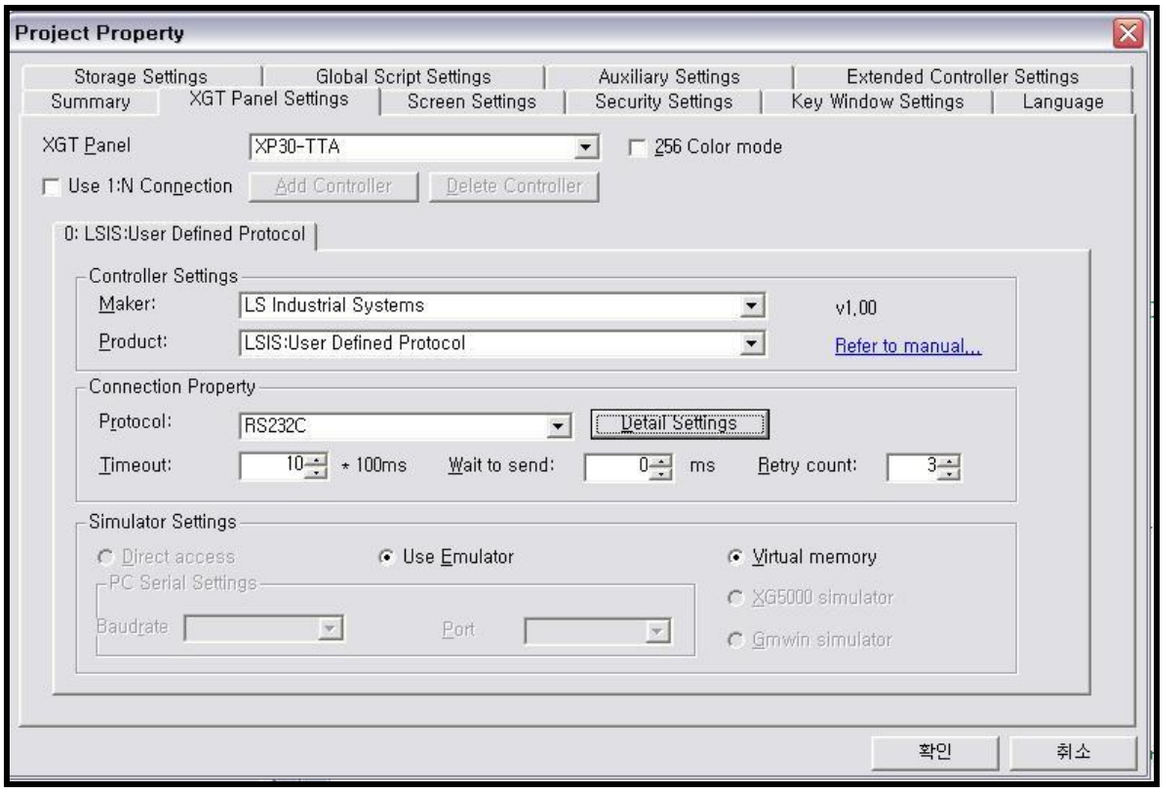
User Defined Communication Protocol

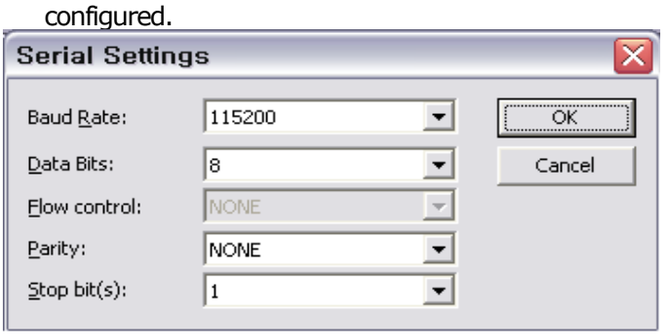
This is manual for user defined communication protocol which supports XP-Builder V1.22 or above, so please install XP-Builder V1.22 before operation. This manual describes only part of MODBUS RTU Master Communication; refer to XP-Builder manual for detailed script functions. For User defined communication, refer to XGT panel communication manual. For understanding MODBUS protocol, refer MODBUS protocol or XGT panel communication user's manual.

1. Communication Setting

(1) Run XP-Builder, [Project property] → [XGT-PANEL Setting], Click 'LS Industrial Systems' → 'LSIS:User Defined Protocol'.



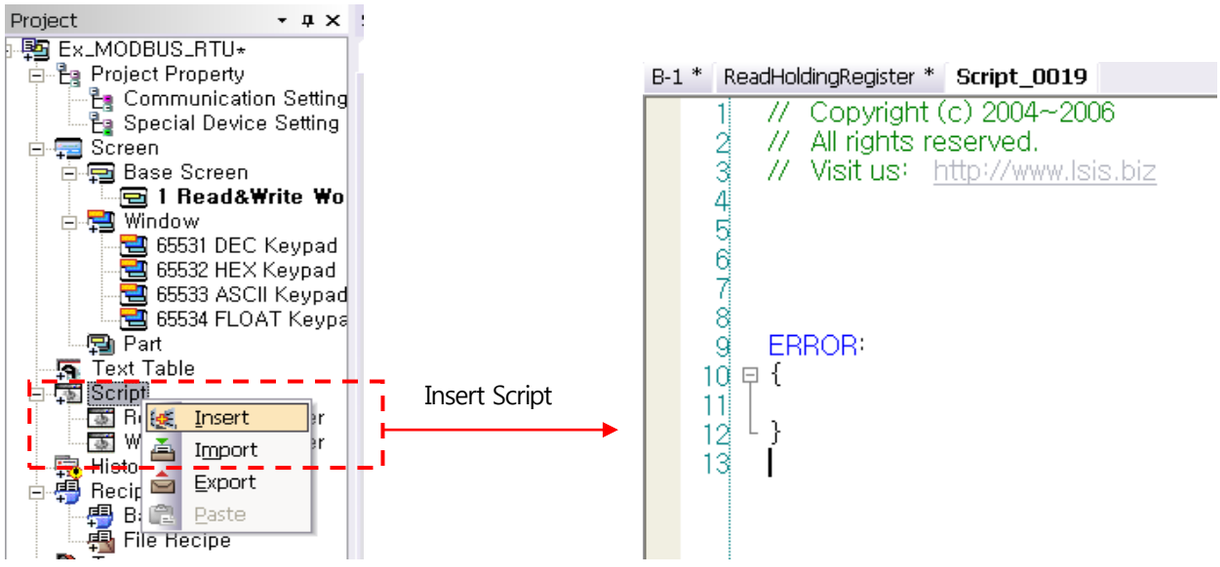
(2) Here, only serial communication is presented. Set connection protocol as 'RS-232C' and click detail settings. Set Baud Rate, Date Bits, Parity and Stop bits on 'Serial Settings'. However, state number cannot be configured because most of protocols include state number so that only physical connection part can be configured.



2. Writing script for reading word

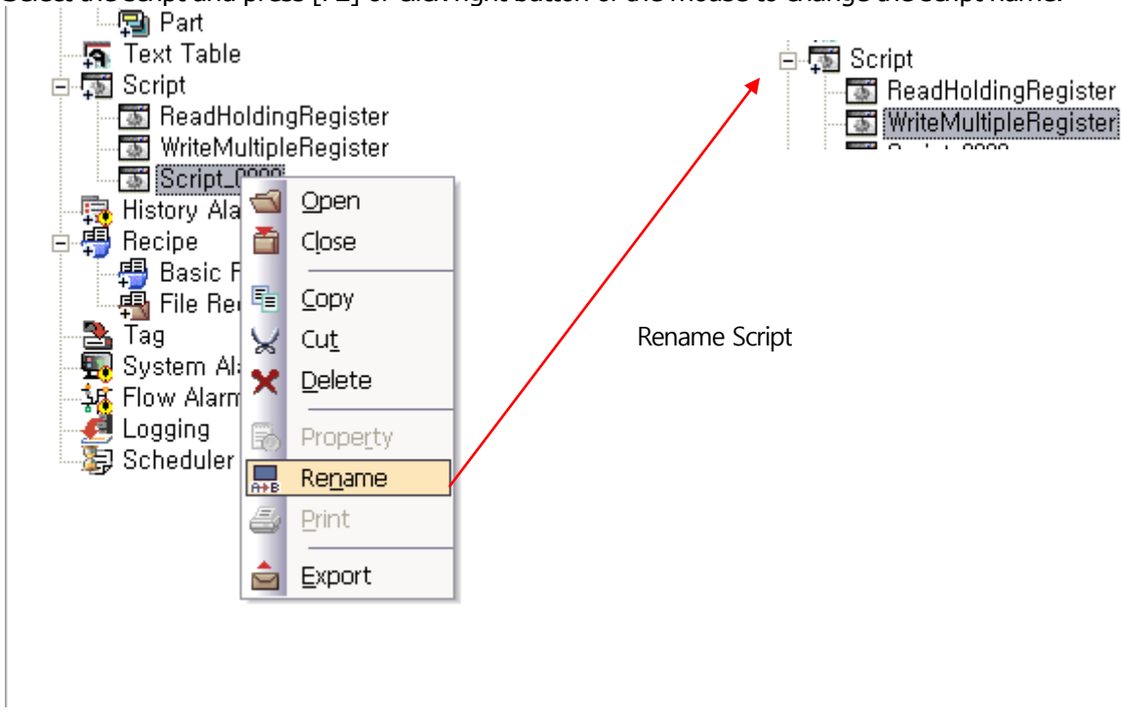
(1) Generate script

[Script] → [Insert] on project tree.



(2) Rename Script

Select the script and press [F2] or click right button of the mouse to change the script name.



3. Making script for writing word
(1) Check other script operations

```
/*********************************************

    MODBUS RTU Master Example

    # Writing Word
      - Function code: 0x10
      - Station number: 1
      - Number of writing : 10

*********************************************/

//Wait if other script is running
//'Set =Running' Tag as 1 when other script is running
// Required same script in the other script (Using as Flag)
if(@[X:=Running] == true)
//When other script is running, check the script  is completed or not every 20ms.
{
    for(int i=0; i<200; i++)
    {
        Sleep(20);
        if(@[X:=Running] == false)
        //Exit Loop when other script is done
            break;
    }
    //When other script is running after 4 seconds, ignore other script and start commu
    //4 seconds setting is just random, no means to be 'END'
}

//Script is running from now on
@[X:=Running] = true; //Set Operation Flag

@[S:HW02000]++; //No meaning, just count the script operation.
```

- Green letters are not related with operation, but described for understanding. There are two ways to make explanation as below. First, write comments between ` **/*** 'and ` ***/*** ' and then all texts are ignored as comments. Second, the line starting with ` **//** ` is considered as a comment same as C language.
- If registering Bit/Word device on tag, it's very convenient to use.

No	Group	Name	Device Type
1	Default	Address	WORD
2	Default	Number	WORD
3	Default	Error	BIT
4	Default	Value	WORD
5	Default	Running	BIT
6	Default	Writing Success	BIT

- 19~31 lines describe the way to set flags which display the bit device status. When the bit tag of '=running' is set to '1'(On) communication gets disable because other scripts are running. In other words, if '=Running' tag is 1 if statement checks the status in every 20ms.
- If '=running' tag is still '1' in 4 seconds, exit for statement and start communication.
- Set '=Running' tag as '1'(On) as operates communication(script occupation) like line 34. If insert same code while making other script, that operates followed by currently running script. (Prevention of collision occurred during scripts operation)
- Line 36 has no meaning (for checking and saving the number of script operation)

(2) Variable declaration and initialization

```
38  char Command[32], Response[32], TX_Data[20];  //Command sequence is Tx buffer, Response sequence is Rx buffer.
39  int Data[20];
40  short Address=0x0000, Checksum=0; //Clear Address, Checksum Variable (Initialization)
41  int SentSize=0, ReadSize=0, nPos=0; //
42
43  // initialize command[0] ~ command[31] to 0
44  @[X:=Error] = !Memset ( &Command[0], 0, 32 ); //32 TX Buffer intialized to 0, Return to 'False' when out of buffer range
45  // initialize response[0] ~ response[31] to 0
46  @[X:=Error] = !Memset ( &Response[0], 0, 32 ); //32 RX Buffer intialized to 0, Return to 'False' when out of buffer range
47
48  for(int i=0;i<10;i++){
49
50      GetData(@[W:HW100], i, &Data[i]);
51
52      TX_Data[i*2] = HIBYTE(Data[i]);
53      TX_Data[i*2 + 1] = LOBYTE(Data[i]);
54
55  }
```

- Line 38~41 initialize variables to use. Those variables are only valid in the current script as local variables.
- Arrays in line 38~39 are newly added since V1.22 and lower version do not support them.
- 'char' is 1byte. 'short' is 2byte(1word). 'int' is 4byte(1 double word).
- There will be warning after declaring unnecessary variables and not using it.
- Line 44 initialized 32 variables of Command[0] ~ Command[31] as 0 using with Memset function. Return '1' (true) when a function is normally proceed or return '0'(false) in case of errors or overflows. '!' is an operator to reverse, set '=error' tag as '1' when return to 'false'
- Line 46 is the same operation.
- Line 48~55 is the operation that stores HW100~HW109 data into Data[0]~Data[9] and again stores Data[0]~Data[9] into the TX_Data buffer as a byte

(3) Communication (Protocol) Data

```
57  Command[nPos++] = 0x01;                 // Station number
58  Command[nPos++] = 0x10;                 // Function code: write multiple register
59
60  Command[nPos++] = HIBYTE ( Address );  // High byte among Word address
61  Command[nPos++] = LOBYTE ( Address ); // Low byte among Word address
62
63  Command[nPos++] = 0;                    //No. of register high
64  Command[nPos++] = 10;                   //No. of register low
65
66  Command[nPos++] = 20;                   //Number of Bytes
67
68  for(i=0;i<20;i++){
69
70      Command[nPos++] = TX_Data[i];       //Writing byte data
71
72  }
73
74  @[X:=에러] = !CRC16 ( &Command[0], 27, &Checksum ); // CRC Calculation, Calculating Command[0] ~ [5] Checksum
75  Command[nPos++] = LOBYTE ( Checksum ); // Low Checksum byte
76  Command[nPos++] = HIBYTE ( Checksum );  // High Checksum byte
```

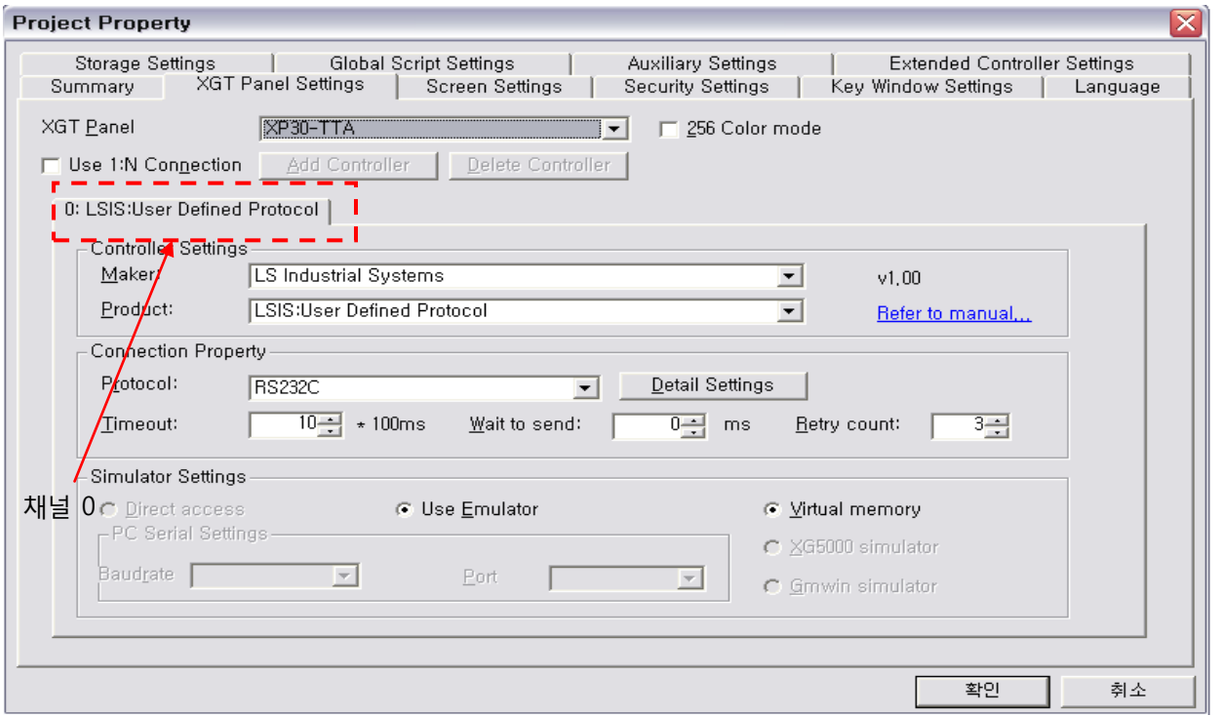
Enter the protocol data on communication data buffer. ([] : 1 byte)
[Station number][Function code][High Address][Low Address][No. of register high][No. of register low]
[No. of bytes][Data#1]…[Data#N][Low Checksum][High Checksum]

- Line 57~62 enter protocol data into command buffer.
- Line 74 calculates checksum(CRC16) of [state number]~[data#N]. It is convenient to use CRC16 function. Return value is a bit and is '1'(true) in case of normal operation.
- Line75~76 separate the checksum as bytes and store to the buffer.

(4) Sending Communication (Protocol)

```
78  SentSize = WriteToIO ( 0, &Command[0], nPos ); // Transmit Tx data on Channel #0, Return Value is as same as number of transmitted data byte
```

- The WirteToIO in line 78 is a command to send data. The return value is the number of data bytes sent. (int type)
- The first argument is the channel to send. (Very important)



(5) Receiving Response

```
80  if(SentSize) //Run when the number of transmitted data is not 0 - Transmission Success
81  {
82      Sleep( 50 );                                    // Waiting response, 50ms is random setting value
83      ReadSize = ReadFromIO ( 0, &Response[0], 8 );   //  Save Rx data(25bytes), Return Value is as same as number of recevied data byte
84
85      if( ReadSize == 8 ) //Run when received the data on success
86      {
87
88          @[X:=Writing Success] = true;
89
90      }
91
92      else if( ReadSize == 5){ //If error code occurred
93
94          @[S:HW130] = Response[1]; //Save error code
95          @[X:=Writing Success] = false;
96
97      }
98  }
99
100 @[X:=Running] = false; //Exit script operation
```

- Line 78 shows the number of data sent is saved in the SentSize variable.
- Line 80 shows a waiting for response from another communication device concluding previous communication was successful in case of not-zero data sent.
- After waiting for 50msec in line 82 start to read buffer in line 83. After reading eight data in channel 0, save them to response buffer. The ReadFromIO function returns the number bytes read.
- In line 85, '=SuccessToWrite' tag is set to '1' in case that the number of data loaded is eight. That means a success to write data. The MODBUS protocol returns 8 bytes in that case.
- Line 92 shows 5 bytes returned in case of error occurred.
- Line94 saves error code in word device which does not display error code on top of the device different from other communication functions. It is necessary to indicate error code to indentify error states.
- Line100 shows an operation which set '=running' tag to '0' in case of completion of operation.

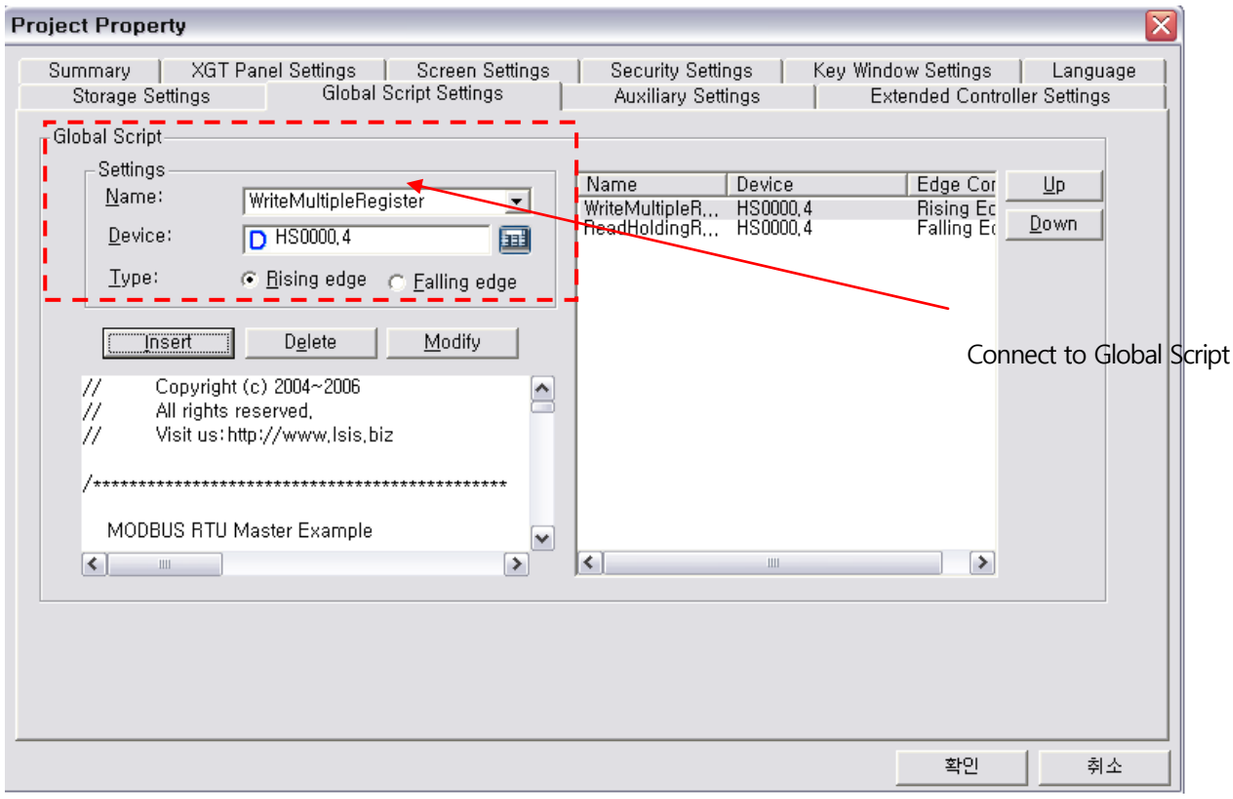
(6) Error handling

```
102  ERROR:
103  {
104      @[S:HW00129] = GetLastError ( ); //Save error number which occurred during script on HW129
105  }
106
```

- Error statement is required for handling unexpected errors like line 102 ~ 105.
- Line 104 shows a way to save return value from GetLastError function to the device. The GetLastError function returns error code.

4. Script Start

(1) Scripts can start in various ways. However, if you want to operate communication in global way you should connect to global script. You can use screen script in case of operation in a specific screen.



(2) Please note that the number of global and screen scripts to register is restricted.

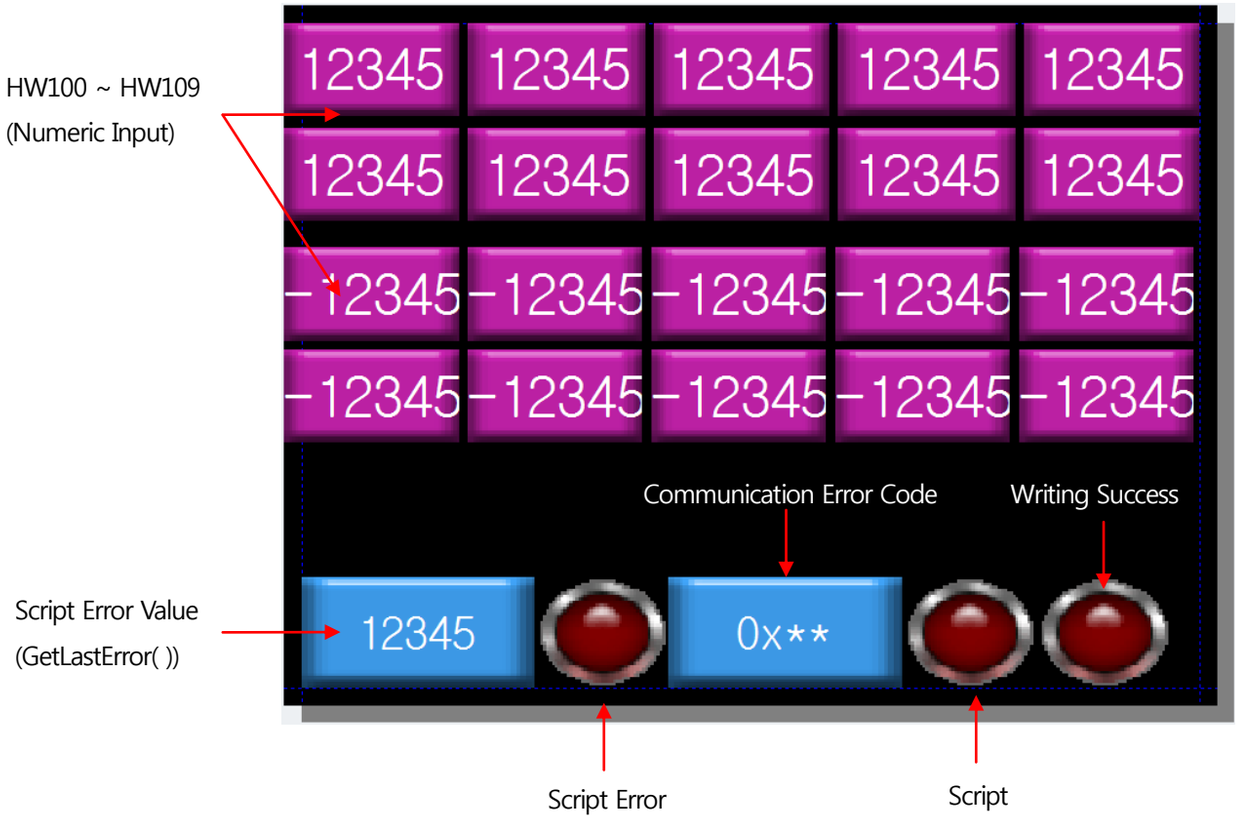
5. Some notes for making word reading operation.

(1) Word reading operation is similar to the word writing operation, but the part to process response is different.

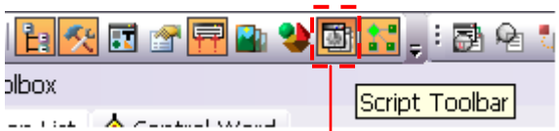
```
62
63    Command[nPos++] = 0;                    //No. of register high
64    Command[nPos++] = 10;                   //No. of register low
65
66    Command[nPos++] = 20;                   //Number of Bytes
67
68  for(i=0;i<20;i++){
69
70      Command[nPos++] = TX_Data[i];         //Writing byte data
71
72  }
73
74  @[X:=Error] = !CRC16 ( &Command[0], 27, &Checksum ); // CRC Calculation, Calculating Command[0] ~ [5] Checksum
75  Command[nPos++] = LOBYTE ( Checksum ); // Low Checksum byte
76  Command[nPos++] = HIBYTE ( Checksum );  // High Checksum byte
77
78  SentSize = WriteToIO ( 0, &Command[0], nPos );// Transmit Tx data on Channel #0, Return Value is as same as number of transmitted data byte
79
80  if(SentSize) //Run when the number of transmitted data is not 0 - Transmission Success
81  {
82      Sleep( 50 );                                    // Waiting response, 50ms is random setting value
```

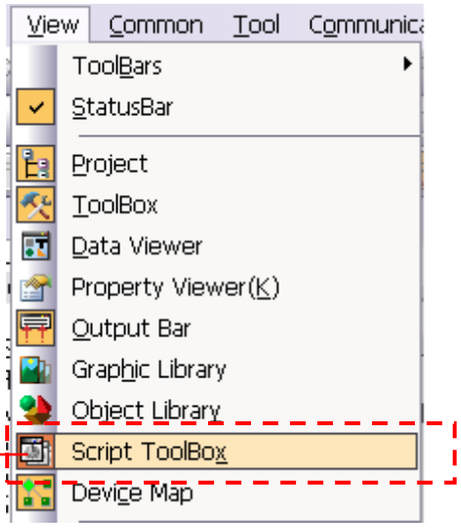
- In line 71 MAKEWORD function is used to save word because received buffer(Response) is based on bytes. The MAKEWRORD function returns the value from two bytes.
- In line 72 MAKEWORD function saves a word value in HW100 device.

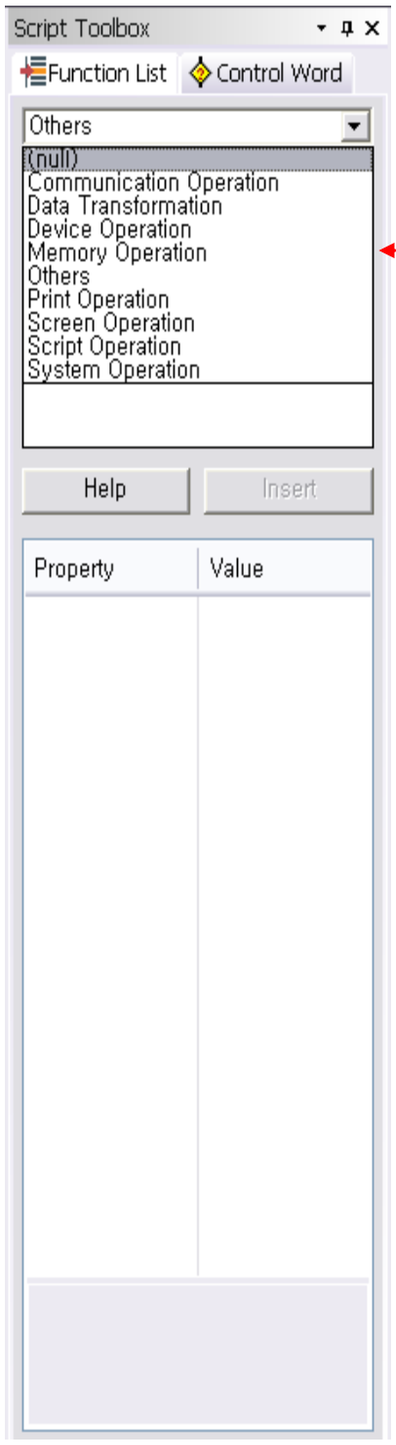
6. Screen data configuration

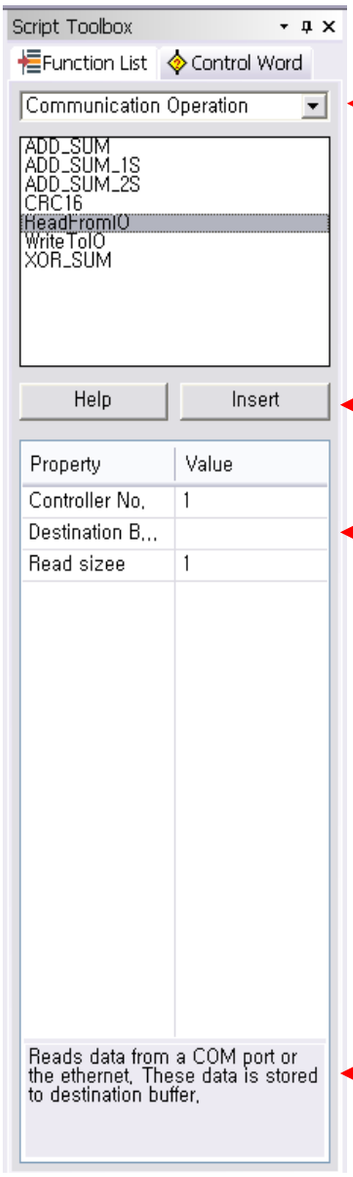


7. Tips for using script!!!
(1) Using Script Toolbox









Function

Add function when click Insert

Input Function (argument)

Explanation of function and Return Value