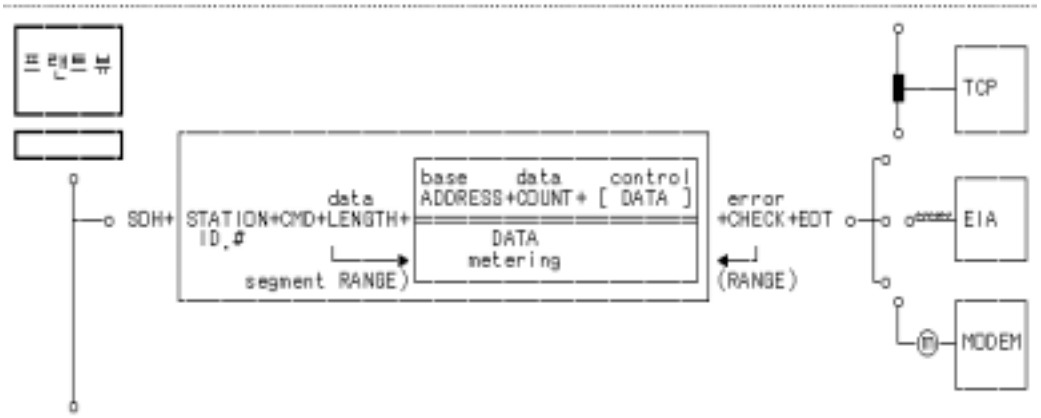
# pv-COMM

# 프레임 편집

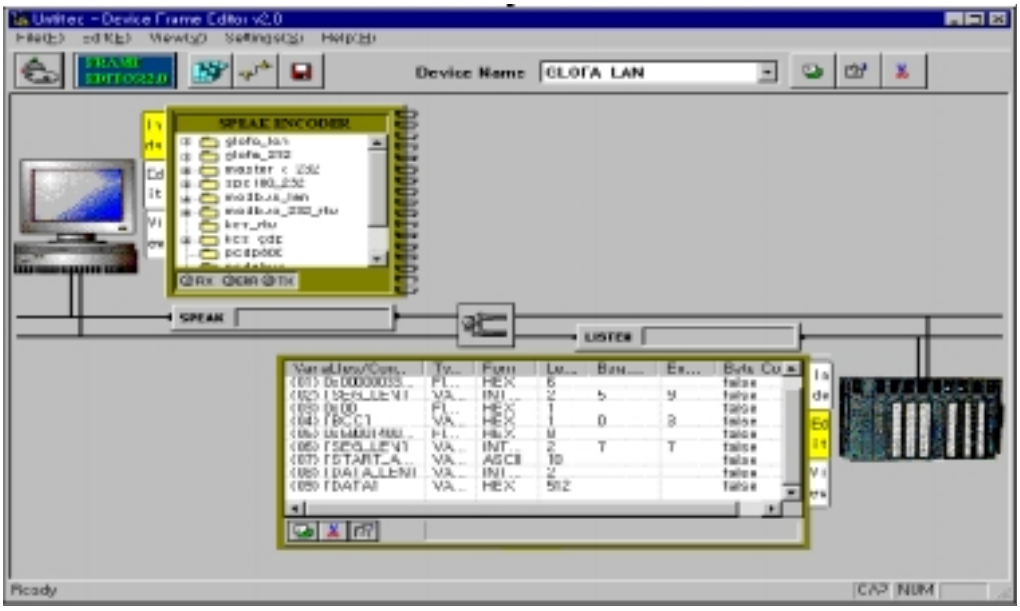
프레임 에디터는 프랜트뷰의 상위(HOST) 혹은 하위(RTU, PLC, DDC등) 시스템간의 데이터 교환 있어, 프로그래밍을 하지않고 통신이 이루어 지도록 하는 기능이다.

즉, 통신선 상에 오가는 프레임을 구성하는 세그먼트들의 요소와 형식을 정의하는 기능이다.

예외는 있으나 대부분 기기들의 프레임은 "<그림 1> 데이터 교환" 같은 형식으로 구성되며, 이를 구현하기 위해 "<그림 2>프레임 편집기"와 같은 기능이 제공된다.




<그림 1> 데이터 교환




<그림 2>프레임 편집기

# 프레임 종류

임의의 기기에 대해 프레임은 "<표 1>프레임의 종류"에서 보듯이 그 기기가 요구하는 프레임의 종류만큼 정의 해야한다. 경우에 따라 요구를 인식하고 다시 응답을 요구하는 경우도 있으며, 이런경우 이들 각각에 대해 프레임이 별도로 정의 되어야한다.

| | | |
|---|---|---|
| 계측 | 요구 | STX+CMD+LENGTH+BASE ADDRESS+COUNT+BCC+ETX |
| | 응답 | STX+CMD+LENGTH+…DATA… +BCC+ETX |
| 제어 | 요구 | STX+CMD+LENGTH+ADDRESS+COUNT+. ..DATA...+BCC+ETX |
| | 응답 | STX + CMD + LENGTH + BCC + ETX |

<표 1> 프레임의 종류

# 세그먼트 편집

## 상수/변수명(QUALIFIER)

세그먼트의 이름을 정의 하는데 있어 상수명(CONSTANT QUALIFIER)은 임의로 정 할 수있으나 변수명인 경우 지정 명칭(PREDEFINED VARIABLE QUALIFIER)만 사용 할 수있음니다. 즉, "<그림 3> 세그먼트 이름의 의미"에서 보듯이 "STATION", "LENGTH", "RANGE", "ADDRESS", "COUNT" 그리고 "CHECK"만으로 제한 됩니다.


```
* Legend :
        SOH       C$SOH         :  Start Of Head
        STATION   V$STATION     :  Station ID, in "1:N" configuration
        CMD       C$CMD         :  Command
        LENGTH    V$LENGTH      :  Data Length
        RANGE                   :
                  SEG_START     :  the index of start/end segment range in the frame
                  SEG_END       :
        ADDRESS   V$ADDRESS     : Base/Start Address
        COUNT     V$COUNT       : Data Count
        CHECK     V$CHECK       : Error Check w/ Algo, BCC, BCC_XOR, CRC, LRC,...
        EOT       C$EOT         : End Of Transmission
```

<그림 3> 세그먼트 이름의 의미

### STATION

다중(MULTI-DROP 혹은 NETWORK) 구성에서 개개 디바이스의 ID.를 프레임에서 지정해야만 해당 디바이스가 써비스를 행한다. 이 경우 동일 선상의 모든 디바이스에 동일한 프레임이 전달되나, 각 디바이스는 자신의 "STATION" ID.에 대해서만 행위를 하게된다.

### LENGTH

데이터 "LENGTH"는 "RANGE"가 지정하는 바이트수를 의미하며, 바이트수는 "PV-FRAME"내에서 계산되어 집니다. 따라서 정의는 디바이스에서 규정하는 바이트수만 정의해야 한다.

### RANGE

PV-FRAME이 데이터 "LENGTH"를 계산하거나, 프레임 전송시 오류를 검사하기 위해, 사용자는 세그먼트의 범위인 시작과 끝을 지정 해야 한다. 이 RANGE의 지정은 디바이스 규정과는 무관하다.

### ADDRESS

계측, 제어 할 상대 디바이스의 메모리 주소를 의미 한다. 계측시에는 보통 블록으로 데이터를 요구하기 때문에 이어지는 데이터 "COUNT" 세그먼트와 함께 "ADDRESS" 범위를 정하게 되고, 제어시에는 이어지는 데이터 세그먼트와 함께 제어값의 주소를 지정한다.

### COUNT

일반적인 경우 계측시에만 이용되고 "ADDRESS" 세그먼트와 함께 계측 할 데이터의 범위를 정하기 위한 데이터 갯수이다.

### CHECK

프레임 전송시 오류를 검사하기 위해 "RANGE"에서 지정한 세그먼트에 대한 "BCC", "BCC_XOR", "CRC", "LRC",... 등의 알고리즘으로 검사 코드를 삽입한다. 검사 코드는 프레임을 ENCODE/DECODE 시 계산되어 지기 때문에 정의는 세그먼트의 범위인 시작과 끝인 RANGE만 정의 합니다.

# 타입(DATA TYPE)

세그먼트의 데이터 타입은 "<표2> 데이터 타입"에서 보듯이 6 종류(FLOAT, DEFAULT, INTEGER, ASCII HEXA, ASCII, ASCII FLOAT)가 있을 수 있으며 이들 만을 지정 할 수있다.

| FLOAT | DEFAULT | INTEGER | ASCII HEXA | ASCII DECIMAL | STRING | ASCII FLOAT |
|---|---|---|---|---|---|---|
| 32767.89 | ???????? | 32767 | 37 46 46 46 | 33 32 37 36 37 | I love you. | 33 32 37 36 37 '.' 38 39 |

<표 2> 데이터 타입

* REMARK :

DEFAULT는 DEVICE에서 규정하고 있지 않는 경우입니다. 보통 FRAME에 실제로 사용하고 있지 않으나 다음에 사용할 수 있도록 여유 공간을 둔 경우입니다. 또한 프랜뷰에서 이용 할 필요가 없을 경우도 있을 수 있읍니다.

### 길이(SEGMENT BYTE SIZE)

세그먼트가 차지하는 바이트 길이로서, 정의는 디바이스에서 규정하는 바이트수 만큼 정의해야 한다.

### 시작/끝 SEG(SEGMENT START/END )

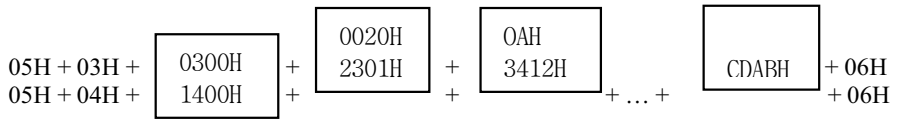
"LENGTH", "CHECK" 세그먼트 계산의 범위를 위해 프레임 내의 세그먼트 시작과 끝의 인덱스를 지정해야한다.

### 바이트 변환(ORDER CONVERSION)

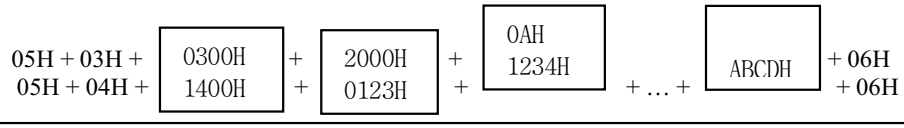
모든 세그먼트의 바이트 표현을 상위, 하위 바이트를 디바이스에서 규 정하는 대로 프레임을 구성해 주기 위함이다."예)"에서 보듯이 **"CROSS ORDER"**로 할 경우 **2** 바이트 이상의 세그먼트는 그 순서가 바뀌게 된다.

**예) "2000H 번지에서 부터 100개의 데이트를 가져오라"**

**NORMAL ORDER : :**




**CROSS ORDER : :**



# 🗁 DRIVER 설정

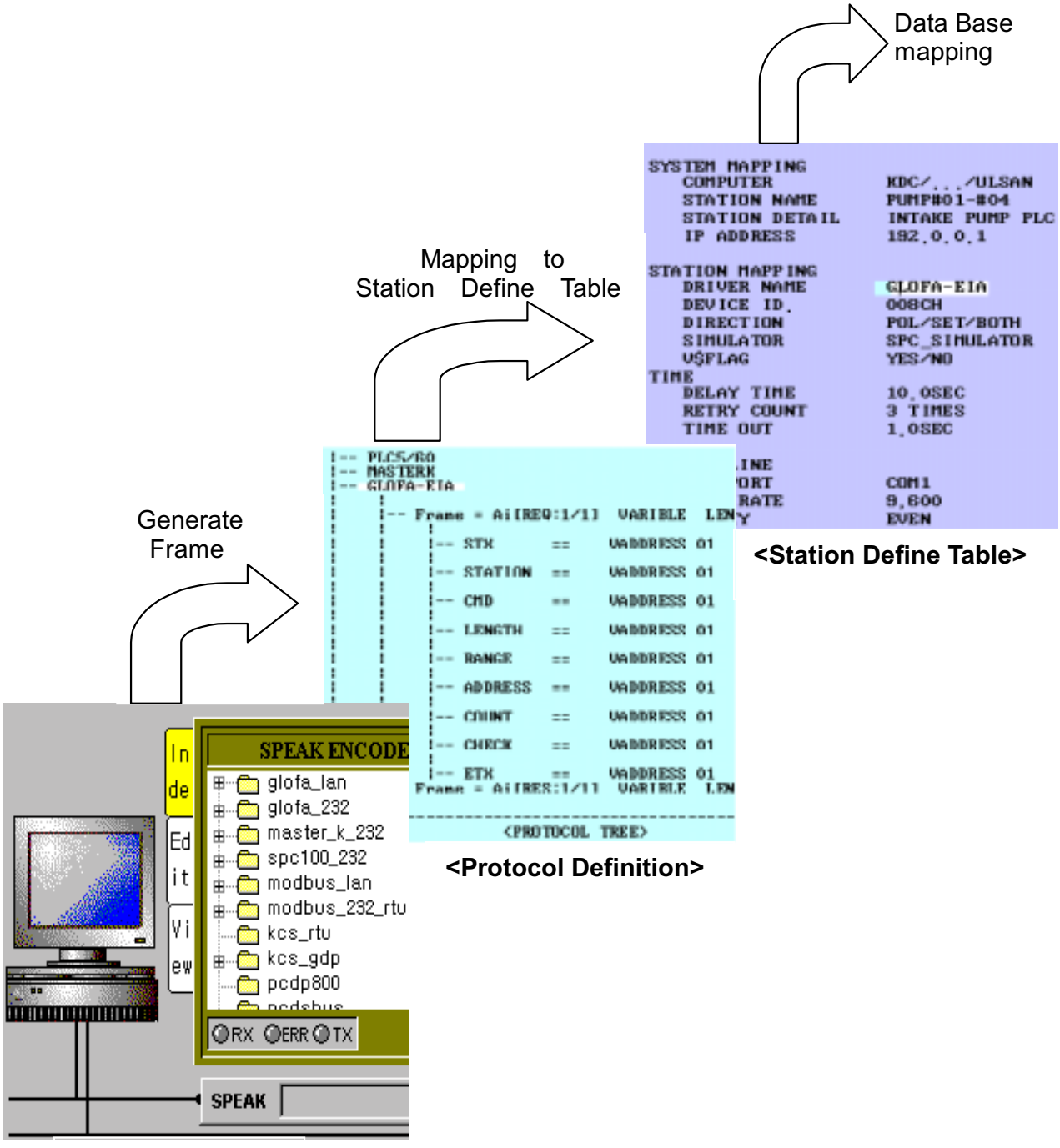
## PROTOCOL 정의

pv-COMM TREE에서 PROTOCOL FORMAT을 정의하기 위해서는 우선 송수신 FRAME마다의 SEGMENT를 정의 해야 한다.

```
PROTOCOL
 |
 |-- TREE RESERVED KEYs
 |     |
 |     |-- TITLE TERMINATOR == { == }
 |     |
 |     |-- CONSTANT == {C$STX=0X05H, C$CMD=0X06H, C$ETX=0X07H }
 |     |
 |     |-- VARIBLE  == {V$STATION, V$ADDRESS, V$LENGTH, V$COUNT, V$CHECK, C$STX,
 |     |                C$CMD, C$ETX, ... }
 |     |
 |     |-- TYPE     == {signed Int, unsigned Int, Float, signed Ascii Int,
|      |               unsigned Ascii Int, Ascii Float, Ascii Hexa, String, Default}
 |     |
 |     |-- ORDER    == {Normal, Cross}
 |     |
 |     |-- FORMAT   == {Ascii float}
 |     |
 |
 |-- MOD-BUS PLUS
 |
 |-- GLOFA-EIA
 |     |
 |     |-- Frame = Ai[REQ:1/1]  VARIBLE  LENGTH TYPE    ORDER  FORMAT RANGE
 |     |     |
 |     |     |-- STX      ==    VADDRESS 01     INTEGER NORMAL 6.2
 |     |     |
 |     |     |-- STATION  ==    VADDRESS 01     INTEGER NORMAL 6.2
 |     |     |
 |     |     |-- CMD      ==    VADDRESS 01     INTEGER NORMAL 6.2
 |     |     |
 |     |     |-- LENGTH   ==    VADDRESS 01     INTEGER NORMAL 6.2
 |     |     |
 |     |     |-- RANGE    ==    VADDRESS 01     INTEGER NORMAL 6.2
 |     |     |
 |     |     |-- ADDRESS  ==    VADDRESS 01     INTEGER NORMAL 6.2
 |     |     |
 |     |     |-- COUNT    ==    VADDRESS 01     INTEGER NORMAL 6.2
 |     |     |
 |     |     |-- CHECK    ==    VADDRESS 01     INTEGER NORMAL 6.2
 |     |     |
 |     |     |-- ETX      ==    VADDRESS 01     INTEGER NORMAL 6.2
 |     |-- Frame = Ai[RES:1/1]  VARIBLE  LENGTH TYPE    ORDER  FORMAT RANGE
 |     |-- Frame = Ao[REQ:1/1]  VARIBLE  LENGTH TYPE    ORDER  FORMAT RANGE
 |     |-- Frame = Ao[RES:1/1]  VARIBLE  LENGTH TYPE    ORDER  FORMAT RANGE
 |     |-- Frame = Di[REQ:1/1]  VARIBLE  LENGTH TYPE    ORDER  FORMAT RANGE
 |     |-- Frame = Di[RES:1/1]  VARIBLE  LENGTH TYPE    ORDER  FORMAT RANGE
 |     |-- Frame = Do[REQ:1/1]  VARIBLE  LENGTH TYPE    ORDER  FORMAT RANGE
```

# PROTOCOL 등록

DEVICE를 관리하는 DEVICE MANAGER는 STATION DEFINE FORM TABLE(SDF)에 정의된 DEVICE의 해당 PROTOCOL FORMAT를 참조하여 통신하게 된다.



# 🗁 CUSTOMER DRIVER

## OVERVIEW

**pv-COMM으로 통신 할 수 없는 경우, 통신 모듈을 플랜트뷰에 BINDING 시키기 위한 방법을 설명하려 한다. 여기서 BINDING이란 RUN TIME시에 결속하게 되어야 되기 때문에 DLL형식으로 플랜트뷰에 LOAD된다.**

## Interconnection

**플랜트뷰와 사용자 프로그램간의 연결은 pv-EDIT에서 "IO PROGRAM" FIELD에 사용자 DRIVER명을 기입하고, "<그림>플랜트뷰 DIRECTORY내의 사용자 DRIVER"에서 보듯이 해당 DLL FILE을 위치하고 "<표> EIA LINE SETUP"에서 적당한 SETUP이 요구된다.**

```
plantVIEW
 |   1_수행 환경                    |   4_프로젝트
 |    |-- A_배치 파일               |    |-- A_포스코 압연 라인
 |    |-- B_실행 파일               |    |    |-- A_DATA 설계 정의
 |    |-- C_파트 파일               |    |    |    |-- a_Eng Ref File
 |    |   DEVICE#01.DLL            |    |    |    |-- b_Serv Ref File
 |    |   DEVICE#02.DLL            |    |    |    |   DEVICE#01.COMSPEC
 |    |   DEVICE#03.DLL            |    |    |    |   DEVICE#02.COMSPEC
 |    |        . . .               |    |    |    |   DEVICE#03.COMSPEC
 |    |        . . .               |    |    |    |
 |    |                            |    |    |    |        . . .
 |    |-- D_bmp 파일               |    |    |    |
 |    |-- E_icon 파일              |    |    |-- D_장비 입력 출력
 |                                 |    |    |    |-- a_Prog Ref File
 |                                 |    |    |    |   SCAN.REF
 |                                 |    |    |    |
```

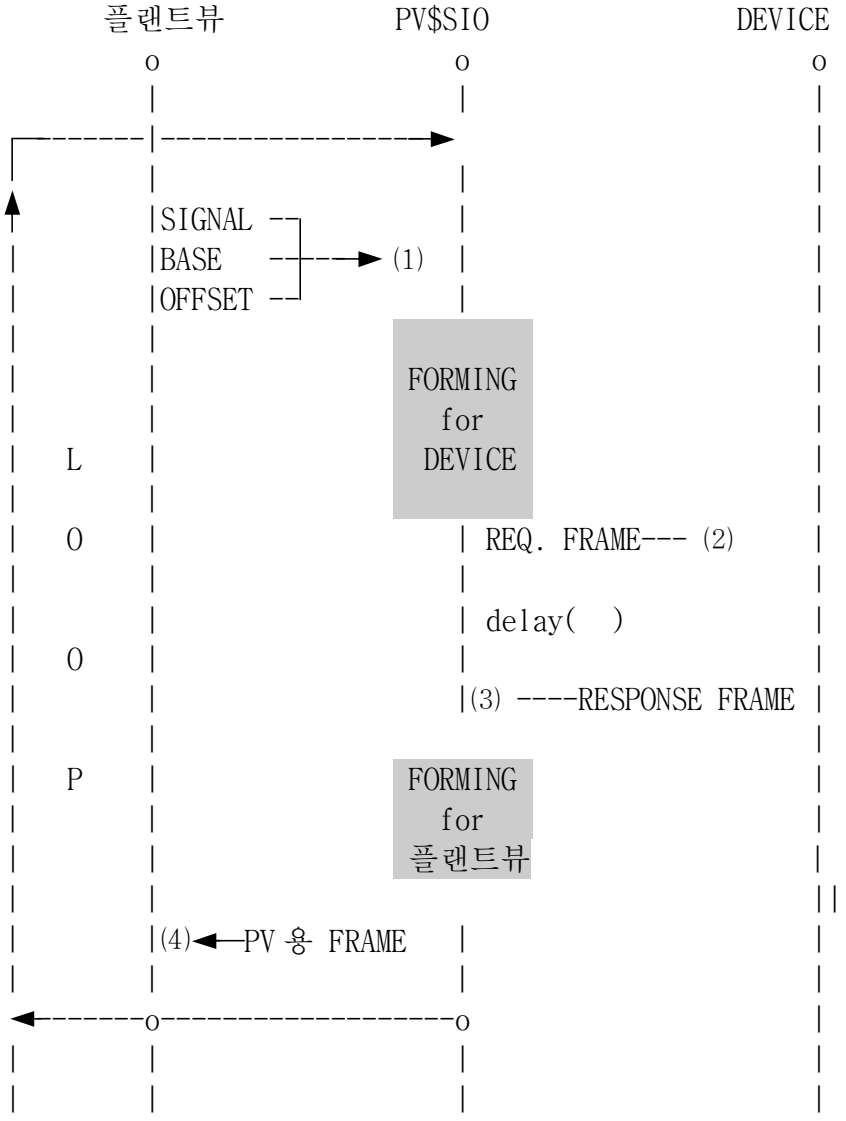
**<그림>플랜트뷰 DIRECTORY 내의 사용자 DRIVER**

```
                              .Baud   /  1200, 2400, 4800, 9600, ...      / BPS
SIO_BAUD  =       9600

                             .Port   /  COM1, COM2, COM3, COM4            / CHAN
SIO_PORT  =       COM2

                              .Data   /    5,    6,    7,    8           / BIT
SIO_DATA_BIT =    8

                              .Parity / None:0  Odd:1 Even:2 Mark:3 Space:4 / BIT
SIO_PARITY_BIT =  0

                              .Stop   /   1Stop:0    1.5Stop:1    2Stop:2   / BIT
SIO_STOP_BIT  =   0
```

**<표> EIA LINE SETUP**

# IMPLEMENT

**플랜트뷰는 "어떤 신호<signal> 종류에 대해 어디<base>부터 얼마만큼 <offset> 데이터를 읽어 플랜뷰<pv_txrx>에 보내라" 또는 "어떤 신호 <signal> 종류에 대해 어디<base>로 얼마<offset>만큼의 데이터<pv_txrx> 를 DEVICE에 보내라"고 요구하고, PV$SIO는 DEVICE가 인식할 수 있는 통신 FORMAT으로 전송 FRAME을 구성하고 이에 해당하는 수신 FRAME을 다시 플랜트뷰 인식할 수 있는 통신 FORMAT으로 재구성해야 한다.**



# 요구 형식

1) 플랜트뷰 "로 부터"

```
INT pv$sio (
  int   signal// signal discriminator eg) Analog Input -> AI
  int   base  // base address
  int   offset       // offset from base address
  char* pv_txrx // TxRx buffer with plantVIEW
);
```

**2) DEVICE "로"**

```
INT COM_SIO_TX(
  frame,        // tx frame to device
  TX_LENGTH     // tx frame length
);
```

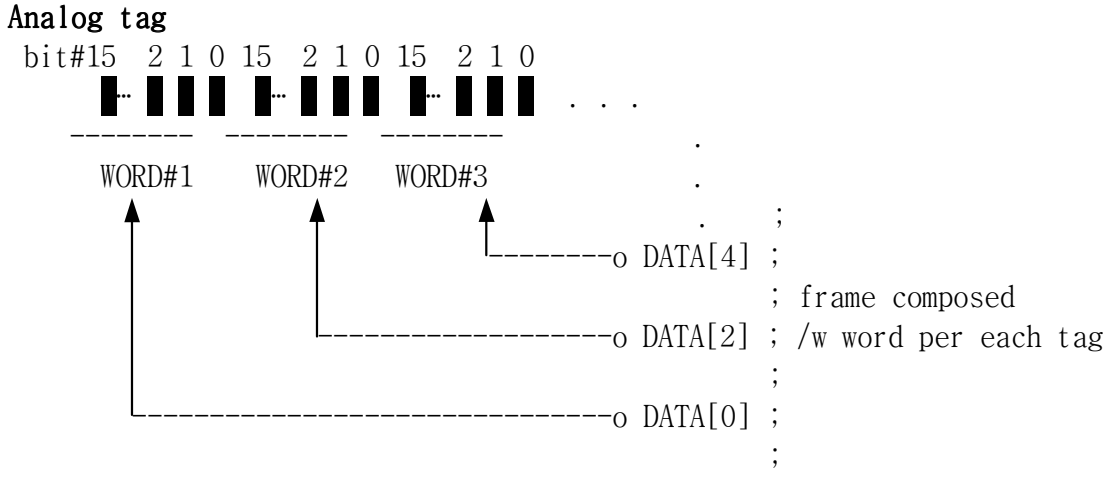
**3) DEVICE "에서"**

```
INT COM_SIO_RX(
  frame,        // rx frame from device
  RX_LENGTH     // rx frame length
);
```

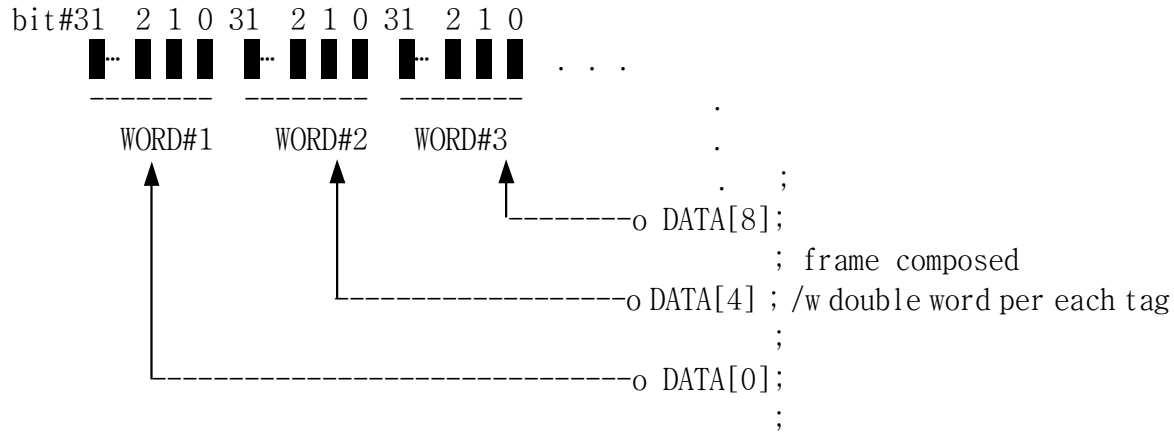
# 통신 형식

**1)** 플랜트뷰 "로"

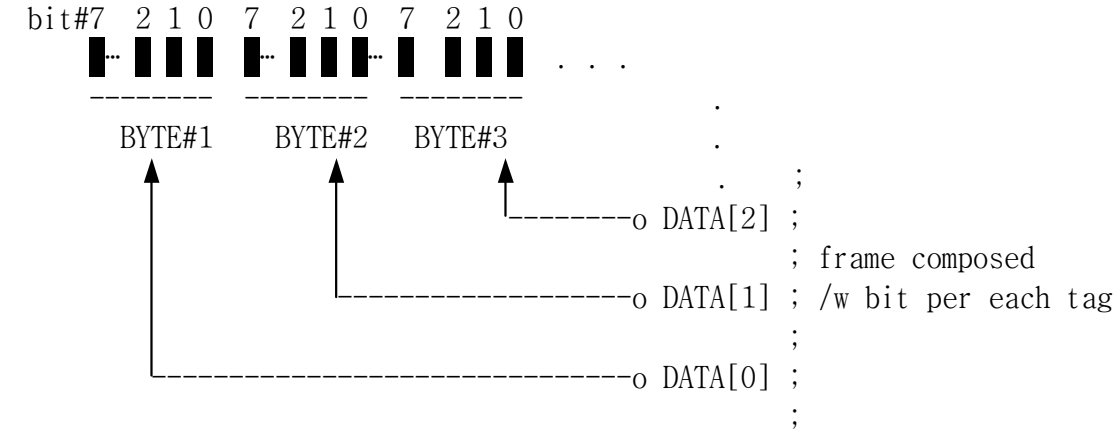
ANALOG TAG는 base ADDRESS로부터 offset 범위에 있는 DATA를 WORD, DISCRETE TAG는 BIT단위로 인접하여 구성된다.



Analog tag (Pulse)



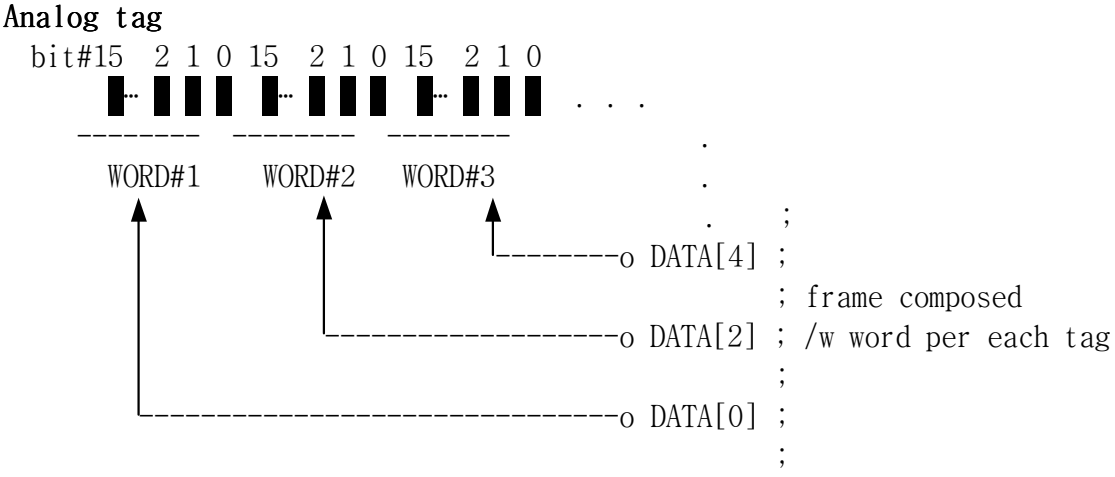

**Discrete tag**



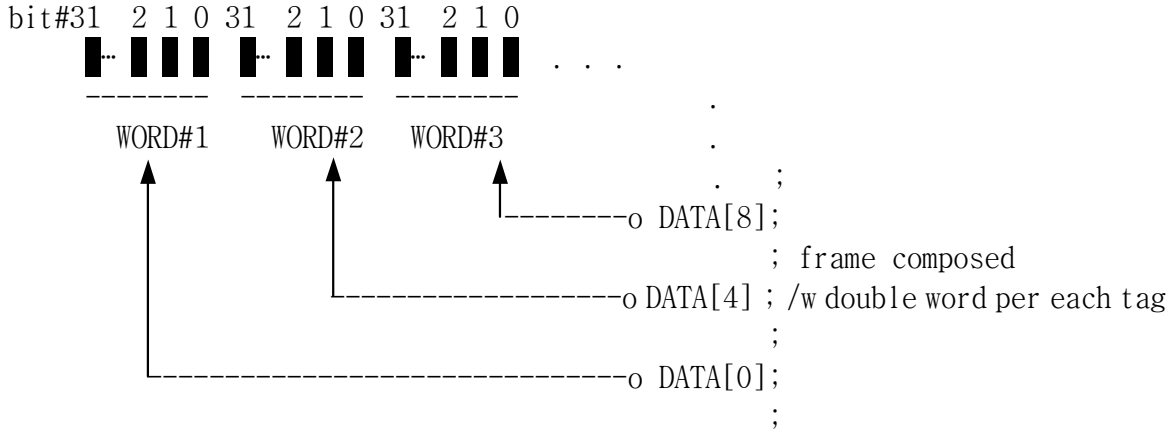

**2) 플랜트뷰 "에서"**

ANALOG TAG는 base ADDRESS로부터 offset 범위에 있는 DATA를 WORD, DISCRETE TAG는 BYTE단위로 인접하여 구성된다.

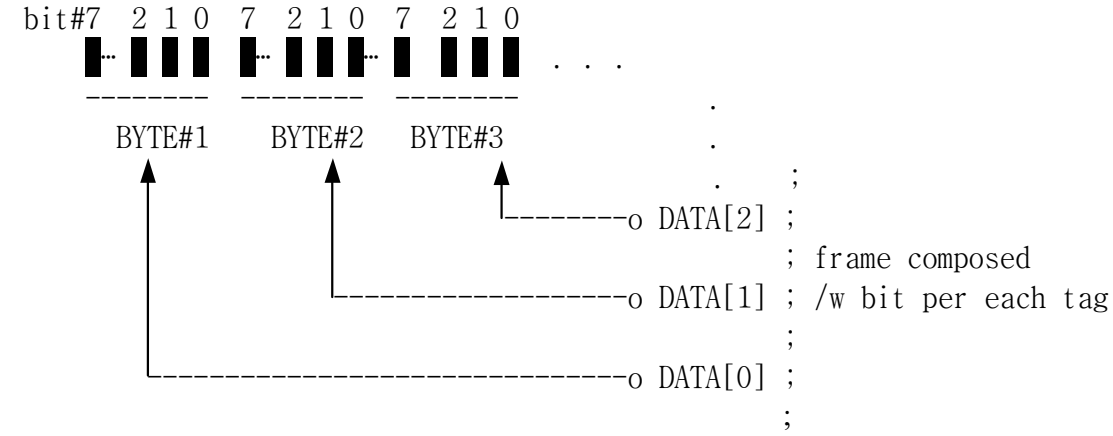
Analog tag



Analog tag (Pulse)




**Discrete tag**




### 3) DEVICE "와"

**DEVICE와의 통신 FORMAT은 해당 MANUAL 참조**

**상태보고**

SUCCESS < 00> ; normal
PARAMETER ; invalid parameter passing from plantVIEW
INVALID_SIGNAL <-11> ; invalid signal passing from plantVIEW
INVALID_BASE <-12> ; invalid base passing from plantVIEW
INVALID_OFFSET <-13> ; invalid offset passing from plantVIEW
INVALID_PV_TXRX <-14> ; invalid pv_txrx passing from plantVIEW
TX_FAIL <-21> ; transmission failure to serial device
RX_FAIL <-31> ; receive failure from serial device

**서비스 빌드**

DLL 명은 사용자가 정의한 POLLER 명을 의미하며, 또한 작성된 program 명과 일치해야 한다.

**C:\ >BUILD DLL_NAME<enter>**

**필요 환경**

Windows-NT 3.51 or later version
WIN-32 SDK(Software Development Kit)

**EXAMPLE**

```
// HR2500.C
/*-------------------------------------------------------------------------------
    Function Name    pv$sio ; this function name is fixed in plantVIEW
    platform
        Windows-NT ver. 3.51 OS and "SDK" required
    Usage
        C: BUILD HR2500 <enter>
    Input
        int     signal      ; signal discremeter      scan       : AI, DI, PI
                                                      control : AO, DO, PO
        int     base         ; base address
        int     offset       ; offset from base address
    Output
        char* pv_txrx     ; data buffer       scan       : send buffer to plantVIEW
                                              control : receive buffer from plantVIEW
    ids
        SUCCESS                 < 00> ; normal
        PARAMETER                     ; invalid parmater passing from plantVIEW
          INVALID_SIGNAL    <-11> ; invalid   signal   passing from plantVIEW
          INVALID_BASE      <-12> ; invalid   base     passing from plantVIEW
          INVALID_OFFSET    <-13> ; invalid   offset   passing from plantVIEW
          INVALID_PV_TXRX <-14> ; invalid   pv_txrx passing from plantVIEW
        TX_FAIL                  <-21> ; transmission failure to serial device
        RX_FAIL                  <-31> ; receive failure from serial device
-------------------------------------------------------------------------------*/
#include      "pv_sio.h"
#define       TX_LENGTH 19                        // transmission frame's length
#define       RX_LENGTH 8 + (offset/2)*6     // receive frame's length
#define       ESC           0x1b

INT pv$sio (int signal, int base, int offset, char* pv_txrx)
{         char frame[512];      int i;          static int first_time = 1;

//1.  C H E C K    I N P U T    P A R A M E T E R    A N D    F I R S T    T I M E
    if ( base<0 || base>4          )                  return INVALID_BASE     ;
    if ( offset<0 || offset>120    )                  return INVALID_OFFSET ;
    if ( pv_txrx == NULL           )                   return INVALID_PV_TXRX;
    //if ( first_time == 1         )                  Initialize(); first_time = 0; }

    switch ( signal ) {
        case      AI :
         //2.   T X    P R O C E D U R E
         // 2.1 MAKE TX FRAME
            sprintf( frame, "TS0;%cT;FM2,%01d%02d,%01d%02d;", ESC, base, 1, base,
offset/2 );

         // 2.2 TRANSMISSION TO CUSTOM SERIAL DEVICE
            if ( COM_SIO_TX(frame, TX_LENGTH) != SUCCESS    )     return TX_FAIL;

         //3.   R X    P R O C E D U R E
         // 3.1 RECEIVE FROM CUSTOM SERIAL DEVICE
            if ( COM_SIO_RX(frame, RX_LENGTH) < RX_LENGTH )     return RX_FAIL;

         //4.   T R A M S M I T T E    T O    p l a n t V I E W
            for ( i = 0 ; i < offset/2 ; i++ ) {
                memcpy( &pv_txrx[i*2], &frame[8+i*6+4], 2 );
```

```
                }
                break;
            case    DI :    // Add your code here
            case    PI :    // Add your code here
            case    AO :    // Add your code here
            case    DO :    // Add your code here
            default :       return INVALID_SIGNAL;
        }
        return SUCCESS ;
    }
```