 데이터를 가져와
코일(릴레이)의 상태 데이터를 읽어온 후
그 값을 Label4에 출력하고
해당 데이터를 한 비트씩 확인하면서 코일의 상태를 배열에 저장합니다.
그리고 이 배열의 데이터를 8개의 CheckBox에 값을 출력합니다..

다음은 각 버튼 클릭 시 코드들입니다.

Coil1(Button1)을 더블 클릭하시고 아래 굵은 글씨를 넣어주십시오
아래 코드는 슬레이브 어드레스가1 인 보드의 &h18(0x18)부터 시작하여 8번째 있는 코일(릴레이)의 상태를 반전시키는 작업을 등록 하는 내용입니다. 이 작업의 ID는 2입니다.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    ModProControl1.ManualBitWrite(2, 1, &H18 + 7, Not m_bArray(7))
End Sub
```

Coil2(Button2)를 더블 클릭하시고 아래 굵은 글씨를 넣어주십시오
아래 코드는 슬레이브 어드레스가 1인 보드의 &h18(0x18)부터 시작하여 7번째 있는 코일(릴레이)의 상태를 반전시키는 작업을 등록 하는 내용입니다. 이 작업의 ID는 2입니다.

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button2.Click
    ModProControl1.ManualBitWrite(2, 1, &H18 + 6, Not m_bArray(6))
End Sub
```

아래 코드들은 위와 같은 형식으로 각 주소에 해당하는 코일(릴레이)의 상태를 반전시키는 내용의 코드들입니다.

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button3.Click
        ModProControl1.ManualBitWrite(2, 1, &H18 + 5, Not m_bArray(5))
End Sub

Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button4.Click
        ModProControl1.ManualBitWrite(2, 1, &H18 + 4, Not m_bArray(4))
End Sub
Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button5.Click
        ModProControl1.ManualBitWrite(2, 1, &H18 + 3, Not m_bArray(3))
End Sub

Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button6.Click
        ModProControl1.ManualBitWrite(2, 1, &H18 + 2, Not m_bArray(2))
End Sub

Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button7.Click
        ModProControl1.ManualBitWrite(2, 1, &H18 + 1, Not m_bArray(1))
End Sub

Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button8.Click
        ModProControl1.ManualBitWrite(2, 1, &H18 + 0, Not m_bArray(0))
End Sub
```

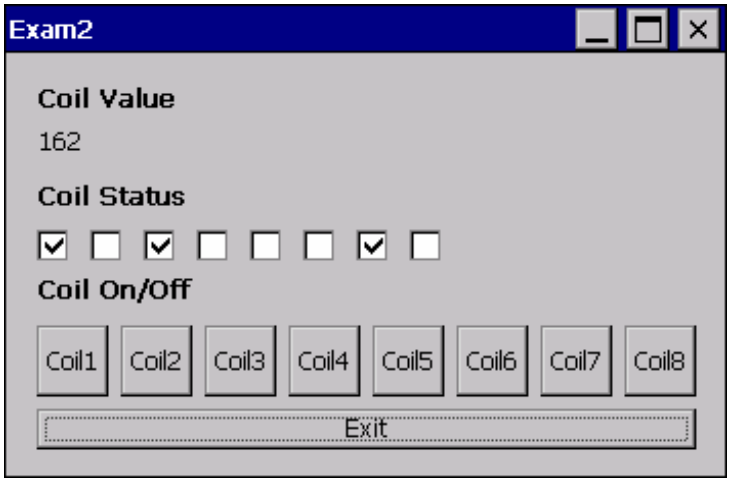
마지막으로 Exit버튼을 더블 클릭하시고 아래 굵은 글씨를 넣어주십시오.

이 코드는 ModProControl1을 닫고 프로그램을 종료시키는 내용입니다.

```
Private Sub Button9_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button9.Click
        ModProControl1.Close()
        Application.Exit()
End Sub
```

4. 실행화면 입니다.

우선 테스트로 1번 코일과 3번 코일과 7번 코일의 상태를 On 상태로 만들어 보겠습니다.



위 그림과 같이 코일의 상태가 표시되었습니다.

상태를 보면 다음과 같은데 이 값을 확인해보면

1	0	1	0	0	0	1	0

128 + 0 + 32 + 0 + 0 + 0 + 2 + 0

= 128+ 32+ 2

= 162

가 나오게 됩니다 그림 상단에 Coil Value의 값이 162로 같은 것을 확인할 수 있습니다.

Exam3. PLC의 복수의 메모리 영역을 액세스하여 이를 이용해 Sin, Cos 그래프를 그리는 예제입니다.

이번 예제는 Exam1에서 사용했던 AutoWordRead를 응용한 예제입니다.
메모리의 0x7000영역부터 0x7004 영역까지 데이터를 사용합니다.

Cubloc을 이용하여 PLC의 0x7000~0x7004 영역을 값을 변경 합니다.
이 때, 0x7000의 값은 x좌표의 값으로 이 값은
2PI를 기준으로 했을 때 2PI를 CUWIN에서 그래프가 디스플레이 할 넓이만큼 나눈 값을 증감값으로 하여 넓이만큼의 루프를 돌았을 때 값이 2PI되는 값을 2PI나눈 후 넓이만큼 곱한 값이다. 말로 풀어 쓰면 어려우나, 예를 들어 넓이가 500일경우

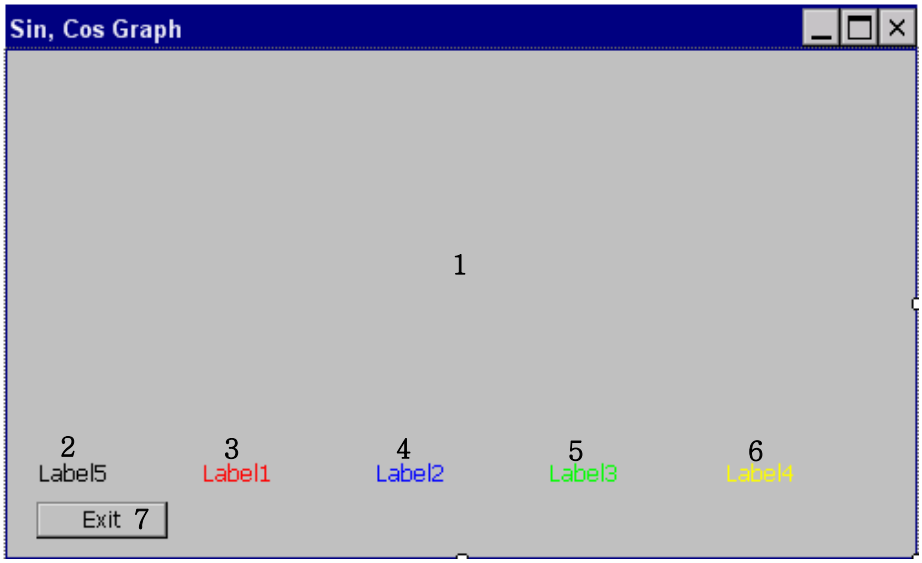
```
A=2PI / 500
F=0
For I= 1 to 250
    F=F+ A
Next
```

이때 F=2PI/2가 된다.
여기서 F에 2PI를 나누고 넓이인 500을 곱하면
F/2PI*500 = (2PI/2)/2PI*500
=(2PI/2PI)/2*500
=1/2*500
=250

루프를 돌았던 횟수. 즉, X값이 나오게 됩니다.
F에 Sin을 적용시키고 그래프가 디스플레이 될 높이만큼 곱하면 Sin그래프의 Y좌표 값을 구할 수 있습니다. 이렇게 0~넓이까지 반복적으로 나오는 X, Y좌표를 연결해주면 Sin 그래프가 완성된다.

우선 프로그램의 구동원리는 이 정도로만 하도록 하겠습니다. 위 내용이 쉽게 이해가 가지 않더라도 직접 따라 해보면서 하면 쉽게 이해갈수 있을 것 입니다.

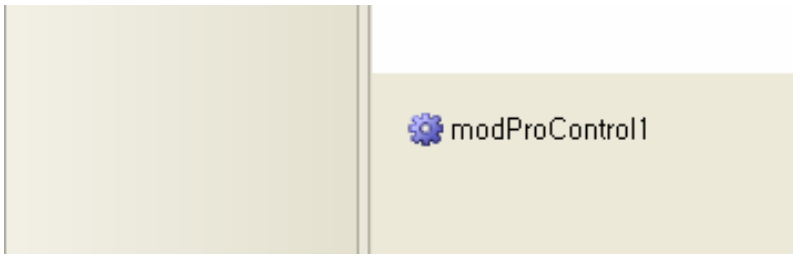
1. 폼 구성은 다음과 같습니다. 위 Exam1의 내용을 참조하여 다음 폼을 작성해 보십시오.



컨트롤의 속성은 다음과 같습니다..

No	Name	Text	Fore Color	Size	Location
1	Form1	Sin, Cos Graph	변경없음	500,300	0,0
2	Label5	변경없음	변경없음	72,20	16,221
3	Label1	변경없음	RED	90,20	106,221
4	Label2	변경없음	BLUE	90,20	201,221
6	Label4	변경없음	YELLOW	90,20	393,221
7	Button1	Exit	변경없음	72,20	16,245

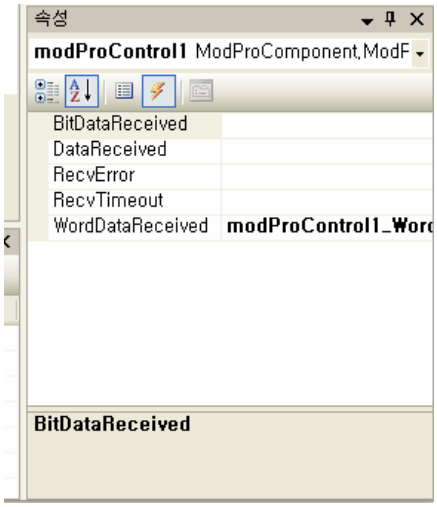
ModProControl을 Form1위로 끌어놓은 후 속성을 변경하여 주십시오.



ModProControl의 속성은 아래와 같습니다.

(Name)	**ModProControl1**
Baudrate	9600
EncodingType	RTU
GenerateMember	True
Modifiers	**Friend**
Ports	**COM1**
QueryInterval	**20**
Timeout	**500**

Ports를 COM1으로 두었고 QueryInterval을 20, Timeout을 500으로 두었습니다.



위는 ModProControl의 이벤트입니다. WordDataReceived이벤트를 더블 클릭하면 그에 해당하는 이벤트 핸들러 함수가 자동으로 생성됩니다.

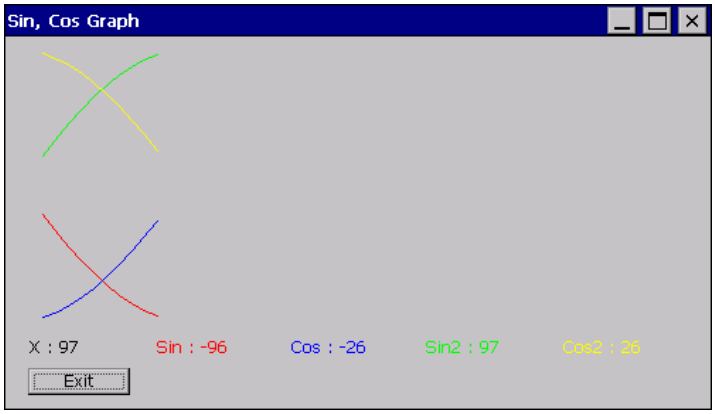
2. 아래는 소스 코드입니다.

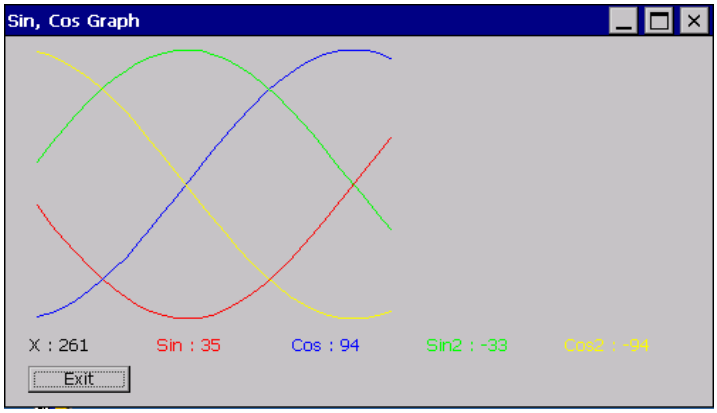
소스코드는 메소드명만 잘 확인하신다면 어떤 부분에 코드를 넣어야 할지 알 수 있습니다.

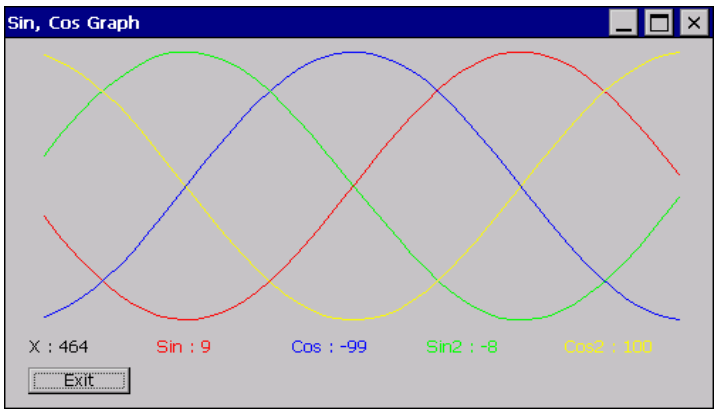
```
Public Class Form1
    Dim fFlag As Boolean = True   '화면의 갱신을 위한 플래그 변수
    Dim oldData(5) As Integer     'DrawLine의 인자로 사용할 과거 좌표값

    '다음은 ModProControl의 WordDataReceived를 더블 클릭하면 생성되는
    '핸들러 함수 입니다.
    Private Sub ModProControl1_WordDataReceived(ByVal e As
      ModProComponent.ModProEventArgs) Handles ModProControl1.WordDataReceived

        '.net에서는 보다 편한 그래픽 사용을 위해 그래픽 객체를 제공합니다.
        '이 Graphics를 이용하여 그래프를 그려 보도록 하겠습니다.
        '우선 폼위에 그림을 그리기위해선 폼의 그래픽 객체를 얻어야하는데 방법은
        '아래와 갔습니다. 현재 클래스를 지칭하는 Me 키워드는 현재 Form클래스를 '의미하며
        Me.CreateGraphics()라는 메소드를 통해 현재 폼의 그림그리기가
        '가능한 Graphics객체를 반환하게 됩니다.

        Dim g As Graphics = Me.CreateGraphics() '선을 그리기위한 그래픽 객체 할당
        '0x7000~0x7004까지의 값
        '각 값은 0x7000 : X좌표의값
        '       0x7001 : Y좌표(Sin)의 값
        '       0x7002 : Y좌표(Cos)의 값
        '       0x7003 : Y좌표(위상을 이동시킨 Sin)의 값
        '       0x7004 : Y좌표(위상을 이동시킨 Cos)의 값

        Dim datas As ArrayList = e.GetDatas()
        Dim data(5) As Integer                    //수신받을 데이터를 저장할 배열
        Dim i As Integer

        //Y축의 좌표는 -100에서 100사이의 값으로 이동하게 되는데
        //수신받은 값은 unsigned 형태의 값입니다
        //이를 signed형태의 값으로 변경 시키기 위한 코드입니다.
        For i = 0 To 4
            Dim a As Integer = Convert.ToInt32(datas(i))
            data(i) = IIf(a > 32767, a - 65536, a)
        Next

        //각 Label에 수신받은 데이터를 출력
        Label5.Text = "X : " + data(0).ToString()
        Label1.Text = "Sin : " + data(1).ToString()
        Label2.Text = "Cos : " + data(2).ToString()
        Label3.Text = "Sin2 : " + data(3).ToString()
        Label4.Text = "Cos2 : " + data(4).ToString()

        //X좌표는 0~471사이의 값을 증가하면서 반복적으로 나오게됩니다
        //만약 좌표의 값이 471에서 0으로 넘어가는 순간엔 화면을 갱신 시켜주기위해
        //fFlag를 두었습니다
        //만약 화면 갱신 플래그 변수가 true라면
        If (fFlag) Then
            //과거좌표에 현재좌표를 저장
            For i = 0 To 4
                oldData(i) = data(i)
            Next
```

```
            //화면갱신 플래그변수를 false로 변경
            fFlag = False
            // 화면을 갱신
            Me.Invalidate()
        End If
        //현재좌표의 값이 처음으로 돌아와 과거좌표 보다 작아진경우
        If (oldData(0) > data(0)) Then
            //화면을 갱신하고 화면 갱신플래그를 True로 변경
            Me.Invalidate()
            fFlag = True
        //그렇지 않은경우(현재좌표의 값이 과거 좌표값 보다 큰 경우)
        Else
            //각 선들의 색깔을 정의하고
            Dim colors() As Color = {Color.Red, Color.Blue, Color.Lime, Color.Yellow}
            //과거좌표에서부터 현재 좌표로 선을 그림
            For i = 1 To 4
                g.DrawLine(New Pen(colors(i - 1), 1.0F), 10 + oldData(0), 110 + oldData(i),
                           10 + data(0), 110 + data(i))
            Next
        End If
        //현재 좌표의 데이터를 과거좌표에 저장
        For i = 0 To 4
            oldData(i) = data(i)
        Next
        //그래픽을 종료시킨다
        g.Dispose()
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
                                        Handles MyBase.Load
        //ModProControl1을 속성값으로 초기화하고
        ModProControl1.InitialIzeModProS()
        //0x7000번지부터 5개의 값을 읽어온다
        ModProControl1.AutoWordRead(1, 1, &H7000, 5)
    End Sub
    //button1클릭시 ModProControl을 닫고 프로그램을 종료한다
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
                                        Handles Button1.Click
        ModProControl1.Close()
        Application.Exit()
    End Sub
    //폼이 종료되기전 ModProControl을 닫는다.
    Private Sub Form1_Closing(ByVal sender As System.Object, ByVal e As
                        System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
        ModProControl1.Close()
    End Sub
End Class
```

아래는 실행 화면입니다.







다음과 같은 4개의 그래프가 그려지는 것을 확인할 수 있습니다 붉은색은 Sin, 파란색은 Cos, 초록색은 위상을 이동시킨 Sin, 노란색은 위상을 이동시킨 Cos 그래프입니다. 이와 같은 방식을 이용한다면 디바이스(PLC)에서 Sensing 하는 데이터를 그래프로 디스플레이 하는 것도 가능합니다.

아래는 Cubloc 소스입니다.

```
Const Device = CB290
 Opencom     1, 9600,3,100,100
 Set Modbus 1,1,50
 Set Rs485   1, 55
 Set Onglobal On
 Ramclear
 Set Outonly On

 Dim pi2 As Single
 pi2 = 2 * 3.14   '2PI Sin, Cos그래프
 Dim itvl As Single
 itvl = pi2 / 471.0       '2PI/넓이로 하여 증가값으로 설정

 Dim f As Single
 Dim f2 As Single
 Dim n As Integer
 Dim nSin As Integer
 Dim nCos As Integer
 Dim nHSin As Integer
 Dim nHCos As Integer
 f=0.0                        '초기값 지정
 f2=pi2*0.5                   '위상을 이동시켜 초기값지정

 Do
        n=Floor (f/pi2*471)              'X좌표값
        nSin=Floor((Sin f)*100)          'Y좌표(Sin)값
        nCos=Floor((Cos f)*100)          'Y좌표(Cos) 값
        nHSin =Floor((Sin f2)*100)     'Y좌표(위상을 이동한 Sin) 값
        nHCos = Floor((Cos f2)*100)    'Y좌표(위상을 이동한 Cos) 값
        _D(0) = n                        'D0(0x7000)영역의 X좌표값 저장
        _D(1) = nSin                     'D1(0x7001)영역의 Y좌표(Sin)값 저장
        _D(2) = nCos                     'D2(0x7002)영역의 Y좌표(Cos)값 저장
        _D(3) = nHSin                    'D3(0x7003)영역의 Y좌표(위상 이동한 Sin)값 저장
        _D(4) = nHCos
        f=f+itvl                   'f를 증가값만큼 증가
        f2=f2+itvl                       'f2를 증가값만큼 증가
        If f >=pi2 Then                  'f가 2Pi를 넘으면 초기화
                f=0
        End If
        If f2 >=pi2 Then           'f2가 2Pi를 넘으면 초기화
                f2=0
        End If
        Delay 3                          '딜레이를 주어 속도조절
 Loop
```